Application Development Guide

Version 8.14

SC18-7185-01

Application Development Guide

Version 8.14

SC18-7185-01

Note:
Before using this information and the product it supports, read the general information under "Notices" on page 349.

Second Edition (October 2003)

This edition applies to Version 8.1 of DB2 Everyplace (product number 5724-D04) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC18-7185-00

Order publications through your IBM representative or the IBM branch office serving your locality or by calling 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.

Table of contents

Part 1. Introduction

Chapter 1. DB2 Everyplace product overview

This section provides an introduction to DB2 Everyplace, a description of the components that make up the DB2 Everyplace solution, and an example of a typical DB2 Everyplace scenario. This section contains the following topics:

What is DB2 Everyplace?

DB2 Everyplace is part of IBM's solution for pervasive computing. With DB2 Everyplace, mobile professionals (such as sales people, inspectors, auditors, field service technicians, doctors, realtors, and insurance claim adjusters) can keep in touch with vital data that they need when away from the office.

Organizations are now able to deliver their DB2 enterprise data to mobile and embedded devices. With DB2 Everyplace, you can access and perform updates to a database on your mobile device. With DB2 Everyplace Sync Server, you can synchronize data from the mobile device to other data sources in your enterprise. The File Adapter capability enables you to distribute files and applications to mobile users.

The DB2 Everyplace database is a relational database that resides on your mobile device. To access data on the mobile device, you can write applications using rapid application development tools, the supported set of DB2 Call Level Interface (CLI) functions, Java Database Connectivity (JDBC) methods, or ADO.NET methods.

Components of the DB2 Everyplace solution

The DB2 Everyplace solution has the following key features and components:

- The DB2 Everyplace mobile database.
- The DB2 Everyplace Sync Server.
- The DB2 Everyplace Sync Client.
- The DB2 Everyplace Mobile Application Builder.
- The DB2 Everyplace sample applications.

The DB2 Everyplace mobile database

This database resides on the mobile device. The mobile database is included with DB2 Everyplace Database Edition, DB2 Everyplace Enterprise Edition, and DB2 Everyplace Software Development Kit. Another component associated with the mobile database is:

- The sample application (engine side)

The DB2 Everyplace mobile database is available for:

- Palm OS
- Symbian OS
- Windows CE/Pocket PC
- Win32 (Windows® 95, Windows® 98, Windows® NT®, Windows® 2000®, and Windows® XP®)
- QNX Neutrino, Linux, and embedded Linux devices.

DB2 Everyplace also supports MIDP mobile devices that use the MIDP database.

The DB2 Everyplace Sync Server

DB2 Everyplace Sync Server is included with DB2 Everyplace Enterprise Edition. Other important components that are associated with the Sync Server include:

- The DB2 Everyplace Mobile Device Administration Center
- The sample applications (server side)

You can synchronize data and applications between DB2 Everyplace mobile devices and enterprise data sources using the DB2 Everyplace Sync Server and DB2 Everyplace Sync Client.

Data synchronization can be bi-directional or uni-directional. Data can be updated at the DB2 Everyplace mobile device or the enterprise database. For example, users could download a subset of data from a DB2 for z/OS database to a DB2 Everyplace database on the mobile device, view the data, make changes to the data, and then synchronize the changed data back to the z/OS server. The DB2 Everyplace Sync Server also provides a mechanism for conflict resolution.

The DB2 Everyplace Sync Server provides an administration tool that helps you manage and deliver synchronization services to groups of users with similar data synchronization needs. More information about the Mobile Devices Administration Center is available in the Sync Server Administration Guide.

DB2 Everyplace Sync Server supports synchronizing relational data with any data source that has a JDBC interface, such as DB2 Universal Database.

DB2 Everyplace Sync Server supports synchronizing relational data with the following data sources:

- DB2 Universal Database for z/OS
- DB2 Universal Database for iSeries
- DB2 Universal Database for Linux, UNIX and Windows
- Any data sources with a JDBC interface

The DB2 Everyplace Sync Client

The DB2 Everyplace Sync Client is included with DB2 Everyplace Enterprise Edition.

The DB2 Everyplace Sync Client, which runs on mobile devices, is comprised of applications that work with the DB2 Everyplace Sync Server. It handles bi-directional synchronization of enterprise relational data with the DB2 Everyplace mobile database on the mobile device. The mobile device also manages operations related to file subscriptions for easy distribution of mobile applications to the device and can execute stored procedures stored on a DB2 database.

The Sync Client is available for the following operating systems:

- Palm OS
- Symbian OS
- Windows CE/Pocket PC
- Win32 (Windows® 95, Windows® 98, Windows® NT®, Windows® 2000®, and Windows® XP®)
- QNX Neutrino, Linux, and embedded Linux devices

For information about the Application Programming Interfaces (APIs) provided with the Sync Client, see *DB2 Everyplace Application Development Guide*.

The DB2 Everyplace Mobile Application Builder

The DB2 Everyplace Mobile Application Builder is included with the Software Development Kit and is also downloadable from the IBM Web site.

You can use the DB2 Everyplace Mobile Application Builder to develop DB2 Everyplace applications for Palm OS, WinCE, Symbian OS, and other platforms that support a user interface and a Java Virtual Machine. With the Mobile Application Builder, you can build applications without writing a single line of code. For information on how to get the Mobile Application Builder, visit the DB2 Everyplace Web site.

Other development tools include WebSphere Studio Device Developer, Visual Age Micro Edition, Metrowerks CodeWarrior, and the GNU Software Developer's Kit.

The DB2 Everyplace sample applications

The sample applications provide an example of an application using DB2 Everyplace. You can use the Visiting Nurse sample application to quickly test bi-directional synchronization between the mobile database and the Sync Server. The sample applications has two parts, one that runs on the Sync Server and another that runs on the mobile database. This mobile database sample application demonstrates the database engine functionality in stand-alone environment. When the Sync Server sample application and the database engine sample application are used together, they work as a complete application that invokes all components of the DB2 Everyplace.

The IBM Sync is also a sample application which demonstrates how to use the DB2 Everyplace Sync Client API to synchronize tables of the subscriptions defined in MDAC.

The Command Line Processor is an application development tool provided as an example application using DB2 Everyplace on platforms with a command line interface. The Command Line Processor is used for the DB2 Everyplace database on mobile devices. It is not used by the Sync Server.

The SQL statements supported by DB2 Everyplace enable you to create and drop a table and index, and delete, insert, and update the rows of a table.

See the *DB2 Everyplace Application Development Guide* for more information about supported SQL statements.

An example DB2 Everyplace scenario

Insurance claims adjusters are responsible for inspecting the damaged property of customers who file claims. In many companies, the adjuster visits the claimant's property, fills out paper forms to validate or refute the claim, and assesses the amount of the damages to be paid to the claimant. Later, when the adjuster returns to the office, the forms are manually entered into the company's computer system in a tedious and expensive process.

Equipping the adjusters with a mobile device running a DB2 Everyplace application can considerably streamline this process. Using their mobile devices wherever they are, the adjusters can access their inspection schedule, route, and claimant policy information. The adjusters can also complete the adjustment form on the mobile device. When the adjusters return to the office, they can synchronize the data on their mobile devices with the company's computer system by uploading the new adjustment form data to the company's enterprise database. If the adjusters need information in the field, they can synchronize the data on their mobile devices with the company's computer system immediately via modem. The claims adjustment process can now be completely paper free, which translates into significant cost savings for the insurance company. Claims are also settled faster because adjusters can have instant access to their company's enterprise databases.

Part 2. Developing DB2 Everyplace applications

Chapter 2. Developing DB2 Everyplace C/C++ applications

This chapter contains information about developing C/C++ applications for DB2 Everyplace. This chapter contains the following sections:

Developing DB2 Everyplace C/C++ applications

To develop a DB2 Everyplace application using C/C++, you use the DB2 Everyplace CLI/ODBC interface. This topic provides an overview of the tasks you must complete to develop C/C++ applications with DB2 Everyplace.

To develop DB2 Everyplace applications using C/C++::

1. Install DB2 Everyplace on the development workstation. See the *DB2 Everyplace Installation and User's Guide* for detailed instructions.
2. Define the application and its data requirements.

   Determine what data the end user needs to see or change, and how that data is retrieved, stored, and updated in the DB2 Everyplace database.
3. Understand the DB2 CLI interface and determine what DB2 CLI functions to use in the application.
4. Write a C/C++ application program using the DB2 CLI functions supported in DB2 Everyplace.
5. Prepare, compile, and link the application code with the DB2 Everyplace header files and operating system library.
6. Test the application:
   a. Copy the DB2 Everyplace libraries to the emulator or device for your operating system.
   b. Test the application on a device or an emulator, if applicable.

Related concepts:

- Chapter 13, "The sample C/C++ applications," on page 87

Related reference:

- "Supported C/C++ development tools"
- "C/C++ supported operating systems" on page 11
- "DB2 CLI function summary" on page 182

Supported C/C++ development tools

You can write a C/C++ application using the DB2 CLI functions supported by DB2 Everyplace.

Supported standard C/C++ development tools for the supported operating systems include:

Palm OS

You can use:

- DB2 Everyplace Mobile Application Builder. For information on Mobile Application Builder, visit the DB2 Everyplace Web site at http://www.ibm.com/software/data/db2/everyplace/
- GNU Software Developer's Kit.
- Metrowerks CodeWarrior for Palm Computing Platform. This commercial development environment allows you to create C/C++ programs for the Palm OS operating system using a Windows workstation.

Recommendation: Register application creator IDs with Palm, Inc. to avoid collisions with other Palm OS applications. The DB2 Everyplace tables and applications have creator IDs like IBDB or DB2*x*, where *x* is a letter from a to z. For more information on creator IDs, go to the following Web site:

http://www.palmos.com/dev/

Symbian OS Version 6.0

You can use Microsoft Visual C++, Version 6, together with the Symbian Version 6.0 C++ Software Developer's Kit (SDK), to develop your applications.

Recommendation: Obtain UIDs from Symbian OS to include in your project file. You can obtain these IDs from the SDK or the following Web site:

http://www.symbian.com/developer/techlib/papers/tn_uid/uidinfo.html

Symbian OS Version 7.0

You can use Metrowerks CodeWarrior for Symbian, together with the Symbian OS Version 7.0 SDK, to develop your applications.

Windows CE

You can use Microsoft Embedded Visual Tools 3.0 to develop your applications for Pocket PC 2000/2002. You can use Microsoft Embedded Visual Tools 4.0 to develop native C/C++ applications for .NET devices.

Windows NT and Windows 2000 operating systems

You can use Microsoft Visual C++ to develop your applications. You can use Microsoft Visual Studio .NET to develop managed .NET applications.

QNX Neutrino

You can use Metrowerks CodeWarrior for QNX Neutrino or the QNX Neutrino Software Developer's Kit (SDK) to develop your applications.

Linux You can use your embedded Linux distribution's cross platform development tools to develop your applications. The embedded Linux kernel needs to have support for ELF binaries enabled.

Related concepts:

- Chapter 13, "The sample C/C++ applications," on page 87

Related tasks:

- "Developing DB2 Everyplace C/C++ applications" on page 9

C/C++ supported operating systems

The C/C++ interface is fully supported on the following operating systems:

- Palm OS
- Symbian OS
- Windows CE® for Pocket PC
- Win32 (Windows 95, Windows 98, Windows NT, Windows 2000, and Windows XP)
- QNX Neutrino
- Linux and embedded Linux

Related concepts:

- Chapter 13, "The sample C/C++ applications," on page 87

Related tasks:

- "Developing DB2 Everyplace C/C++ applications" on page 9

Related reference:

- "Supported C/C++ development tools" on page 9
- "DB2 CLI function summary" on page 182

Preparing, compiling, and linking a C/C++ project

This task is part of the larger task of Developing DB2 Everyplace applications using C/C++. When you complete the steps for Preparing, compiling, and linking a C/C++ project, return to "Developing DB2 Everyplace C/C++ applications" on page 9.

Procedure:

DB2 Everyplace includes header files and operating system library files for application development.

To prepare a project file and compile and link a DB2 Everyplace application using the correct complier:

1. Create a project file. This procedure varies depending on the development tools and operating system that you are developing for.
2. Include the following DB2 Everyplace header files in the project. The header files contain the constants, data types, and C/C++ function prototypes provided with DB2 Everyplace. The header files are:

   ```
   \db2everyplace\Clients\include\sqlcli.h
   \db2everyplace\Clients\include\sqlcli1.h
   \db2everyplace\Clients\include\sqlext.h
   \db2everyplace\Clients\include\sqlsystm.h
   ```

3. Include any header files specific to you application.
4. Include the appropriate DB2 Everyplace library in the project.

   The following table summarizes the DB2 Everyplace libraries and lists additional information for each operating system.

Table 1. DB2 Everyplace libraries

Operating system	Required library files and additional information
Palm OS	\db2everyplace\clients\palmos\database\DB2e.libOptional: Increase the stack size to 8 KB. The default is 4 KB. Palm OS applications have a limited default application stack size. Depending on the application, you may encounter a stack overflow problem at run time. To avoid this problem, specify a larger stack size in the palm-pref.r file that is included with DB2 Everyplace. Follow the instructions in the palm-pref.r file and include it in the project file. If you are developing an application using PRC-Tools, add stack=0x8000 in the .def file for your application. For example: application {"MyApplicationName" APID stack=0x8000 }
Symbian OS v6	Emulator applications: \db2everyplace\clients\symbian6\database\wins\DB2e.lib Device applications: \db2everyplace\clients\symbian6\database\armi\DB2e.lib
Symbian OS v7	Emulator applications: \db2everyplace\clients\Symbian7\database\wins\DB2e.lib Device applications: \db2everyplace\clients\Symbian7\database\armi\DB2e.lib
Windows CE	ARM processor: • V3.00 \db2everyplace\clients\wince\database\wce300\armrel\DB2e.lib • V4.00 \db2everyplace\clients\wince\database\wce400\ARM4VRel\DB2e.lib MIPS processor: • V3.00 \db2everyplace\clients\wince\database\wce300\mipsrel\DB2e.lib • V4.00 \db2everyplace\clients\wince\database\wce400\MIPSIVRel\DB2e.lib SH3 processor: • V3.00 \db2everyplace\clients\wince\database\wce300\sh3rel\DB2e.lib • V4.00 \db2everyplace\clients\wince\database\wce400\SH3Rel\DB2e.lib Windows CE emulator: • V3.00 \db2everyplace\clients\wince\database\wce300\x86emrel\DB2e.lib (for Pocket PC emulator) \db2everyplace\clients\wince\database\wce300\x86rel\DB2e.lib (for Pocket PC 2002 emulator) • V4.00 \db2everyplace\clients\wince\database\wce400\emulatorRel\DB2e.lib (for WinCE.NET emulator) Verify that UNICODE is enabled for the project. Add UNICODE and _UNICODE to the **Preprocessor Definition** of the Project Settings. XScale processor: • v3.00 \db2everyplace\clients\wince\database\wce300\xscale\DB2e.lib
Win32	\db2everyplace\clients\Win32\database\x86\DB2e.lib
Neutrino	libdb2e.so This file is located in the \db2everyplace\clients\neutrino\database*proc*\ directory.
Linux	libdb2e.so This file is located in the /db2everyplace/Clients/Linux/database/*proc* directory.

5. Optional: Define the macro UNICODE and _UNICODE in your project file to get UNICODE support.

   See "DB2 Everyplace UNICODE support" on page 342 for more information on UNICODE

6. Compile the project and link the object code with the appropriate DB2 Everyplace library.

   Many of the application development tools provide automatic compiling and linking from within a integrated development environment. For additional information on compiling and linking a project, see the documentation included with your application development software.

Related concepts:

- Chapter 13, "The sample C/C++ applications," on page 87

Related tasks:

- "Testing a C/C++ application"

Related reference:

- "Supported C/C++ development tools" on page 9
- "C/C++ supported operating systems" on page 11
- "DB2 CLI function summary" on page 182
- "DB2 Everyplace UNICODE support" on page 342

Testing a C/C++ application

This task is part of the larger task of Developing DB2 Everyplace applications using C/C++. When you complete the steps for Testing a C/C++ application, return to "Developing DB2 Everyplace C/C++ applications" on page 9.

Procedure:

To test an application:

1. Copy the DB2 Everyplace libraries to the emulator or device for your operating system. Without these files, a DB2 Everyplace application will fail to load. The following table summarizes the required DB2 Everyplace files for each operating system.

Table 2. Required DB2 Everyplace files for testing

Operating system	Required files on device or emulator
Palm OS	\db2everyplace\clients\palmos\database\DB2eCat.prc \db2everyplace\clients\palmos\database\DB2eCLI.prc \db2everyplace\clients\palmos\database\DB2eComp.prc \db2everyplace\clients\palmos\database\DB2eRunTime.prc \db2everyplace\clients\palmos\database\DB2eDMS.prc
Symbian OS Version 6.0	For emulator testing, copy the file \db2everyplace\clients\symbian6\database\wins\DB2e.dll to each of the following emulator directories: \EPOCROOT%EPOC32\Release\wins\udeb\ (for debug emulator) \EPOCROOT%EPOC32\Release\wins\urel\ (for release emulator) For device testing, install the following file using the PsiWin connection software: \db2everyplace\clients\symbian6\database\armi\DB2e.sis

Table 2. Required DB2 Everyplace files for testing (continued)

Operating system	Required files on device or emulator

Windows CE	Install the appropriate library for your operating system.
	
	ARM processor:
	• V3.00
	`db2everyplace\clients\wince\database\wce300\armrel\DB2e.dll`
	
	MIPS processor:
	• V3.00
	`\db2everyplace\clients\wince\database\wce300\mipsrel\DB2e.dll`
	
	SH3 processor:
	• V3.00
	`\db2everyplace\clients\wince\database\wce300\sh3rel\DB2e.dll`
	
	Windows CE emulator:
	• V3.00
	For Pocket PC emulator:
	`db2everyplace\clients\wince\database\wce300\x86emrel\DB2e.dll`
	
	For Pocket PC 2002 emulator:
	`\db2everyplace\clients\wince\database\wce300\x86rel\DB2e.dll`

Win32	Copy `\db2everyplace\clients\win32\database\x86\DB2e.dll` to either the current directory of the application or the PATH environment variable of the system.
Neutrino	`/db2e/database/x86/libdb2e.so (for x86 processor type)` and `/db2e/database/strongarm/libdb2e.so (for strongarm processor type)`
Linux	`/db2e/database/x86/libdb2e.so (for x86 processor type)` and `/db2e/database/strongarm/libdb2e.so (for strongarm processor type)`

2. **For Linux and Neutrino**: Add `libdb2e.so` to the library search path, using one of the following methods:
   - Copy `libdb2e.so` to a directory that is in the library search path. This might require root permissions.
   - Copy `libdb2e.so` to another directory, and add that directory to the library search path. Adding a directory to the library search path permanently requires an entry in `/etc/ld.config`. Temporarily adding a directory to the library search path can be done by setting the LD_LIBRARY_PATH environment variable appropriately.

     For example, type the following command (bash, with `libdb2e.so` in the current directory): `export LD_LIBRARY_PATH=`
3. Load the files for the application being tested. For example, to test the Visiting Nurse sample application on Palm OS, load the `NurseInit.prc` and `Nurse.prc` files.
4. Test the application.

Related concepts:

• Chapter 13, "The sample C/C++ applications," on page 87

Related tasks:

• "Preparing, compiling, and linking a C/C++ project" on page 11

Related reference:

- "Supported C/C++ development tools" on page 9
- "C/C++ supported operating systems" on page 11
- "DB2 CLI function summary" on page 182

Developing DB2 Everyplace Sync Client applications using C/C++

This topic provides an overview of how to develop DB2 Everyplace Sync Client applications using C/C++ based on the IBM Sync Client C-API for Version 8.1. "IBM Sync Client C-API function summary" on page 296 provides the function specifications for all the C API functions

Prerequisites:

Install DB2 Everyplace on the development workstation. See the *DB2 Everyplace Installation and User's Guide* for details.

Procedure:

To develop a DB2 Everyplace Sync Client application using C/C++:

1. Define the synchronization application including:
   - the data it will be synchronizing;
   - the operations allowed;
   - the users and the user groups;
   - data security (for example, data encryption over the wire and local data encryption)

     See the *DB2 Everyplace Sync Server Administration Guide* for more details about the definition of the data to be synchronized and the administration of the users.
2. Include the DB2 Everyplace Sync Client header file (`isyncore.h`) in the C application programs, and use the DB2 Everyplace Sync Client C API functions following the function specifications.
3. Prepare, compile, and link the application code with the DB2 Everyplace Sync Client operating system libraries, `isyncconf` and `isyncore`.
4. Test the application:
   - Install the DB2 Everyplace libraries on the emulator or device for your operating system.
   - Test the application on an emulator, if applicable.
   - Test the application on a device.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related reference:

- "Supported C/C++ development tools" on page 9

Chapter 3. Developing DB2 Everyplace Java applications

This chapter describes how to develop DB2 Everyplace Java applications. The topics covered are:

- "JDBC interface supported operating systems"
- "Developing DB2 Everyplace Java applications"

JDBC interface supported operating systems

The JDBC interface is supported on the following operating systems:

- Palm OS
- Symbian OS
- Windows CE® for Pocket PC
- Win32 (Windows 95, Windows 98, Windows NT, Windows 2000, and Windows XP)
- QNX Neutrino
- Linux and embedded Linux

Developing DB2 Everyplace Java applications

To develop a DB2 Everyplace application using Java, you can use the Java Software Developer's Kit together with the DB2 Everyplace Java Database Connectivity (JDBC) interface for Java.

This topic provides a high-level overview of the tasks required to develop Java applications with DB2 Everyplace.

Restrictions:

DB2 Everyplace does not support multitasking on Symbian. In order to access a database from a second thread, the Connection object from the first thread must be closed before the connection can be established in the second thread. The same Connection object cannot be shared between threads.

Prerequisites:

A Java application that accesses DB2 Everyplace uses the DB2 Everyplace JDBC driver. Install Java and JDBC on your workstation if you have not already done so.

Procedure:

To develop DB2 Everyplace applications using Java:

1. Import the `java.sql` package and any other necessary Java classes.
2. Load the DB2 Everyplace JDBC driver. The class name is `com.ibm.db2e.jdbc.DB2eDriver`.
3. Connect to the database using a URL of the form `jdbc:`*`subprotocol:subname`*. The DB2 Everyplace *subprotocol* is `db2e`. If the database is in `c:\dir1\dir2`, use the URL `jdbc:db2e:c:/dir1/dir2/`. You may also use a relative path for *`subname`*.
4. Create a Statement object.

5. Access the database (your application logic goes here):
   - Execute a SQL statement using the Statement object.
   - Retrieve data from the returned ResultSet object (if the SQL statement you executed is a query).
6. Release database and JDBC resources by closing the ResultSet, Statement, and Connection objects.

Related concepts:

- "Overview of the sample Java applications" on page 89

Related tasks:

- "Compiling and running sample Java applications on Palm OS targets" on page 91
- "Compiling and running sample Java applications on non-Palm OS targets" on page 95

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267

Chapter 4. Developing Java Sync Client applications

This chapter describes how to develop DB2 Everyplace Sync Client Java applications. The topics covered are:

- "Java Sync API supported operating systems"
- "IBM Java Sync APIs"

Java Sync API supported operating systems

Java Sync APIs are available on the following operating systems:

- Win32
- Symbian OS
- Windows CE (with MIPS and ARM processors)
- Palm OS
- Linux
- QNX Neutrino

IBM Java Sync APIs

You can create Java applications using Java Database Connectivity (JDBC) and the Java interface in order to integrate DB2 Everyplace Database and Sync Server functionality.

For detailed information about the interfaces, classes, and exceptions that are supplied with the IBM Java Sync APIs supported by DB2 Everyplace, refer to the Javadoc documentation in the `Clients\javadoc` directory.

Related concepts:

- "Overview of the sample Java applications" on page 89

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Overview of DB2 Everyplace synchronization providers

This topic describes the Sync Client Java-API that is supported by DB2 Everyplace. The API is a set of libraries that allow developers to build applications that synchronize data bi-directionally between DB2 Everyplace and enterprise relational databases. It works in conjunction with the DB2 Everyplace Sync Server to simplify the synchronization of relational data and files. The Sync Server provides conflict resolution and manages the movement of data to and from the mobile PDA, embedded or MIDP 1.0 enabled device.

The Sync Client Java API consists of two types of synchronization providers:

- "DB2 Everyplace native synchronization" on page 20
- "DB2 Everyplace Java synchronization" on page 26

Information on how to create Java applications on the client device based on these providers are provided in the sample files.

Related tasks:

- "Installing the JNI-based native synchronization provider"
- "Installing and verifying the trap-based native synchronization provider" on page 23

Related concepts:

- "The sample Java native synchronization applications" on page 113
- "The sample Java MIDP synchronization applications" on page 117

DB2 Everyplace native synchronization

:

The native synchronization providers provide the Java interface that invokes the native synchronization client libraries.

Note: The native synchronization providers do not support thread safety in this release, it is the application's responsibility to coordinate thread synchronization.

There are two types of DB2 Everyplace native synchronization providers:

- Java Native Interface (JNI) - based native synchronization provider
- Palm OS trap-based native synchronization provider

Related tasks:

- "Installing the JNI-based native synchronization provider"
- "Installing and verifying the trap-based native synchronization provider" on page 23

Related concepts:

- "Overview of DB2 Everyplace synchronization providers" on page 19

Installing DB2 Everyplace native synchronization providers

This chapter describes how to install DB2 Everyplace native synchronization providers. The topics covered are:

- "Installing the JNI-based native synchronization provider"
- "Installing and verifying the trap-based native synchronization provider" on page 23

Installing the JNI-based native synchronization provider

The JNI-based sync provider works with Java VM that supports the Java Native Interface. .

This provider is supported on the following operating systems:

- Win32
- Symbian Release 6 (for Nokia 9210/9290 Communicator devices)
- Symbian Release 7 (for Sony Ericsson P800 devices)
- Windows CE (for Pocket PC devices)
- Linux

- QNX Neutrino

Prerequisites:

The JNI-based synchronization provider requires the following files:

- the `isync4j.jar` file
- the following native sync client binaries:
  - `isyncore.dll`
  - `isyncconf.dll`
  - `imsadb2e.dll`
  - `imsafile.dll`
  - `imsaconfig.dll`
  - `wbxmllib.dll`
  - `isync4j.dll`
  - `isyncxpt.dll`

If your application is using the JNI-based native synchronization provider, you must import the following isync4j Java packages:

- `com.ibm.mobileservices.isync`
- `com.ibm.mobileservices.isync.event`
- `com.ibm.mobileservices.isync.sql`

Verify that the following software is installed on your system:

- DB2 Everyplace Sync Server Version 8
- DB2 Everyplace Sync Client Libraries Version 8

  See the *DB2 Everyplace Installation and User's Guide* for more information.
- Java VM that supports the Java Native Interface

Read the following topics for more information on how to install the JNI-based synchronization provider on each of the supported operating systems:

- "Installing the JNI-based synchronization provider on Win32"
- "Installing the JNI-based synchronization provider on Nokia 9210/9290 Communicator devices using Symbian V6" on page 22
- "Installing the JNI-based synchronization provider on Windows CE" on page 22

Installing the JNI-based synchronization provider on Win32

To install the JNI-based synchronization provider on a Win32 operating system, you must compile and run the ISyncSample program. JNI-based implementations for Win32 devices have been tested on Sun MicroSystems Java™ VM and the IBM Java™ 2 Standard Edition Developer Kit .

Procedure:

1. Compile the ISyncSample program.
   a. Change the PATH system variable to include the following directories:

      ```
      <DB2e_InstDir>\Clients\Win32\database\x86
      <DB2e_InstDir>\Clients\Win32\sync
      ```

   b. Change the CLASSPATH variable to include the `isync4j.jar` file:

      ```
      <DB2e_InstDir>\Clients\Win32\Sync\isync4j.jar
      ```

c. Compile the sample files included in the <DB2e_InstDir>\Clients\clientapisample\Java_API directory. For example:

```
javac ISyncSample.java
```

2. Edit the isyncdb2e.properties file to specify the server URL, user, and password.
3. Run the ISyncSample program.
   a. Type the following command:

```
java.exe ISyncSample <property file>
```

   where <*property file*> is the property file for your client database. For example:

```
java.exe -classpath .; isync4j.jar ISyncSample isyncdb2e.properties
```

Related tasks:

- "Installing the JNI-based native synchronization provider" on page 20

Installing the JNI-based synchronization provider on Nokia 9210/9290 Communicator devices using Symbian V6

To install the JNI-based synchronization provider on Nokia 9210/9290 Communicator devices using Symbian V6, you must compile and run the ISyncSample program. JNI-based implementations for Nokia 9210/9290 devices using Symbian V6 have been tested on Symbian OS 6.0 PersonalJava JVM.

Procedure:

1. Edit and compile the ISyncSample program on your workstation.
   a. Edit ISyncSample.java to take isyncdb2e.properties as a parameter.
   b. Compile ISyncSample.java with isync4j.jar in your classpath by typing the following command:

```
javac -classpath isync4j.jar ISyncSample.java
```

   c. Edit isyncdb2e.properties to specify the server URL, user, and password.
2. Run the ISyncSample program.
   a. Make sure the DB2 Everyplace database and Sync Client libraries are installed on the device.
   b. Copy the ISyncSample.class and isyncdb2e.properties files to the C:\System\Apps\ISync directory on the device.
   c. Using the Windows File Manager, locate and select the isync4j.jar file. Click **Enter**. Use the Redirect program, which is installed on your Nokia device, to transfer the output from the Java program and then either display this output on your console or write this output to a file.

Related reference:

- "Supported JSP Version 1.1 subsets" on page 41
- "IBM custom tags for JSP application database access" on page 45

Installing the JNI-based synchronization provider on Windows CE

To install the JNI-based synchronization provider on Windows CE operating systems you must compile and run the ISyncSample program. The JNI-based Sync Provider for Windows CE devices have been tested on OTI J9 JVM.

Procedure:

1. Compile the ISyncSample program on your workstation.
   a. Type the following command to compile `ISyncSample.java` with `isync4j.jar` in the classpath:

      ```
      javac -classpath isync4j.jar ISyncSample.java
      ```

   b. Edit `isyncdb2e.properties` to specify the server URL, user, and password.
2. Run the ISyncSample program.
   a. Verify that the J9 Java Virtual Machine (JVM) run-time environment is installed on the device (for example, `\wsdd`). In addition, the DB2 Everyplace and Sync Client libraries must be installed.
   b. Copy the `ISyncSample.class` and `isyncdb2e.properties` files to the device (for example, `\`).
   c. Use one of the following two methods to invoke the ISyncSample program with `isync4j.jar` in the classpath.

      Java console

      Type the following command:

      ```
      j9.exe -bp:\wsdd\classes.zip -cp:\wsdd;\Windows\isync4j.jar ISyncSample
        <property file>
      ```

      For example:

      ```
      j9.exe -bp:\wsdd\classes.zip -cp:\wsdd;\Windows\isync4j.jar
       ISyncSample isyncdb2e.properties
      ```

      Windows shortcut

      Create and edit a Windows shortcut called `ISyncSample.lnk` on your workstation. For example:

      ```
      255#"\wsdd\j9.exe" "-bp:\wsdd;\Windows\isync4j.jar;\wsdd\classes.zip"
      "ISyncSample" "isyncdb2e.properties"
      ```

      Enter the shortcut on a single line, and enclose each field in double quotation marks. The first field that you type *must* be the name of the executable. The files and directories that you specify must be fully qualified.
   d. Run the sample program, and verify that the synchronized data resides in the target directory as specified in the property file.

Related tasks:

- "Installing the JNI-based native synchronization provider" on page 20

Installing and verifying the trap-based native synchronization provider

The trap-based native synchronization provider is used with WebSphere Studio Device Developer's (WSDD) J9 JVM on the Palm OS platform only.

This topic describes how the DB2 Everyplace isync4j for Palm OS can be used with the J9's jclMidp (J2ME MIDP) configuration. This synchronization provider references the com.ibm.oti.palmos package, so it will only run on WSDD J9 JVM for PalmOS v1.5 or above.

The following table describes where the programs that are used to install the API on Palm devices are located, where %DSYINSTDIR% represents the install directory for DB2 Everyplace.

Directory	Description
%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/lib	The folder that contains the isync4j for Palm OS Java classes. These classes are imported during the implementation.
%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/sample	The folder that contains the source code for the sample isync4j application.
%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/bin/ISyncSample.prc	The sample isync4j application used with the J9 Palm OS CLDC library.

Prerequisites:

The trap based native synchronization provider requires the following Sync Client native shared libraries and DB2 Everyplace libraries:

- isyncore.prc
- isyncconf.prc
- imsaconfig.prc
- imsafile.prc
- imsadb2e.prc
- wbxmllib.prc
- isyncxpt.prc

In addition, you need to install the J9 Palm OS JVM binaries on your device.

If your application is using the trap-based native synchronization provider, you must import the following isync4j Java packages:

- com.ibm.mobileservices.isync
- com.ibm.mobileservices.isync.db2e.sti
- com.ibm.mobileservices.isync.event
- com.ibm.mobileservices.isync.sql

Verify that the following software is installed on your system:

- Palm OS Version 3.5 or later (with at least eight MB of memory)
- WebSphere Studio Device Developer (WSDD) Version 4.0
- DB2 Everyplace database for Palm OS Version 7.1 or later
- DB2 Everyplace Sync Client libraries Version 8.1 or later

After you have installed WSDD, you must set up a Palm OS target. To set up a Palm OS target, see the WSDD Development Environment & Tools Product Documentation, in the chapter called ″Getting Started with Palm OS Targets″. The WSDD documentation is located on the product CD-ROM in IBM\wsdd\wsdd4.0\doc\wsddCustomer.pdf. Finally, verify that WSDD is properly installed by building and running a WSDD sample application.

Procedure:

To verify proper WSDD installation:

1. Create a new project for the isync4j sample application:
   a. Open the Java Perspective in WSDD.
   b. Select **File –> New–> Other**.
   c. Select the wizard for J2ME for J9 and Create MIDlet Suite.
   d. Name the custom project, MIDlet name, and MIDlet class name in the MIDlet Suite Creation dialog.

e. Click **Next**.

f. Click **Next** again to go to Java Settings.

g. On the Java Settings, click on the **Libraries** tab and click **Create Folder...**. Type **lib** in the New Class Folder dialog.

h. Click **Finish**.

2. Import the DB2 Everyplace ISYNC4J Java classes and set up the build path.

   a. Click on the project in the **Packages** view, then click the menu item **File->Import...**

   b. Import the `%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/lib` folder and select `%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/lib` as the source directory.

   c. Expand the `lib` directory and select the checkbox for the com directory under `/lib`. Under **Select the destination for imported resources:** type the name of the project followed by /lib in the **Folder:** field. For example, if the project name is ISyncSample, then the field should appear as ISyncSample/lib.

   d. Click **Finish**.

   e. Expand the lib folder, you should see the following ISYNC4J Java packages:

      ```
      com.ibm.mobileservices.isync
      com.ibm.mobileservices.isync.db2e.sti
      com.ibm.mobileservices.isync.event
      com.ibm.mobileservices.isync.sql
      ```

3. Verify the isync4j library setup by building and running the sample application.

   a. Import the sample application.

      - Click on the src folder for the project in the **Packages** view, and then click **File >Import** from the main menu.
      - Import ISyncSample.java. Select `%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/samples/ISyncSample/` as the source directory, then select the checkbox for ISyncSample.java. Verify that the destination for imported resources is <project>/src.).

   b. Create a build file for the sample application.

      - In the editor, click the **in/exclusion** tab, then click **New**.
      - Enter **ISyncSample** for the main class, select **J9 for Palm 68k** for the platform. Click **Next**.
      - Enter the Creator id, and ISyncSample for the App Name. Click **Next** twice.
      - Select **Prc Application on PalmOS emulator**. Click **Finish**.

   c. Modify ISyncSample.jxeLinkOptions file.

      - Expand the **palm68k** folder for the project in the **Packages** view.
      - Double click `ISyncSample.jxeLinkOptions`.
      - In the editor, click the **in/exclusion** tab, then click **New**.
      - Enter **com.ibm.mobileservices.isync.db2e.sti.DB2eISyncProvider** for the Rule pattern, then click **OK**.
      - In the editor, click the **source** tab
      - Type **-vmOption -ms:15** to set the stack size.
      - Save your changes.

   d. Run the sample application.

- Click the **Run** icon in the menu, then select **Run –>Build –>Launch** from the build file.
- Select the target for the sample application, and click **Finish**.
- If there are no errors, the Palm OS emulator should start and run the application.

You can now create your own application. When you create a new application, include a new project name for the DB2 Everyplace isync4j in the build path for your project. After you create a build file for your application, modify its `jxeLinkOptions` file to meet the requirements of your application.

Related concepts:

- "Overview of DB2 Everyplace synchronization providers" on page 19

DB2 Everyplace Java synchronization providers

This chapter describes DB2 Everyplace Java synchronization providers. The topics covered are:

- "DB2 Everyplace Java synchronization"
- "DB2 Everyplace J2ME MIDP synchronization"
- "DB2 Everyplace Java Sync Client for Cloudscape" on page 28

DB2 Everyplace Java synchronization

:

The Java synchronization providers provide the Java interface that invokes the Java synchronization client libraries.

There are two types of DB2 Everyplace Java synchronization providers:

- "DB2 Everyplace J2ME MIDP synchronization"
- "DB2 Everyplace Java Sync Client for Cloudscape" on page 28

Related tasks:

- "Installing the JNI-based native synchronization provider" on page 20
- "Installing and verifying the trap-based native synchronization provider" on page 23

Related concepts:

- "Overview of DB2 Everyplace synchronization providers" on page 19

DB2 Everyplace J2ME MIDP synchronization

The J2ME MIDP ISync Client allows you to build applications that synchronize subscriptions to the MIDP Record Store Management System (RMS). The J2ME MIDP ISync Client is a set of libraries that work with the DB2 Everyplace Sync Server to simplify the synchronization of relational data between enterprise databases and MIDP 1.0 enabled devices. The Sync Server manages the movement of data to and from the MIDP device.

This topic includes the following information about the J2ME MIDP ISync Client:

- Required Web server software to install the J2ME MIDP ISync Client

- Required software and hardware to run the J2ME MIDP ISync Client on Motorola iDEN phones
- J2ME MIDP ISync Client installation directory layout

Required Web server software to install the J2ME MIDP ISync Client:

In order to install the J2ME MIDP Sync Client, you need one of the following two software products:

- WebSphere Application Server, Advanced Single Server Edition Version 4.x or later. You can download a free trail version of this software from the IBM Web site at http://www-3.ibm.com/software/webservers/appserv/advanced.html.
- Apache Tomcat Version 4.0.x or later. You can download a free copy of this software at http://jakarta.apache.org/tomcat/.

Required software and hardware to run the J2ME MIDP ISync Client on Motorola iDEN phones:

To install and run the MIDP synchronization provider on Motorola iDEN phones, you need the following hardware and software:

- Sun Microsystems JavaTM 2 Platform Micro Edition, Wireless Toolkit
- iDEN Update and data cable (for loading applications on the phone)
- Apache ANT
- RetroGuard Ofuscator
- isync4j Java packages (included with DB2 Everyplace)
  - com.ibm.mobileservices.isync
  - com.ibm.mobileservices.isync.midp
  - com.ibm.mobileservices.isync.event

J2ME MIDP ISync Client installation directory layout:

The J2ME MIDP ISync Client installation process creates four initial directories:

- `bin` - `%DSYINSTDIR%\Clients\Midp\bin`, contains a script for running the WTK emulator from the command line.
- `lib` - `%DSYINSTDIR%\Clients\Midp\lib` contains the MIDP ISync API jars, Servlet for MIDP, the FilterServlet.jar file, and sample MIDLets with associated JAD files.
- `docs` - `%DSYINSTDIR%\Clients\Midp\doc` contains the J2ME MIDP ISync Client Javadoc.
- `samples` - `%DSYINSTDIR%\Clients\Midp\samples`, contains the source code for the sample isync4j application where `%DSYINSTDIR%` is the install directory for DB2 Everyplace.

If you decide to re-compile the samples, you will find several build*Classes directories, which are used for pre-verification and obfuscation.

Related tasks:

- "Installing the JNI-based native synchronization provider" on page 20
- "Installing and verifying the trap-based native synchronization provider" on page 23

Related concepts:

- "Overview of DB2 Everyplace synchronization providers" on page 19

DB2 Everyplace Java Sync Client for Cloudscape

The DB2 Everyplace Java Sync Client for Cloudscape allows you to build applications that synchronize subscriptions to a Cloudscape database. The Java Sync Client for Cloudscape is a set of libraries that work with the DB2 Everyplace Java Sync Server to simplify the synchronization of relational data between enterprise databases and a Cloudscape client. The Sync Server manages the movement of data to and from the device.

This topic includes the following information about the Java Sync Client for Cloudscape:

- Required software to run the Java Sync Client for Cloudscape
- Java Sync Client for Cloudscape directory layout
- Setting the CLASSPATH environment variable

Required software to run the Java Sync Client for Cloudscape:

In order to run the Java Sync Client for Cloudscape, you need the following software products:

- DB2 Everyplace version 8.1.4 or higher
- The Cloudscape database

Java Sync Client for Cloudscape installation directory layout:

The Java Sync Client for Cloudscape files are in the following directories:

- `%DSYINSTDIR%\Clients\javaclient` contains the Cloudscape ISync API jar file.
- `%DSYINSTDIR%\doc\javadoc\javaclient` contains the Java Sync Client for Cloudscape Javadoc.

Setting the CLASSPATH environment variable:

To use the Java Sync Client for Cloudscape, set your CLASSPATH environment variable to include the following files:

- The Cloudscape jar files (`db2j.jar, db2jtools.jar`) from your Cloudscape installation.
- The Cloudscape ISync API jar file (`db2jisync.jar`).
- Optional: The sample applications (ISyncSample and GoISyncConsole).

Related tasks:

- "Compiling and running the GoISyncConsole sample Java synchronization application" on page 124

Chapter 5. Developing Visual Basic applications

This chapter provides information about developing DB2 Everyplace applications using Visual Basic. The topics covered are:

Developing DB2 Everyplace Visual Basic applications

To develop a DB2 Everyplace application in Visual Basic, you use the DB2 Everyplace CLI/ODBC interface. This topic provides a high-level overview of the tasks you must complete in order to develop Visual Basic applications with DB2 Everyplace.

Restrictions:

When you develop applications for DB2 Everyplace using Visual Basic, consider the following restrictions and requirements:

- Do not use the function SQLAllocHandleVer directly in the code of your application. SQLAllocHandleVer is used by DB2 Everyplace internally. If you use it in your application code, it might cause program failures.
- Debugging might not work because of the way Visual Basic loads and handles calls to functions inside a DLL.
- Visual Basic functions that call DB2 Everyplace functions in `db2e.dll` must have the statement "On Error Resume Next", otherwise the program will not work properly.

Procedure:

The basic steps to develop a DB2 Everyplace Visual Basic application are:

1. Create a new Visual Basic project.
2. Copy the file `db2ecli.bas` (from the DB2 Everyplace Visual Basic project directory) into your project folder, and insert the file into your Visual Basic project.
3. Copy `DB2e.dll` into your project folder. If you don't want to place `DB2e.dll` in your project folder, modify the path to `DB2e.dll` in the function declarations in the `db2ecli.bas` file.
4. Write your own application code. You can use the DB2 Everyplace sample Visual Basic program to help you.
5. Create the executable program for your application by selecting the menu item **File –> Make** *project*.

Related concepts:

- "Overview of the sample Visual Basic application" on page 103

Related reference:

- "DB2 CLI function summary" on page 182
- "Visual Basic Interface supported operating systems" on page 30

Visual Basic Interface supported operating systems

The Visual Basic Interface is fully supported on the following operating systems:

- Windows CE® for Pocket PC
- Win32 (Windows 95, Windows 98, Windows NT, Windows 2000, and Windows XP)

Chapter 6. Developing JSP applications

This chapter provides information about developing DB2 Everyplace applications using JavaServer Pages. The topics covered are:

- "JSP supported operating systems"
- "Developing DB2 Everyplace JSP applications"
- "DB2 Everyplace JSP support overview" on page 32

JSP supported operating systems

JSP support is available for the following operating systems:

- Win32 (Windows® NT® and Windows® 2000®)
- Windows CE® for Pocket PC
- Symbian OS

Related tasks:

- "Verifying JSP support on a Windows workstation" on page 33
- "Setting up for JSP development on a Windows CE device" on page 34

Related concepts:

- "DB2 Everyplace JSP support overview" on page 32
- Chapter 16, "The sample JSP applications," on page 109

Developing DB2 Everyplace JSP applications

DB2 Everyplace supports JavaServer Pages (JSP) to enable you to create DB2 Everyplace Web-based applications easily. The applications that you create are independent of the operating system and can run on mobile devices in a disconnected mode or while disconnected from a Local Area Network.

JSP technology provides a simplified and fast method to develop and maintain dynamic Web pages. The JSP technology separates the user interface from content generation so that you can create and update page layouts without changing the underlying dynamic content.

JSP uses JDBC technology. The Web-based applications that you create using JSP can access DB2 Everyplace databases through the DB2 Everyplace JDBC Driver.

Refer to the accompanying documentation in `<DB2Everyplace>\SDK\JSP\doc` for information on how to generate JSP pages that access DB2 Everyplace using WebSphere Studio.

Test your JSP application by following the steps in "Running a JSP application on a Windows workstation" on page 39. You should always run your JSP application on the workstation before transferring it to the device. Running the JSP application on the workstation does the necessary preprocessing of some of the application files.

Related tasks:

- "Verifying JSP support on a Windows workstation" on page 33
- "Setting up for JSP development on a Windows CE device" on page 34

Related concepts:

- "DB2 Everyplace JSP support overview"
- Chapter 16, "The sample JSP applications," on page 109

Related reference:

- "Supported JSP Version 1.1 subsets" on page 41
- "IBM custom tags for JSP application database access" on page 45

DB2 Everyplace JSP support overview

DB2 Everyplace JSP support consists of two components:

- The mini HTTP Web server
- The JSP processor

The mini HTTP Web server receives requests from a Web browser and sends responses back to the Web browser (using HTTP 1.1 as the protocol for requests and responses). The JSP processor parses a JSP file, generates corresponding Java source code, and compiles the source code. The Java source code may include JavaBeans that generate dynamic content when the JSP page is requested. When a JSP page is requested, the mini HTTP Web server executes the corresponding Java code, and sends the output back to the Web browser as the response to the request.

When you enter a URL such as `http://localhost/request.jsp` in a Web browser (where **request.jsp** is the JSP page that you are requesting), the Web browser sends the request to the mini HTTP Web server. The request is forwarded to the JSP processor if one of these conditions is true:

- There is no corresponding .class file for the JSP page (for example, if the JSP page is a newly created page).
- There is a corresponding .class file for the JSP page, but the timestamp on the JSP file is more recent than the timestamp on the .class file (for example, if the JSP page has been modified).

  Note: For **request.jsp**, the corresponding .class file is **_request_jsp_.class**.

If the request is forwarded to the JSP processor and the JSP file syntax is valid, the mini HTTP Web server sends output to the Web browser indicating that the JSP page is valid. Click the **request.jsp** link in the output to view the JSP page.

If the request is forwarded to the JSP processor and the JSP file syntax is invalid, the mini HTTP Web server sends diagnostic information to the Web browser.

If the request does not need to be forwarded to the JSP processor, the mini HTTP Web server executes the corresponding .class file for the JSP page and sends the output to the Web browser.

JSP application development should be done on a Windows workstation. It is important to test your JSP pages throughout development. The JSP processor will catch any syntax errors in a JSP page. After the errors are fixed, test the JSP page again by clicking the Refresh button in the Web browser. You may need to delete the files in the Web browser's Temporary Internet Files folder or cache before the changes to the JSP page are reflected. After you complete your application, you can transfer the application to a device and run it on the device.

Note: The JSP processor runs on the workstation and is not needed on the device.

Related tasks:

- "Transferring a JSP application to a Windows CE device" on page 37
- "Verifying JSP support on a Windows workstation"
- "Setting up for JSP development on a Windows CE device" on page 34

Related concepts:

- "Developing DB2 Everyplace JSP applications" on page 31
- Chapter 16, "The sample JSP applications," on page 109

Setting up for JSP development

This section provides information about how to set up to develop DB2 Everyplace applications using Java Server Pages. The topics covered are:

- "Verifying JSP support on a Windows workstation"
- "Setting up for JSP development on a Windows CE device" on page 34
  - "Installing the J9 JVM run-time environment on a Windows CE device" on page 34
  - "Installing and verifying JSP support on a Windows CE device" on page 35
- "Installing and verifying JSP support on a Symbian OS Version 6 device" on page 36

Verifying JSP support on a Windows workstation

Prerequisites:

Verify that the following software is installed on your Windows workstation:

- DB2 Everyplace Software Development Kit
- Java 2 Standard Development Kit, Standard Edition
  - Version 1.1.8 if your target is a Symbian OS Version 6 device
  - Version 1.2.2 or later for other targets
- Web browser
  - Internet Explorer Version 5.50 or later *or*
  - Netscape Navigator Version 6.2.1 or earlier

Procedure:

To verify that your Windows workstation is set up to use DB2 Everyplace JSP support:

- Run the Visiting Nurse sample JSP application:
  1. Start the mini HTTP Web server:
     a. Open a MS-DOS window.
     b. Change to the `<DB2Everyplace>\SDK\JSP\Win32` directory using the `cd` command.
     c. Type `runJspServer`.
  2. Open a Web browser to the following URL:
     http://localhost/VisitingNurse/schedule.jsp

     Alternatively, you can enter the URL http://localhost/ and browse to the VisitingNurse/schedule.jsp page.

If you successfully set up your workstation to use DB2 Everyplace JSP support, a Visiting Nurse Schedule table appears in the Web browser.

Related tasks:

- "Running a JSP application on a Windows workstation" on page 39

Related reference:

- "Supported JSP Version 1.1 subsets" on page 41
- "IBM custom tags for JSP application database access" on page 45

Setting up for JSP development on a Windows CE device

Prerequisites:

Verify that the following software is installed on your workstation:

- DB2 Everyplace Software Development Kit

Verify that the `\Windows` directory on your device contains the following files:

```
<DB2Everyplace>\Clients\WinCE\database\ver\proc\CryptoPlugin.dll
<DB2Everyplace>\Clients\WinCE\database\ver\proc\DB2e.dll
<DB2Everyplace>\Clients\WinCE\database\ver\proc\DB2eJDBC.dll
<DB2Everyplace>\Clients\WinCE\database\jdbc\db2ejdbc.jar
```

where *ver* is the version number of the Windows CE operating system on your device and *proc* is the processor type .

Verify that the following software is installed on your device:

- Web browser

Procedure::

1. Install the J9 runtime environment on the device.
2. Install and verify DB2 Everyplace JSP support on the device.

Related tasks:

- "Running a JSP application on a Windows CE device" on page 40

Related reference:

- "Supported JSP Version 1.1 subsets" on page 41
- "IBM custom tags for JSP application database access" on page 45

Installing the J9 JVM run-time environment on a Windows CE device

Currently, only StrongARM devices are supported on Pocket PC. If your device has a different processor type, you can try another JVM that supports JNI (for example, Sun PersonalJava, Insignia Jeode, NSIcom CrEme). If you use a JVM other than the J9 JVM, you must modify `\SDK\JSP\WinCE\MiniHttpServer.lnk` accordingly.

This installation of J9 allows you to run the sample JSP applications. You may need to install additional J9 files for your own applications.

This task is part of the main task of Setting up for JSP development on a Windows CE device. After you complete these steps, return to "Setting up for JSP development on a Windows CE device."

Procedure:

To install the J9 JVM run-time environment:

1. Install WebSphere Studio Device Developer v5.5 (WSDD) on your workstation. The evaluation version of WSDD is available for download at http://www.ibm.com/software/pervasive/products/wsdd/. In the steps below, <WSDD> refers to the directory where you installed WSDD.
2. In WSDD, use the Update Manager to install WCE Tooling for WSDD.
   a. Click **Help —> Software Updates —> Update Manager** to open the Install/Update Perspective.
   b. In the Features Updates view, expand the following nodes: **Sites to Visit -> WebSphere Custom Environment -> WebSphere Custom Environment**.
   c. Click on WCE Tooling for WSDD 5.5.0.
   d. In the Preview view, click the **Install** button and follow the steps to install.
   e. Install the following additional features by following similar steps:
      - WCE jclMax Class Library
      - WCE Database Enabler Library
      - WCE Personal Configuration Class Library
3. Create the following directory structure on your device:

   ```
   \wsdd\bin
   \wsdd\lib
   \wsdd\lib\jclMax
   ```

4. Copy the following files to \wsdd\bin:

   ```
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9.exe
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9dyn20.dll
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9int20.dll
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9max20.dll
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9prt20.dll
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9thr20.dll
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9vm20.dll
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9zlib20.dll
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\iverel20.dll
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\swt-win32-2104.dll
   ```

5. Copy the following file to \wsdd\lib:

   ```
   <WSDD>\wsdd5.0\ive\runtimes\common\ive\lib\charconv.zip
   ```

6. Copy the following files to \wsdd\lib\jclMax:

   ```
   <WSDD>\wsdd5.0\ive\runtimes\common\ive\lib\jclMax\classes.zip
   <WSDD>\wsdd5.0\ive\runtimes\common\ive\lib\jclMax\database_enabler.jar
   <WSDD>\wsdd5.0\ive\runtimes\common\ive\lib\jclMax\locale.zip
   <WSDD>\wsdd5.0\ive\runtimes\pocketpc\common\ive\lib\jclMax\prsnlwin.jar
   ```

Return to “Setting up for JSP development on a Windows CE device” on page 34.

Installing and verifying JSP support on a Windows CE device

This task is part of the main task of Setting up for JSP development on a Windows CE device. After you complete these steps, return to “Setting up for JSP development on a Windows CE device” on page 34.

Procedure:

To install and verify JSP support::

1. Install the following file onto the device:

   ```
   <DB2 Everyplace>\SDK\JSP\WinCE\DB2eJSP.CAB
   ```

2. Verify that your device is set up to use DB2 Everyplace JSP support by running the Visiting Nurse sample JSP application:

a. Optional depending on your JVM: Modify `MiniHttpServer.lnk` on the device. You need to modify the shortcut if you are using a JVM other than the J9 JVM. The number that the shortcut begins with is the number of characters after the '#' character. The maximum number of characters after the '#' character is 256.

b. Start the mini HTTP Web server:

   1) Open a File Explorer.

   2) Navigate to the root directory.

   3) Click the MiniHttpServer shortcut.

c. Open a Web browser to the following URL:
   http://localhost/VisitingNurse/schedule.jsp

   Alternatively, you can enter the URL http://localhost/ and browse to the VisitingNurse/schedule.jsp page.

If you successfully set up your device to use DB2 Everyplace JSP support, a Visiting Nurse Schedule table appears in the Web browser.

Return to "Setting up for JSP development on a Windows CE device" on page 34.

Installing and verifying JSP support on a Symbian OS Version 6 device

Prerequisites:

Verify that the following software is installed on your workstation:

- DB2 Everyplace Software Development Kit

Verify that the following software is installed on the device:

- Web browser
- Java run-time environment (`Java.sis`)

  `Java.sis` can be downloaded from the Internet. Older devices may provide `Java.sis` on the sales package CD-ROM for the device.

Verify that the `\system\libs` directory on the device contains the following files:

- `<DB2Everyplace>\Clients\Symbian6\database\armi\CryptoPlugin.dll`
- `<DB2Everyplace>\Clients\Symbian6\database\armi\DB2e.dll`
- `<DB2Everyplace>\Clients\Symbian6\database\armi\db2ejdbc.dll`
- `<DB2Everyplace>\Clients\Symbian6\database\armi\ECSPKCS11.DLL`

Verify that the `\system\java\ext` directory on the device contains the following file:

- `<DB2Everyplace>\Clients\Symbian6\database\armi\db2ejdbc.jar`

Procedure:

To install and verify JSP support on a Symbian OS Version 6 device:

1. Install the following file onto the device:
   `<DB2Everyplace>\SDK\JSP\Symbian6\DB2eJSP.sis`.
2. Verify that the device is set up to use DB2 Everyplace JSP support by running the Visiting Nurse sample JSP application:

   a. Start the mini HTTP Web server:

1) Go to the Extras screen on the device.
2) Start the DB2eJSP application.

b. Open a Web browser to the following URL:
http://localhost/VisitingNurse/schedule.jsp

Alternatively, you can enter the URL http://localhost/ and browse to the VisitingNurse/schedule.jsp page.

If you successfully set up your device to use DB2 Everyplace JSP support, a Visiting Nurse Schedule table appears in the Web browser.

Related tasks:

- "Running a JSP application on a Symbian OS Version 6 device" on page 41

Related reference:

- "Supported JSP Version 1.1 subsets" on page 41
- "IBM custom tags for JSP application database access" on page 45

Transferring a JSP application to a Windows CE device

Prerequisites:

Before you transfer your JSP application to a device, complete the following tasks:

- Install JSP support on the device.
- Develop and test the application on the workstation.

Procedure:

To transfer a JSP application to a Windows CE device:

1. Copy the database used by the application to the device, if it doesn't already exist on the device. Make sure to put the database in the directory that the application expects it to be in. That is, the directory specified by the URL that the application connects to.
2. Copy the application files to the directory specified for the JspPath property in `MiniHttpConfig.properties` for the device. If the application is in a subdirectory under JspPath on the workstation, create the same subdirectory structure under JspPath on the device. Use the following rules when copying the application files to the device:

Table 3.

File on the workstation	File to copy to the device
`<page>.jsp`	`<page>_jsp_.class`
`<webapp>\web.xml`	`<webapp>\<webapp>_config_.class`

Notes:

- The .class files to copy to the device are generated by the JSP processor when you run the application on the workstation. They are located in the same directory as the corresponding `.jsp` and `web.xml` files.
- Your application may not include a `web.xml` file. Applications generated by WebSphere Studio Application Developer use a `web.xml` file.
- The maximum length of a shortcut is 256 characters after the '#' character at the beginning of the file. To stay under the maximum length, copy the

following WebSphere Studio Application Developer sample files to the root directory instead of the sample application's directory:

– `dbbeans.jar`

– any ViewBean class file

To run applications generated by WebSphere Studio that use these files, the classpath in `MiniHttpServer.lnk` must include `dbbeans.jar`, as well as any directory that contains ViewBean class files.

- Application files that are not `.jsp` or `web.xml` files should be copied to the device as well.

For example, to copy the Visiting Nurse sample JSP application to the device:

a. Copy the Visiting Nurse sample database to the device:
   1) Create the following directory structure on the device: `\sample\data`. This is the directory specified by the URL that the application connects to.
   2) Copy the contents of the `<DB2Everyplace>\SDK\JSP\sample\data` directory on the workstation to `\sample\data` on the device.

b. Copy the application files to the device:
   1) Create the following directory structure on the device: `\sample\jsp`. This is the directory specified for JspPath in the default `MiniHttpConfig.properties` file.
   2) Create the subdirectory `\VisitingNurse` under `\sample\jsp`.
   3) Copy the following files to `\sample\jsp\VisitingNurse`:

```
<DB2Everyplace>\SDK\JSP\sample\jsp\VisitingNurse\_schedule_jsp_.class
<DB2Everyplace>\SDK\JSP\sample\jsp\VisitingNurse\_contact_jsp_.class
<DB2Everyplace>\SDK\JSP\sample\jsp\VisitingNurse\_medrecord_jsp_.class
<DB2Everyplace>\SDK\JSP\sample\jsp\VisitingNurse\_person_jsp_.class
```

Related tasks:

- "Setting up for JSP development on a Windows CE device" on page 34

Related reference:

- "Supported JSP Version 1.1 subsets" on page 41
- "IBM custom tags for JSP application database access" on page 45

Running a JSP application

This section describes how to run a JSP application. The following topics are covered:

- Optional depending on your workstation or application configuration: "Configuring the mini HTTP Web server"
- "Running a JSP application on a Windows workstation" on page 39
- "Running a JSP application on a Windows CE device" on page 40
- "Running a JSP application on a Symbian OS Version 6 device" on page 41

Configuring the mini HTTP Web server

Procedure:

The following file is the default `MiniHttpConfig.properties` file for Windows workstations. You can use this default file or modify it to meet the requirements of

your application and system. You may also modify the default `MiniHttpConfig.properties` file for the devices.

```
# Mini HTTP Web server properties - Win32

# JspPath

# Specifies: the path that contains the JSP pages (.jsp and .class files).
# Default: JspPath=<directory where the mini HTTP Web server is started>
#
# Note: use \\ to denote the directory separator
#
JspPath=sample\\jsp

# Port
#
# Specifies: port that the server listens on
# Default: Port=80
#
# Note: If you are not using the default Port, start URL requests with:
#  http://localhost:Port/
# (instead of http://localhost/) where Port is the specified Port.

# Mime
#
# Specifies: Mime types
# Default: Mime=text/html wml htm html,text/plain txt,image/gif gif,image/jpeg jpg
#
# Note: Additional Mime types can be added using the following format:
# Mime=mime_type_A ext1 ext2 ext3 ...,mime_type_B ext4 ext5 ...,...
#
Mime=application/octet-stream exe class,image/jpeg jpeg jpg

# LogFile
#
# Specifies: log file for the server
# Values:
# "" - log entries are written to the console
# "no" - no log entries are kept
# "<log_file_name>" - log entries are written to <log_file_name>
# Default: LogFile=
#
LogFile=JspServer.log

# Index
#
# Specifies: the index page (displayed if http://localhost is requested)
# Values:
# "" - the JspPath directory will be loaded
# "no" - no page will be loaded
# "<index_file>" - <index_file> will be loaded
# Default: Index=
```

Related tasks:

- "Installing and verifying JSP support on a Symbian OS Version 6 device" on page 36

Related reference:

- "Supported JSP Version 1.1 subsets" on page 41
- "IBM custom tags for JSP application database access" on page 45
- "JSP supported operating systems" on page 31

Running a JSP application on a Windows workstation

Procedure:

To run a JSP application on a Windows workstation:

1. If necessary, configure the `MiniHttpConfig.properties` file in the `<DB2Everyplace>\SDK\JSP\Win32 directory`.
2. Start the mini HTTP Web server:
   a. Open an MS-DOS window.
   b. Change the directory to `<DB2Everyplace>\SDK\JSP\Win32` using the 'cd' command.
   c. Type `runJspServer`.
3. Open a Web browser and enter the URL to start your JSP application. For example, if the start page of your application is `start.jsp`, type `http://localhost/start.jsp`. If you configured the Port property in the `MiniHttpConfig.properties` file, type `http://localhost:Port/start.jsp` instead, where Port is the specified port number.
4. To stop the mini HTTP Web server when you are done running the JSP application:
   a. Go to the MS-DOS window that you started the server in.
   b. Type Ctrl+C, and then type 'y' to terminate the batch job.

Related tasks:

- "Verifying JSP support on a Windows workstation" on page 33

Related reference:

- "Supported JSP Version 1.1 subsets" on page 41
- "IBM custom tags for JSP application database access" on page 45

Running a JSP application on a Windows CE device

Procedure:

To run a JSP application on a Windows CE device:

1. Transfer the application to your device.
2. If necessary, modify `MiniHttpServer.lnk` on the device. You need to modify the shortcut if you are using a JVM other than the J9 JVM. The number that the shortcut begins with is the number of characters after the '#' character.
3. If necessary, configure the `MiniHttpConfig.properties` file for the Windows CE device.
4. Start the mini HTTP Web server:
   a. Open a File Explorer.
   b. Navigate to the root directory.
   c. Click the MiniHttpServer shortcut.
5. Open a Web browser and enter the URL to start your JSP application. For example, if the start page of your application is `start.jsp`, type `http://localhost/start.jsp`. If you configured the Port property in the `MiniHttpConfig.properties` file, type `http://localhost:Port/start.jsp` instead, where Port is the specified port number.
6. To stop the mini HTTP Web server when you are done running the JSP application:
   a. Go to the J9 console window.
   b. Click **File —> Close**.

Related tasks:

- "Setting up for JSP development on a Windows CE device" on page 34

Related reference:
- "Supported JSP Version 1.1 subsets"
- "IBM custom tags for JSP application database access" on page 45

Running a JSP application on a Symbian OS Version 6 device

Procedure:
1. Transfer the application to the device.
2. If necessary, configure the `MiniHttpConfig.properties` file for the Symbian device.
3. Start the mini HTTP Web server:
   a. Go to the Extras screen on the device.
   b. Start the DB2eJSP application.
4. Open a Web browser and enter the URL to start your JSP application. For example, if the start page of your application is `start.jsp`, type http://localhost/start.jsp. If you configured the Port property in the `MiniHttpConfig.properties` file, type http://localhost:Port/start.jsp instead, where Port is the specified port number.
5. To stop the mini HTTP Web server when you are done running the JSP application, do a soft reset.

Related tasks:
- "Installing and verifying JSP support on a Symbian OS Version 6 device" on page 36

Related reference:
- "Supported JSP Version 1.1 subsets"
- "IBM custom tags for JSP application database access" on page 45
- "JSP supported operating systems" on page 31

Supported JSP Version 1.1 subsets

This section lists and describes the directives, implicit objects, scripting elements, and standard actions included in DB2 Everyplace JSP support.

Directives:

Page directive:

Description

The page directive defines page-dependent attributes.

Syntax

```
<%@ page page_directive_attr_list %>
page_directive_attr_list ::=
{language="scriptingLanguage"}
{extends="className" }
{import="importList" }
{contentType="ctinfo" }
```

Attributes

The four valid attributes for this directive are:
- **language** - Must be 'java' if specified.

- **extends** - A fully qualified Java programming language class name that names the superclass of the class to which this JSP page is transformed.
- **import** - The default import list is `com.ibm.db2e.jsp.server.*, java.io.*, java.sql.*, and java.util.*.`
- **contentType** - Can be any value (such as text/html, text/xml, application/x-octet).

Example

```
<%@ page contentType="text/html" %>
```

Include directive:

Description

Use the include directive to include data in a JSP page. The included file may have elements which will also be processed.

Syntax

```
<%@ include file="relativeURLspec" %>
```

Attributes

The attribute for this directive is **file** which must be a page-relative path. The path cannot not start with a "/", and is to be interpreted relative to the current JSP page.

Example

```
<%@ include file="copyright.html" %>
```

Implicit objects:

When you create JSP pages, you have access to certain implicit objects. They can be used within scriptlets and expressions, without first being declared. Each implicit object has a class defined in a core Java technology or `com.ibm.db2e.jsp.server` package, shown in Table 4.

Table 4. Implicit objects

Implicit Variable	Type	Representation	Method Summary
request	com.ibm.db2e.jsp.server.MimeHttpRequest	The request for the JSP page.	`java.lang.String getParameter(java.lang.String name)` `java.lang.String getQueryString()`

response	com.ibm.db2e.jsp.server.MimeHttpResponse	The response to the request	`java.lang.String encodeURL(java.lang.String url)`
			`void setDateHeader(java.lang.String name, long date`
			`void setHeader(java.lang.String name, java.lang.Str`

in	java.io.BufferedReader	This object is currently unavailable.	
out	java.io.PrintStream	An object that writes into the Web browser.	
config	com.ibm.db2e.jsp.server.DB2eJspConfig	DB2eJspConfig for this JSP page.	`java.lang.String getInitParameter(java.lang.String n`
exception	java.lang.Throwable	An exception thrown during execution of the JSP page.	

Note: Some of the types for the implicit objects above differ from those in JSP 1.1 due to the implementation of DB2 Everyplace JSP support.

Scripting elements:

Declarations:

Description

Use declarations to declare Java variables and methods used in a JSP page. Declarations are member variables (fields and methods) of the Java class for the JSP page.

Syntax

```
<%!declaration(s) %>
```

Example

```
<%!
String name = "Joe Smith";

public String getName() {
 return name;
}
%>
```

Standard actions:

<jsp:useBean>:

Description

A jsp:useBean action associates an instance of a Java programming language object defined within the ″page″ scope available with a given id via a newly declared scripting variable of the same id.

Syntax

```
<jsp:useBean id="name" scope="page|request|session|application" typeSpec/>
typeSpec ::=
class="className"
```

Attributes

The three attributes for this tag are:

- **id** - Specifies the identifier of this bean. Do not reuse this name in the JSP page. This attribute is required
- **scope** - This attribute is ignored if specified. The default scope ″page″ is used.
- **class** - Specifies the class that this bean represents. This attribute is required.

Example

<jsp:useBean id=″masterViewDBBean″
class=″Query1HTMLResultsMasterViewBean″ />

<jsp:setProperty>:

Description

The jsp:setProperty action sets the value of properties in a Bean.

Syntax

```
<jsp:setProperty name="beanName" prop_expr />
prop_expr ::=
property="propertyName" value="propertyValue"
propertyValue ::= string
The value propertyValue can also be a request-time attribute value.
propertyValue ::= expr_scriptlet
```

Attributes

The three attributes for this tag are:

- **name** - The name of a Bean instance defined by a <jsp:useBean> element before this action appears. The Bean instance must contain the property you want to set. This attribute is required.

- **property** - The name of the Bean property whose value you want to set. This attribute is required.
- **value** - The value to assign to the given property. This attribute can accept a request-time attribute expression as a value. This attribute is required.

Example

```
<jsp:setProperty name="masterViewDBBean" property="username"
value='<%=config.getInitParameter("username")%>' />
```

Scriptlets:

Description

Use scriptlets to hold any valid Java code fragments. These code fragments are placed in the source code for the JSP page, and are relative to other elements of the JSP page.

Syntax

```
<% scriptlet %>
```

Example

```
<%
try {
String name = Query1DBBean.getString(1);
out.println("Name = " + name);
}
catch (SQLException e) {

}
%>
```

Expressions:

Description

Expressions are string representations of data types. You can use expressions in queries and in HTML comments. The application evaluates the expressions at run-time and converts the expressions into strings.

Syntax

```
<%= expression %>
```

Example

```
<%= new java.util.Date() %>
```

Note: Variable names that begin with *__db2ejsp__* are keywords and are used internally. Do not use these variables in the JSP page.

Related concepts:

- "DB2 Everyplace JSP support overview" on page 32
- "Developing DB2 Everyplace JSP applications" on page 31
- Chapter 16, "The sample JSP applications," on page 109

Related tasks:

- "Verifying JSP support on a Windows workstation" on page 33
- "Setting up for JSP development on a Windows CE device" on page 34

Related reference:

- "IBM custom tags for JSP application database access" on page 45

IBM custom tags for JSP application database access

The following tags are IBM custom tags that you can use in your JSP application to access a DB2 Everyplace database.

<tsx:dbconnect>:

Description

This tag establishes a connection to a specified DB2 Everyplace database using the DB2 Everyplace JDBC driver.

Syntax

```
<tsx:dbconnect
id="connection_id"
driver="com.ibm.db2e.jdbc.DB2eDriver"
url="jdbc:db2e:database"
userid="db_user"
passwd="user_password">
</tsx:dbconnect>
```

Attributes

The five attributes for this tag are:

- **id** - Specifies the identifier of this connection. Do not reuse this name in the JSP page. This attribute is required.
- **driver** - Specifies the DB2 Everyplace JDBC driver. This attribute is required.
- **url** - Specifies the DB2 Everyplace database. The term *database* refers to the path of the DB2 Everyplace database. This attribute is required.
- **userid** - Specifies a valid user ID for the database to be accessed. This attribute is optional.
- **passwd** - Specifies the user password for the userid attribute. This attribute is required if userid is specified.

Example

```
<tsx:dbconnect
id="conn"
driver="com.ibm.db2e.jdbc.DB2eDriver"
url="jdbc:db2e:sample/data/" >
</tsx:dbconnect>
```

<tsx:dbquery>:

Description

This tag submits a query to the database using the connection specified through the <tsx:dbconnect> tag and produces a java.sql.ResultSet object in which the cursor points to the first row of the result set. You can reference this result set using the identifier of this query and the DB2 Everyplace JDBC interface for java.sql.ResultSet.

Syntax

```
<tsx:dbquery id="query_id" connection="connection_id" limit="value">
select_SQL_statement
</tsx:dbquery>
```

Attributes

The attributes for this tag are:

- **id** - Specifies the identifier of this query. Do not reuse this query identifier in the JSP page. This attribute is required.
- **connection** - Specifies the identifier of a <tsx:dbconnect> tag in this JSP file. This attribute is required.

- **limit** - Specifies the maximum number of rows that the query can return. This attribute is optional.

Parameter

The valid parameter for this tag is:

- **select_SQL_statement** - Specifies the SQL query that you want to submit to the database. This SQL query statement can contain dynamic data.

Example

```
<tsx:dbquery id="Query1DBBean" connection="conn">
select <%= request.getParameter("column") %> from vnperson
</tsx:dbquery>
```

<tsx:dbmodify>:

Description

This tag submits a command to modify data in the database using the connection specified through the <tsx:dbconnect> tag. There is no result for this tag.

Syntax

```
<tsx:dbmodify connection="connection_id">
modify_command
</tsx:dbmodify>
```

Attribute

The attribute for this tag is:

- **connection** - Specifies the identifier of a <tsx:dbconnect> tag in this JSP file. This attribute is required.

Parameter

The valid parameter for this tag is:

- **modify_command** - Specifies the SQL command that you want to submit to the database to modify data. This modify command can contain dynamic data.

Example

```
<tsx:dbmodify connection="conn">
update vnperson set Name = '<%=Name%>' where ID = '<%=id%>'
</tsx:dbmodify>
```

<tsx:repeat>:

Description

Use this tag to loop through each row in the query results. The start and stop attributes control the looping process. If you do not specify the start and stop attributes, the loop terminates when the cursor of the result set, which is referenced by the <tsx:getProperty> tag, reaches the end of the result set. This tag can be nested.

Syntax

```
<tsx:repeat index="name" start="starting_index" stop="ending_index">
repeat_block
</tsx:repeat>
```

Attributes

The attributes for this tag are:

- **index** - Specifies the identifier for the index of this tag. This attribute is optional.
- **start** - Specifies the number of rows to skip before processing the repeat block. The default is 0. This attribute is optional.

- **stop** - Specifies the ending index value for this repeat block. The default is 2,147,483,647. This attribute is optional.

Parameter

The valid parameter for this tag is:

- **repeat_block** - Specifies the block of HTML tagging that contains the <tsx:getProperty> tag syntax and the HTML tags used to format the content. If you place a <tsx:getProperty> tag in the repeat block, the cursor advances to the next row each time the repeat block is processed.

Example

```
<TABLE border="1">
<TR>
<TH>Name</TH>
</TR>

<tsx:repeat>
<TR>
<TD>
<tsx:getProperty name="Query1DBBean" property="Name" />
</TD>
</TR>
</tsx:repeat>
</TABLE>
```

<tsx:getProperty>:

Description

This tag retrieves the value of a ResultSet bean to be displayed in a JSP page (that is, the HTML result page). If you place this tag inside a <tsx:repeat> block tag, the cursor of the ResultSet bean advances to the next row each time the repeat block is processed.

Syntax

```
<tsx:getProperty name="bean_name" property="property_name" />
```

Attributes

The attributes of this tag are:

- **name** - Specifies the name of the ResultSet bean that you declared previously using a <tsx:dbquery> tag in this JSP file.
- **property** - Specifies the column of the ResultSet bean to access.

Example

```
<tsx:getProperty name="Query1DBBean" property="FIRSTNAME" />
```

Related concepts:

- "DB2 Everyplace JSP support overview" on page 32
- "Developing DB2 Everyplace JSP applications" on page 31
- Chapter 16, "The sample JSP applications," on page 109

Related tasks:

- "Verifying JSP support on a Windows workstation" on page 33
- "Setting up for JSP development on a Windows CE device" on page 34

Related reference:

- "Supported JSP Version 1.1 subsets" on page 41

Troubleshooting JSP applications

Troubleshooting:

Mini HTTP Web server console output:

1. WebSphere Studio Application Developer related
   - java.lang.NoClassDefFoundError: javax/sql/DataSource

     WSAD v4.0.3+ contains the fix for this error. You can use the `dbbeans.jar` in `<DB2Everyplace>\SDK\JSP\sample\jsp\VNSchedule_wsad40, which is the jar from WSAD v4.0.3.`
   - ″Cannot determine search value for column type name <name>. Assumed searchable = true.″

     This message is generated by `dbbeans.jar` and can be ignored.

Web browser:

- ″The page you are looking for cannot be found.″

  If Pocket Internet Explorer displays this message when you try to connect to URL ″http://localhost/″, your Windows CE device is probably older than Pocket PC v3.0.11171 and you need to obtain a fix for the Web browser. (On the Internet, search for how to connect to localhost through Pocket Internet Explorer.) Make sure the mini HTTP Web server is running.

Application development:

- Changes to JSP page are not reflected in the Web browser

  If this occurs, you may need to delete the files in your Web browser's Temporary Internet Files folder or cache.

Related concepts:

- "DB2 Everyplace JSP support overview" on page 32
- "Developing DB2 Everyplace JSP applications" on page 31
- Chapter 16, "The sample JSP applications," on page 109

Related tasks:

- "Verifying JSP support on a Windows workstation" on page 33
- "Setting up for JSP development on a Windows CE device" on page 34

Related reference:

- "IBM custom tags for JSP application database access" on page 45

Chapter 7. Developing .NET applications

This chapter provides information about developing DB2 Everyplace applications using .NET. The topics covered are:

Synchronization support

This section provides information about DB2 Everyplace .NET synchronization support. The topics covered are:

ISync.Net API file locations

The DB2 Everyplace Sync Client provides two APIs that enable developers to build managed applications for the DB2 Everyplace Sync Server. These APIs include:

- ISyncComponent
- ISync.Net

ISyncComponent is smaller then ISync.Net, but provides visual design support for developers who want to use this function.

For information about the .Net providers for the database engine, see "Overview of .NET support for building applications on the client database" on page 52.

Table 5. ISync.NET managed provider location and namespaces

Available Providers	Namespaces	Supported Platforms	Location, `\DB2Everyplace\Clients`
Non-Unicode for .NET Framework	IBM.Data.Sync IBM.Data.Sync.DB2e	Win 32	`Win32\Sync\NMP\IBM.Data.Sync.DB2e.dll`
Unicode for .NET Framework	IBM.Data.Sync IBM.Data.Sync.DB2e	Win32 Unicode	`Win32\Sync\NMP\unicode\IBM.Data.Sync.DB2e.dll`
.NET Compact Framework	IBM.Data.Sync IBM.Data.Sync.DB2e.CF	WinCE	`WinCE\Sync\NMP\IBM.Data.Sync.DB2e.CF.dll`

Sample application ISync.NET

There are two sample applications located in \DB2Everyplace\Clients\clientapisample\NMP*lang*:

- GoISync
- sample1

where *lang* is either C# (cs) or Visual Basic (vb)

ISync.NET API specification

You can find the API specification for ISync.NET in \DB2Everyplace\Clients\clientapisample\NMP\ISync.NET.chm

Related concepts:

- "Simple example application using the ISync.NET API" on page 51
- "Overview of .NET support for building applications on the client database" on page 52
- "Sample DB2 Everyplace .NET Data Provider application code for WinCE and Win32" on page 57

Related tasks:

- "Using the ISync.NET API"
- "Using ISyncComponent" on page 51
- "Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 53

Using the ISync.NET API

You can find the API specification for ISync.NET in \DB2Everyplace\Clients\clientapisample\NMP\ISync.NET.chm

Prerequisite:

Software Requirements

- DB2 Everyplace Version 8.1.4, beta 1
- Microsoft. NET Standard Framework 1.0 (included with Visual Studio 2002) — needed for developing applications on Win32
- Microsoft .NET Compact Framework (included with Visual Studio 2003) — needed for developing applications on WinCE

Although the ISync.NET provider is platform and language independent, it still depends on the underlying native Sync Client libraries. Both the provider and the Sync Client libraries must be included in the user path at application runtime. During the installation of DB2 Everyplace the user paths should be updated.

For information on the client libraries, see the *DB2 Everyplace Installation and User's Guide* in Chapter 3., "Installing DB2 Everyplace on a mobile device".

Procedure:

To use Isync.NET in your applications, the following steps must be performed for all Visual Studio Frameworks and application types.

1. In Microsoft Visual Studio .NET, create a new project in the language of your choice.
2. In your application, import the DB2 Everyplace namespaces. The following is an example for the Standard Framework:

   ```
   [Visual Basic]
        Imports IBM.Data.Sync
        Imports IBM.Data.Sync.DB2e
        [C#]
        using IBM.Data.Sync;
        using IBM.Data.Sync.DB2e;
   ```

   For more information, you can view the sample synchronization application located in \DB2Everyplace\Clients\clientapisample\NMP
3. Add a reference:
   a. In Visual Studio, right click on the project name and select **Add Reference**.
   b. Under the Projects tab, browse for the location of **IBM.Data.Sync.DB2e.dll**.

c. On command line, type: **csc /t:exe /r:IBM.Data.Sync.DB2e.dll ISyncSample.cs**.

Related concepts:

- "ISync.Net API file locations" on page 49
- "Simple example application using the ISync.NET API"
- "Overview of .NET support for building applications on the client database" on page 52
- "Sample DB2 Everyplace .NET Data Provider application code for WinCE and Win32" on page 57

Related tasks:

- "Using ISyncComponent"
- "Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 53

Using ISyncComponent

Procedure:

ISyncComponent also provides minimal design support in the Standard Framework. This basic support enables you to drag and drop into a form, and to modify the ConnectionString (server, port, and user name) and TargetPath (target directory for the data) properties. When developing a Visual Studio Windows Application, be sure to add the DB2 Everyplace component **IBM.Data.Sync.DB2e.dll** to your **Toolbox**.

Note: The native Sync Client libraries must already be in the user path for this process to complete successfully.

For the Standard Framework, there is an option to use a simpler API by using IBM.Data.Sync.DB2e.ISyncComponent.

```
ISyncComponent comp1 = new ISyncComponent();
comp1.ConnectionString = SERVER=localhost;PORT=80;UID=username;PWD=password;
comp1.TargetPath = data;
comp1.Sync();
comp1.Close();
```

Related concepts:

- "ISync.Net API file locations" on page 49
- "Simple example application using the ISync.NET API"
- "Overview of .NET support for building applications on the client database" on page 52
- "Sample DB2 Everyplace .NET Data Provider application code for WinCE and Win32" on page 57

Related tasks:

- "Using the ISync.NET API" on page 50
- "Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 53

Simple example application using the ISync.NET API

This topic includes an example which provides a quick reference of how to use the ISync.NET API.

```
// Synchronization properties
private Hashtable userProps = new Hashtable();

// Get an instance DB2eISyncProvider
ISyncProvider provider = DB2eISyncProvider.GetInstance();

// Set up properties
userProps.Add("isync.user", username);
userProps.Add("isync.password", password);

// Get an instance of synchronization service from the provider
ISyncService service = provider.CreateSyncService(http://localhost:80, userProps);

// Get an instance of the configuration store
ISyncConfigStore config = service.GetConfigStore(data);

// Get an instance of the sync driver to perform synchronization
ISyncDriver syncer = config.GetSyncDriver();

// Perform synchronization
syncer.Sync();

// Close objects
syncer.Close();
config.Close();
service.Close();
```

Related concepts:

- "ISync.Net API file locations" on page 49
- "Overview of .NET support for building applications on the client database"
- "Sample DB2 Everyplace .NET Data Provider application code for WinCE and Win32" on page 57

Related tasks:

- "Using the ISync.NET API" on page 50
- "Using ISyncComponent" on page 51
- "Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 53

Support for building .NET applications

This section provides information about Support for building .NET applications. The topics covered are:

- "Overview of .NET support for building applications on the client database"
- "Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 53
- "Sample DB2 Everyplace .NET Data Provider application code for WinCE and Win32" on page 57

Overview of .NET support for building applications on the client database

DB2 Everyplace provides the tools to enable developers to build applications that use the ADO.NET API to manipulate data managed by the DB2 Everyplace database. DB2 Everyplace contains two .NET Data Providers. One provider runs on the .NET Framework 1.0 and the other provider runs on .NET Compact Framework. You will find these providers or APIs in:

- **For Win32:**

  `\DB2Everyplace\Clients\Win32\database\nmp\IBM.Data.DB2.DB2e.dll`

- **For WinCE:**

  `\DB2Everyplace\Clients\WinCE\database\nmp\IBM.Data.DB2.DB2e.CF.dll`

The API specifications are located in

`\DB2Everyplace\Clients\Win32\database\nmp\doc\readme.html`

To simplify the transition for programmers that have used Microsoft ODBC .NET Data Provider in the past, the new DB2 Everyplace .NET Data Provider interfaces are almost identical to those of the Microsoft ODBC .NET Data Provider. For instance, the Microsoft ODBC .NET Data Provider has the OdbcConnection class, while IBM DB2 Everyplace .NET Data Provider has DB2eConnection as an equivalent function class. Similarly you can replace 'Odbc' with 'DB2e' in the other class names to get the corresponding DB2 Everyplace .NET Data Provider classes.

Related concepts:

- "ISync.Net API file locations" on page 49
- "Simple example application using the ISync.NET API" on page 51
- "Sample DB2 Everyplace .NET Data Provider application code for WinCE and Win32" on page 57

Related tasks:

- "Using ISyncComponent" on page 51
- "Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider"

Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider

The namespaces for the DB2 Everyplace .NET Data Provider are as follows:

- **Running on the .NET Compact Framework:** IBM.Data.DB2.DB2e.CF
- **Running on the .NET Framework:** IBM.Data.DB2.DB2e

The DB2 Everyplace .NET Data Provider provides functionality for connecting to a DB2 Everyplace data source, executing commands, and retrieving results. Those results can be processed directly, or placed in an ADO.NET DataSet for further processing while in a disconnected state. While in the DataSet, data can be exposed to the user, combined with other data from multiple sources, or passed remotely between tiers. Any processing performed on the data while in the DataSet can then be reconciled to the data source.

The DB2 Everyplace .NET Data Provider is designed to be lightweight. It consists of a minimal layer between DB2 Everyplace and your code that extends functionality without sacrificing performance.

DB2 Everyplace .NET Data Provider classes inherit or implement members from other .NET Framework classes or interfaces. This provider documentation includes a summary of the supported members within each of these classes. For more detailed information about a specific inherited member, see the appropriate topic in the Microsoft® .NET Framework SDK.

Prerequisite:

Table 6. Prerequisites for using the DB2 Everyplace .NET Data Provider

Component	Minimum requirement
Microsoft.NET Framework	Microsoft.NET Framework 1.0 Must be installed prior to installing the DB2 Everyplace .NET Data Provider for application development
Microsoft Visual Studio.NET 2003	Microsoft Visual Studio.NET 2003 for developing mobile applications
Microsoft.NET Compact Framework	Microsoft .NET Compact Framework 1.0 for mobile development Must be installed on the device prior to installing the DB2 Everyplace .NET Data Provider for mobile application development.
DB2 Everyplace product	• `DB2e.dll` of version 8.1.4 or above • `AgentProxy.dll` of version 8.1.4 or above required for remoted stored procedure call • `wbxmllib.dll` of version 8.1.4 or above required for remoted stored procedure call. • DB2 Everyplace Sync Server version 8.1.4 or above required for remoted stored procedure call `DB2e.dll`, `AgentProxy.dll`, and `wbxmllib.dll` are native libraries and thus are processor dependent; thus, the operating system needs to locate these native libraries (setting the environment variable PATH, for example) in order for DB2 Everyplace .NET Data Provider to function properly.

Restrictions:

Provider Limitations:

- Update on primary key columns is not currently allowed in DB2 Everyplace.
- Result set retrieval using a remote stored procedure call has a limitation on the size of the result set.
- Local stored procedure calls are not supported.
- For methods or properties that are not supported, a `System.NotSupportedException` will be thrown

Thread Safety:

Any public non-instance members of this provider are safe for multithreaded operations. Any instance members are not guaranteed to be thread safe.

Procedure:

There are four core objects that make up DB2 Everyplace .NET data provider. The following table describes these objects and their function.

Table 7. DB2 Everyplace .NET Data Provider, core objects

Object	Desciption
DB2eConnection	Establishes a connection to a DB2 Everyplace data source and can begin a *Transaction*.
DB2eCommand	Executes a command at a DB2 Everyplace server, and exposes *Parameters*.
DB2eDataAdapter	Populates a *DataSet* and resolves updates with the DB2 Everyplace data source.
DB2eDataReader	Exposes and reads a forward-only stream of data from a DB2 Everyplace data source.

Along with the core classes listed in the preceding table, the DB2 .NET data provider also contains the classes listed in the following table.

Table 8. DB2 Everyplace .NET Data Provider, additional classes

Object	Desciption
DB2eCommandBuilder	A helper object that will automatically generate command properties of the *DB2eDataAdapter* or will derive parameter information from a stored procedure and populate the *DB2eParameters* collection of a *DB2eCommand* object. Note: Use of the *DB2eCommandBuilder* is not recommended as it can generate very inefficient and, in some cases, invalid SQL statements.
DB2eError	Exposes the information from a warning or error returned by a DB2 Everyplace data source.
DB2eException	Returned when an error is encountered at the DB2 Everyplace data source. For an error encountered at the client, .NET data providers throw a .NET Framework exception.
DB2eParameter	Defines input, output, and return value parameters for commands and stored procedures.
DB2eTransaction	Enables you to enlist commands in transactions at the DB2 Everyplace data source.

To use the DB2 Everyplace .NET Data Provider, you must add an imports or using statement for the IBM.Data.DB2.DB2e or namespace to your application .DLL, as the following code illustrates:

[Visual Basic]

```
Imports IBM.Data.DB2.DB2e
```

[C#]

```
using IBM.Data.DB2.DB2e;
```

You also must include a reference to the .DLL when you compile your code. For example, if you are compiling a Microsoft® Visual C#™ program, your command line should include:

```
csc /r:IBM.Data.DB2.DB2e.dll
```

For the .NET Compact Framework, the namespace is IBM.Data.DB2.DB2e.CF, and the application needs to reference the IBM.Data.DB2.DB2e.CF.dll assembly.

For information about how to best use this namespace, see the documentation on the following DB2 Everyplace.NET Data Provider classes:

- DB2eDataAdapter
- DB2eCommand
- DB2eConnection
- DB2eDataReader

For more information about how the DB2 Everyplace .NET Data Provider functions within the .NET Framework, see IBM.Data.DB2.DB2e Hierarchy.

Table 9. Classes

Object	Desciption
DB2eCommand	Represents an SQL statement or stored procedure to execute against a data source. This class cannot be inherited.
DB2eCommandBuilder	Automatically generates single-table commands used to reconcile changes made to a *DataSet* with the associated data source. This class cannot be inherited.
DB2eConnection	Represents an open connection to a data source.
DB2eDataAdapter	Represents a set of data commands and a connection to a data source that are used to fill the *DataSet* and update the data source. This class cannot be inherited.
DB2eDataReader	Provides a way of reading a forward-only stream of data rows from a data source. This class cannot be inherited.
DB2eError	Collects information relevant to a warning or error returned by the data source. This class cannot be inherited.
DB2eException	The exception that is generated when a warning or error is returned by a DB2 Everyplace data source. This class cannot be inherited.
DB2eParameter	Represents a parameter to a *DB2eCommand* and optionally, its mapping to a *DataColumn*. This class cannot be inherited.
DB2eTransaction	Represents an SQL transaction to be made at a data source. This class cannot be inherited.

Table 10. Delegates

Delegate	Desciption
DB2eInfoMessageEventHandler	Represents the method that will handle the InfoMessage event of a DB2eConnection.
DB2eRowUpdatedEventHandler	Represents the method that will handle the RowUpdated event of a DB2eDataAdapter.
DB2eRowUpdatingEventHandler	Represents the method that will handle the RowUpdating event of an DB2eDataAdapter.

Table 11. Enumerations

Enumeration	Desciption
DB2eType	Specifies the data type of a field, property, or DB2eParameter.

Table 12. DB2 Everyplace .NET Provider Connection String Keywords

Keyword	Desciption
Database	Database location. For example: `C:\data1\`
UID	User ID
PWD	Password

C# Example:

```
string connString = @"Database=C:\data1\; UID=user; PWD=userpwd";
```

Related concepts:

- "ISync.Net API file locations" on page 49
- "Simple example application using the ISync.NET API" on page 51
- "Overview of .NET support for building applications on the client database" on page 52
- "Sample DB2 Everyplace .NET Data Provider application code for WinCE and Win32"

Related tasks:

- "Using the ISync.NET API" on page 50
- "Using ISyncComponent" on page 51

Sample DB2 Everyplace .NET Data Provider application code for WinCE and Win32

There are two sample applications that illustrate how to develop applications for WinCE and Win32 using the DB2 Everyplace .NET Data Provider:

- DB2eSample1.cs
- DB2eSample2.sc

Both files are located in:

- **For Win32 and WinCE**

  `\DB2Everyplace\Clients\Win32\database\nmp\samples`

The following is an example of one of the sample applications.

```
using System;
using System.Text;
using System.Data;

using IBM.Data.DB2.DB2e;

/*
 * Sample1
 *
 * The following example creates a table, insert some rows to it, fetches
 * all the rows from the table, and finally drops the table.
 *
 */
namespace IBM.Data.DB2.DB2e.Samples
{
        class DB2eSample1
        {
```

```
          /// <summary>


 /// The main entry point for the application.
/// </summary>
[STAThread]
static void Main(string[] args)
  {
    DB2eConnection conn   = null;
    DB2eCommand cmd       = null;
    DB2eDataReader reader = null;
     String connString     = @"database=.\; uid=user1; pwd=user1";
     int rowsAffected      = 0;

      try
           {
            conn = new DB2eConnection(connString);
            conn.Open();

              Console.WriteLine("creating table t1...");
              cmd = new DB2eCommand("create table t1 (c1 int primary key not null,
   c2 smallint, c3 char(10), c4 varchar(10), c5 decimal(8,2), c6 date,
   c7 time, c8 timestamp )", conn);
                 rowsAffected = cmd.ExecuteNonQuery();
                 Console.WriteLine("inserting a row into table t1...");
                 cmd.CommandText = "insert into t1 values (1, 10, 'John',
    'Yip', null, current date, current time, current timestamp)";
            rowsAffected = cmd.ExecuteNonQuery();
            Console.WriteLine("inserting a row into table t1...");
            cmd.CommandText = "insert into t1 values (2, 20, 'Mary', 'Jann',
    2.2, current date, current time, current timestamp)";
            rowsAffected = cmd.ExecuteNonQuery();
            cmd.CommandText = "select * from t1";
            Console.WriteLine("fetching resultset from table t1...");
            reader = cmd.ExecuteReader();
                while (reader.Read())
                  {
                    if (!reader.IsDBNull(0))
                       Console.Write(reader.GetInt32(0) + "\t");
                    else
                        Console.Write("NULL " + "\t");
                    if (!reader.IsDBNull(1))
                        Console.Write(reader.GetInt16(1) + "\t");
                    else
                        Console.Write("NULL " + "\t");
                    if (!reader.IsDBNull(2))
                        Console.Write(reader.GetString(2) + "\t");
                    else
                        Console.Write("NULL " + "\t");
                    if (!reader.IsDBNull(3))
                        Console.Write(reader.GetString(3) + "\t");
                    else
                        Console.Write("NULL " + "\t");
                    if (!reader.IsDBNull(4))
                        Console.Write(reader.GetDecimal(4) + "\t");
                    else
                        Console.Write("NULL " + "\t");
                    if (!reader.IsDBNull(5))
                        Console.Write(reader.GetDate(5) + "\t");
```

```
                    else
                        Console.Write("NULL " + "\t");
                    if (!reader.IsDBNull(6))
                        Console.Write(reader.GetTime(6) + "\t");
                    else
                        Console.Write("NULL " + "\t");
                    if (!reader.IsDBNull(7))
                        Console.Write(reader.GetDateTime(7) + "\t");
                    else
                        Console.Write("NULL " + "\t");
                        Console.WriteLine();
                    }
                reader.Close();
                reader = null;
                Console.WriteLine("dropping table t1...");
                cmd.CommandText = "drop table t1";
                cmd.ExecuteNonQuery();
              }
                catch (DB2eException e1)
               {
                    int cnt = e1.Errors.Count;
                    for (int i=0; i < cnt; i++)
                      {
                        Console.WriteLine("Error #" + i + "\n" +
                           "Message: " + e1.Errors[i].Message + "\n" +
                           "Native: " + e1.Errors[i].NativeError.ToString() + "\n" +
                           "SQL: " + e1.Errors[i].SQLState + "\n");
                      }
                   }
                catch (Exception ex)
                  {
                        Console.WriteLine(ex.Message);
                  }
                finally
                   {
                        if (reader != null)
                           {
                              reader.Close();
                              reader = null;
                           }
                        if (conn != null)
                           {
                              conn.Close();
                              conn = null;
                           }
                   }
            }  // end of Main

        }  // end of class

} // end of namespace
```

Related concepts:

- "ISync.Net API file locations" on page 49
- "Simple example application using the ISync.NET API" on page 51
- "Overview of .NET support for building applications on the client database" on page 52

Related tasks:

Chapter 8. Connecting to a DB2 Everyplace database

This chapter explains how to connect to a DB2 Everyplace database. The following topics are covered:

Overview of the DB2 Everyplace database tables

A DB2 Everyplace database is comprised of several system catalog tables and a number of user-defined tables. Each table is stored in two files, one for the data itself and the other for indexes. All indexes will be kept in the same index file. Unlike DB2 Universal Database, DB2 Everyplace databases do not have names and cannot be cataloged or uncataloged. That is, the database name is ignored. In DB2 Everyplace, a database is merely a number of files that can be copied or moved to another location. System catalog tables contain metadata on user-defined tables. For example, if you remove files for a user-defined table without deleting a corresponding entry in the catalog tables this action will cause an inconsistency. A DB2 Everyplace database must contain the following system catalog tables:

- DB2eSYSTABLES
- DB2eSYSCOLUMNS
- DB2eSYSRELS
- DB2eSYSUSERS (this table is created if you use local data encryption)

To access catalog tables in a query, you must use delimited identifiers. For example, the following query returns if there exists a table T:

```
SELECT 1 FROM "DB2eSYSTABLES" WHERE TNAME = 'T'
```

Related reference:

- Chapter 19, "DB2 Everyplace System Catalog base tables," on page 333

Handling naming conflicts

Procedure:

This topic shows some examples of ways that you can handle file naming conflicts for user-defined tables. Suppose that an application executes the following CREATE TABLE statement:

```
CREATE TABLE T (PK INT NOT NULL PRIMARY KEY, A INT)
```

Once this statement is executed, DB2 Everyplace will create the following two files for table *T*:

- DSY_T (data)
- DSY_iT (index)

If you create another table and use the name *iT*, DB2 Everyplace will create two additional files: DSY_iT (data) and DSY_iiT (index). The index file for table *T* and the data file for table *iT* are in conflict because they both have the same name. Both files are named DSY_iT. To avoid this problem, DB2 Everyplace supports file name mapping. That is, the file names will be completely created and managed by DB2 Everyplace. To use this feature, applications must set the connection attribute and it must be executed prior to the creation of the first table. For example, in CLI:

```
SQLSetConnectAttr(hdbc, SQL_ATTR_FILENAME_FORMAT,
                  (SQLPOINTER)SQL_FILENAME_FORMAT_83, 0)
```

Or in Command Line Processor:

```
DISABLE LONG FILENAME
```

Once this command is executed and the first table is created, the resulting files will be for table *T*:

- 0001.DBd
- 0001.DBi

Related concepts:

- "Overview of the DB2 Everyplace database tables" on page 61
- "Connection serialization" on page 63

Related tasks:

- "Connecting to the DB2 Everyplace database"

Connecting to the DB2 Everyplace database

Applications often require creating and accessing tables in a specific location such as `C:\TEMP` folder on Win32 platforms. You can use the following CLI call to connect to the database specifying the database path:

```
rc = SQLConnect(hdbc, "C:\\TEMP\\db", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

where *db* is the database name (which is ignored by DB2 Everyplace).

In addition, you can use this CLI call:

```
rc = SQLConnect(hdbc, "C:\\TEMP\\", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

Connecting to extended memory on devices or WinCE object stores requires a special path specification.

- For Sony Memory Stick on PalmOS

  ```
  rc = SQLConnect(hdbc, "#0:\\", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
  ```

- For WinCE Object Store

  ```
  rc = SQLConnect(hdbc, "@:\\", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
  ```

Using Command Line Processor, users can connect to a specific location using the "CONNECT TO" command. For example, the following commands will connect to the database that is kept in C:\TEMP\ folder on Win32 platform:

```
CONNECT TO C:\TEMP\
```

Related concepts:

- "Overview of the DB2 Everyplace database tables" on page 61

Related tasks:

• “Handling naming conflicts” on page 61

Connection serialization

You can have only one connection at a time to a DB2 Everyplace database. On all platforms *except* Palm, DB2 Everyplace supports connection serialization. With connection serialization, the connection requests to a data source are serialized. Only one active connection can be made to a given data source at a time. The other connection requests are put into a queue. The timeout period can be set using the SQL_ATTR_LOGIN_TIMEOUT attribute of the SQLSetConnectAttr() function. Typical serialization scenarios include:

- Multiple processes trying to open connections to a single data source are serialized; a given data source can only be open to one process at a time.
- Multiple threads within a single process trying to open connections to a single data source are serialized.
- Multiple different processes might each have one connection open to multiple different data sources concurrently.

As an example, the following CLI /JDBC calls will set the connection timeout period to10. That is, the application will receive an error if there is another connection to the same database.

For **CLI:**

```
 int i = 10; // 10 seconds timeout
rc = SQLSetConnectAttr(hdbc, SQL_ATTR_LOGIN_TIMEOUT, (SQLPOINTER) i, 0);
```

For **JDBC:**

```
int waitTime = 10;
String url = "jdbc:db2e:mysample";

Properties prop = new Properties();
prop.setProperty("LOGIN_TIMEOUT", Integer.toString(waitTime));

Connection con = driver.connect(url,prop);
```

Notes:

1. The default timeout period is 0 second.
2. A multi-thread application may connect to a database using one thread and disconnect from the database using a different thread.
3. A file named ’DSYLOCK’ will be created when an application successfully connects to a database. If the application process terminates abnormally, the file DSYLOCK will be removed automatically.
4. Limitation: connection serialization may not work for database residing on network drive.
5. In a JDBC program, The timeout value will be ignored and set to zero if it is passed in a property to the DriverManager.getConnection() method.

Related tasks:

• “Connecting to the DB2 Everyplace database” on page 62

DB2 Everyplace databases on read-only media

The DB2 Everyplace database and an application can be run directly from read-only media such as CD-ROMs or ROM chips in embedded devices. For example, a sample application of a quarterly product catalog might be distributed on a CD-ROM to sales representatives. Each quarter the sales representatives receive a CD-ROM containing the complete company product catalog and a DB2

Everyplace application to browse, display, and query product information for products meeting a customers specific needs. The DB2 Everyplace application would run directly from the CD-ROM without having to be installed on a workstation first.

When DB2 Everyplace detects that an application is running on read-only media (or that the files are write protected), it is set to read-only mode. In this mode, updates, inserts, deletes, create and drop statements are prohibited, and will return an error. Note that for some select queries DB2 Everyplace creates temporary tables and files. These are created in the default temporary directory. On Win32 platforms this directory is designated by the environment variable *TEMP*. If the *TEMP* environment variable does not exist, it might also be designated *TMP*. In Linux, the `/tmp/` directory is used.

This feature is supported only on Win32 and Linux platforms.

Chapter 9. Piecemeal retrieval of data through CLI

There are two ways for a user to retrieve data from a DB2 Everyplace table through CLI:

- The application may choose to allocate the maximum memory the column value could occupy (based on its knowledge of a column in the result set via *SQLDescribeCol()* or prior knowledge), and bind it via *SQLBindCol()*.
- The application may call *SQLGetData()* to get column data into the application allocated buffer.

In the case of binary data (BLOB) or character data (CHAR or VARCHAR), the column can be very long. The application developer may not want to allocate a buffer big enough to hold the whole column, or may not be able to afford to allocate a buffer this large. Additionally, in some cases the application only requires some pieces of the column. In these scenarios, piecemeal retrieval of data is needed.

Procedure:

A feature of *SQLGetData()* allows the application to use repeated calls to obtain, in sequence, the value of a single column in more manageable pieces. Essentially, a call to *SQLGetData()* 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.

Syntax:

```
SQLRETURN   SQLGetData       (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLSMALLINT       TargetType,        /* fCType */
                SQLPOINTER        TargetValuePtr,    /* rgbValue */
                SQLINTEGER        BufferLength,      /* cbValueMax */
                SQLINTEGER   *FAR StrLen_or_IndPtr); /* pcbValue */
```

This will retrieve *BufferLength* bytes at a time, and *StrLen_or_IndPtr* indicates the number of bytes *remaining*. The return value of the function is SQL_SUCCESS_WITH_INFO (with SQLSTATE 01004) if there are bytes remaining. Otherwise, if the return value is SQL_SUCCESS, *StrLen_or_IndPtr* indicates the number of bytes that DB2 Everyplace CLI has available to return in the *TargetValuePtr* buffer. *SQLGetData()* can be used this way to retrieve long columns if the C data type (*TargetType*) is SQL_C_CHAR, SQL_C_BINARY, or if *TargetType* is SQL_C_DEFAULT and the column type denotes a binary or character string.

To use this feature of *SQLGetData()*, you must first set a statement attribute SQL_ATTR_GETDATA_MODE to SQL_PIECEMEAL_DATA. The default value of this attribute is SQL_CHUNK_DATA. The difference between these two modes is that, in SQL_CHUNK_DATA mode (which is the default mode), and when truncation occurs, `SQLGetData()`'s return value *StrLen_or_IndPtr* indicates the *total* number of bytes of this column, and the second call still retrieves data from the very beginning of the column.

Example code fragment:

```
sqlrc = SQLSetStmtAttr(hstmt, SQL_ATTR_GETDATA_MODE, (SQLPOINTER) SQL_PIECEMEAL_DATA, 0);
    SQLCHAR * stmt = (SQLCHAR *) "SELECT blobColumn FROM t1 where c1 = ?";
    sqlrc = SQLPrepare( hstmt, stmt, SQL_NTS ) ;
    sqlrc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, ...);
    sqlrc = SQLExecute( hstmt ) ;
    sqlrc = SQLFetch( hstmt );
    /* get BUFSIZ bytes at a time, bufInd indicates number of Bytes LEFT */
    sqlrc = SQLGetData (hstmt, 1, SQL_C_BINARY, (SQLPOINTER) buffer, BUFSIZ, &bufInd);
    while( sqlrc == SQL_SUCCESS_WITH_INFO ) {
        // handle BUFSIZ bytes of blob data in buffer
            :
        sqlrc = SQLGetData (hstmt, 1, SQL_C_BINARY, (SQLPOINTER) buffer, BUFSIZ, &bufInd);
    }
    if (sqlrc == SQL_SUCCESS) {  /* partial buffer on last GetData */
        // handle bufInd bytes of blob data in buffer
            :
    }
```

Chapter 10. Parameter markers

This chapter provides information about using parameter markers in DB2 Everyplace queries. The topics covered are:

- "Overview of parameter markers"
- "Examples of parameter marker usage"
- "DB2 Everyplace supported parameter markers" on page 72

Overview of parameter markers

For SQL statements that need to be executed many times, it is often beneficial to prepare the SQL statement once, and reuse the query plan by using parameter markers to substitute the input values during runtime.

In DB2 Everyplace, a parameter marker is represented by a "?" character, and indicates where an application variable is to be substituted inside an SQL statement. Parameter markers are referenced by number, and are numbered sequentially from left to right, starting at one. Before the SQL statement is executed, the application must bind a variable storage area to each parameter marker specified in the SQL statement. In addition, the bound variables must be a valid storage area, and must contain input data values when the prepared statement is executed against the database.

The following example illustrates an SQL statement containing two parameter markers.

```
SELECT * FROM customers WHERE custid = ? AND lastname = ?
```

Related concepts:

- "Examples of parameter marker usage"

Examples of parameter marker usage

DB2 Everyplace provides a rich set of standard interfaces including CLI/ODBC, JDBC, and ADO.NET to access data efficiently. The following example code snippets show the use of prepared statement with parameter markers for each data access API.

Consider the following table schema for table t1, where column c1 is the primary key for table t1.

Table 13. Example table schema

Column name	DB2 Everyplace data type	Nullable
c1	INTEGER	false
c2	SMALLINT	true
c3	CHAR(20)	true
c4	VARCHAR(20)	true
c5	DECIMAL(8,2)	true
c6	DATE	true

Table 13. Example table schema (continued)

Column name	DB2 Everyplace data type	Nullable
c7	TIME	true
c8	TIMESTAMP	true
c9	BLOB(30)	true

The following examples illustrate how to insert a row into table t1 using a prepared statement.

CLI Example

```
void parameterExample1(void)
{
   SQLHENV henv;
   SQLHDBC hdbc;
   SQLHSTMT hstmt;
   SQLRETURN rc;
   TCHAR server[] = _T("C:\\mysample\\");
   TCHAR uid[] = _T("db2e");
   TCHAR pwd[] = _T("db2e");
   long p1 = 10;
   short p2 = 100;
   TCHAR p3[100];
   TCHAR p4[100];
   TCHAR p5[100];
   TCHAR p6[100];
   TCHAR p7[100];
   TCHAR p8[100];
   char  p9[100];
   long len = 0;

   _tcscpy(p3, _T("data1"));
   _tcscpy(p4, _T("data2"));
   _tcscpy(p5, _T("10.12"));
   _tcscpy(p6, _T("2003-06-30"));
   _tcscpy(p7, _T("12:12:12"));
   _tcscpy(p8, _T("2003-06-30-17.54.27.710000"));

   memset(p9, 0, sizeof(p9));
   p9[0] = 'X';
   p9[1] = 'Y';
   p9[2] = 'Z';

   rc = SQLAllocEnv(&henv);
   // check return code ...

   rc = SQLAllocConnect(henv, &hdbc);
   // check return code ...

   rc = SQLConnect(hdbc, (SQLTCHAR*)server, SQL_NTS,
    (SQLTCHAR*)uid, SQL_NTS, (SQLTCHAR*)pwd, SQL_NTS);
   // check return code ...

   rc = SQLAllocStmt(hdbc, &hstmt);
   // check return code ...

   // prepare the statement
   rc = SQLPrepare(hstmt, _T("INSERT INTO t1 VALUES (?,?,?,?,?,?,?,?,?)"), SQL_NTS);
```

```
    // check return code ...

    // bind input parameters
    rc = SQLBindParameter(hstmt, (unsigned short)1, SQL_PARAM_INPUT,
        SQL_C_LONG, SQL_INTEGER, 4, 0, &p1, sizeof(p1), &len);
    // check return code ...

    rc = SQLBindParameter(hstmt, (unsigned short)2, SQL_PARAM_INPUT, SQL_C_LONG,
        SQL_SMALLINT, 2, 0, &p2, sizeof(p2), &len);
    // check return code ...

    len = SQL_NTS;
    rc = SQLBindParameter(hstmt, (unsigned short)3, SQL_PARAM_INPUT, SQL_C_TCHAR,
        SQL_CHAR, 0, 0, &p3[0], 100, &len);
    // check return code ...

    rc = SQLBindParameter(hstmt, (unsigned short)4, SQL_PARAM_INPUT, SQL_C_TCHAR,
        SQL_VARCHAR, 0, 0, &p4[0], 100, &len);
    // check return code ...

    rc = SQLBindParameter(hstmt, (unsigned short)5, SQL_PARAM_INPUT, SQL_C_TCHAR,
        SQL_DECIMAL, 8, 2, &p5[0], 100, &len);
    // check return code ...

    rc = SQLBindParameter(hstmt, (unsigned short)6, SQL_PARAM_INPUT, SQL_C_TCHAR,
        SQL_TYPE_DATE, 0, 0, &p6[0], 100, &len);
    // check return code ...

    rc = SQLBindParameter(hstmt, (unsigned short)7, SQL_PARAM_INPUT, SQL_C_TCHAR,
        SQL_TYPE_TIME, 0, 0, &p7[0], 100, &len);
    // check return code ...

    rc = SQLBindParameter(hstmt, (unsigned short)8, SQL_PARAM_INPUT, SQL_C_TCHAR,
        SQL_TYPE_TIMESTAMP, 0, 0, &p8[0], 100, &len);
    // check return code ...

    len = 3;
    rc = SQLBindParameter(hstmt, (unsigned short)9, SQL_PARAM_INPUT, SQL_C_BINARY,
        SQL_BINARY, 0, 0, &p9[0], 100, &len);
    // check return code ...

    // execute the prepared statement
    rc = SQLExecute(hstmt);
    // check return code ...

    rc = SQLFreeStmt(hstmt, SQL_DROP);
    // check return code ...

    rc = SQLDisconnect(hdbc);
    // check return code ...

    rc = SQLFreeConnect(hdbc);
    // check return code ...

    rc = SQLFreeEnv(henv);
    // check return code ...

}
```

JDBC Example

```
public static void parameterExample1() {

   String driver = "com.ibm.db2e.jdbc.DB2eDriver";
   String url    = "jdbc:db2e:mysample";
   Connection conn = null;
   PreparedStatement pstmt = null;

   try
   {
      Class.forName(driver);

      conn = DriverManager.getConnection(url);

      // prepare the statement
      pstmt = conn.prepareStatement("INSERT INTO t1 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)");

      // bind the input parameters
      pstmt.setInt(1, 1);
      pstmt.setShort(2, (short)2);
      pstmt.setString(3, "data1");
      pstmt.setString(4, "data2");
      pstmt.setBigDecimal(5, new java.math.BigDecimal("12.34"));
      pstmt.setDate(6, new java.sql.Date(System.currentTimeMillis() ) );
      pstmt.setTime(7,  new java.sql.Time(System.currentTimeMillis() ) );
      pstmt.setTimestamp (8,  new java.sql.Timestamp(System.currentTimeMillis() ) );
      pstmt.setBytes(9, new byte[] { (byte)'X', (byte)'Y', (byte)'Z' } );

      // execute the statement
      pstmt.execute();

      pstmt.close();

      conn.close();
   }
   catch (SQLException sqlEx)
   {
      while(sqlEx != null)
      {
          System.out.println("SQLERROR: \n" + sqlEx.getErrorCode() +
             ", SQLState: " + sqlEx.getSQLState() +
             ", Message: " + sqlEx.getMessage() +
             ", Vendor: " + sqlEx.getErrorCode() );
         sqlEx = sqlEx.getNextException();
      }
   }
   catch (Exception ex)
   {
      ex.printStackTrace();
   }
}
```

ADO.NET Example

[C#]

```
public static void ParameterExample1()
{
   DB2eConnection conn = null;
```

```
DB2eCommand cmd  = null;
String connString   = @"database=.\; uid=db2e; pwd=db2e";
int i = 1;

try
{
   conn = new DB2eConnection(connString);

   conn.Open();

   cmd = new DB2eCommand("INSERT INTO t1 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)", conn);

   // prepare the command
   cmd.Prepare();

   // bind the input parameters
   DB2eParameter p1 = new DB2eParameter("@p1", DB2eType.Integer);
   p1.Value = ++i;
   cmd.Parameters.Add(p1);

   DB2eParameter p2 = new DB2eParameter("@p2", DB2eType.SmallInt);
   p2.Value = 100;
   cmd.Parameters.Add(p2);

   DB2eParameter p3 = new DB2eParameter("@p3", DB2eType.Char);
   p3.Value = "data1";
   cmd.Parameters.Add(p3);

   DB2eParameter p4 = new DB2eParameter("@p4", DB2eType.VarChar);
   p4.Value = "data2";
   cmd.Parameters.Add(p4);

   DB2eParameter p5 = new DB2eParameter("@p5", DB2eType.Decimal);
   p5.Value = 20.25;
   cmd.Parameters.Add(p5);

   DB2eParameter p6 = new DB2eParameter("@p6", DB2eType.Date);
   p6.Value = DateTime.Now;
   cmd.Parameters.Add(p6);

   DB2eParameter p7 = new DB2eParameter("@p7", DB2eType.Time);
   p7.Value = new TimeSpan(23, 23, 23);
   cmd.Parameters.Add(p7);

   DB2eParameter p8 = new DB2eParameter("@p8", DB2eType.Timestamp);
   p8.Value = DateTime.Now;
   cmd.Parameters.Add(p8);

   byte []barr = new byte[3];
   barr[0] = (byte)'X';
   barr[1] = (byte)'Y';
   barr[2] = (byte)'Z';

   DB2eParameter p9 = new DB2eParameter("@p9", DB2eType.Blob);
   p9.Value = barr;
   cmd.Parameters.Add(p9);

   // execute the prepared command
   cmd.ExecuteNonQuery();
}
```

```
   catch (DB2eException e1)
   {
      for (int i=0; i < e1.Errors.Count; i++)
      {
         Console.WriteLine("Error #" + i + "\n" +
            "Message: " + e1.Errors[i].Message + "\n" +
            "Native: " + e1.Errors[i].NativeError.ToString() + "\n" +
            "SQL: " + e1.Errors[i].SQLState + "\n");
      }
   }
   catch (Exception e2)
   {
       Console.WriteLine(e2.Message);
   }
   finally
   {
      if (conn != null && conn.State != ConnectionState.Closed)
      {
            conn.Close();
            conn = null;
      }
   }
}
```

Related concepts:

- "Overview of parameter markers" on page 67

DB2 Everyplace supported parameter markers

A parameter marker, denoted by a question mark (?), is a place holder in an SQL statement whose value is obtained during statement execution. An application uses `SQLBindParameter()` to associate bind parameter markers to application variables. During the execution of the `SQLExecute()` and `SQLExecDirect()` DB2 CLI functions, the values of these variables replace each respective parameter marker. Data conversion might take place during the process. See Table 95 on page 266 for more information on the supported data type conversions.

DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. Table 14 lists the restrictions on parameter marker usage.

Table 14. Restrictions on parameter marker usage

Untyped parameter marker location	Data type
Expression: Alone in a select list	Error
Expression: Both operands of an arithmetic operator	Error
Predicate: Left-hand side operand of an IN predicate	Error
Predicate: Both operands of a relational operator	Error
Function: Operand of an aggregation function	Error

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

Parameter markers

Chapter 11. Cursor behavior within the context of a connection

General read cursor under write conflicts from another statement handle:

An application can have multiple statement handles doing read and write operations on the same table at the same time. Conflicts occur when one handle is performing a write operation on the table (for example, UPDATE, DELETE, or INSERT) while another handle is in the middle of a read or write operation. The DB2 Everyplace behavior is that the read cursor is stable and always reading the most current data. It survives the write conflicts, regardless of whether it is using an index or not. For example, suppose an application has two statement handles. Handle #1 is used to fetch rows from a table T whereas handle #2 is used to delete some rows from the same table. It is likely that each handle may have been created by different threads (for example, in a Java thread environment).

Here is a possible scenario:

```
// Fetch 2 rows from table T
   Statement handle 1: execute "SELECT A FROM T WHERE primary_key < 10"
   Statement handle 1: fetch one row; fetch another row
    // Delete some rows in table T
   Statement handle 2: prepare "DELETE FROM T WHERE primary_key = ?"
   Statement handle 2: execute
    // Continue to fetch one more row from T
   Statement handle 1: fetch one row
```

At this point in the execution, the statement handle #1 will be able to continue fetching the next row (if any), regardless of whether an index is used. In the scenario above, an index *is* used because there is a primary key. The idea is that DB2 Everyplace will try to reposition the cursor position of handle #1, using its current position, before advancing. If the current position does not exist anymore (for example, the row was deleted by another statement handle), then the cursor simply advances to the next position upon fetching. Likewise, if the next position was deleted by another statement handle, the cursor can skip over the ″hole″ to the following position.

Scrollable cursor under write conflicts from another statement handle:

Consider an example similar to the one in the previous section, but in which the read cursor is a scrollable cursor. If it is an ″insensitive″ scrollable cursor, this is not an issue because the result set does not change by definition. If the cursor is not ″insensitive″, the behavior matches a regular read cursor described above. Essentially, the read cursor behavior after the conflict is that the result set is recomputed according to the most current table data, and the start of the current row set is maintained. The cursor is advanced to the next row if the current row is deleted.

The following example illustrates the case with a scrollable cursor using CLP. Suppose table T has six rows:

```
create table T (a int, b int)
     create index idx1 on T(a)
     insert into T values (1, 1)
     insert into T values (2, 2)
```

```
insert into T values (3, 1)
insert into T values (3, 2)
insert into T values (3, 3)
insert into T values (4, 4)
```

Without loss of generosity, consider an example where the application has two statement handles, one for read and the other for delete.

```
Statement handle 1: enable scrollable cursor;
Statement handle 1: execute "SELECT A FROM T WHERE a < 10"
Statement handle 2: prepare "DELETE FROM T WHERE a = ?"
Statement handle 1: fetchscroll with SQL_FETCH_FIRST
-- get (1, 1)
Statement handle 1: fetchscroll with SQL_FETCH_NEXT
-- get (2, 2)
Statement handle 1: fetchscroll with SQL_FETCH_NEXT
-- get (3, 1)
Statement handle 2: execute
--- suppose delete row (2, 2)
Statement handle 1: fetchscroll with SQL_FETCH_NEXT
-- re-compute previous rows, and return (3, 2)
Statement handle 1: fetchscroll with SQL_FETCH_PRIOR
-- get (3, 1)
Statement handle 1: fetchscroll with SQL_FETCH_PRIOR
-- get (1, 1)  note that (2, 2) is gone
Statement handle 1: fetchscroll with SQL_FETCH_ABSOLUTE, offset 2
-- get (3, 1)  note that (2, 2) is gone
Statement handle 1: fetchscroll with SQL_FETCH_ABSOLUTE, offset 5
-- get (4, 4)
```

Cursor under commit and rollback, including autocommit mode:

Regardless of transaction or autocommit mode, an open cursor remains open across commit, and an open cursor is closed upon rollback.

Object dependency:

Preparing an SQL statement via a statement handle H may put some dependency on certain objects. For example, selecting rows from a table T via an index Idx requires the existence of the table T and the index Idx. If these objects were deleted by another statement handle (for example, if the index Idx was dropped), re-executing the statement through H will force a re-compilation of the SQL statement. As a result, the query plan may be different or an error may be returned.

Chapter 12. Encrypting local data

This chapter explains how to encrypt local data in a DB2 Everyplace database. The following topics are covered:

Overview of local data encryption

Encryption in DB2 Everyplace is designed for securing data on a mobile or embedded device. This topic provides a quick overview of local data encryption to help get you started. The following topics are discussed:

- Why use local data encryption?
- Local data encryption goals
- Creating the first encrypted table
- Subsequent access to encrypted tables
- Managing user privileges

Why use local data encryption?:

Consider a corporate sales application that contains customer contact data. A mobile salesperson might bring this data in their PDA to a customer visit. Unless the application or PDA provides a secure storage system, the data can easily be accessed using the application or by investigating the native file system of the mobile device. Encrypting sensitive data becomes a crucial aspect of protecting corporate information.

Local data encryption goals:

DB2 Everyplace provides a solution that allows for an application to implement a corporate security policy. The first goal is to encrypt secret or sensitive information stored in DB2 Everyplace tables. Data is encrypted using standard encryption methods like DES which implements encryption keys. The second goal is to provide a secure framework to be able to manage the keys used to encrypt the data. The user is required to provide a user ID and password at the time of database connection. For more information, see "Managing encryption privileges" on page 80.

For more information on using data encryption, see "Encryption using the DB2eCLP" on page 80.

Prerequisites:

This section describes how encryption is enabled for each platform and lists the libraries that are needed in addition to those required by the DB2 Everyplace database.

For Win32:

- plug-in library: CryptoPlugin.dll (provided by DB2 Everyplace)
- encryption library: Crypt32.dll (128-bit Cypher strength Encryption package, comes with IE5.5 or above). Go to http://www.microsoft.com/windows/ie/downloads/critical /q313675/download.asp.

For Windows CE/Pocket PC

- plug-in library: CryptoPlugin.dll (provided by DB2 Everyplace)
- encryption library: Microsoft High Encryption Pack for Pocket PC V1.0. Go to http://www.microsoft.com/mobile/pocketpc/downloads/ssl128.asp.

For Palm OS

- plug-in library: CryptoPlugin.PRC (provided by DB2 Everyplace)
- encryption library: PBSPKcs11.prc (provided by DB2 Everyplace)

For Linux/Neutrino

- plug-in library: libcryptoplugin.so (provided by DB2 Everyplace)
- encryption library: libpvcpkcs11.so (provided by DB2 Everyplace)

For Symbian

- plug-in library: CRYPTOPLUGIN.DLL (provided by DB2 Everyplace)
- encryption library: ECSPKCS11.DLL (provided by DB2 Everyplace)

Procedure:

To use data encryption:

1. Establish a connection to the DB2 Everyplace database.
2. Grant a user encryption privileges.
3. Create the first encrypted table.

   Subsequent access to encrypted tables: If a database contains the DB2eSYSUSERS table, any subsequent database connection will go through user authentication with the provided user ID and password. If authenticated fails, the application can access only non-encrypted tables. The application cannot create new encrypted tables, cannot drop existing encrypted tables, or access and update encrypted data.
4. Manage encryption privileges.

Establishing a connection to the DB2 Everyplace database

This task is part of the main task of Encrypting local data. After you have complete these steps, return to “Overview of local data encryption” on page 77.

Procedure:

Any interaction with the DB2 Everyplace database requires a connection to be established. In addition, in order for a user to access or create encrypted tables, the application must connect to DB2 Everyplace with non-empty user ID and password, using the following CLI function:

```
rc = SQLConnect(hdbc, "C:\temp\", SQL_NTS, "user1", SQL_NTS, "pwd1", SQL_NTS)
```

where *"C:\temp\"* is the directory of the database that the application is connect to, using the user ID *"user1"* and the password *"pwd1"*.

For a JDBC interface, a database connection can be established similarly.

Related concepts:

- "Connection serialization" on page 63

Granting a user encryption privileges

This task is part of the main task of Encrypting local data. After you have complete these steps, return to "Overview of local data encryption" on page 77.

Procedure:

Before creating the first encrypted table, the application must grant a user encryption privileges. For example, the application can issue the following SQL statement:

```
rc = SQLExecDirect(..., "GRANT ENCRYPT ON DATABSE TO \" user1\"" +
                         " using \"pwd1\" new \"pwd1\"", SQL_NTS)
```

Upon executing this SQL statement, DB2 Everyplace will create a system catalog table called DB2eSYSUSERS, and a row will be inserted into this table. This means that the user *"user1* is now registered with the corresponding password, and will now have all encryption privileges such as creating and accessing encrypted tables.

This table is tightly bound to the database and the encrypted data, and thus it cannot just be moved to another DB2 Everyplace database to access encrypted data. This is because a different databases will have different keys for encryption or decryption. As a result, if a person is allowed to access encrypted tables in a database, that person cannot access a different database using the same user ID and password. Like other system catalog tables, an application can retrieve rows using the SQL select statement but it cannot modify the data in this table using the INSERT, DELETE, UPDATE, CREATE, or DROP statements.

Related concepts:

- "Encryption using the DB2eCLP" on page 80

Related tasks:

- "Encryption using the DB2eCLP" on page 80
- "Creating an encrypted table"
- "Managing encryption privileges" on page 80

Creating an encrypted table

This task is part of the main task of Encrypting local data. After you have complete these steps, return to "Overview of local data encryption" on page 77.

Procedure:

Once you have established a connection to the DB2 Everyplace database and granted a user encryption privileges, the application can create encrypted tables using an extended CREATE TABLE statement. For example, you can create the following employee table:

```
SQLExecDirect(..., "CREATE TABLE EMPLOYEES (EMPNO INT PRIMARY KEY, NAME VARCHAR(30),
SALARY DECIMAL(10,2)) WITH ENCRYPTION", SQL_NTS)
```

Related concepts:

- "Encryption using the DB2eCLP"

Related tasks:

- "Encryption using the DB2eCLP"
- "Granting a user encryption privileges" on page 79
- "Managing encryption privileges"

Managing encryption privileges

This task is part of the main task of Encrypting local data. After you have complete these steps, return to "Overview of local data encryption" on page 77.

Procedure:

Once an application connects to a database with the authenticated user ID and password, the application can create new users, change passwords, or remove a registered user from the system. The syntax for creating a new user or changing a password is:

```
GRANT ENCRYPT ON DATABASE TO "newuser" USING "grantorpassword" NEW "newpassword"
```

The syntax for removing a registered user is:

```
REVOKE ENCRYPT ON DATABASE FROM "user"
```

Note: If all registered users are removed from the DB2eSYSUSERS table (using the REVOKE statement), no more encryption operations can be performed, including accessing existing encrypted table. There is no recovery mechanism.

Related concepts:

- "Encryption using the DB2eCLP"

Related tasks:

- "Encryption using the DB2eCLP"
- "Creating an encrypted table" on page 79
- "Granting a user encryption privileges" on page 79

Encryption using the DB2eCLP

This section contain an example of an interactive session designed to show you how to use data encryption in your applications. Comments have been added to explain each operation.

```
-- Encryption using DB2eCLP
--
-- This is an example encryption session using the provided sample
-- command line interface program DB2eCLP.
--
-- We only show the return code of a statement if it
-- failed, if it completed successfully we only show the results
-- of selects.
-- Commands which can be typed into DB2 Everyplace are
-- prefixed by the string "CLP:> ".
--
-- -- (CLI:SQLConnect, SQL:CREATE TABLE, SQL:GRANT, SQL:REVOKE)
--
-- When you start DB2eCLP you are automatically
```

```
-- connected to the default database (in the current directory).
-- This is equivalent to:
--
CLP:> CONNECT TO anything;

-- Since no specific path is given, just a name "anything", it connects
-- to the current directory.
--
-- We will now create a non-encrypted table containing a mapping of
-- some numbers to Swedish counting words.

CLP:> CREATE TABLE swedish(nummer INT, ord VARCHAR(32));
CLP:> INSERT INTO swedish VALUES(1, 'ett');
CLP:> INSERT INTO swedish VALUES(3, 'tre');
CLP:> INSERT INTO swedish VALUES(4, 'fyra');
CLP:> INSERT INTO swedish VALUES(5, 'fem');
CLP:> INSERT INTO swedish VALUES(7, 'sju');
CLP:> INSERT INTO swedish VALUES(99, 'nittionio');

-- Just have a look at the data
CLP:> SELECT * FROM swedish;

NUMMER      ORD
----------- --------------------------------
          1 ett
          3 tre
          4 fyra
          5 fem
          7 sju
         99 nittionio
6 row(s) returned.

-- We will now try to create the corresponding table for English,
-- but using encryption.
--
CLP:> CREATE TABLE english(number INT, word VARCHAR(32)) WITH ENCRYPTION;
Statement failed [sqlstate = 42501].

-- This fails because we are not authorized yet. As indicated by the error code.
-- So we need to connect again:
--
CLP:> CONNECT TO something USER jsk USING hemligt;

-- This connects to the same database (default/current directory) but with
-- a specific user identity "jsk" and using the password "hemligt".
-- The CONNECT TO command is not an SQL statement, thus is
-- interpreted by the DB2eCLP application. It will
-- disconnect and connect again to the DB2 Everyplace database
-- using:
--    SQLConnect(hdbc, "something", SQL_NTS, "jsk", SQL_NTS, "hemligt", SQL_NTS);
--
-- Now, we have to create the first authorized user. When the
-- first user is created it has to have the same name as the
-- logged in user and the same password:
--
CLP:> GRANT ENCRYPT ON DATABASE TO "jsk" USING "hemligt" NEW "hemligt";

-- Notice that for GRANT the name and passwords need to be inside
-- double quotes. This is because they are case-sensitive, and
-- the statement is passed directly to DB2 Everyplace.
--
-- Now that we have an authorized encryption user we can create the
-- encrypted table:
--

CLP:> CREATE TABLE english(number INT, word VARCHAR(32)) WITH ENCRYPTION;
CLP:> INSERT INTO english VALUES(1, 'one');
CLP:> INSERT INTO english VALUES(3, 'three');
CLP:> INSERT INTO english VALUES(4, 'four');
CLP:> INSERT INTO english VALUES(5, 'five');
CLP:> INSERT INTO english VALUES(7, 'seven');
CLP:> INSERT INTO english VALUES(99, 'ninety nine');

-- Just have a look at the data.
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
```

```
6 row(s) returned.

-- Select a large random number in Swedish:
--
CLP:> SELECT * FROM swedish WHERE nummer > 42;

NUMMER      ORD
----------- --------------------------------

         99 nittionio
1 row(s) returned.

-- Select a large random number in English:
--
CLP:> SELECT * FROM english WHERE number > 42;

NUMBER      WORD
----------- --------------------------------
         99 ninety nine
1 row(s) returned.

-- Translate 'fyra' to english:
--
CLP:> SELECT word FROM swedish, english WHERE number = nummer AND ord = 'fyra';

WORD
--------------------------------
four
1 row(s) returned.

-- Get a translation table:
--
CLP:> SELECT number, ord, word FROM swedish, english WHERE number = nummer;

NUMBER      ORD                              WORD
----------- -------------------------------- --------------------------------
          1 ett                              one
          3 tre                              three
          4 fyra                             four
          5 fem                              five
          7 sju                              seven
         99 nittionio                        ninety nine
6 row(s) returned.

--Attempt to authorize another user to access the encrypted data with her
-- own password:
--
CLP:> GRANT ENCRYPT ON DATABASE TO "xin" USING "notKnown" NEW "notKnown";
Statement failed [sqlstate = 42506].

-- Failed because the user who is logged in must validate himself
-- in order to add a new user, this is done by providing his password
-- after the USING clause.
--
CLP:> GRANT ENCRYPT ON DATABASE TO "xin" USING "hemligt" NEW "notKnown";

-- Let's reconnect with the new user:
--
CLP:> CONNECT TO samething USER xin USING notknown;
Statement failed [sqlstate = 42505].

-- This fails, because the password is not the same, thus will not generate
-- the same key and access is denied.
--
CLP:> CONNECT TO samething USER ksin USING notKnown;

-- This will not fail, because the user ksin does not exist, and we therefore
-- do not attempt to authenticate the user.
-- However, using SQLGetInfo one can distinguish this case
-- from the case where an user was successfully authenticated.
--
CLP:> SELECT * FROM swedish;

NUMMER      ORD
----------- --------------------------------
          1 ett
          3 tre
          4 fyra
          5 fem
          7 sju
         99 nittionio
6 row(s) returned.

-- Selecting non-encrypted data works fine, however encrypted data cannot
```

```
-- be read/updated unless an authorized user is connected:
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

-- Connect as the new user, finally with correct username and password.
--
CLP:> CONNECT TO samething USER xin USING notKnown;

-- Verify that we are authenticated and can access the data.
--
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- Add another user:
--
CLP:> GRANT ENCRYPT ON DATABASE TO "thf" USING "notKnown" NEW "heimlich";

-- List currently existing users:
--
CLP:> SELECT username, grantorname FROM "DB2eSYSUSERS";

USERNAME            GRANTORNAME
------------------- -------------------
jsk                 jsk
xin                 jsk
thf                 xin
3 row(s) returned.

-- Again connect as "jsk":
--
CLP:> CONNECT TO itagain USER jsk USING hemligt;
Statement completed successfully.

-- Attempt to change the password to "secret":
--
CLP:> GRANT ENCRYPT ON DATABASE TO "jsk" USING "secret" NEW "secret";
Statement failed [sqlstate = 42506].

-- Ah, we failed because we need to supply first our old password and then
-- the new password:
--
CLP:> GRANT ENCRYPT ON DATABASE TO "jsk" USING "hemligt" NEW "secret";

-- Try the new password:
--
CLP:> CONNECT TO itagain USER jsk USING secret;

-- Make sure we can access encrypted ata:
--
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- Let's remove encryption privilege from "xin":
--
CLP:> REVOKE ENCRYPT ON DATABASE FROM "xin";

-- List users
--
CLP:> SELECT username, grantorname FROM "DB2eSYSUSERS";

USERNAME            GRANTORNAME
------------------- -------------------
jsk                 jsk
thf                 xin
2 row(s) returned.
```

```
-- Connect again to the now non-existing user, without error.
--
CLP:> CONNECT TO thedatabase USER xin USING idontknow;

-- Attempt to do encryption operations without authorization:
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

CLP:> DROP TABLE english;
Statement failed [sqlstate = 42501].

CLP:> REVOKE ENCRYPT FROM "jsk";
Statement failed [sqlstate = 42601].

CLP:> GRANT ENCRYPT ON DATABASE TO "xin" USING "idontknow" NEW "idontknow";
Statement failed [sqlstate = 42502].

-- Connect as "thf":
--
CLP:> CONNECT TO thedatabase USER thf USING heimlich;

-- Check that we can read encrypted data:
--
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- Let's remove the connected user's privilege:
--
CLP:> REVOKE ENCRYPT ON DATABASE FROM "thf";

-- Make sure he cannot access the data anymore:
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

-- If we connect to the database as the only remaining user "jsk"
--
CLP:> CONNECT TO thedatabase USER jsk USING secret;

-- We remove the connected user, that user can not access the data anymore.
-- Actually, there is no way to access the encrypted data ever again.
--

CLP:> REVOKE ENCRYPT ON DATABASE FROM "jsk";

-- Make sure there are no users left:
--
CLP:> SELECT username, grantorname FROM "DB2eSYSUSERS";

USERNAME            GRANTORNAME
------------------- -------------------
0 row(s) returned.

-- We should now not be able to access the encrypted data.
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

-- This concludes the example session.
```

Related tasks:

- "Overview of local data encryption" on page 77
- "Granting a user encryption privileges" on page 79
- "Creating an encrypted table" on page 79
- "Managing encryption privileges" on page 80

Part 3. Sample applications

Chapter 13. The sample C/C++ applications

At least one sample C/C++ application is provided for each operating system. See the appropriate client directory for the complete sample applications with source code.

Related tasks:

- "Developing DB2 Everyplace C/C++ applications" on page 9

Related reference:

- "Supported C/C++ development tools" on page 9
- "C/C++ supported operating systems" on page 11

Chapter 14. The sample Java applications

This chapter provides information about the sample Java applications. The topics covered are:

Overview of the sample Java applications

This topic describes the `DB2eAppl.java` and the `DB2eJavaCLP.java` sample applications.

Sample 1: DB2eAppl.java:

`DB2eAppl.java` demonstrates how to code a JDBC application for DB2 Everyplace.

The major steps of the `DB2eAppl.java` application are:

Step 1:
Import the `java.sql` package.

Step 2:
Load the DB2 Everyplace JDBC driver `com.ibm.db2e.jdbc.DB2eDriver`.

Step 3:
Connect to the database in the current directory, the directory that the `DB2eAppl.java` application will be run in.

Step 4:
Create a Statement object.

Step 5:
Set up the very simple sample database, consisting of an EMPLOYEE table that contains two records. This is done using the `executeUpdate(String sql)` method of the `java.sql.Statement` interface.

Step 6:
Select all records from the EMPLOYEE table, and retrieve the rows by using the `next()` method of the `java.sql.ResultSet` interface.

Step 7:
Drop the EMPLOYEE table from the database and release database and JDBC resources.

The `DB2eAppl.java` source code below contains comments that show where the steps explained above are being used.

```
import java.sql.*; //Step 1

public class DB2eAppl
{
    public static void main(String[] args) {

        String driver = "com.ibm.db2e.jdbc.DB2eDriver";
        String url    = "jdbc:db2e:mysample";
```

```
        try {
            Class.forName(driver); //Step 2
            Connection con = DriverManager.getConnection(url); //Step 3
            Statement st = con.createStatement(); //Step 4

            //Create table: employee  //Step 5
            st.executeUpdate("CREATE TABLE employee (EMPNO CHAR(6), FIRSTNAME VARCHAR(12))");
            System.out.println("*** Created table: employee");

            //Add records to employee
            st.executeUpdate("INSERT INTO employee VALUES ('112233','John')");
            st.executeUpdate("INSERT INTO employee VALUES ('445566','Mary')");
            System.out.println("*** Inserted two records");

            //Query and display results //Step 6
            ResultSet rs = st.executeQuery("SELECT * FROM employee");
            System.out.println("*** Query results:");
            while (rs.next()) {
                System.out.print("EMPNO=" + rs.getString(1) + ", ");
                System.out.println("FIRSTNAME=" + rs.getString(2));
            }

            //Delete table: employee  //Step 7
            st.executeUpdate("Drop table employee");
            System.out.println("*** Deleted table: employee");

            rs.close();
            st.close();
            con.close();

        } catch (SQLException sqlEx) {
            while(sqlEx != null)
            {
                System.out.println("[SQLException] " +
                                   "SQLState: " + sqlEx.getSQLState() +
                                   ", Message: " + sqlEx.getMessage() +
                                   ", Vendor: " + sqlEx.getErrorCode() );
                sqlEx = sqlEx.getNextException();
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Sample 2: DB2eJavaCLP.java:

`DB2eJavaCLP.java` is a Java command line processor for DB2 Everyplace.

Restriction: On Palm OS, the `DB2eJavaCLP.java` sample application is not supported.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17
- "Compiling and running sample Java applications on Palm OS targets" on page 91
- "Creating a WSDD project for DB2eAppl.java for Palm OS targets" on page 92
- "Creating a WSDD project and adding jar files to the build path for DB2eAppl.java for non-Palm OS targets" on page 97
- "Running DB2eAppl.java on Win32" on page 98
- "Running DB2eAppl.java on Windows CE" on page 99
- "Running DB2eAppl.java on a Palm OS emulator" on page 94
- "Running DB2eAppl.java on QNX Neutrino or embedded Linux" on page 101
- "Running DB2eAppl.java on Symbian" on page 101
- "Compiling and running sample Java applications on non-Palm OS targets" on page 95

Related concepts:

- "The sample Java native synchronization applications" on page 113

Compiling and running sample Java applications on Palm OS targets

The following set of topics describe how to compile and run the sample `DB2eAppl.java` Java code on Palm OS targets.

It is recommended to use WebSphere Studio Device Developer (WSDD) as your development environment. WSDD uses the J9 VM, which may not support the processor type of your device. If you use a different development environment and JVM, make sure the JVM supports the JNI, because the DB2 Everyplace JDBC driver uses the JNI. Other compatible JVMs include Sun PersonalJava, Insignia Jeode, and NSIcom CrEme. Currently, if your target is Palm OS, you *must* use the J9 VM that comes with WSDD. The evaluation version of WSDD can be downloaded from http://www.ibm.com/software/pervasive/products/wsdd/.

Prerequisites:

1. Ensure that you have the following software installed:
   - WSDD 5.5, which includes the J9 Java Virtual Machine
2. Prepare your target and development environment according to WSDD documentation. Verify your WSDD installation by building and running WSDD sample applications.
3. Install DB2 Everyplace on your target device. See the *DB2 Everyplace Installation and User's Guide* for detailed instructions.

Procedure:

To compile and run the sample Java code on Palm OS targets:

1. Install WCE Tooling for WSDD.
2. Create a WSDD project for `DB2eAppl.java`.
3. Add the DB2 Everyplace JDBC Driver and java.sql package to the build path.
4. Import `DB2eAppl.java` into WSDD.
5. Run `DB2eAppl.java` on a Palm OS emulator.

Related tasks:

- "Compiling and running sample Java applications on non-Palm OS targets" on page 95

Installing WCE Tooling for WSDD for Palm OS targets

This task is part of the main task of Compiling and running sample Java applications on Palm OS targets. After you complete these steps, return to "Compiling and running sample Java applications on Palm OS targets."

Procedure:

1. In WSDD, click **Help —> Software Updates —> Update Manager** to open the Install/Update Perspective
2. In the Feature Updates View of the Install/Update Perspective, expand the following: **Sites to Visit —> WebSphere Custom Environment —> WebSphere Custom Environment**.
3. Select WCE Tooling for WSDD 5.5.0.
4. In the Preview View, click **Install**.

5. Follow the installation instructions to install the WCE Tooling for WSDD feature.
6. If you select to use the jclXtr class library, you must also install the WCE jclXtr Class Library by following similar steps.

Return to "Compiling and running sample Java applications on Palm OS targets" on page 91.

Creating a WSDD project for DB2eAppl.java for Palm OS targets

Two versions of the DB2 Everyplace JDBC Driver for Palm OS are available. One version is compatible with the J2ME CLDC Configuration. The other version is compatible with the JCL Extreme Palm Custom Configuration provided by WSDD. Follow the appropriate steps to create the type of project you need.

This task is part of the main task of Compiling and running sample Java applications on Palm OS targets. After you complete these steps, return to "Compiling and running sample Java applications on Palm OS targets" on page 91.

Procedure:

To create a WSDD Palm OS project using the jclCldc configuration::

1. In WSDD, click **Window —>Open Perspective —>Java** to switch to the Java Perspective.
2. Create the DB2 Everyplace Sample for Palm OS CLDC project:
   a. Click **File —>New —>Other**.
   b. On the Select page of the New Project window, select J2ME for J9 in the left pane, select Create J2ME project in the right pane, then click **Next**.
   c. On the J2ME Project page of the New Project window, type `DB2 Everyplace Sample for Palm OS CLDC` in the Project name field, then click **Next**.
   d. On the Library selection page of the New Project window, select WME jclCldc (jclCldc), then click **Finish**.

To create a WSDD Palm OS project using the jclXtr configuration::

1. In WSDD, click **Window —>Open Perspective —>Java** to switch to the Java Perspective.
2. Create the DB2 Everyplace Sample for Palm OS XTR project:
   a. Click **File —>New —>Other**.
   b. On the Select page of the New Project window, select WCE for J9 in the left pane, select Create WCE project in the right pane, then click **Next**.
   c. On the Custom Project page of the New Project window, type `DB2 Everyplace Sample for Palm OS XTR` in the Project name field, then click **Next**.
   d. On the Library selection page of the New Project window, select WCE jclXtr (jclXtr), then click **Finish**.

Return to "Compiling and running sample Java applications on Palm OS targets" on page 91.

Adding the DB2 Everyplace JDBC Driver and java.sql package to the build path

The following steps apply to the DB2 Everyplace Sample for Palm OS CLDC project. The DB2 Everyplace Sample for Palm OS XTR project involves similar steps.

This task is part of the main task of Compiling and running sample Java applications on Palm OS targets. After you complete these steps, return to "Compiling and running sample Java applications on Palm OS targets" on page 91.

Procedure:

To add the DB2 Everyplace JDBC Driver (and `java.sql` package) to the build path:

1. Right-click the DB2 Everyplace Sample for Palm OS CLDC project in the Package Explorer view in the Java Perspective, then click **Properties** from the pop-up menu.
2. In the properties window that opens, click Java Build Path in the left pane, then click the Libraries tab in the right pane.
3. Click **Add External JARs**. In the JAR Selection window, browse to `<DB2Everyplace>\Clients\PalmOS\database\JDBC\cldc\db2ejdbc.jar`, then click **Open**.
4. Repeat the previous step to add `database_enabler_cldc.jar` and `DB2eJDBC_Cldc_maps.jar` to the build path.
5. Back in the Properties for DB2 Everyplace Sample for Palm OS CLDC window, click **OK**.

Return to "Compiling and running sample Java applications on Palm OS targets" on page 91.

Importing DB2eAppl.java into WSDD for Palm OS

The following steps apply to the DB2 Everyplace Sample for Palm OS CLDC project. The DB2 Everyplace Sample for Palm OS XTR project involves similar steps.

This task is part of the main task of Compiling and running sample Java applications on Palm OS targets. After you complete these steps, return to "Compiling and running sample Java applications on Palm OS targets" on page 91.

Procedure:

To import `DB2eAppl.java` into WSDD:

1. In the Package Explorer View in the Java Perspective, right-click the `src` folder in the DB2 Everyplace Sample for Palm OS CLDC project, then click **Import** from the pop-up menu.
2. On the Select page of the Import window, select File system as the import source, then click **Next**.
3. On the File system page of the Import window, click **Browse**.
4. Browse to the `<DB2Everyplace>\Clients\PalmOS\database\JDBC\cldc\sample` folder, then click **OK**.
5. Select the **DB2eAppl.java** check box in the right pane, then click **Finish**.

Return to “Compiling and running sample Java applications on Palm OS targets” on page 91.

Running DB2eAppl.java on a Palm OS emulator

The following steps apply to the DB2 Everyplace Sample for Palm OS CLDC project. The DB2 Everyplace Sample for Palm OS XTR project involves similar steps.

This task is part of the main task of Compiling and running sample Java applications on Palm OS targets. After you complete these steps, return to “Compiling and running sample Java applications on Palm OS targets” on page 91.

Prerequisites:

If you have not already set up your system to use the DB2 Everyplace JDBC driver, install the following files for the JDBC driver on your device:

<DB2 Everyplace>\Clients\PalmOS\database\JDBC\cldc\DB2eJDBC.prc
<DB2 Everyplace>\Clients\PalmOS\database\JDBC\cldc\DB2eJDBC_Cldc.prc

If you are working with the DB2 Everyplace Sample for Palm OS XTR project, install the following files for the JDBC driver on your device instead:

<DB2 Everyplace>\Clients\PalmOS\database\JDBC\xtr\DB2eJDBC.prc
<DB2 Everyplace>\Clients\PalmOS\database\JDBC\xtr\DB2eJDBC_Xtr.prc

Procedure:

To run `DB2eAppl.java` on a Palm OS emulator:

1. Configure the Palm OS emulator:
   a. Click **Devices —> Configure**.
   b. In the Device Configurations window, select Palm Emulator in the left pane, then click **New**.
   c. In the configuration that appears on the right, enter the following information:
      - In the **Device name** field, type `DB2 Everyplace Palm Emulator`.
      - In the **Palm emulator executable** field, browse to <*PalmEmulator*>\Emulator.exe, where <*PalmEmulator*> is the directory where you installed the Palm Emulator.
      - In the **Emulator run arguments** field, type -psf <*file*>.psf, where <*file*>.psf is a .psf file that has DB2 Everyplace and the J9 VM installed.
   d. Click Apply, then click OK.
2. Build `DB2eAppl.java`.
   a. In the Package Explorer View in the Java Perspective, double-click the `wsddbuild.xml` file for the DB2 Everyplace Sample for Palm OS CLDC project.
   b. In the editor for `wsddbuild.xml`, click **Add Build**.
   c. Select J9 for Palm 68k from the **Platform** list, keep the default values in the **Main class** and **Buildname** fields, then click **Next**.
   d. On the PalmOS settings page, type `DB2e` in the **Creator id** field, and type `DB2eAppl` in the **App Name** field, then click **Next**.
   e. On the Jxelink Options page, keep the defaults and click **Finish**.
3. Modify the `DB2eAppl.jxeLinkOptions` file:

a. In the Package Explorer in the Java Perspective, expand the `palm68k` folder for the DB2 Everyplace Sample for Palm OS CLDC project. Double-click DB2eAppl.jxeLinkOptions to open the editor for DB2eAppl.jxeLinkOptions.
b. In the editor, click the Input tab. Click **New** for the Read classes from map files (prereq) section. In the Add prereq window that appears, enter `DB2eJDBC_Cldc` for the prereq, then click **OK**. **Note:** If you are working with the DB2 Everyplace Sample for Palm OS XTR project, enter `DB2eJDBC_Xtr` for the prereq instead, and skip the next two steps.
c. In the editor, click the Jxe tab. Under Jxe Platform Information, click **New** for the Use VM options when running jxe section.
d. In the Add VM option window that appears, enter `-jcl:cldc:loadlibrary=db2ejdbc` for the VM option, then click **OK**.
e. Type Ctrl+S to save your changes.
f. In the editor for `wsddbuild.xml`, select jxe2prc palm68k/DB2eAppl, then click **Perform Build**.

4. Run `DB2eAppl.java`.
   a. Click **Run —>Run** from the main menu. The Launch Configurations window opens.
   b. In the Launch Configurations window, select Device Java Application in the left pane, then click **New**.
   c. In the configuration that appears in the right pane, type `DB2eAppl Palm CLDC` in the **Name** field.
   d. In the Java Application panel, enter the following information:
      1) In the Project field, browse to DB2 Everyplace Sample for Palm OS CLDC.
      2) Click **Search** in the Java Application field.
      3) In the Select Target window, select `DB2eAppl.prc (Target "jxe2prc palm68k/DB2eAppl" in wsddbuild.xml)`, then click **Finish**.
      4) Select DB2 Everyplace Palm Emulator from the **Device or JRE** list.
      5) Back in the Launch Configurations window, click **Apply**, then click **Run**. A Palm Emulator should start and run DB2eAppl. You should see output for the sample application either on the Palm Emulator screen, or in the `j9stdout.txt` file in the directory that your .psf file is in. If you did not modify the J9 Java VM ″Display Stdout on Screen″ preference, the output will be in the `j9stdout.txt` file. Also check `j9stderr.txt` for errors.

Return to “Compiling and running sample Java applications on Palm OS targets” on page 91.

Compiling and running sample Java applications on non-Palm OS targets

The following set of topics describe how to compile and run the sample Java code using WebSphere Studio Device Developer (WSDD) 5.5 and the J9 Java Virtual Machine.

It is recommended to use WSDD as your development environment. WSDD uses the J9 VM, which may not support the processor type of your device. If you use a different development environment and JVM, make sure the JVM supports the JNI, because the DB2 Everyplace JDBC driver uses the JNI. Other compatible JVMs include Sun PersonalJava, Insignia Jeode, and NSIcom CrEme. The evaluation

version of WSDD can be downloaded from http://www.ibm.com/software/pervasive/products/wsdd/.

Prerequisites:

1. Ensure that you have the following software installed:
   - WSDD 5.5, including the J9 Java Virtual Machine, or some other compatible JVM
2. Prepare your target and development environment according to WSDD documentation. Verify your WSDD installation by building and running WSDD sample applications.
3. Install DB2 Everyplace on your target device. See the *DB2 Everyplace Installation and User's Guide* for detailed instructions.

Procedure:

To compile and run the sample Java code on non-Palm OS targets:

1. Install WCE Tooling for WSDD.
2. Create a WSDD project and add jar files to the build path for `DB2eAppl.java`.
3. Import `DB2eAppl.java` into WSDD.
4. Run `DB2eAppl.java`. The steps vary depending on your operating system.
   - "Running DB2eAppl.java on Win32" on page 98
   - "Running DB2eAppl.java on Windows CE" on page 99
   - "Running DB2eAppl.java on QNX Neutrino or embedded Linux" on page 101
   - "Running DB2eAppl.java on Symbian" on page 101

Related tasks:

- "Compiling and running sample Java applications on Palm OS targets" on page 91

Installing WCE Tooling for WSDD for non-Palm OS targets

This task is part of the main task of Compiling and running sample Java applications on non-Palm OS targets. After you complete these steps, return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95.

Procedure:

1. In WSDD, click **Help —> Software Updates —> Update Manager** to open the Install/Update Perspective
2. In the Feature Updates View of the Install/Update Perspective, expand the following: **Sites to Visit —> WebSphere Custom Environment —> WebSphere Custom Environment**.
3. Select WCE Tooling for WSDD 5.5.0.
4. In the Preview View, click **Install**.
5. Follow the installation instructions to install the WCE Tooling for WSDD feature.
6. Install the WCE Database Enabler Library and the WCE jclMax Class Library by following similar steps.

Return to “Compiling and running sample Java applications on non-Palm OS targets” on page 95.

Creating a WSDD project and adding jar files to the build path for DB2eAppl.java for non-Palm OS targets

This task is part of the main task of Compiling and running sample Java applications on non-Palm OS targets. After you complete these steps, return to “Compiling and running sample Java applications on non-Palm OS targets” on page 95.

Procedure:

1. In WSDD, click **Window —>Open Perspective —>Java** to switch to the Java Perspective
2. Create the DB2 Everyplace Sample project:
   a. Click **File —>New —>Other**.
   b. On the Select page of the New Project window, select WCE for J9 in the left pane, select Create WCE project in the right pane, then click **Next**.
   c. On the Custom Project page of the New Project window, type `DB2 Everyplace Sample` for the project name, then click **Next**.
   d. On the Library selection page of the New Project window, select WCE jclMax (jclMax), then click **Next**.
   e. On the Java Settings page of the New Project window, click the **Libraries** tab.
   f. Add `db2ejdbc.jar` to the build path:
      1) Click **Add External JARs**.
      2) In the JAR Selection window, browse to ***<DB2Everyplace>***`\Clients\Win32\database\jdbc\db2ejdbc.jar`, then click **Open**.

      .
   g. Back on the Java Settings page, add `database_enabler.jar` to the build path.
      1) Click **Add External JARs**.
      2) In the JAR Selection window, browse to *<WSDD>*`\wsdd5.0\ive\lib\jclMax\database_enabler.jar`, then click **Open**.
   h. Back on the Java Settings page of the New Project window, click **Finish**.

Return to “Compiling and running sample Java applications on non-Palm OS targets” on page 95

Importing DB2eAppl.java into WSDD for non-Palm OS targets

This task is part of the main task of Compiling and running sample Java applications on non-Palm OS targets. After you complete these steps, return to “Compiling and running sample Java applications on non-Palm OS targets” on page 95.

Procedure:

To import `DB2eAppl.java` into WSDD for non-Palm OS targets:

1. In the Package Explorer in the Java Perspective, expand the DB2 Everyplace Sample project, select the `src` folder, then click **File —>Import**.
2. On the **Select** page of the Import window, select File system as the import source, then click **Next**.
3. On the **File system** page of the Import window, click **Browse** for the **Directory** field, browse to the *<DB2Everyplace>*`\Clients\Win32\database\jdbc` folder, then click **OK**.
4. Back on the **File system** page of the Import window, select the **DB2eAppl.java** check box in the right pane, then click **Finish**.

Return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95

Running the sample Java applications

This chapter describes how to run the sample Java applications. The topics covered are:

Running DB2eAppl.java on Win32

This task is part of the main task of Compiling and running sample Java applications on non-Palm OS targets. After you complete these steps, return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95.

Prerequisites:

If you have not already set up your system to use the DB2 Everyplace JDBC driver:

1. Using the set command, include the following directory in your PATH system variable: `<DB2Everyplace>\Clients\Win32\database\x86`
2. Using the set command, include the following file in your CLASSPATH system variable: `<DB2Everyplace>\Clients\Win32\database\jdbc\db2ejdbc.jar`

Note: If WSDD is open, you will need to restart it for these changes to be reflected in WSDD.

Procedure:

To run `DB2eAppl.java` on a Windows workstation:

1. Build `DB2eAppl.java`:
   a. In the Package Explorer view in the Java Perspective, double-click the `wsddbuild.xml` file for the DB2 Everyplace Sample project.
   b. In the editor for `wsddbuild.xml`, click **Add Build**.
   c. In the Create New Ant Build Target window, click **Browse** for the **Main class** field.
   d. In the window that opens, select DB2eAppl - (default package) - DB2 Everyplace Sample/src, then click **Finish**.

e. Back in the Create New Ant Build Target window, select J9 for Windows X86 in the platform list, keep the default in the Buildname field, then click **Next**.

f. On the Jxelink Options page, keep the defaults and click **Finish**.

g. Back in the editor for `wsddbuild.xml`, select smartlink winx86/DB2eAppl, then click **Perform Build**.

2. Run `DB2eAppl.java`.

   a. Click **Run —> Run**. The Launch Configurations window opens.

   b. In the Launch Configurations window, select Java Application in the left pane, then click **New**.

   c. In the configuration that appears in the right pane, type `DB2eAppl Win32` in the **Name** field.

   d. On the Main page, complete the following steps:

      - Click **Browse** for the **Project** field. In the Project Selection window, select DB2 Everyplace Sample, then click **OK**.
      - Click **Search** for the **Main class** field. In the Choose Main Type window, select DB2eAppl, then click **OK**.

   e. Back in the Launch Configurations window, click **Apply**, then click **Run**. You should see output for the sample application in the WSDD Console.

Return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95.

Running DB2eAppl.java on Windows CE

This task is part of the main task of Compiling and running sample Java applications on non-Palm OS targets. After you complete these steps, return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95.

Prerequisites:

If you have not already set up your system to use the DB2 Everyplace JDBC driver, complete the following steps:

1. Copy the following files to the \Windows directory on your device:`<DB2Everyplace>\Clients \WinCE \database \`*`proc`* `\`*`ver`* `\db2ejdbc.dll<DB2Everyplace>\Clients \WinCE \database \jdbc \db2ejdbc.jar`where *proc* is the processor type and *ver* is the version number of the Windows CE operating system on your device.
2. Using the Windows CE Remote Registry Editor, modify the registry for your device to include the following files on the classpath of the device:

   ```
   \Windows\db2ejdbc.jar
   \wsdd\lib\jclMax\database_enabler.jar
   ```

   assuming you installed J9 under the root directory of your device.

   Alternatively, you can update the DB2eAppl shortcut generated by WSDD to include the above files on the classpath:

```
256#"\wsdd\bin\j9.exe" "-Xbootclasspath:\Windows\db2ejdbc.jar;
\wsdd\lib\jclMax\database_enabler.jar;\wsdd\lib\jclMax\classes.zip;
\wsdd\lib\jclMax\locale.zip;\wsdd\lib\charconv.zip" "-jcl:max" "-jxe:\Temp\DB2eAppl.jxe"
```

Procedure:

To run DB2eAppl.java on a Windows CE device:

1. Configure your Windows CE device.
   a. Click **Devices —> Configure**.
   b. In the Device Configurations window, select PocketPC Handheld in the left pane, then click **New**.
   c. In the configuration that appears on the right, complete the following steps:
      1) In the **Device name** field, type DB2 Everyplace PocketPC Handheld.
      2) Click **Browse** for the **J9 runtime location** field. In the Browse for Folder on Device window, select wsdd (assuming that you installed J9 under the root directory of your device), then click **OK**.
      3) Click **Browse** for the **Application location install** field. In the Browse for Folder on Device window, select Temp, then click **OK**.
      4) Click **Browse** for the **Shortcut install location** field. In the Browse for Folder on Device window, select Temp, then click **OK**.
   d. Back in the Device Configurations window, click **Apply**, then click **OK**.
2. Build DB2eAppl.java.
   a. In the Package Explorer pane in the Java Perspective, double-click the wsddbuild.xml file for the DB2 Everyplace Sample project.
   b. In the editor for wsddbuild.xml, click **Add Build**.
   c. In the Create New Ant Build Target window, click **Browse** for the **Main class** field.
   d. In the window that opens, select DB2eAppl - (default package) - DB2 Everyplace Sample/src, then click **Finish**.
   e. Back in the Create New Ant Build Target window, select J9 for PocketPC ARM from the **Platform** list, keep the default in the **Buildname** field, then click **Next**.
   f. On the **Jxelink Options** page, keep the defaults and click **Finish**.
   g. In the editor for wsddbuild.xml, select smartlink ppcarm/DB2eAppl, then click **Perform Build**.
3. Run DB2eAppl.java:
   a. Click **Run —> Run**. The Launch Configurations window opens.
   b. In the Launch Configurations window, select Device Java Application in the left pane, then click **New**.
   c. In the configuration that appears in the right pane, type DB2eAppl WinCE in the **Name** field.
   d. On the Main page, complete the following steps:
      1) Click **Browse** for the **Project** field. In the Project Selection window, select DB2 Everyplace Sample, then click **OK**.
      2) Click **Search** for the Java Application field. In the Select Target window, select DB2eAppl.jxe (Target "smartlink ppcarm/DB2eAppl" in wsddbuild.xml), then click **Finish**.
      3) Select DB2 Everyplace PocketPC Handheld from the **Device or JRE** list.
   e. Back in the Launch Configurations window, click **Apply**, then click **Run**. You should see output for the sample application in the J9 Console on your device.

Return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95.

Running DB2eAppl.java on QNX Neutrino or embedded Linux

This task is part of the main task of Compiling and running sample Java applications on non-Palm OS targets. After you complete these steps, return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95.

Prerequisites:

If you have not already set up your system to use the DB2 Everyplace JDBC driver:

1. Using the export command, include the directory (or directories) that contain the appropriate libdb2e.so and libdb2ejdbc.so native libraries on your device in your LD_LIBRARY_PATH system variable.

Procedure:

To run `DB2eAppl.java` on a QNX Neutrino or embedded Linux device:

1. Build `DB2eAppl.java`.
   a. In the Package Explorer pane in the Java Perspective, double-click the `wsddbuild.xml` file for the DB2 Everyplace Sample project.
   b. In the editor for `wsddbuild.xml`, click **Add Build**.
   c. In the Create New Ant Build Target window, click **Browse** to browse for the main class. In the Select Target window that opens, select DB2eAppl - (default package) - DB2 Everyplace Sample/src, then click **Finish**.
   d. Back on the Set up build page of the Create New Ant Build Target window, select the appropriate platform from the **Platform** list, keep the default in the Buildname field, then click **Next**.
   e. On the Jxelink Options page, keep the defaults and click **Finish**.
   f. Back in the editor for wsddbuild.xml, select the appropriate build and click **Perform Build**.
2. Run `DB2eAppl.java`.
   a. Copy the appropriate DB2eAppl.jxe file to your device from

      ```
      <WSDD>\workspace\DB2 Everyplace Sample\<target>
      ```

      , where <target> represents the target device and processor type.
   b. Start the application by using the following command:

      ```
      j9 -Xbootclasspath:/wsdd/lib/jclMax/classes.zip:/wsdd/lib/jclMax/database_enabler.jar -cp:
      /DB2e/db2ejdbc.jar:. -jxe:DB2eAppl.jxe
      ```

Return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95.

Running DB2eAppl.java on Symbian

This task is part of the main task of Compiling and running sample Java applications on non-Palm OS targets. After you complete these steps, return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95.

Procedure:

Some Symbian devices come with a JVM. If you want to run a text-based Java application (for example, the sample Java programs), you need to install Redirect

(supplied as `Redirect.sis` in the Symbian SDK for Java) and start the Redirect application before you start the text-based application. The text output will be captured by Redirect.

Return to "Compiling and running sample Java applications on non-Palm OS targets" on page 95.

Chapter 15. The sample Visual Basic application

This chapter provides information about the sample Visual Basic application. The topics covered are:

Overview of the sample Visual Basic application

The sample Visual Basic application shows you how to access DB2 Everyplace data using Visual Basic. You can develop applications that have the same application logic and user interface on both Pocket PC (WinCE) and Win32 operating systems. Two Visual Basic sample applications are provided with DB2 Everyplace. One is for the Pocket PC (WinCE) operating system and the other is for Win32 operating systems. The application logic and user interface for both these sample applications are the same. The file `db2evb.bas`, which contains the application logic, is common between the two operating systems. See 103 for more details.

Files included in the sample application:

The Visual Basic project directory, which contains the sample application, is located under the directory where you installed DB2 Everyplace. For Pocket PC (WinCE), you can find the files in `\db2everyplace\clients\`**`wince`**`\database\visualbasic`. For Win32 operating systems, you can find the files in `\db2everyplace\clients\`**`win32`**`\database\visualbasic`.

The sample Visual Basic application includes the following files:

`db2evb.bas`
: The `db2evb.bas` file contains the sample Visual Basic application. You can use the sample application to help you write your own Visual Basic application.

`db2ecli.bas`
: The `db2ecli.bas` file is the Visual Basic interface that connects to the DB2 Everyplace database. It also defines various DB2 Everyplace constraints that are found in `sqlcli.h`, `sqlcli1.h`, `sqlext.h`, and `sqlsystm.h`. Only the most commonly used constraints are in this file. You can add other constraints from `sqlcli.h`, `sqlcli1.h`, `sqlext.h`, and `sqlsystm.h` if you need to.

DB2eForms (extensions vary depending on the operating system)
: Application user Interface file.

DB2eSample.exe (For WinCE, DB2eSample.vb)
: Application executable file.

DB2eSample.vbw
: Application project file.

DB2eSample.vbp (For WinCE, DB2eSample.ebp)
: Application project file.

Visual Basic example: db2evb.bas:

The major steps used in the sample application (db2evb.bas) are:

Connect to the DB2 Everyplace database.

- Step 1: Allocate an environment handle.
- Step 2: Allocate a database handle.
- Step 3: Connect to the database.
- Step 4: Allocate a statement handle.

Access DB2 Everyplace data.

- Step 5: Create a table.
- Step 6: Insert data into the table.
- Step 7: Retrieve data from table.

Terminate the application application.

Note: Make sure that the application closes the connection to the DB2 Everyplace database before exiting.

Comments have been added to this example to illustrate the sample application steps.

```
Option Explicit

Public henv    As Long          ' Environment handle
Public hdbc    As Long          ' Database handle
Public hstmt   As Long          ' Statement handle
Public rc      As Integer       ' Return code

Public dbpath As String        ' filesystem path where DB2e will create tables.
Public userid As String        ' Userid: not used by DB2 Everyplace.
Public pass   As String        ' Password: not used by DB2 Everyplace

'--------------------------------------------------------------------------------
' Function: DB2eTest
'
' Description: Function illustrating how calls to DB2 Everyplace can be made.
'
'--------------------------------------------------------------------------------

Public Function DB2eTest() As Integer
    Dim errmsg As String
    Dim numCols As Integer
    Dim i As Integer
    Dim retLen As Long
    Dim data As String
    Dim crtStmt As String
    Dim insStmt1 As String
    Dim insStmt2 As String
    Dim selStmt As String
    On Error Resume Next 'Important: don't ask me why, but this line is needed
                         'in every function that calls functions from db2e.dll
                         'otherwise visual basic does strange mysterious things.
    '
    dbpath = ""
    userid = ""
    pass = ""
    '
    crtStmt = "CREATE TABLE x(a INT, b TIMESTAMP)"
    insStmt1 = "INSERT INTO x VALUES(1, CURRENT TIMESTAMP)"
    insStmt2 = "INSERT INTO x VALUES(2, CURRENT TIMESTAMP)"
    selStmt = "SELECT * FROM x"
    '
    data = String(80, " ")
    ' Step 1: allocate an environment handle.
    '

    DB2eForm.DB2eText.Text = vbCrLf & vbCrLf & " Allocating an environment handle"

    rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HENV, henv)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function
    End If
```

```
'
' Step 2: allocate database handle
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf &
                         " Allocating a database handle"

rc = SQLAllocHandle(SQL_HANDLE_DBC, henv, hdbc)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' Step 3: connect to the database
'
DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf &
                         "  Connecting to the database"

rc = SQLConnect(hdbc, dbpath, SQL_NTS, userid, SQL_NTS, pass, SQL_NTS)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' Step 4: allocate a statment handle.
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf &
                         "  Allocating a statement handle"

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, hstmt)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf

'
' Now we can use CLI function calls to execute SQL statements.
'
' Step 5: Create a table
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & crtStmt
rc = SQLExecDirect(hstmt, crtStmt, SQL_NTS)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' Create the same table again to force an error message and
' see if DB2eError works.
'
'rc = SQLExecDirect(hstmt, "create table p(a int)", SQL_NTS)
'If (rc <> 0) Then
'   testmsg = MsgBox("BLA1", 1, "DB2 Everyplace Visual Basic")
'   rc = DB2eError()
'   testmsg = MsgBox("BLA2", 1, "DB2 Everyplace Visual Basic")
'   rc = DB2eTerminate()
'   testmsg = MsgBox("BLA3", 1, "DB2 Everyplace Visual Basic")
'   Exit Function
'End If
'

'
' Step 6: Insert data into the table.
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & insStmt1
rc = SQLExecDirect(hstmt, insStmt1, SQL_NTS)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function

End If
```

```
    DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & insStmt2
    rc = SQLExecDirect(hstmt, insStmt2, SQL_NTS)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function

    End If

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf

    '
    ' Step 7: Retrieve data from table.
    '

    DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & selStmt
                           & vbCrLf

    rc = SQLExecDirect(hstmt, selStmt, SQL_NTS)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function
    End If


    rc = SQLNumResultCols(hstmt, numCols)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function
    End If

    Do While (SQLFetch(hstmt) = SQL_SUCCESS)
        For i = 1 To numCols
            rc = SQLGetData(hstmt, i, SQL_C_CHAR, data, 80, retLen)
            DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & "  " & data & vbCrLf
            If (rc <> 0) Then
                rc = DB2eError()
                rc = DB2eTerminate()
                Exit Function
            End If
        Next
        data = String(80, " ")
        DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf
    Loop

    '
    ' Step 8: Close connection to DB2e database before application terminates.
    '
    rc = DB2eTerminate()

    DB2eTest = 0
End Function
```

Related tasks:

- "Developing DB2 Everyplace Visual Basic applications" on page 29

Compiling and testing the sample Visual Basic program

Procedure:

To compile and test the DB2 Everyplace sample program:

1. Open the Visual Basic project file `DB2eSample.vbp` (For WinCE, `DB2eSample.ebp`).
2. Build the sample program.
   - For Win32: Select **File –> DB2eSample.exe.**. `DB2eSample.exe` will be built.
   - For WinCE: Select **File –> DB2eSample.vb**. `DB2eSample.vb` will be built.
3. Copy the following files:

- For Win32: Copy `DB2e.dll` (for your Win32 operating system) into your current project directory or the path of `DB2e.dll` in the environment variable PATH.
- For WinCE: Copy `DB2eSample.vb`, `DB2e.dll` (for your Pocket PC operating system), and Visual Basic Runtime into the directory of your choice.

4. Run `DB2Sample.exe` (For WinCE, `DB2Sample.vb`).

Related concepts:

- "Overview of the sample Visual Basic application" on page 103

Related reference:

- "DB2 CLI function summary" on page 182
- "Visual Basic Interface supported operating systems" on page 30

Chapter 16. The sample JSP applications

The files listed below are relative to the <DB2Everyplace>\SDK\JSP\sample\jsp directory. All of the sample JSP applications use the Visiting Nurse sample database in the <DB2Everyplace>\SDK\JSP\sample\data directory. Tutorials are in the <DB2Everyplace>\SDK\JSP\doc directory.

Applications developed using WebSphere Studio Professional/Entry Edition v4.0
Visiting Nurse Schedule

Description
This sample dynamically queries the Visiting Nurse database and displays the results in a table.

Start page
VNSchedule_ws40\scheduleHTMLResults.jsp

Tutorial
ws40.pdf

Applications developed using WebSphere Studio Application Developer v4.0
Visiting Nurse Schedule

Description
This sample dynamically queries the Visiting Nurse database and displays the results in a table. This sample application requires JDBC 2.0 and cannot be run on Symbian OS Version 6.

Start page
VNSchedule_wsad40\scheduleMasterView.jsp

Other files
VNSchedule_wsad40\web.xml
VNSchedule_wsad40\dbbeans.jar

Tutorial
wsad40.pdf

Applications developed using WebSphere Studio Application Developer v5.0
Visiting Nurse Schedule

Description
This sample dynamically queries the Visiting Nurse database and displays the results in a table.

Start page
VNSchedule_wsad50\scheduleMasterView.jsp

Other files
VNSchedule_wsad50\web.xml
VNSchedule_wsad50\dbbeans.jar
VNSchedule_wsad50\scheduleMasterViewBean.class

Tutorial
wsad50.pdf

Applications developed outside of WebSphere Studio
Visiting Nurse

Description

See the *DB2 Everyplace Installation and User's Guide* for a description of the Visiting Nurse sample application.

Start page

```
VisitingNurse\schedule.jsp
```

Other files

```
VisitingNurse\contact.jsp
  VisitingNurse\medrecord.jsp
  VisitingNurse\person.jsp
```

Related tasks:

- "Verifying JSP support on a Windows workstation" on page 33
- "Setting up for JSP development on a Windows CE device" on page 34

Related concepts:

- "Developing DB2 Everyplace JSP applications" on page 31
- "DB2 Everyplace JSP support overview" on page 32

Chapter 17. Sample synchronization applications

This chapter provides information about the sample synchronization applications. The topics covered are:

The sample Sync Client C/C++ application

The following example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application. You can find more source code examples in `\DB2e\Clients\clientapisample\C_API`.

```
/*****************************************************************/
/**
 * This function defines the sync listener. See isyncore.h for more
 * information.
 * param: listenerData, your personal data.
 * param: event, event object
 * param: pExtraInfo (reserved)
 * return: integer, when event type is ISCEVTTYPE_Retry:
 *   . ISCRTNCB_ReplyYes    : retry less than 3 times
 *   . ISCRTNCB_ReplyNo     : retry more than or equal to 3 times
 *     when event type is ISCEVTTYPE_Info:
 *   . ISCRTNCB_Done
 *     when event type is ISCEVTTYPE_Query and its event code is ISCEVT_QueLogin:
 *   . ISCRTNCB_Done        : username and password are entered correctly
 *   . ISCRTNCB_Default     : username and password are not entered
 *     others (ISCEVTTYPE_Fatal, ISCEVTTYPE_Error, ISCEVTTYPE_Query,
 *             and ISCEVTTYPE_Conflict)
 *   . ISCRTNCB_Default     : take default action
 **/
static isy_INT32 syncListener(
       isy_UINT32 listenerData,
       ISCEVT    *event,
       isy_VOID  *pExtraInfo)
{
    // appEventCodeToMessage is some user function to map an event code to
    // some descriptive event message
    char *statusMsg = appEventCodeToMessage(event);
    int   timesRetried;

    switch (event->type) {
        case ISCEVTTYPE_Fatal:
        case ISCEVTTYPE_Error:
            printf("Error: %s\n", statusMsg);
            return ISCRTNCB_Default ;

        case ISCEVTTYPE_Retry:
            timesRetried = event->retry;
            if (timesRetried >= 3)
                return ISCRTNCB_ReplyNo;
            else {
                char ans;
                printf("%s [Y/N] ", statusMsg);
```

```
                ans = getchar();
                getchar();
                if(tolower(ans) == 'y')
                        return ISCRTNCB_ReplyYes;
                else
                        return ISCRTNCB_ReplyNo;
            }

          case ISCEVTTYPE_Info:
              switch (event->code) {
                  case ISCEVT_InfSucceeded:
                  case ISCEVT_InfFailed:
                  case ISCEVT_InfCanceled:
                      printf("Conclusion: %s\n", statusMsg);
                      break;
                  case ISCEVT_InfGeneral:
                  case ISCEVT_InfCancelingSync:
                  case ISCEVT_InfPrepMsg:
                  case ISCEVT_InfSendMsg:
                  case ISCEVT_InfWaitMsg:
                  case ISCEVT_InfApplyMsg:
                      printf("Status: %s\n", statusMsg);
                      break;
                  default: // ignore other event code
                      break;
              } // switch (event->code)
              return ISCRTNCB_Done;

          case ISCEVTTYPE_Query:
              if (event->code == ISCEVT_QueLogin) {
                  ISCLISTENARG *args = event->info;
                  isy_TCHAR *target = args->argv[0];
                  // Just an example, not intended to be free of memory leaks.
                    isy_TCHAR *username = (isy_TCHAR *) calloc(18, sizeof(isy_TCHAR));
                    isy_TCHAR *password = (isy_TCHAR *) calloc(254, sizeof(isy_TCHAR));
         char c;
                    int  i;

                  printf("Query on target data(%s): %s ...\n", target, statusMsg);
                  // Ask for the username
                  printf("Username: ");
                  for(i = 0; (c = getchar()) != '\n'; i++) username[i] = c;
                  username[i] = '\0';
                  if (i == 0) return ISCRTNCB_Default; // username not entered
                  // Ask for the password
                  printf("Password: ");
                  for(i = 0; (c = getchar()) != '\n'; i++) password[i] = c;
                  password[i] = '\0';
        args->argv[1] = username;
                    args->argv[2] = password;
                 return ISCRTNCB_Done;
              }
              return ISCRTNCB_Default;

            // all other event types, don't care
            default:
              return ISCRTNCB_Default;
      } // switch (event->type)
}

// Sample SyncClient
main()
{
    isy_TCHAR user[]     = isy_T("user1");
    isy_TCHAR password[] = isy_T("password");
    HISCSERV  hServ;
    HISCCONF  hConf;
    HISCENG   hEngine;
    isy_INT32 rc;

    rc = iscConfigOpen(hServ, isy_T(".\isyncPath"), &hConf;);
    rc = iscEngineOpen(hConf, &hEngine;);
    iscEngineSetListener(hEngine, syncListener, NULL);
```

```
    iscEngineSyncConfig(hEngine); // get the configuration first
    iscConfigEnableSubsSet(hConf, NULL); // enable all subscription sets
    rc = iscEngineSync(hEngine); // sync config + subscription sets

    if (rc == ISCRTN_Failed) {
        HISCCSR   hCursor;
        isy_TCHAR  id[ISCLEN_SubsSetID];
        isy_TCHAR  name[ISCLEN_SubsSetName];
        isy_INT32  enabled;

        iscConfigOpenCursor(hConf, &hCursor;);
        while (iscConfigGetNextSubsSet(hConf, hCursor, id, name)
               == ISCRTN_Succeeded) {
            enabled = iscConfigSubsSetIsEnable(hConf, id);
            if (enabled != ISCRTN_True) continue; // forget about those which have
                                                  // been disabled
            rc = iscConfigGetSubsSetStatus(hConf, id);
            if (rc != ISCRTN_Succeeded)
// Then, the application can have some code
// processing the failing subscription sets here.
// To disable the subscription set, call:

                iscConfigDisableSubsSet(hConf, id);
        }
        iscConfigCloseCursor(hConf, hCursor);
        rc = iscEngineSync(hEngine); // sync config + subscription sets
    }
    // close all handles
    iscEngineClose(hEngine);
    iscConfigClose(hConf);
    iscServiceClose(hServ);
} // main
```

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Supported C/C++ development tools" on page 9

The sample Java native synchronization applications

There are a number of sample Java programs available to help you write Java synchronization applications for DB2 Everyplace.

See the section called "Overview of sample applications" in the *DB2 Everyplace Installation and User's Guide* for information on where the samples are located.

The sample program ISyncSample.java demonstrates how to code a Sync Client application for DB2 Everyplace native synchronization provider.

The major steps of the ISyncSample.java sample application are:

Step 1:

Import the DB2 Everyplace synchronization packages.

```
import com.ibm.mobileservices.isync.*;
import com.ibm.mobileservices.isync.event.*;
```

For JNI-based synchronization provider, import com.ibm.mobileservices.isync.db2e.jni.*;

For Trap-based synchronization provider, import com.ibm.mobileservices.isync.db2e.sti.*;

Step 2:

Implement the eventIssued method of the ISyncListener interface for event notification during synchronization.

Step 3:

Get an instance DB2eISyncProvider

Step 4:

Get an instance of synchronization service from the provider object

Step 5:

Get an instance of the configuration store from the service object

Step 6:

Get an instance of the synchronization driver from the configuration store object

Step 7:

Register your application listener object that implements the ISyncListener interface for event notification from the synchronization driver object during synchronization

Step 8:

Perform synchronization on all enabled subscription sets. Check return code and exception for status of the synchronization.

Step 9:

Close and free all resources allocated by the synchronization provider

```
// Example 1:  ISync Java - Simple API usage
//

 // Step 1:  import the Sync Client Java packages
      //
      import com.ibm.mobileservices.isync.*;
      import com.ibm.mobileservices.isync.event.*;
      import com.ibm.mobileservices.isync.db2e.jni.*;


      // Step 2:  implement the eventIssued() method in the ISyncListener
      //    interface if you are interested in event notification (optional)
      //
      public class ISyncSample implements ISyncListener {

         public ISyncSample () {}

         public int eventIssued(ISyncEvent evt) {

            int evtType = evt.getEventType();

            switch(evtType) {

            // display event status
            case ISync.EVTTYPE_INFO:
            case ISync.EVTTYPE_ERROR:

               System.out.println ("**********************");
               System.out.println ("SubsSet:    " + evt.getSubscriptionSetName() );
               System.out.println ("Subs:       " + evt.getSubscriptionName() );
               System.out.println ("SubsType:   " + evt.getSubscriptionType() );
               System.out.println ("Event Type: " + evtType );
               System.out.println ("Event Code: " + evt.getEventCode() );
               System.out.println ("Progress:   " + evt.getSyncProgress());
               System.out.println ("**********************\n");

               return ISync.RTNCB_DONE;

            case ISync.EVTTYPE_RETRY:
               return ISync.RTNCB_REPLY_YES;

            case ISync.EVTTYPE_CONFLICT:
               return ISync.RTNCB_DONE;

            // ignore other event types
            default:
               break;
```

```
         }

         // let sync engine take default action
         return ISync.RTNCB_DEFAULT ;

      }

      public void runSample(String host, String port,
                            String userID, String passwrd) {

         ISyncProvider provider = null;
         ISyncService  service  = null;
         ISyncConfigStore config = null;
         ISyncDriver syncer = null;
         String      path = "data";         // a data directory under current dir
         ISyncSubscriptionSet ssArr[] = null;
         int rc = 0;

         try {

            // Step 3:  get an instance DB2eISyncProvider
            //
            provider = DB2eISyncProvider.getInstance();

            // Step 4:  get an instance of synchronization service from the provider
            //
/*
 For the DB2j sync client, the JDBC driver and url are required

 String driver = "com.ibm.db2j.jdbc.DB2jDriver";
 String jdbcUrl = jdbc:db2j:crtlDb;create=true;
*/

if (driver != null)
    userProps.put("target.db.driver", driver);
if (jdbcUrl != null)
    userProps.put("target.db.url", jdbcUrl);

            Properties userProps = new Properties();

    userProps.put("isync.user", user);
    userProps.put("isync.password", password);
    userProps.put("isync.trace", "detailed");

    service = provider.createSyncService(uri, userProps);

            // Step 5:  get an instance of the configuration store
            //
            config = service.getConfigStore(path);

            // Step 6:  get an instance of the sync driver to perform
            //                                  synchronization
           syncer = config.getSyncDriver();

            // Step 7:  set the listener object for event notification from the
                   syncer object
            //          during synchronization (optional)
            syncer.setSyncListener(this);

            // Step 8:  perform synchronization on all enabled subscription sets
            //
            rc = syncer.sync();

            switch (rc) {
               case ISync.RTN_SUCCEEDED:
                  System.out.println("Synchronization succeeded");
                  break;

               case ISync.RTN_CANCELED:
                  System.out.println ("Synchronization canceled");
                  break;

               default:
                  System.out.println ("Synchronization failed");
                  break;
            }

            ssArr = config.getSubscriptionSets();

            for (int i=0; i < ssArr.length; i++ ) {

               System.out.print ("Subscription Set: " +
                                 ssArr[i].getName() + " Status: ");
```

```
            switch(ssArr[i].getStatus()) {

               case ISync.STATUS_READY:
                  System.out.println("READY");
                  break;

               case ISync.STATUS_COMPLETED:
                  System.out.println ("COMPLETED");
                  break;

               case ISync.STATUS_CANCELED:
                  System.out.println ("CANCELED");
                  break;

               default:
                  System.out.println ("FAILED");
                  break;
            }
         }
      }
      catch (ISyncException ie) {
         System.out.println("Exception code: " + ie.getCode());
         ie.printStackTrace();
      }
      catch (Exception e) {
         e.printStackTrace();
      }
      finally {
         // Step 9:  close and free all allocated resources
         //

         try {

            if (syncer != null) {
               syncer.close();
               syncer = null;
            }

            if (config != null) {
               config.close();
               config = null;
            }

            if (service != null) {
               service.close();
               service = null;
            }
         }
         catch(ISyncException ie2) {

            System.out.println("Exception code: " + ie2.getCode());
            ie2.printStackTrace();
         }
      }
   }  // end runSample()


   public static void main(String args[]) {

      String host     = "localhost";
      String port     = "8080";
      String userID   = "nurse1";
      String passwrd  = "nurse1";

      ISyncSample isa = new ISyncSample();

      if (args.length > 0) {
      if (args.length == 4)
      {
        host     = args[0];
        port     = args[1];
        userID   = args[2];
        passwrd  = args[3];
      }
else
        System.out.println("Usage: java ISyncSample [host] [port] " +
                           "[userid] [password]");
   }
      isa.runSample(host, port, userID, passwrd);

   }  // end main()

 }  // end ISyncSample class
```

Related tasks:

- "Installing and verifying the trap-based native synchronization provider" on page 23

Related concepts:

- "The sample Java MIDP synchronization applications"

Related reference:

- "IBM Java Sync APIs" on page 19

The sample Java MIDP synchronization applications

There are a number of sample Java applications available to help you write Java synchronization applications for DB2 Everyplace.For the MIDP synchronization provider, the samples are located in:

```
%DSYINSTDIR%/Clients/Midp/samples
```

The primary sample is the Visiting Nurse application under `com/ibm/mobileservices/demo`, `VNurse.java` and `NursesAid.jar`. Under the same samples directory are two files which make up a simple application. This application does not provide Record Store Management (RMS) code or a solid user interface. The files are:

- ISyncSample.java : driving MIDlet
- ISyncWorker.java : worker responsible with synchronizing data

Details on the ISyncWorker.java file:

The sample program `SyncWorker.java` demonstrates how to code a Sync Client application for DB2 Everyplace MIDP synchronization provider.

The Java sample application performs the following steps:

1. Import the DB2 Everyplace synchronization packages.

   ```
   import com.ibm.mobileservices.isync.*;
   import com.ibm.mobileservices.isync.event.*;
   import com.ibm.mobileservices.isync.midp.*;
   ```

2. Implement the eventIssued method of the ISyncListener interface for event notification during synchronization.
3. Get an instance MIDPISyncProvider
4. Get an instance of synchronization service from the provider object
5. Get an instance of the configuration store from the service object
6. Get an instance of the synchronization driver from the configuration store object
7. Register your application listener object that implements the ISyncListener interface for event notification from the synchronization driver object during synchronization
8. Perform synchronization on all enabled subscription sets. Check the return code and exception for the status of the synchronization.
9. Close and free all resources allocated by the synchronization provider.

The ISyncSample.java example:

The following example contains comments that refer to the steps in the previous section.

```
// Example 1:  ISync Java - Simple API usage
//

 // Step 1:  import the Sync Client Java packages
      //
      import com.ibm.mobileservices.isync.*;
      import com.ibm.mobileservices.isync.event.*;
      import com.ibm.mobileservices.isync.midp.*;



/**
 Supporting class which handles all of the synchronization tasks.
 Called by ISyncSample.
*/

public class SyncWorker extends Thread implements ISyncListener
{
 private ISyncSample midlet;
 private boolean mCancel;
 private ISyncProvider provider;
 private ISyncService service;
 private ISyncConfigStore config;
 private ISyncDriver syncer;
 private String eventString;

 public SyncWorker(ISyncSample midlet)
 {
  this.midlet = midlet;
  mCancel = false;
 }

   // Step 2:  implement the eventIssued() method in the ISyncListener interface
   //          if you are interested in event notification (optional)
   //

 public int eventIssued(ISyncEvent evt)
 {
  int evtType = evt.getEventType();
  int evtCode = evt.getEventCode();
  int evtProg = evt.getSyncProgress();
         String ssName = evt.getSubscriptionSetName();
  Object listenerInfo = evt.getEventInfo();

  Exception e = null;
  ConflictReader cr = null;

  if (listenerInfo instanceof Exception)
   e = (Exception) listenerInfo;
  else if (listenerInfo instanceof ConflictReader)
   cr = (ConflictReader) listenerInfo;

  eventString += evtCode + ":";

  switch(evtType)
  {
   // display event status
    case ISync.EVTTYPE_INFO:

              switch (evtCode)
      {
                case ISync.EVT_INF_SYNCING_SUBS:
                    midlet.updateSyncStat1("Synchronizing " + ssName);
                    midlet.updateSyncStat2(" ");
                    break;
                case ISync.EVT_INF_SYNC_STARTED:
                    midlet.updateSyncStat1("Synchronization started");
                    midlet.updateSyncStat2(" ");
                    break;
                case ISync.EVT_INF_PREP_MSG:
                    midlet.updateSyncStat2("Preparing message...");
                    break;
                case ISync.EVT_INF_SEND_MSG:
                    midlet.updateSyncStat2("Sending message...");
                    break;
                case ISync.EVT_INF_WAIT_MSG:
                   midlet.updateSyncStat2("Awaiting server reply...");
                   break;
                case ISync.EVT_INF_APPLY_MSG:
                    midlet.updateSyncStat2("Applying server message...");
```

```
                break;
            case ISync.EVT_INF_SYNC_CANCELED:
        midlet.updateSyncStat1("Synchronization canceled");
        midlet.updateSyncStat2(" ");
        break;
         case ISync.EVT_INF_SYNC_SUCCEEDED:
        midlet.updateSyncStat1("Synchronization succeeded");
        midlet.updateSyncStat2(" ");
        break;
            case ISync.EVT_INF_SYNC_FAILED:
        midlet.updateSyncStat1("Synchronization failed");
        midlet.updateSyncStat2(" ");
        break;
            default:
                break;
          }

  return ISync.RTNCB_DONE;

  case ISync.EVTTYPE_ERROR:
   midlet.updateSyncStat2("Error: " + evtCode);
   return ISync.RTNCB_DONE;

  case ISync.EVTTYPE_RETRY:
   midlet.updateSyncStat2("Retry: " + evtCode);
   return ISync.RTNCB_REPLY_YES;

  case ISync.EVTTYPE_CONFLICT:
  if (evtCode == ISync.EVT_CFT_REJECT)
  {
   String tabName = evt.getSubscriptionName();
                    midlet.updateSyncStat2("Conflict: " + tabName);

   /*
    Application needs to do the right thing with conflictRow.
   */

   // System.out.println("Conflict table " + tabName
    // + " row: " + conflictRow);
  }
  return ISync.RTNCB_DONE;

  // ignore other event types
  default:
   break;
  }

  // let sync engine take default action
 return ISync.RTNCB_DEFAULT ;

}  // end of eventIssued()
/*
 Synchronization is implemented in a thread to allow the
 user to cancel the request which single-threaded, might
 be hung on in IO request
*/

public void run()
{
 sync();
}

public void cancel()
{
 try
 {
  if (syncer != null)
   syncer.cancelSync();
 }
 catch (ISyncException iex)
 {}

 mCancel = true;
}

private void sync()
{
 try
 {
  eventString = " ";

  String user = "nurse1";
  String password = "nurse1";
  String host = "localhost";
```

```
String port = "9080";

/*
 If jad file has values, use them, see
 DeployManifest.java in tools.

 In the Sun WirelessToolkit, under Settings, you can enter
 values in the User Defined tab.
*/

String x = midlet.getAppProperty("Db2eSyncUserName");
 if (x != null)
 user = x;
x = midlet.getAppProperty("Db2eSyncPassword");
if (x != null)
 password = x;
x = midlet.getAppProperty("Db2eSyncHost");
if (x != null)
 host = x;
x = midlet.getAppProperty("Db2eSyncPort");
if (x != null)
 port = x;
midlet.appendForm(host + ":" + port + " " + user + "/" +
        password);


      // Step 3:  get an instance MIDPISyncProvider
      //
provider = MIDPISyncProvider.getInstance();

      // Step 4:  get an instance of synchronization service from the provider
      //
 Hashtable ht = new Hashtable();
 ht.put("isync.user", userName);
 ht.put("isync.password", password);
 ht.put("isync.trace", "detailed");

 service = provider.createSyncService(URI, ht);

      // Step 5:  get an instance of the configuration store
      //
config = service.getConfigStore(null);

      // Step 6:  get an instance of the sync driver to perform synchronization
      //
syncer = config.getSyncDriver();

      // Step 7:  set the listener object for event notification
      //     from the syncer object during synchronization
      //
syncer.setSyncListener(this);

// Step 8:  perform synchronization on all enabled subscription sets
         //

int rc = syncer.sync();

switch (rc)
{
  case ISync.RTN_SUCCEEDED:
 midlet.reportSyncStatus("Synchronization succeeded "
  + eventString);
 break;

  case ISync.RTN_CANCELED:
 midlet.reportSyncStatus("Synchronization canceled "
   + eventString);
 break;

  default:
 midlet.reportSyncStatus("Synchronization failed "
   + eventString);
 break;
}
//  Step 9: Close all resources
//

 close();
}
catch (ISyncException iex)
{
 midlet.reportSyncStatus("Exception Code: "
  + iex.getCode() + ", Event codes: " + eventString);
}
```

```
  catch (Exception e)
  {
   midlet.reportSyncStatus(e.toString());
  }
  finally
  {
   mCancel = false;
  }
 }

 private void close() throws ISyncException
 {
  if (syncer != null)
  {
   syncer.close();
   syncer = null;
  }
  if (config != null)
  {
   config.close();
   config = null;
  }

  if (service != null)
  {
   service.close();
   service = null;
  }

  provider = null;
 }
}
```

Related tasks:

- "Developing the isync4j for MIDP application with the Sun Wireless Toolkit"
- "Developing the isync4j for MIDP application with ANT and the Sun Wireless Toolkit Command Line" on page 122

Related concepts:

- "The sample Java MIDP synchronization applications" on page 117

Developing the isync4j for MIDP application with the Sun Wireless Toolkit

This topic describes how to develop the DB2 Everyplace ISYNC4J for MIDP within the Sun Wireless Toolkit application. The examples used in this section are based on the VNurse sample application.

Prerequisites:

See "Overview of DB2 Everyplace synchronization providers" on page 19 for detailed information on hardware and software prerequisites for using the DB2 Everyplace J2ME MIDP synchronization provider.

Procedure:

1. Launch the Wireless Toolkit. From a command line prompt, change to the `bin` directory where the Sun Wireless Toolkit is installed. Type **ktoolbar.bat**.

   Note: It is recomended to use a command prompt instead of the Windows **Start** menu.
2. Create a new project for the `isync4j` sample application:
   a. Open the J2ME Wireless Toolkit
   b. Select **New Project**.
   c. Type the project name (for example, VNurse)

d. Type a MIDlet class name (for example, com.ibm.mobileservices.demo.VNurse)

e. Click **Create Project**.

f. Copy the `ISyncMidp.jar` file to the J2ME project library. For example:

```
c:\>copy %DSYINSTDIR%\Clients\Midp\lib\ISyncMidp.jar \
  j2me_install_dir\apps\VNurse\lib.
```

g. Optional: If you want to view the trace output while the MIDlet is running, copy `ISyncMidpDebug.jar` to the *j2me_install_dir* `\apps \VNurse \lib`.

Note: Do not use trace when building a JAR file that will be installed on the phone.The resultant JAR file will be too large to install.

h. Optional: To use obfuscation (to reduce the code size), copy the `retroguard.jar file` to the bin directory where J2ME is installed.

i. Click **Settings** . The Settings for project window opens. Click the **User Defined** tab and click **Add** to type the following **Key** and **Value** entries:

- Db2eSyncPassword, nurse1 [default]
- Db2eSyncUserName, nurse1 [default]
- PacketDownSize, 2800 [default 30000]
- PacketUpSize, 1400 [default 30000]
- Db2eSyncHost, localhost [default]
- Db2eSyncPort, 9080 [default]

These values are placed in the .jad file by the Sun Wireless Toolkit and the MIDlet reads their values at run time.

j. The DB2 Everyplace sample application (VNurse) will display a PNG image file. Click the **MIDlets** tab and select **MIDlet-1**. Click **Edit** and change `VNurse.png` to `ibm.png`. You will need to copy `ibm.png` from the `Midp\samples\images` directory to the *J2ME_ install_ dir*`\apps\VNurse\res` directory.

3. Import the DB2 Everyplace sample Java files into the project. For example, copy the directory structure from `%DSYINSTDIR%\Clients \Midp \samples \com` to the *J2ME_ install_ dir* `\apps \VNurse \src` directory.
4. Build and run the VNurse sample application. From the Sun Wireless Toolkit window, click **Build** and click **Run**.

Related tasks:

- "Developing the isync4j for MIDP application with ANT and the Sun Wireless Toolkit Command Line"

Related concepts:

- "Overview of DB2 Everyplace synchronization providers" on page 19

Developing the isync4j for MIDP application with ANT and the Sun Wireless Toolkit Command Line

This topic describes how to develop the DB2 Everyplace ISYNC4J for MIDP with ANT and the Sun Wireless Toolkit Command Line.

Prerequisites:

Download and install the following software to work with the examples provided:

- Sun Microsystems JavaTM 2 Platform Micro Edition, Wireless Toolkit

- Apache ANT
- RetroGuard Ofuscator

Procedure:

1. Optional: Recompile the demos if you want to modify them.

   The `lib` directory contains precompiled JAD and JAR files. The `build.bat` and `build.xml` scripts are provided to illustrate the use of Apache ANT, the DeployManifest tool, and the RetroGuard obfuscator.

   a. Add `retroInstallDir\lib\retroguard.jar` to your CLASSPATH variable. Set the following variables in your environment:
      - ANT_HOME – to the root of your ANT installation
      - DB2m_HOME – to the %DSYINSTDIR%\Clients\Midp directory
      - J2MEWTK_HOME – to the root of your Sun Wireless Toolkit installation
      - JAVA_HOME – to the root of your jdk13 or jdk131 (only) installation
      - JAVA14_HOME – set to the root of your jdk14 directory.

   b. Execute the `build.bat` file in the root of the MIDP clients directory to re-populate the `lib` directory under MIDP with new JAR and JAD files. There is one JAR file and several JAD files for each user and device id configuration.

      You will find several new build*classes directories, which are used for pre-verification and obfuscation. There is one JAR file and several JAD files for each user and device ID configuration. View the JAD files to see how the user ID, password, and device ID are set and passed into the MIDLet application.

   c. The DeployManifest class is included in the lib\FilterServlet*.jar and is called from the `build.xml` file. Use this class to generate both the JAR Manifest file and the JAD file. Use the following syntax when generating the files.

      To generate the Manifest file:

      ```
      java DeployManifest -m <midletName> <className> <imageFileName> <outputfileName>
      ```

      To generate the JAD file:

      ```
      java DeployManifest -j <midletJarName> -U <uploadMaxPacket> -D \
        <downloadMaxPacket> -n <numClients> <JadBaseName> <outputFileName>
      ```

   d. The DeployManifest class is called internally from `build.xml` by Apache ANT. Edit the setJad entries in the `build.xml` file to permanently change the user ID, password or other attributes. The default is *nurse1* and *nurse1*.

2. Run the insync4j application.

   The DB2 Everyplace installation creates a Vnurse database with subscription sets, users, and groups.

   a. Select **Start –> DB2 Everyplace –> Start MDAC** and verify that a user named *nurse1* exists. The password of this user is set to *nurse1*. You can use this user name, or you can edit the `lib\<midlet>.jad` file that you are passing to the run script. Note that each time you compile your changes, the JAD files are overwritten. See `samples\DeployManifest.java` to change the user and password permanently.

   b. You must start the Sync Server using either Tomcat or Websphere Version 4.0 or later. HTTP Connections from MIDP phones use HTTP Transfer-encoding, which requires a servlet engine that supports the HTTP Servlet 2.3 specification and HTTP 1.1.

   c. Execute the BATCH file in the `Midp\bin` directory passing it the name of a JAD file from the `Midp\lib` dir:

- To run the non-debug version of the demo, type:
  ```
  run VNurse
  ```
- To run the debug version using ″nurse3″ as the user ID and password for device # 213, type:
  ```
  run VNurseDebug3
  ```

The J2ME MIDP Sync Client uses the interfaces and classes that are defined in the com.ibm.mobileservices.isync.midp as well as those in the com.ibm.mobileservices.isync and com.ibm.mobileservices.isync.event packages.

Related tasks:

- "Developing the isync4j for MIDP application with the Sun Wireless Toolkit" on page 121

Related reference:

- "IBM Java Sync APIs" on page 19
- "Java Sync API supported operating systems" on page 19

Compiling and running the GoISyncConsole sample Java synchronization application

GoISyncConsole is a Java sample application to demonstrate the use of the DB2 Everyplace Sync Client Java API.

File contents for GoISyncConsole:

- `GoISyncConsole.java`
- `GoISyncConstants.java`
- `GoISyncListener.java`
- `isyncdb2.properties`
- `isyncdb2e.properties`
- `isyncdb2j.properties`

Prerequisites:

- Installation and configuration of the DB2 Everyplace Sync Server.
- Installation of the Sync Client binaries on the device. These are located in the `Clients\`*`platform`*`\sync` directories.
- If you are using a Cloudscape client, installation of Cloudscape on the device.

Procedure:

1. Compile the GoISyncConsole application:

   This requires the `isync4j.jar` file, which is one of the Sync Client binaries.

   a. Open a command prompt.
   b. Type the following command:
      ```
      javac -classpath isync4j.jar *.java
      ```
2. Set up the environment:

   The path environment must be setup so that the Sync Client binaries can be located.

   - For Win32: Set your PATH variable to include the folder where the Sync Client binaries are located.

- For Linux or Neutrino: Export the LD_LIBRARY_PATH to include the folder where the Sync Client binaries are located.

3. Run the sample:

   GoISyncConsole can be used with either the C client or with the Java DB2j client. A property file is used to determine which client to use. Sample properties files for both DB2e and DB2j are provided.

   - To use the C client, pass in the provided `isyncdb2e.properties` file by typing the following command:

     ```
     java -classpath isync4j.jar;. GoISyncConsole isyncdb2e.properties
     ```

   - To use the Java DB2j client, include the DB2j Sync Client jar, the Cloudscape jar file, and pass in the `isyncdb2j.properties` file by typing the following command (modify the text in italics if your Cloudscape installation directory is different):

```
java -classpath c:\cloudscape_5.1\lib\db2j.jar;
db2jisync.jar GoISyncConsole isyncdb2j.properties
```

   The application starts with a text menu containing the following options:

   - (1) Perform Synchronization
   - (2) Enable, Disable or Reset Subscription Sets
   - (3) Change Server Settings
   - (4) View The Log
   - (5) About Sync Client
   - (6) Exit

4. Specify option (3) to configure the server settings. This option will allow you to specify the Sync Server's IP address, your sync user name and password, and other options.
5. Specify option (1) to perform the synchronization.

   The GoISyncConsole application creates another property file called `ISync.properties` to save your preferences. If you change a property in `isyncdb2e.properties` or `isyncdb2j.properties`, you should delete `ISync.properties` before you rerun GoISyncConsole to ensure that the new changes go into effect.

Related concepts:

- "Overview of DB2 Everyplace synchronization providers" on page 19

Part 4. Reference

Chapter 18. Application programming interfaces (APIs)

DB2 Everyplace SQL statement support

This chapter contains the supported syntax diagrams, semantic descriptions, rules, and examples of the use of the SQL statements supported by DB2 Everyplace. The topics covered are:

Overview of DB2 Everyplace SQL statement support

Supported executable SQL statements can be issued interactively from the mobile device by using the command line processor (CLP), or they can be used in application programs to access data in a DB2 Everyplace database. Table 15 lists the SQL statements supported by DB2 Everyplace.

Table 15. Supported SQL statements

SQL statement	Function
CALL	Calls a remote stored procedure using the DB2 Everyplace Sync Server Remote Query and Stored Procedure Adapter (AgentAdapter)
CREATE INDEX	Creates an index.
CREATE TABLE	Defines a table.
DELETE	Deletes one or more rows from a table.
DROP	Deletes a table or index from a database.
EXPLAIN	Obtains information about access path selection for a SELECT statement.
GRANT	Grants encryption privileges to a user.
INSERT	Inserts one or more rows into a table.
REORG TABLE	Removes or reduces the wasted storage associated with the specified table.
REVOKE	Revokes a user's encryption privileges.
SELECT	Specifies a result table queried from one or more tables.
UPDATE	Updates the values of one or more columns in one or more rows of a table.

"SQLState messages reported by SQL" on page 171 lists all of the SQLSTATEs reported by the DB2 Everyplace SQL engine.

The length of an SQL statement cannot exceed 64,000 characters.

The catalog includes the following DB2 Everyplace system tables that are managed by DB2 Everyplace: DB2eSYSTABLES, DB2eSYSRELS, and DB2eSYSCOLUMNS.

Related reference:

- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

CALL

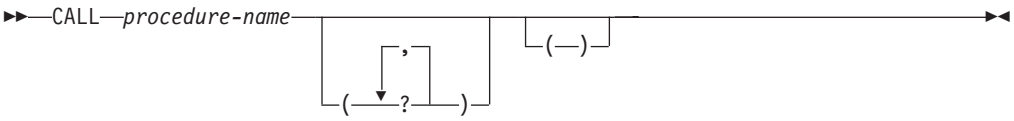
Invokes a stored procedure defined with the Remote Query and Stored Procedure Adapter for the DB2 Everyplace Sync Server. A stored procedure, for example, executes at the location of the remote database, and returns data to the DB2 Everyplace client application.

Programs using the SQL CALL statement are designed to run in two parts, one on the client and the other on the server.

Invocation:

Remote stored procedures are invoked from a DB2 Everyplace application by passing the following CALL statement syntax to `SQLPrepare()` followed by `SQLExecute()`.

Syntax:



Description:

procedure-name
Identifies the procedure to call at the remote server. The procedure identified must be defined in the AgentAdapter subscription at the current Sync Server.

? The *?* in the CALL statement syntax diagram denotes a parameter marker corresponding to an argument for a stored procedure. All arguments must be passed using parameter markers.

Rules:

none

Notes:

The CALL statement uses the remote query and stored procedure adapter included with DB2 Everyplace Sync Server. DB2 Everyplace Sync Server is required to use the CALL statement in DB2 Everyplace applications. DB2 Everyplace does not support local stored procedures.

For additional information, see the data sources section of the *DB2 Everyplace Sync Server Administration Guide*.

Example:

A compete example of how to use the CALL statement and remote query and stored procedure adapter is available in the *DB2 Everyplace Sync Server Administration Guide*. The following sample shows only the coding of the CALL statement in a sample application.

A stored procedure `MYPROC()` is defined at the source server for database `mysample`. An AgentAdapter subscription is defined at the DB2 Everyplace Sync Server with the following attributes:

```
User ID:  db2admin
Password: db2admin
Other:    dbname=mysample;procname= db2e.MYPROC
```

Sample program using the CALL statement:

```
int main(int argc, char * argv[])
{
  SQLHENV  henv;
  SQLHDBC  hdbc;
  SQLHSTMT hstmt;
  SQLRETURN rc;
  SQLCHAR  strSQL[] = "CALL db2e.MYPROC(?,?,?,?,?)";
  int   nInd4, nInd5;
  int   nSaving = 0, nChecking =0 ;
  int   nCmd =0, nAmount=0;
  SQLCHAR  strConnect[254];

 //*********************************************************************
 //* Check input parameters
 //*********************************************************************
 if ( argc < 4 ){
   printf("\nUsage : myClient AccountName Cmd Amount");
   printf("\n     cmd 1 : query balance");
   printf("\n     cmd 2 : Transfer from Saving to Checking");
   printf("\n     cmd 3 : Trnasfer from Checking to Saving");
   return (99);
 }
 nCmd = atoi(argv[2]);
 nAmount = atoi(argv[3]);

 //*********************************************************************
 //* Allocate handles
 //*********************************************************************
 rc = SQLAllocHandle( SQL_HANDLE_ENV,
      SQL_NULL_HANDLE,
      &henv; //checkerror
 rc = SQLAllocHandle( SQL_HANDLE_DBC,
      henv,
      &hdbc); //checkerror
 if (argc == 5){
 strcpy(strConnect,"http://");
 strcat(strConnect,argv[4]);
   strcat(strConnect,"/servlet/com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample");
 }else{
  strcpy(strConnect,
             "http://127.0.0.1:8080/db2e/servlet/

              com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample");
 }

 //*********************************************************************
 //* Connect to remote database
 //*********************************************************************
 rc = SQLConnect(hdbc,
  strConnect,
  SQL_NTS,
  "userex", SQL_NTS,
  "userex", SQL_NTS  ); //checkerror
 rc = SQLAllocHandle( SQL_HANDLE_STMT,
      hdbc,
      &hstmt); //checkerror
 //*********************************************************************
 //* Prepare, Bind , and Execute the statement
 //*********************************************************************
 rc = SQLPrepare(hstmt,strSQL, SQL_NTS); //checkerror
 rc = SQLBindParameter(hstmt,
   1,
   SQL_PARAM_INPUT,
   SQL_C_CHAR,
```

```
  SQL_CHAR,
  0,
  0,
  (SQLPOINTER)argv[1],
  0,
  NULL ); //checkerror
rc = SQLBindParameter(hstmt,
  2,
  SQL_PARAM_INPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nCmd,
  sizeof(int),
  NULL); //checkerror
rc = SQLBindParameter(hstmt,
  3,
  SQL_PARAM_INPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nAmount,
  sizeof(int),
  NULL ); //checkerror
rc = SQLBindParameter(hstmt,
  4,
  SQL_PARAM_OUTPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nSaving,
  sizeof(int),
  &nInd4 ); //checkerror
rc = SQLBindParameter(hstmt,
  5,
  SQL_PARAM_OUTPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nChecking,
  sizeof(int),
  &nInd5 ); //checkerror
rc = SQLExecute(hstmt); //checkerror
//****************************************************************
//* Print the balance
//****************************************************************
printf("\nSaving = %d",nSaving);
printf("\nChecking = %d",nChecking);

SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return 0;
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

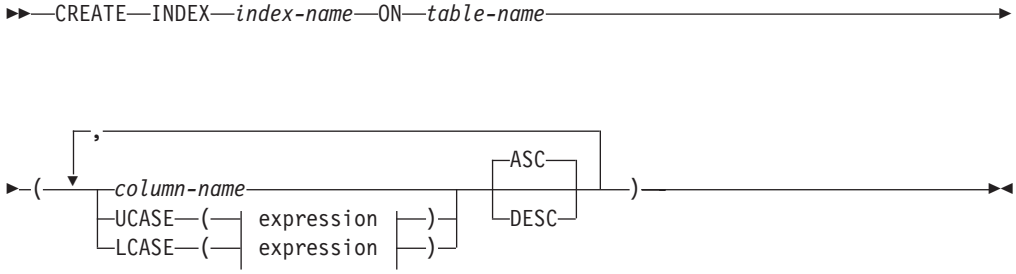
CREATE INDEX

The CREATE INDEX statement is used to create an index on a DB2 Everyplace table.

Invocation:

This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:



Description:

INDEX *index-name*
Names the index.

ON *table-name*
The *table-name* names a table on which an index is to be created.

column-name
For an index, column name identifies a column that is to be part of the index key.

Each column name must be an unqualified name that identifies a column of the table. Use eight columns or fewer; the column names cannot be repeated (SQLSTATE 42711).

The length of each specified column must not be greater than 1024 bytes.

ASC Puts the index entries in ascending order by the column. This is the default.

DESC Puts the index entries in descending order by the column.

LCASE / UCASE

The LCASE or LOWER function returns a string in which all the SBCS characters have been converted to lowercase characters. That is, the characters A-Z will be translated to the characters a-z, and characters with diacritical marks will be translated to their lowercase equivalents if they exist.

The argument must be an expression whose value is a CHAR or VARCHAR data type.

The result of the function has the same data type and length attribute as the argument. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Ensure that the characters in the value of column JOB in the EMPLOYEE table are returned in lowercase characters. For example:

```
SELECT LCASE(JOB)
     FROM EMPLOYEE
     WHERE EMPNO = '000020';
```

Rules:

- A maximum of 15 indexes can be created on a table without a primary key. A maximum of 14 indexes can be created on a table with a primary key.
- The CREATE INDEX statement will fail if attempting to create an index that matches an existing index. Two index descriptions are considered duplicates if:
  - The set of columns and their order in the index is the same as that of an existing index.

  - The ordering attributes are the same.
- Columns with a BLOB data type cannot be used in a CREATE INDEX statement.

Notes:

- The CREATE INDEX statement can contain a maximum of 8 columns.
- DB2 Everyplace supports bi-directional scanning of indexes. The following two indexes serve the same purpose although they have different definitions.

```
CREATE INDEX IDX1 ON EMPLOYEE (JOB ASC)
CREATE INDEX IDX1 ON EMPLOYEE (JOB DESC)
```

  In general, indexes should be created without specifying the order direction. Fewer indexes typically incurs lower index maintenance cost.
- DB2 Everyplace supports prefix-scanning of indexes. Consider the following example. The following index is created.

```
CREATE INDEX J1 ON T (A, B, C, D, E, F, G, K)
```

  There is no need to create another index on `T (A,B,C,D)`.
- If the table does not contain data, CREATE INDEX creates a description of the index; the index entries are created when data is inserted into the table.
- To create an index for the dirty bit index, use the following example:

```
CREATE INDEX <index name>
  ON <table name>
  ($dirty)
```

  See 259 for more information about the dirty bit.

Example:

Create an index named JOB_BY_DPT on the EMPLOYEE table. Arrange the index entries in ascending order by job title (JOB) within each department (WORKDEPT).

```
CREATE INDEX JOB_BY_DPT
ON EMPLOYEE (WORKDEPT, JOB)
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

CREATE TABLE

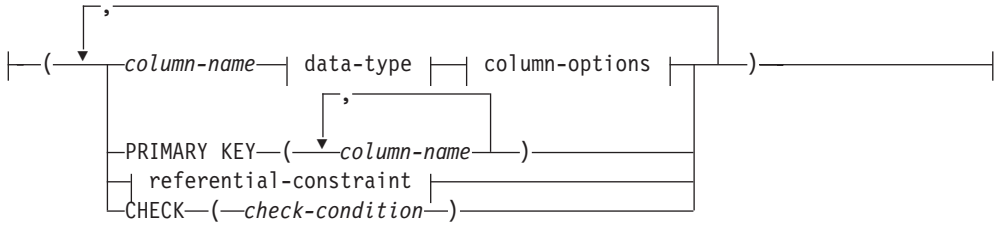
The CREATE TABLE statement defines a table. The definition must include its name and the names and attributes of its columns. The definition can also include other attributes of the table, such as its primary key.

Invocation:

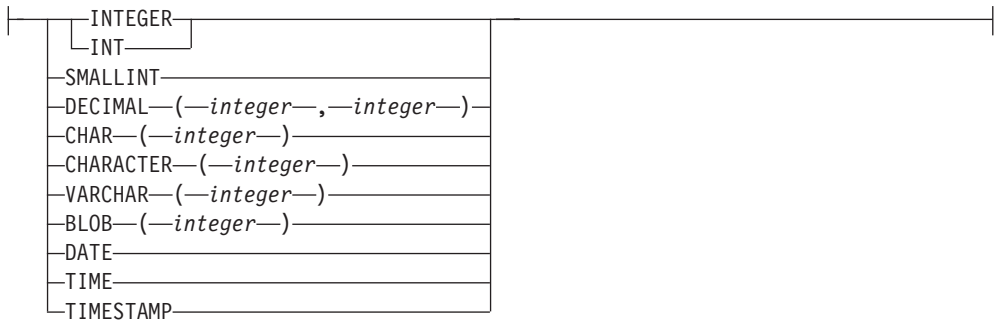
This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:

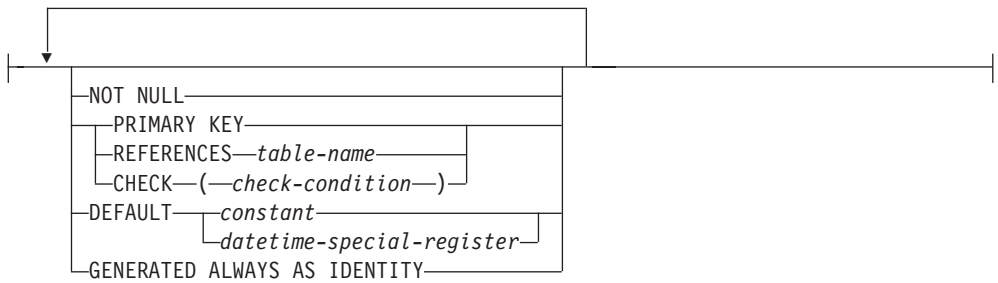
element-list:



data-type:



column-options:



referential-constraint:

Description:

table-name
Names the table. The name can be up to 18 bytes long. The name must not identify a table in the catalog. The name must be unique for the mobile device.

Table names are converted to uppercase before being stored in the catalog. You can use delimited identifiers (with double quotation marks) to prevent such conversion. You must use delimited identifiers when a table name contains blanks or special characters.

The table name can include Double Byte Character Set characters.

Restriction: The system-created data files that correspond to tables created and named by user names do not distinguish between upper and lowercase characters. For example, the data file for a table named TB is named DSY_TB. The data file for a table named ″tb″ is also DSY_TB. Therefore, to ensure data integrity, it is strongly recommended that you do not name a table using a series of characters identical, except for character case, to an existing table name.

WITH ENCRYPTION
Creates an encrypted user table. To encrypt a table, you must be authenticated and connected. You must be explicitly granted encryption. (For more information, see “GRANT” on page 147.)

A user table can be encrypted only at the time it is created. After a table is created, encryption cannot be added or removed except by deleting the table.

column-name
Names a column of the table. The name can be up to 18 bytes long. The name cannot be qualified and the same name cannot be used for more than one column of the table.

Column names are converted to uppercase before being stored in the catalog. You can use delimited identifiers (with double quotation marks) to prevent such conversion. You must also use delimited identifiers when a column name contains blanks or special characters.

The column name can include DBCS characters.

data-type
Is one of the types in the following list. Use:

INTEGER or **INT**
For a four-byte signed integer in the range of 2147483647 to -2147483648.

SMALLINT
For a two-byte signed integer in the range of -32768 to 32767.

DECIMAL*(precision-integer, scale-integer)*
For a decimal number. The first integer is the precision of the number; that is, the total number of digits; it might range from 1 to 31. The second integer is the scale of the number; that is, the number of digits to the right of the decimal point; it might range from 0 to the precision of the number.

CHAR*(integer)*
For a fixed-length character string of length *integer*, which might range from 1 to 32767.

CHARACTER*(integer)*
For a fixed-length character string of length *integer*, which might range from 1 to 32767.

VARCHAR*(integer)*
For a varying-length character string of maximum length *integer*, which might range from 1 to 32767.

BLOB*(integer)*
For a binary large object string of the specified maximum length in bytes.

The length might be in the range of 1 byte to 32767 bytes.

integer is the maximum length.

DATE
For a date. An input value can be in one of the following formats: MM/DD/YYYY, YYYY-MM-DD, or DD.MM.YYYY. The date value is printed out in only the ISO format, YYYY-MM-DD.

The special register CURRENT DATE also produces the current date in ISO format.

TIME
For a time. An input value can be in one of the following formats: HH:MM AM (or PM), HH:MM:SS, HH.MM AM (or PM), or HH.MM.SS. The SS, seconds, is optional with HH:MM:SS or HH.MM.SS formats. A time value is printed out only in ISO format, HH:MM:SS.

The special register CURRENT TIME also produces the current time in ISO format.

TIMESTAMP
For a timestamp. An input value must be in the following format: YYYY-MM-DD-HH.MM.SS.ZZZZZZ. A timestamp value is printed out in the following format: YYYY-MM-DD-HH.MM.SS.ZZZZZZ.

The special register CURRENT TIMESTAMP also produces the current timestamp.

column-options
Defines additional options related to columns of the table.

NOT NULL
Prevents the column from containing null values.

If NOT NULL is not specified, the column can contain null values, and its default value is either the null value or the value provided by the DEFAULT clause.

PRIMARY KEY
This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause is specified as a separate clause.

See the description of PRIMARY KEY on page 138.

REFERENCES *table-name*
See the description of REFERENCES on page 139.

CHECK (*check-condition***)**
See the description of CHECK on page 139.

DEFAULT
Provides a default value in the event that a value is not supplied on an INSERT statement.

Omission of DEFAULT from a column-definition results in the use of the null value as the default for the column. If such a column is defined NOT NULL, then the column does not have a valid default.

constant
Specifies the constant as the default value for the column. The specified constant must:

- Represent a value that could be assigned to the column.
- Not have non-zero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column).

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified).

GENERATED ALWAYS AS IDENTITY
When creating a table, a user can specify a column as ″GENERATED ALWAYS AS IDENTITY″. Subsequently, the value of this column will be generated by DB2 Everyplace each time the user performs an INSERT or INSERT with sub–SELECT. This column has to be a numeric type, (INTEGER, SMALLINT, or DECIMAL type), and DB2 Everyplace automatically generates unique serial numbers starting from 1, incremented by 1 each time.

The generated value for IDENTITY column starts from 1, and increases by 1 each time a row is inserted into the table. Thus, uniqueness is guaranteed, although DB2 Everyplace does not automatically create an index on an IDENTITY column. If you want to have an index on an IDENTITY column, you must either create an index explicitly, or specify the column as PRIMARY KEY. When the range of the values of an IDENTITY column is exhausted (the maximum value is reached), further INSERT statements will cause an error (SQLSTATE 23522). The maximum value of an IDENTITY column of INT and SMALLINT types are the maximum values allowed by those 2 types. The maximum value of an IDENTITY column of a DECIMAL type is determined by (1) definition of the data type (precision, scale) and (2) maximum value allowed for IDENTITY column: 2.15* (10∧18) (19 decimal digits). The smaller of the (1) and (2) is the range limit. For an IDENTITY column of a DECIMAL type, the value's fractional part is always 0, and the integral part is increased by 1 each time.

The IDENTITY specification can only be defined on columns whose data type is one of the 3 numeric types: INT, SMAIINT, DECIMAL. Otherwise, an error is raised (SQLSTATE 42815). There can be at most one IDENTITY column per table (otherwise error SQLSTATE 428C1). The user can not provide a value for an IDENTITY column in an INSERT statement (must default to DB2 Everyplace system generated value), nor can the user UPDATE an IDENTITY column.

PRIMARY KEY (*column-name***, ...)**
Defines a primary key composed of the identified columns. The clause must not be specified more than once and the identified columns must be defined as NOT NULL. Each column-name must identify a column of the table, and the same column must not be identified more than once.

The number of identified columns must not exceed 8.

A unique index will be automatically created on the specified columns.

Only one primary key can be defined on a table.

The length attribute of each specified column must not be greater than 1024 bytes.

referential-constraint
Defines a referential constraint.

FOREIGN KEY (*column-name***, ...)**
Defines a referential constraint with the specified constraint-name.

Let T1 denote the object table of the statement. The foreign key of the referential constraint is composed of the identified columns. Each name in the list of column names must identify a column of T1, and the same column must not be identified more than once. The number of identified columns must not exceed 8. Foreign keys are not enforced by DB2 Everyplace.

REFERENCES *table-name*
The table specified in a REFERENCES clause must identify a base table that is described in the catalog, but must not identify a catalog table.

A referential constraint is a duplicate if its foreign key is the same as the foreign key table of a previously specified referential constraint.

In the following discussion, let T2 denote the identified parent table, and let T1 denote the table being created.

The specified foreign key must have the same number of columns as the parent key of T2 and the description of the nth column of the foreign key must be comparable to the description of the nth column of that parent key. Datetime columns are not considered to be comparable to string columns for the purposes of this rule. Foreign keys are not enforced by DB2 Everyplace.

CHECK (*check-condition***)**
Defines a check constraint. A *check-condition* is a search condition. A column reference must be a column of the table being created. Values being inserted or updated into a table must satisfy any check constraints.

If a check constraint is specified as part of a column-definition then a column reference can be made only to the same column. Check constraints specified as part of a table definition can have column references identifying columns previously defined in the CREATE TABLE statement. Check constraints are not checked for inconsistencies, duplicate conditions, or equivalent conditions. Therefore, contradictory or redundant check constraints can be defined.

The check-condition ″IS NOT NULL″ can be specified, however it is recommended that nullability be enforced directly using the NOT NULL attribute of a column. For example, CHECK (salary + bonus > 30000) is accepted if salary is set to NULL, because CHECK constraints must be either satisfied or unknown and in this case salary is unknown. However, CHECK (salary IS NOT NULL) would be considered false and a violation of the constraint if salary is set to NULL.

Check constraints are enforced when rows in the table are inserted or updated.

All check constraints defined in a CREATE TABLE statement are combined and stored in the system catalog. DB2 Everyplace has a limit of 512 bytes for this combined check constraint.

Rules:

- The actual total of byte counts of a row must not be greater than 65 536.
  See 140 for more information.
- Columns with the BLOB data type cannot have check, default, referential, or foreign key constraints (SQLSTATE 42962).
- Columns with the BLOB data type cannot be used in the primary key of a CREATE TABLE statement.

Notes:

- Tables and columns should be created using uppercase names. Mixed case and lowercase names might cause errors to occur with some languages.
- If you create a new table on your mobile device, the table will not be automatically created on an enterprise database by synchronizing your mobile device with the server. The table must be created on the enterprise database before synchronization.
- Byte counts for data: The following list contains the byte counts of columns by data type. This count might change with each release. Each record also includes information about NULLs. NULL information requires 4 bytes for each group of 32 columns. A NULL value still uses the fixed size column size.

Data type	Column byte count
INTEGER	4
SMALLINT	4
DECIMAL(n, m)	4 – 20
CHAR(n)	n+1
VARCHAR(n)	i+5 where i is the actual length
BLOB	i+4 where i is the actual length
DATE	4
TIME	4
TIMESTAMP	12

Example:

Create table EMPLOYEE with column names EMPNO, FIRSTNAME, LASTNAME, DEPT, PHONENO, SALARY, and HIREDATE. CHAR means that the column will contain character data. NOT NULL means that the column cannot contain a null value. VARCHAR means that the column will contain varying-length character data. The primary key consists of the column EMPNO.

```
CREATE TABLE EMPLOYEE
  (EMPNO       CHAR(3)    PRIMARY KEY,
   FIRSTNAME   VARCHAR(12)   NOT NULL,
   LASTNAME    VARCHAR(15)   NOT NULL,
   DEPT        CHAR(3),
   PHONENO     CHAR(4),
   SALARY      INT,
   HIREDATE    DATE)
```

Related reference:

- “Overview of DB2 Everyplace SQL statement support” on page 129
- “Data type compatibility for assignments and comparisons” on page 166
- “DB2 Everyplace supported parameter markers” on page 72
- “SQLState listing” on page 170
- “Summary of SQLState class codes” on page 170

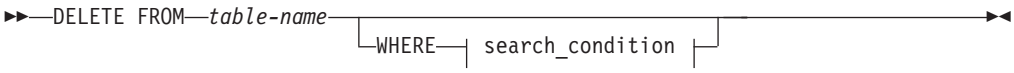
DELETE

The DELETE statement deletes one or more rows from a table.

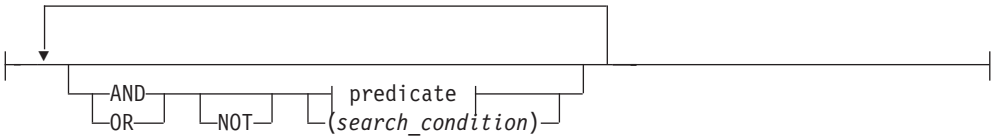
Invocation:

This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:

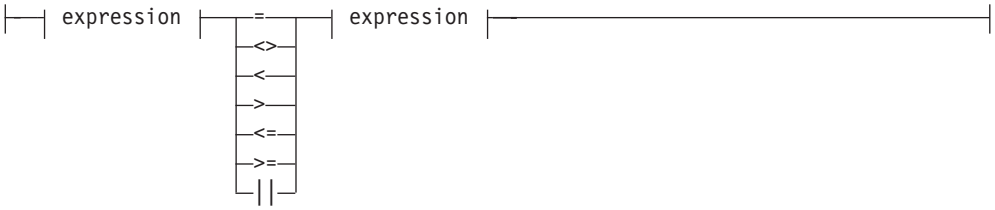


search_condition:

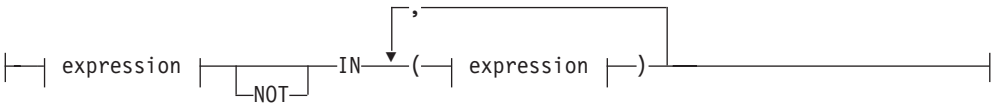


predicate:

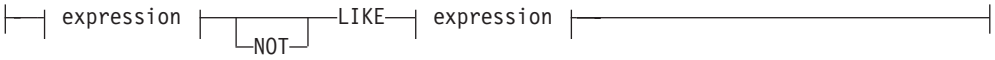
basic predicate:



IN predicate:

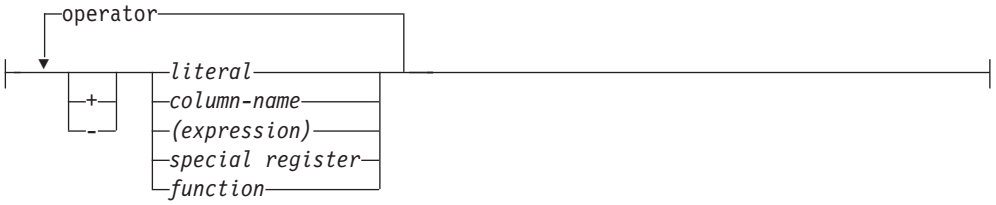


LIKE predicate:



NULL predicate:

expression:



operator:

Notes:

1 BLOB expressions are only allowed in NULL predicates.

Description:

FROM *table-name*
Identifies the table from which rows are to be deleted. The name must identify a table that exists in the catalog, but it must not identify a catalog table.

WHERE
Specifies a condition that selects the rows to be deleted. The clause can be omitted or a search condition specified. If the clause is omitted, all rows of the table are deleted.

search_condition
A *search_condition* specifies a condition that is true, false, or unknown about a given row.

The result of a *search_condition* is derived by applying the specified *logical operators* (AND, OR, NOT) to the result of each specified predicate. A predicate compares two values. If logical operators are not specified, the result of the search condition is the result of the specified predicate.

Search conditions within parentheses are evaluated first. If the order of evaluation is not specified by parentheses, NOT is applied before AND, and AND is applied before OR. The order in which operators at the same precedence level are evaluated is undefined to allow for optimization of search conditions.

The *search_condition* is applied to each row of the table and the deleted rows are those for which the result of the *search_condition* is true.

Each *column-name* in the search condition must identify a column of the table.

NOT
If NOT is specified, the result of the predicate is reversed.

expression
Identifies an operand of the predicate. The *expression* can be a literal, column name, special register, or function.

Arithmetic operations on BLOB(n), DATE, TIME, and TIMESTAMP data types are not supported.

literal
A *literal* can be a value of data type INTEGER, SMALLINT, DECIMAL, CHAR(n), VARCHAR(n), BLOB(n), DATE, TIME, or TIMESTAMP.

column-name
Identifies the column that is an operand of the predicate.

special register
Identifies the special register that is an operand of the predicate. The special registers CURRENT DATE, CURRENT TIME, and CURRENT TIMESTAMP can be used to produce the current date, time, or timestamp.

function
Can include only the MOD, LENGTH, and RTRIM functions.

relational operator
Can be any of the following operators:

= Equal to.

<> Not equal to.

< Less than.

> Greater than.

<= Less than or equal to.

>= Greater than or equal to.

LIKE Matches one character string. Use a single-byte character-set (SBCS) underscore to refer to one SBCS character. Use a double-byte character-set (DBCS) underscore to refer to one DBCS character. For example, the condition `WHERE PART_NUMBER LIKE '_0'` returns all 2–digit part numbers ending in 0 (20, 30, and 40, for example). Use a percent (either SBCS or DBCS) to refer to a string of zero or more SBCS or DBCS characters. For example, the condition `WHERE DEPT_NUMBER LIKE '2%'` returns all department numbers beginning with the number 2 (20, 27, or 234, for example).

NOT LIKE
Does not have at least one of the same characters.

IS NULL
Contains the null value.

IS NOT NULL
Does not contain the null value.

AND
If specified, the logical operator AND is applied to the result of each specified predicate.

OR
If specified, the logical operator OR is applied to the result of each specified predicate.

Rules:

None.

Notes:

- A logical DELETE never applies to logically deleted records.

Example:

Delete employee number (EMPNO) 003002 from the EMPLOYEE table.

```
DELETE FROM EMPLOYEE
  WHERE EMPNO = '003002'
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

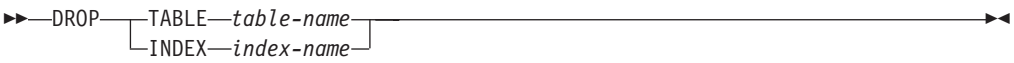
DROP

The DROP statement deletes a table or index.

Invocation:

This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:



Description:

TABLE *table-name*
Identifies the base table that is to be dropped. *table-name* must identify a table that is described in the catalog (SQLSTATE 42704).

INDEX *index-name*
Identifies the index that is to be dropped. The *index-name* must identify an index that is described in the catalog (SQLSTATE 42704). It cannot be an index required by the system for a primary key (SQLSTATE 42704).

Rules:

None.

Notes:

- Tables and indexes should not be dropped when a table is in use (a statement handle is active on a query which uses that table or index). Dropping tables and indexes that are in use will invalidate statement handles which involve the table or index.

Example:

Drop table EMPLOYEE.

```
DROP TABLE EMPLOYEE
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

EXPLAIN

The EXPLAIN statement obtains information about access path selection for a SELECT statement. The information obtained is placed in a user table named DB2ePLANTABLE.

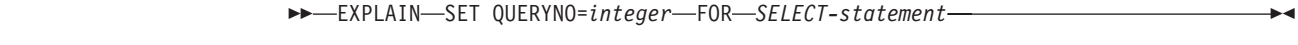
The EXPLAIN statement is supported on the following platforms:

- Win32 (Windows 95, Windows 98, Windows NT, Windows 2000, and Windows XP)
- Linux

Invocation:

This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:



Description:

SET QUERYNO = integer
Associates *integer* with the SELECT statement. The column QUERYNO is given the value *integer* in every row inserted into the plan table by the EXPLAIN statement.

SELECT-statement
Specifies a set of new rows in the form of the result table of a select statement.

Rules:

The *integer* value must be positive.

Notes:

- When you use the EXPLAIN statement, by default DB2ePLANTABLE is automatically created if it does not exist.
- To explicitly create DB2ePLANTABLE, use the following example:

```
create table "DB2ePLANTABLE"
 (query_no int, plan_no int, table_name char(18), index_name char(18), sort_temp char(1),
   expl_timestamp timestamp, remarks varchar(300))
```

Table 16 describes DB2ePLANTABLE columns.

Table 16. DB2ePLANTABLE column information

Column name	Description
query_no	The integer that connects the EXPLAIN statement to the output within DB2ePLANTABLE.
plan_no	The integer that represents the steps that the statement is executed in (in ascending order).
table_name	The name of the table or correlated name that uniquely identifies the table or null if not applicable.
index_name	The name of the index (if used) on the table access. Returns a null if no index is used.
sort_temp	'Y' means that a sort on a temporary table is needed to handle a GROUP BY or ORDER BY. If a null is returned it indicates that no sort temporary table is necessary.
expl_timestamp	The timestamp value when the EXPLAIN statement is executed.
remarks	The remarks column contains the null value. You can add remarks to this column for bookkeeping purposes.

- DB2ePLANTABLE is a user table that can be modified or dropped by any application.

Example:

When developing a new application, it is desirable to determine what access path is chosen for a SELECT statement. In this example, a new application queries the SALES and EMPLOYEES tables. The EXPLAIN statement shows whether the appropriate indexes are chosen for the SELECT statement.

```
 EXPLAIN SET QUERYNO = 100 FOR
  SELECT E.EMPNAME, S.SALES_AMOUNT
      FROM SALES S, EMPLOYEES E
            WHERE S.EMPNO = E.EMPNO
                  AND S.MONTH = ?

Index XSALES on SALES(MONTH)
Index XEMP on EMPLOYEES(EMPNO)

SELECT QUERY_NO, PLAN_NO, TABLE_NAME, INDEX_NAME,  SORT_TEMP
   FROM "DB2ePLANTABLE"

QUERY_NO  PLAN_NO  TABLE_NAME  INDEX_NAME  SORT_TEMP
--------------------------------------------------------------------------------
   100     1        SALES        XSALES        -
   100     2        EMPLOYEE     XEMP          -
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

GRANT

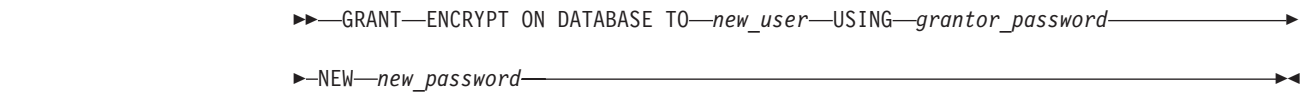
The GRANT statement gives you the permission to create, query, and manipulate encrypted tables within the database. To perform the GRANT operation, you must be currently connected and authenticated. If a database is not encrypted, you (as the first user) can grant yourself the authentication necessary to perform the GRANT operation. (See example 1 below for more information on how to do this.)

To change your own password, you should perform a GRANT operation on your own user ID.

Invocation:

This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:



Description:

new_user
Identifies the user being granted the encryption privileges.

grantor_password
The password of the authenticated user who is granting the new user encryption privileges.

new_password
The password of the user being granted the encryption privileges

Rules:

- Both the user name and the password parameter are limited in length to 254 bytes.
- For multi-byte characters, the UTF-8 encoding is used internally for storage. Therefore, user names written using international character sets are limited in length.
- DB2 Everyplace requires the grantor (that is, the currently-connected user) to re-enter the grantor password to be able to grant privileges to a new user. This restrictions ensures that the grantor is physically present at the device.
- Passwords and userids must be delimited by double quotes.

Notes:

- If you are an existing user, you must be connected and authenticated to change your own password. You can change your own password only.
- The GRANT statement cannot be used with parameter markers or the SQLPrepare() function.
- Attempting to GRANT privileges while connected with an unauthorized user returns SQLSTATE 42502. Specifying a wrong password with the GRANT statement causes a SQLSTATE 42506.

Example:

Example 1: The first user grants herself the authentication necessary to perform the GRANT operation, on a database that has not yet been encrypted:

```
GRANT ENCRYPT ON DATABASE TO "jsk" USING "foo" NEW "foo"
```

*Example 2:*Now the user "jsk" (in *Example 1*, above) is created and authenticated and owns the connection. For "jsk" to add another user:

```
GRANT ENCRYPT ON DATABASE TO "xin" USING "foo" NEW "bar"
```

*Example 3:*The user "jsk", currently connected, changes her own password:

```
GRANT ENCRYPT ON DATABASE TO "jsk" USING "foo" NEW "fie"
```

*Example 4:*The user "jsk", still currently connected, uses her new password to add another user:

```
GRANT ENCRYPT ON DATABASE TO "thf" USING "fie" NEW "fum"
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

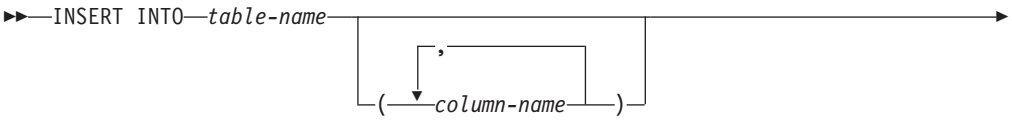
INSERT

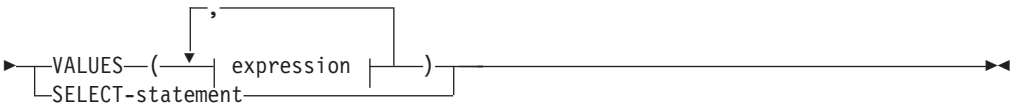
The INSERT statement inserts one or more rows into a table using the values provided.

Invocation:

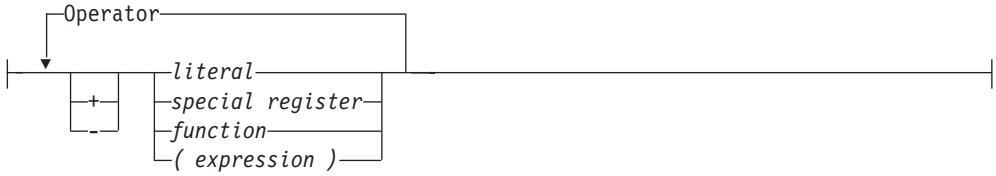
This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:





expression:



operator:

Description:

INTO *table-name*
Identifies the table of the insert operation. The name must identify an existing table, but it must not identify a catalog table.

(*column-name***,...)**
Specifies the columns for which insert values are provided. Each name must be an unqualified name that identifies a column of the table. The same column must not be identified more than once.

Omission of the column list is an implicit specification of a list in which every column of the table is identified in left-to-right order.

VALUES
Introduces one row of values to be inserted.

The number of values for each row must equal the number of names in the column list. The first value is inserted in the first column in the list, the second value in the second column, and so on.

expression
The expression can be a literal, special register, function, or a complex expression.

Arithmetic operations on CHAR, VARCHAR, BLOB(n), DATE, TIME and TIMESTAMP data types are not supported.

literal
A literal can be a value of any supported data type INTEGER, SMALLINT, DECIMAL, CHAR(n), VARCHAR(n), BLOB(n), DATE, TIME, or TIMESTAMP.

special register
The special registers CURRENT DATE, CURRENT TIME, and CURRENT TIMESTAMP can be used to produce the current date, time, and timestamp.

SELECT-statement
Specifies a set of new rows in the form of the result table of a select statement. There may be one, more than one, or none. If the result table is empty, SQLCODE is set to +100 and SQLSTATE is set to '02000'. The base object of the select statement cannot be the base object of the INSERT.

Rules:

Default values
A default or null value is inserted in any column that is not in the column list. Columns that do not allow default or null values must be included in the column list.

Length
If the insert value of a column is a number, the column must be a numeric column with the capacity to represent the integral part of the number. If the insert value of a column is a string, the column must be a string column with a length attribute at least as great as the length of the string.

Assignment

Insert values are assigned to columns in accordance with the assignment rules described in the *DB2 Universal Database SQL Reference*.

Examples:

Example 1: Insert an employee with the following specifications into the EMPLOYEE table:

- Employee number (EMPNO) is 002001
- First name (FIRSTNAME) is John
- Last name (LASTNAME) is Harrison
- Department number (DEPT) is 600
- Phone number (PHONENO) is 4900
- Salary (SALARY) is 50000
- Date of hire (HIREDATE) is 01/12/1989

```
INSERT INTO EMPLOYEE
 VALUES ('002001', 'John', 'Harrison', '600', '4900', 50000, '01/12/1989')
```

Example 2: Insert a new employee with the following specifications into the EMPLOYEE table:

- Employee number (EMPNO) is 003002
- First name (FIRSTNAME) is Jim
- Last name (LASTNAME) is Gray

```
INSERT INTO EMPLOYEE (EMPNO, FIRSTNAME, LASTNAME)
  VALUES ('003002', 'Jim', 'Gray')
```

Example 3: Create a table EMP_ACT_COUNT. Load EMP_ACT_COUNT with the rows from the EMP_ACT table with an employee number (EMPNO) with the number of projects involved.

```
CREATE TABLE EMP_ACT_COUNT
   ( EMPNO CHAR(6)  NOT NULL,
     COUNT          INTEGER)

INSERT INTO EMP_ACT_COUNT
    SELECT EMPNO, COUNT(*)
   FROM EMP_ACT
   GROUP BY EMPNO
```

Restrictions:

1. The column data types of SELECT-statement must be identical to the column definition of the target table (except nullability).
2. ORDER BY and LIMIT clauses are not allowed.

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

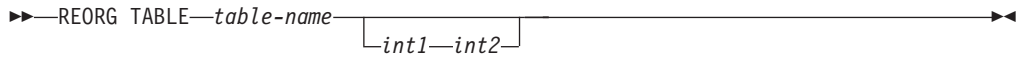
REORG TABLE

The REORG TABLE statement compresses the data associated with the specified table.

Invocation:

This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:



Description:

REORG TABLE *table-name*
Identifies the table of the reorganization operation. The name must identify an existing table.

int1
The optional minimal percentage of bytes that need to be recovered.

int2
The minimal number of bytes that need to be recovered for the table compression to be executed.

Rules:

- The optional values *int1* and *int2* must be used together or not at all.
- The optional value *int1* must be a non-negative number.
- The optional value *int1* must be between 0 and 100.

Notes:

- A table reorganization can be invoked by DB2 Everyplace internally.
- The first optional parameter is the percentage of unusable bytes that the table must contain (i.e.10 percent means ″at least 10 percent of the space is unusable″). The second optional parameter is the number of unusable bytes that the table must contain (i.e. 1000 would mean ″at least 1000 bytes must be unusable space). Both criteria must be met before an actual reorganization of the table takes place.
- If there are no parameters specified, DB2 Everyplace uses default values for these options. The default percentage is 30 and the default bytes is 6144. Thus, ″reorg table t1″ is the same as ″reorg table t1 30 6144″.
- If the reorganization mode is set to enabled, then DB2 Everyplace will automatically reorganize a table. If reorganization is enabled, on a DELETE or UPDATE, a ″reorg table table_name 50 30270″ is executed for the target table after the statement is executed. If reorganization is enabled, on a DROP TABLE, a ″reorg table DB2eSYSTABLES 30 10240″ (also for DB2eSYSCOLUMNS and DB2eSYSRELS) is executed at the end of the drop table processing.
- In a C/C++ program the reorganization mode is set by using the CLI/ODBC function SQLSetStmtAttr with the attribute SQL_ATTR_REORG_MODE. In a JAVA program the reorganization mode is set by the DB2eStatement interface enableReorg method. The default value is reorganization is enabled.

- Reorganizing a table compresses the data file that contains the table by physically reclaiming unusable space create by deletes and updates. Then indexes for the table are updated to point to the new physical location of the rows.
- DB2 Everyplace System Catalog base tables can be reorganized.
- No other activity should be occurring in the database while a REORG TABLE statement is being executed.

Examples:

The VNNURSE table is compressed using the default values.

```
REORG TABLE VNNURSE
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

REVOKE

The REVOKE statement permits a connected and authenticated user to revoke encryption privileges from an existing user.

Invocation:

This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:

Description:

user
Identifies the user whose encryption privileges are being revoked.

Rules:

- The user parameter must be delimited identifier. It is limited in length to 254 bytes.
- For multi-byte characters, the UTF-8 encoding is used internally for storage. Therefore, user names written using international character sets are limited in length.
- If all users with encryption privileges are removed, encrypted tables can continue to be accessed during the current session. After the current session is terminated, the encrypted tables are no longer accessible.

Notes:

- A user must be connected and authenticated to revoke privileges from an existing user. If you are a connected and authenticated user, you can revoke privileges from any user including yourself.

- The REVOKE statement cannot be used with parameter markers or the SQLPrepare() function.
- Attempting to REVOKE privileges while connected as an unauthorized user returns SQLSTATE 42502. Trying to REVOKE privileges from a non-existing user results in SQLSTATE 42501.

Example:

The currently connected, authenticated user removes encryption privileges from user ″jsk″:

```
REVOKE ENCRYPT ON DATABASE FROM "jsk"
```

Related reference:

- “Overview of DB2 Everyplace SQL statement support” on page 129
- “Data type compatibility for assignments and comparisons” on page 166
- “DB2 Everyplace supported parameter markers” on page 72
- “SQLState listing” on page 170
- “Summary of SQLState class codes” on page 170

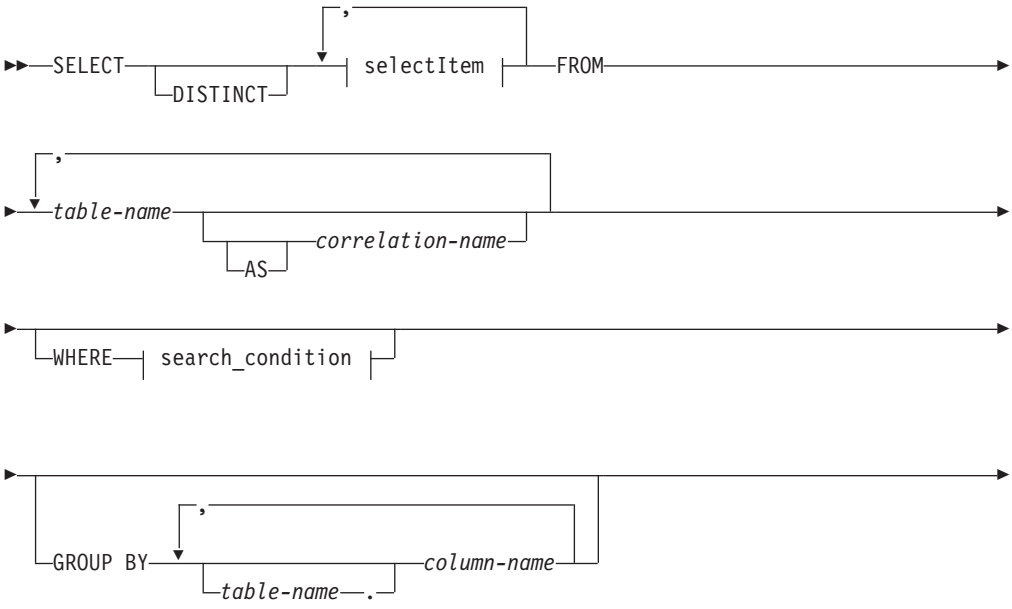
SELECT

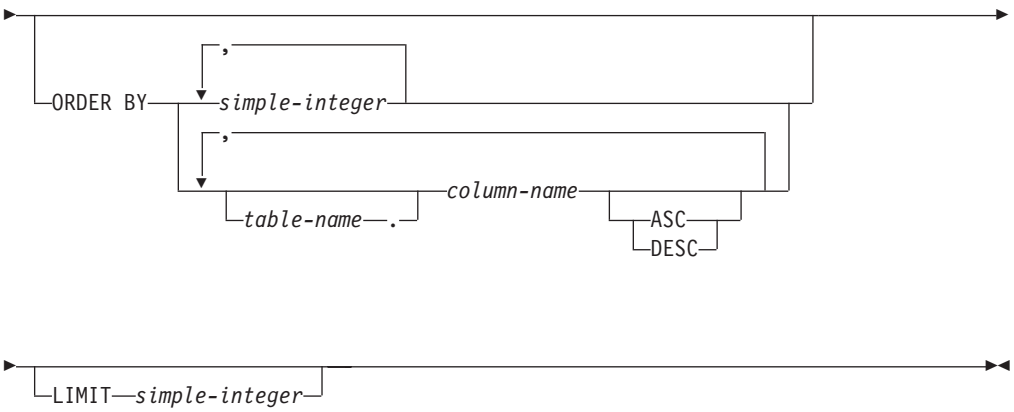
The SELECT statement is a form of query.

Invocation:

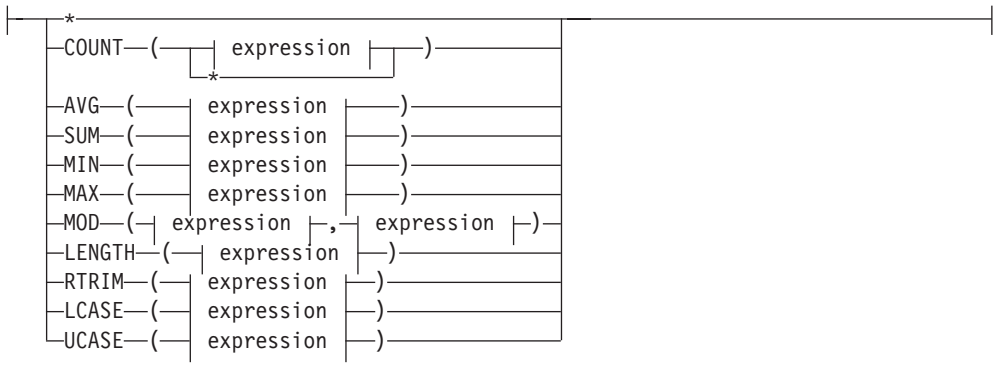
This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

Syntax:

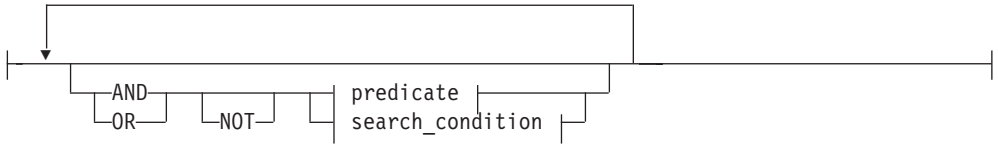




selectItem:

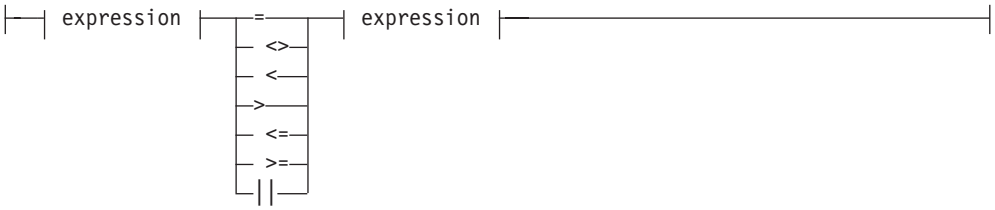


search_condition:



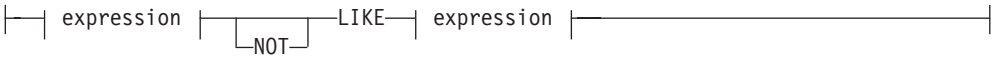
predicate:

basic predicate:



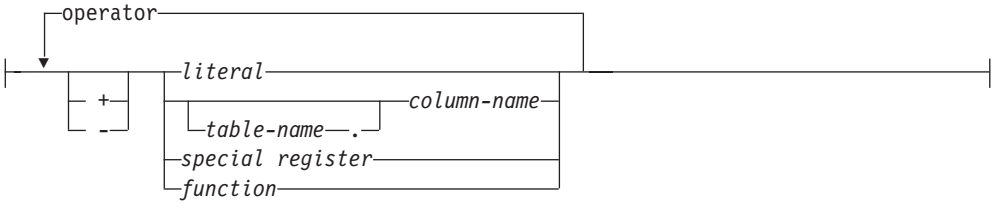
IN predicate:

LIKE predicate:



NULL predicate:

expression:



operator:

Notes:

1 BLOB expressions are allowed only in NULL predicates.

Description:

selectItem

* Specifies all columns. If * is specified, it must be the only select item.

COUNT(*)
The COUNT function returns the number of rows or values in a set of rows or values. The argument of COUNT(*) is a set of rows. The result is the number of rows in the set. A row that includes only NULL values is included in the count.

expression
The *expression* can be a literal, column name, function, or special register. Valid functions are: COUNT, AVG, SUM, MIN, MAX, MOD, LENGTH, and RTRIM.

Arithmetic operations on CHAR, VARCHAR, BLOB(n), DATE, TIME, and TIMESTAMP data types are not supported.

literal
A *literal* can be a value of data type INTEGER, SMALLINT, DECIMAL, CHAR(n), VARCHAR(n), BLOB(n), DATE, TIME, and TIMESTAMP.

table-name
Identifies the table containing the column that you are querying.

. Separator in the two-part column identifier, *table-name.column-name*.

column-name
Identifies the column that you are querying.

COUNT(*expression***)**
The argument of COUNT(*expression*) is a set of rows. The function is applied to the set of rows derived from the argument values by the elimination of null values. The result is the number of non-null values in the set, including duplicates.

AVG(*expression***)**
The AVG(*expression*) function returns the average of the values of *expression*. The argument values must be numbers and their sum must be within the range of the data type of the result. The function is applied to the set of values derived from the argument values by the elimination of null values. The result can be null.

SUM(*expression***)**
The SUM(*expression*) function returns the sum of the values of *expression*. The argument values must be numbers and their sum must be within the range of the data type of the result. The function is applied to the set of values derived from the argument values by the elimination of null values.

MIN(*expression***)**
The MIN(*expression*) function returns the minimum value in the set of values of *expression*. The argument values can be of any built-in type other than a BLOB. The function is applied to the set of values derived from the argument values by the elimination of null values.

MAX(*expression***)**
The MAX(*expression*) function returns the maximum value in the set of values of *expression*. The argument values can be of any built-in type other than a BLOB. The function is applied to the set of values derived from the argument values by the elimination of null values.

MOD(*expression, expression***)**
The MOD(*expression, expression*) function returns the remainder of the first argument divided by the second argument. The result is negative only if the first argument is negative.

The first and second arguments can be either SMALLINT or INTEGER.

The result of the function is SMALLINT if both arguments are SMALLINT; otherwise, it is an INTEGER. The result can be null; if any argument is null, the result is the null value.

(*expression* || *expression***)**
The (*expression* || *expression*) returns the concatenation of two string arguments. The two arguments must be compatible types.

The result of the function is a string. Its length is sum of the lengths of the two arguments. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

LENGTH(*expression***)**

The LENGTH(*expression*) function returns the length of a value.

The argument can be an expression that returns a value of the following built-in data types:

- VARCHAR
- CHAR
- BLOB

The result of the function is an integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result is the length of the argument. The length of a varying-length string is the actual length, not the maximum length.

The length of a BLOB is the number of bytes used to represent the value.

Consider a VARCHAR(50) column named ADDRESS with a value of '895 Don Mills Road'. `LENGTH(ADDRESS)` returns the value 18.

RTRIM(*expression***)**

The RTRIM(*expression*) function removes blanks from the end of the string.

The argument can be a CHAR or VARCHAR data type.

The result data type of the function is always VARCHAR.

The length parameter of the returned type is the same as the length parameter of the argument data type.

The actual length of the result for character strings is the length of the string-expression minus the number of bytes removed for blank characters. The actual length of the result for graphic strings is the length (in number of double byte characters) of string-expression minus the number of double byte blank characters removed. If all of the characters are removed, the result is an empty, varying-length string (length is zero).

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Consider a CHAR(50) column named NAME with a value of 'Cliff '. `RTRIM(NAME)` returns 'Cliff'. `LENGTH(RTRIM(NAME))` returns 5.

LCASE / UCASE

The LCASE or LOWER function returns a string in which all the SBCS characters have been converted to lowercase characters. That is, the

characters A-Z will be translated to the characters a-z, and characters with diacritical marks will be translated to their lowercase equivalents if they exist.

The argument must be an expression whose value is a CHAR or VARCHAR data type.

The result of the function has the same data type and length attribute as the argument. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Ensure that the characters in the value of column JOB in the EMPLOYEE table are returned in lowercase characters. For example:

```
SELECT LCASE(JOB)
     FROM EMPLOYEE
     WHERE EMPNO = '000020';
```

special register
The special registers CURRENT DATE, CURRENT TIME, and CURRENT TIMESTAMP can be used to produce the current date, time, and timestamp.

FROM
The FROM clause specifies an intermediate result table.

If one table-reference is specified, the intermediate result table is simply the result of that table-reference. If more than one table-reference is specified, the intermediate result table consists of all possible combinations of the rows of the specified table-references (the Cartesian product). Each row of the result is a row from the first table-reference concatenated with a row from the second table-reference, concatenated in turn with a row from the third, and so on. The number of rows in the result is the product of the number of rows in all the individual table-references. A maximum of 20 tables can be specified in the FROM clause.

table-name
Each *table-name* specified as a table-reference must identify an existing table.

AS
Identifies the table definition.

correlation-name
Each *correlation-name* is defined as a designator of the immediately preceding *table-name*. If a correlation name is specified for a table, any qualified reference to a column of the table must use the correlation name rather than the table name. If the same *table-name* is specified twice, at least one specification should be followed by a *correlation-name*. The *correlation-name* is used to qualify references to the columns of the table. As a qualifier, a correlation name can be used to avoid ambiguity or to establish a correlated reference. It can also be used merely as a shorter name for a table.

WHERE
Specifies a condition that selects the rows. The clause can be omitted or a search condition specified. If the clause is omitted, all rows of the table are selected.

search_condition
A *search_condition* specifies a condition that is true, false, or unknown about a given row.

The result of a *search_condition* is derived by applying the specified *logical operators* (AND, OR, NOT) to the result of each specified predicate. A predicate

compares two values. If logical operators are not specified, the result of the search condition is the result of the specified predicate.

Search conditions within parentheses are evaluated first. If the order of evaluation is not specified by parentheses, NOT is applied before AND, and AND is applied before OR. The order in which operators at the same precedence level are evaluated is undefined to allow for optimization of search conditions.

The *search_condition* is applied to each row of the table, and the selected rows are those for which the result of the *search_condition* is true.

Each *column-name* in the search condition must identify a column of the table.

NOT
If NOT is specified, the result of the predicate is reversed.

expression
The *expression* can be a literal, column name, special register, or function.

Arithmetic operations on CHAR, VARCHAR, BLOB(n), DATE, TIME and TIMESTAMP data types are not supported.

literal
A *literal* can be a value of data type INTEGER, SMALLINT, DECIMAL, CHAR(n), VARCHAR(n), BLOB(n), DATE, TIME, or TIMESTAMP.

table-name
Identifies the table containing the column that is an operand of the predicate.

. Separator in the two-part column identifier, *table-name.column-name*.

column-name
Identifies the column that is an operand of the predicate.

special register
Identifies the special register that is an operand of the predicate. The special registers CURRENT DATE, CURRENT TIME, and CURRENT TIMESTAMP can be used to produce the current date, time, and timestamp.

function
Can include LCASE, UCASE, MOD, LENGTH, and RTRIM functions.

operator
Can be any of the following operators:

= Equal to.

<> Not equal to.

< Less than.

> Greater than.

<= Less than or equal to.

>= Greater than or equal to.

|| Returns the concatenation of two string arguments.

LIKE Matches one character string. Use a single-byte character-set (SBCS) underscore to refer to one SBCS character. Use a double-byte character-set (DBCS) underscore to refer to one DBCS character. For example, the condition `WHERE PART_NUMBER LIKE '_0'` returns all

2–digit part numbers ending in 0 (20, 30, and 40, for example). Use a percent (either SBCS or DBCS) to refer to a string of zero or more SBCS or DBCS characters. For example, the condition `WHERE DEPT_NUMBER LIKE '2%'` returns all department numbers beginning with the number 2 (20, 27, or 234, for example).

NOT LIKE
Does not have at least one of the same characters.

IN Matches a collection of values. The IN predicate compares a value with a collection of values.

Examples:

SELECT lname, fname FROM emp WHERE state IN ('CA', 'AZ', 'OR');

SELECT c1 FROM t1 WHERE c1*5-6 IN (mod(c2,2)+5,c3+4/2);

NOT IN
Does not match a collection of values. The NOT IN predicate compares a value with a collection of values.

Examples:

SELECT empid FROM emp WHERE city NOT IN ('San Jose', 'Morgan Hill', 'Santa Clara');

IS NULL
Contains the null value.

IS NOT NULL
Does not contain the null value.

AND
If specified, the logical operator AND is applied to the result of each specified predicate.

OR
If specified, the logical operator OR is applied to the result of each specified predicate.

GROUP BY
Specifies an intermediate result table that consists of a grouping of the rows of R. R is the result of the previous clause of the subselect.

ORDER BY
Specifies an ordering of the rows of the result table.

column-name
Usually identifies a column of the result table. In this case, *column-name* must be the column name of a named column in the select list.

simple-integer
Must be greater than 0 and not greater than the number of columns in the result table. The integer *n* identifies the *n*-th column of the result table.

ASC
Uses the values of the column in ascending order.

DESC
Uses the values of the column in descending order.

LIMIT *simple-integer*
Limits the number of rows to be returned to the application to the first *n* number of rows in the answer set where *n* is an integer. Must be greater than 0.

Relational operators
Can be one of the following operators

+	Add
-	Subtract
*	Multiple
/	Divide by

Rules:

BLOB data type columns cannot be used in GROUP BY, ORDER BY, and DISTINCT clauses.

Notes:

- A SELECT DISTINCT statement can contain a maximum of 8 columns.
- A GROUP BY clause can contain a maximum of 8 columns.
- An ORDER BY clause can contain a maximum of 8 columns.
- All columns specified in the ORDER BY clause must appear in the select list. For example, the following query is not valid:

  ```
  SELECT EMPNO, FIRSTNAME FROM EMPLOYEE ORDER BY LASTNAME
  ```

  The following query is valid:

  ```
  SELECT LASTNAME, EMPNO, FIRSTNAME FROM EMPLOYEE ORDER BY LASTNAME
  ```

Examples:

Example 1: Select the employees (EMPNO and LASTNAME) from the EMPLOYEE table who were hired after 01/01/1980 and put them in order of their last name (LASTNAME).

```
SELECT EMPNO, LASTNAME FROM EMPLOYEE
 WHERE HIREDATE > '01/01/1980'
 ORDER BY LASTNAME
```

Example 2: Compute the average salary for each department in the EMPLOYEE table.

```
SELECT DEPT, AVG(SALARY) FROM EMPLOYEE
 GROUP BY DEPT
```

Example 3: Compute the maximum sales volume for each sales region, and display the results by region, in order of highest to lowest sales volume.

```
SELECT REGION, MAX(SALES_VOL) FROM SALES
 GROUP BY REGION ORDER BY 2 DESC
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166

- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

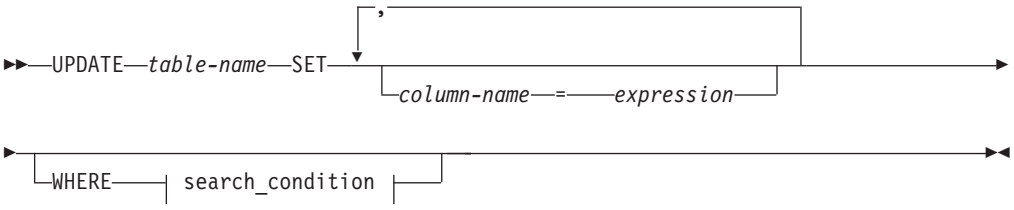
UPDATE

The UPDATE statement updates the values of specified columns in rows of a table.

Invocation:

This statement can be used in an application program using the DB2 CLI functions or issued through the CLP.

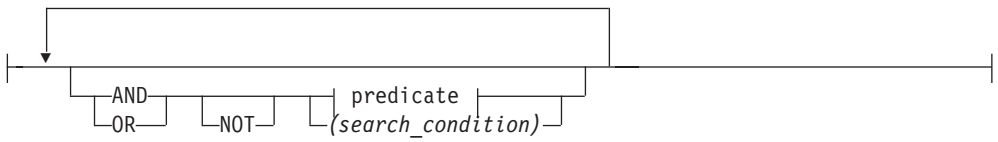
Syntax:



expression:

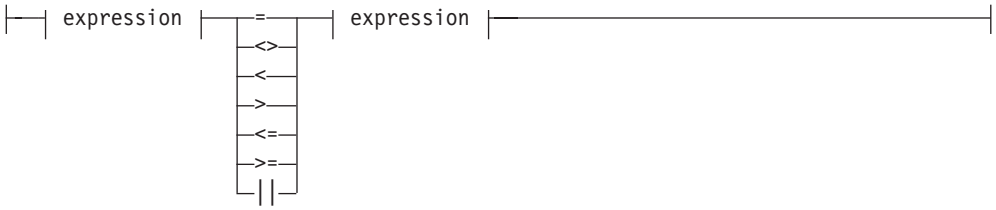
operator:

search_condition:

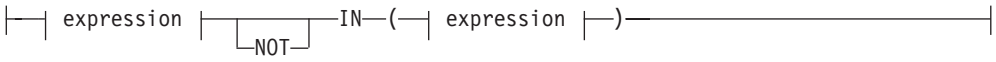


predicate:

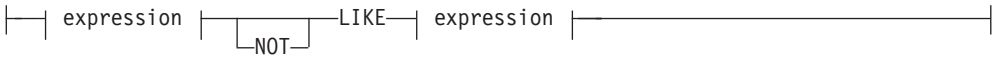
basic predicate:



IN predicate:



LIKE predicate:



NULL predicate:

relational operator:

Notes:

1 BLOB expressions are allowed only in NULL predicates.

Description:

table-name
Is the name of the table to be updated. The name must identify a table described in the catalog, but not a catalog table.

SET
Introduces the assignment of values to column names.

column-name
Identifies a column to be updated. The *column-name* must identify a column of the specified table. A column must not be specified more than once (SQLSTATE 42701).

expression
An *expression* can be a literal, column name, or special register.

Arithmetic operations on BLOB(n), DATE, TIME, and TIMESTAMP data types are not supported.

literal
A *literal* can be a value of data type INTEGER, SMALLINT, DECIMAL, CHAR(n), VARCHAR(n), BLOB(n), DATE, TIME, or TIMESTAMP.

special register
The special registers CURRENT DATE, CURRENT TIME, and CURRENT TIMESTAMP can be used to produce the current date, time, and timestamp.

WHERE
Introduces a condition that indicates what rows are updated. You can omit the clause or give a search condition. If the clause is omitted, all rows of the table are updated.

search_condition
A *search_condition* specifies a condition that is true, false, or unknown about a given row.

The result of a *search_condition* is derived by applying the specified *logical operators* (AND, OR, NOT) to the result of each specified predicate. A predicate compares two values. If logical operators are not specified, the result of the search condition is the result of the specified predicate.

Search conditions within parentheses are evaluated first. If the order of evaluation is not specified by parentheses, NOT is applied before AND, and AND is applied before OR. The order in which operators at the same precedence level are evaluated is undefined to allow for optimization of search conditions.

The *search_condition* is applied to each row of the table and the updated rows are those for which the result of the *search_condition* is true.

Each *column-name* in the search condition must identify a column of the table.

You can use the CONCAT, MOD, LENGTH, and RTRIM functions in the predicate expression of the search condition. For more information about the MOD function, see page 156.

NOT
If NOT is specified, the result of the predicate is reversed.

relational operator
Can be any of the following operators:

=	Equal to.
<>	Not equal to.
<	Less than.
>	Greater than.

<= Less than or equal to.

>= Greater than or equal to.

LIKE Matches one character string. Use a single-byte character-set (SBCS) underscore to refer to one SBCS character. Use a double-byte character-set (DBCS) underscore to refer to one DBCS character. For example, the condition `WHERE PART_NUMBER LIKE '_0'` returns all 2–digit part numbers ending in 0 (20, 30, and 40, for example). Use a percent (either SBCS or DBCS) to refer to a string of zero or more SBCS or DBCS characters. For example, the condition `WHERE DEPT_NUMBER LIKE '2%'` returns all department numbers beginning with the number 2 (20, 27, or 234, for example).

NOT LIKE
Does not have at least one of the same characters.

IS NULL
Contains the null value.

IS NOT NULL
Does not contain the null value.

AND
If specified, the logical operator AND is applied to the result of each specified predicate.

OR
If specified, the logical operator OR is applied to the result of each specified predicate.

Rules:

- ***Assignment***: Update values are assigned to columns under the assignment rules described in the *DB2 Universal Database SQL Reference*.
- UPDATE never applies to logically deleted records.

Notes:

- In system mode the dirty bit is set by default. If you are running your application in system mode (SQL_DIRTYBIT_SET_BY_SYSTEM), you cannot manually set the dirty bit. If you try to set the dirty bit, an error will occur.

See on page 259 for more information.

Example:

Change the phone number (PHONENO) of employee number (EMPNO) '003002' in the EMPLOYEE table to '1234'.

```
UPDATE EMPLOYEE
 SET PHONENO = '1234'
 WHERE EMPNO = '003002'
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170
- "Summary of SQLState class codes" on page 170

Data type compatibility for assignments and comparisons

Assignment operations are performed during the execution of INSERT and UPDATE statements. Comparison operations are performed during the execution of statements that include predicates. The data types of the operands involved must be compatible, as shown in Table 17 through Table 19.

If the data type column contains:

X The data types of the operands are compatible.

blank The data types of the operands are not compatible.

Table 17. Data type compatibility, table 1

SQL data type	INT	SMALLINT	DECIMAL	BLOB
INT	X	X	X	
VARCHAR				
BLOB				X
DECIMAL	X	X	X	
CHAR				
SMALLINT	X	X	X	
DATE				
TIME				
TIMESTAMP				

Table 18. Data type compatibility, table 2

SQL data type	CHAR	VARCHAR
INT		
VARCHAR	X	X
BLOB		
DECIMAL		
CHAR	X	X
SMALLINT		
DATE	X	X
TIME	X	X
TIMESTAMP	X	X

Table 19. Data type compatibility, table 3

SQL data type	DATE	TIME	TIMESTAMP
INT			
VARCHAR	X	X	X
BLOB			
DECIMAL			
CHAR	X	X	X
SMALLINT			
DATE	X		
TIME		X	

Table 19. Data type compatibility, table 3 (continued)

SQL data type	DATE	TIME	TIMESTAMP
TIMESTAMP			X

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "DB2 Everyplace supported parameter markers" on page 72
- "SQLState listing" on page 170

SQL symbolic and default data types

Table 20. SQL symbolic and default data types

SQL Data Type	Symbolic SQL Data Type	Default Symbolic C Data Type
BLOB	SQL_BLOB	SQL_C_BINARY
CHAR	SQL_CHAR	SQL_C_CHAR
DATE	SQL_TYPE_DATE	SQL_C_TYPE_DATE
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TYPE_TIME	SQL_C_TYPE_TIME
TIMESTAMP	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP
VARCHAR	SQL_VARCHAR	SQL_C_CHAR

Data type attributes

Information is shown for the following data type attributes:

- Precision
- Scale
- Length
- Display Size

Precision:

The precision of a numeric column or parameter refers to the maximum number of digits used by the data type of the column or parameter. The precision of a non-numeric column or parameter generally refers to the maximum length or the defined length of the column or parameter. The following table defines the precision for each SQL data type.

Table 21. Precision

fSqlType	Precision
SQL_CHAR SQL_VARCHAR	The defined length of the column or parameter. For example, the precision of a column defined as CHAR(10) is 10.
SQL_DECIMAL	The defined maximum number of digits. For example, the precision of a column defined as DECIMAL(10,3) is 10.

Table 21. Precision (continued)

fSqlType	Precision
SQL_SMALLINT a	5
SQL_INTEGER a	10
SQL_BLOB	The defined length of the column or parameter. For example, the precision of a column defined as BLOB(10), is 10.
SQL_DATE a	10 (the number of characters in the yyyy-mm-dd format).
SQL_TIME a	8 (the number of characters in the hh:mm:ss format).
SQL_TIMESTAMP	26 (The number of characters in the ″yyyy-mm-dd-hh.mm.ss.ffffff″ format used by the TIMESTAMP data type.)

a: The *cbParamDef* argument of `SQLBindParameter()` is ignored for this data type.

Scale:

The scale of a numeric column or parameter refers to the maximum number of digits to the right of the decimal point. The following table defines the scale for each SQL data type.

Table 22. Scale

fSqlType	Scale
SQL_CHAR SQL_VARCHAR	Not applicable.
SQL_DECIMAL	The defined number of digits to the right of the decimal place. For example, the scale of a column defined as DECIMAL(10, 3) is 3.
SQL_SMALLINT SQL_INTEGER	0
SQL_BLOB	Not applicable.
SQL_DATE SQL_TIME	Not applicable.
SQL_TIMESTAMP	6 (The number of digits to the right of the decimal point in the ″yyyy-mm-dd-hh.mm.ss.ffffff″ format.)

Length:

The length of a column is 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. Note that the length of a column may be different than the number of bytes required to store the data on the data source.

The following table defines the length for each SQL data type.

Table 23. Length

fSqlType	Length
SQL_CHAR SQL_VARCHAR	The defined length of the column. For example, the length of a column defined as CHAR(10) is 10.
SQL_DECIMAL	The maximum number of digits plus two. Since these data types are returned as character strings, characters are needed for the digits, a sign, and a decimal point. For example, the length of a column defined as DECIMAL(10,3) is 12.
SQL_SMALLINT	2 (two bytes).
SQL_INTEGER	4 (four bytes).
SQL_BLOB	The defined length of the column. For example, the length of a column defined as BLOB(10) is 10.
SQL_DATE SQL_TIME	6 (the size of the DATE_STRUCT or TIME_STRUCT structure).
SQL_TIMESTAMP	16 (the size of the TIMESTAMP_STRUCT structure).

Display size:

The display size of a column is the maximum number of bytes needed to display data in character form. The following table defines the display size for each SQL data type.

Table 24. Display size

fSqlType	Display size
SQL_CHAR SQL_VARCHAR	The defined length of the column. For example, the display size of a column defined as CHAR(10) is 10.
SQL_DECIMAL	The precision of the column plus two (a sign, precision digits, and a decimal point). For example, the display size of a column defined as DECIMAL(10,3) is 12.
SQL_SMALLINT	6 (a sign and 5 digits).
SQL_INTEGER	11 (a sign and 10 digits).
SQL_BLOB	The defined length of the column times 2 (each binary byte is represented by a 2 digit hexadecimal number). For example, the display size of a column defined as BLOB(10) is 20.
SQL_DATE	10 (a date in the format yyyy-mm-dd).
SQL_TIME	8 (a time in the format hh:mm:ss).
SQL_TIMESTAMP	26 (a timestamp in the format yyyy-mm-dd-hh.mm.ss.ffffff).

SQLState listing

This section will help you interpret error messages generated from SQL or CLI.

- "Summary of SQLState class codes" contains a listing of the general categories for errors.
- "SQLState messages reported by SQL" on page 171, "SQLState messages reported by CLI" on page 174, and "SQLState messages reported by JDBC" on page 182 contain descriptions of each error, and for SQL, provide the name of the function that generated it.

You can also find SQLSTATE descriptions by using a DB2 command line processor, if you have DB2 UDB installed:

1. To open the command line processor, select **Start —> Program —> DB2 —> Command Line Processor**.
2. On the command line, type `? [SQLSTATE]`.

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "Summary of SQLState class codes"

Summary of SQLState class codes

The first two characters of the SQLState messages in Table 27 on page 174 are in **bold** to indicate the class code. These class codes are *summarized* in Table 25.

Table 25. SQLState Class Codes

Code	Class
00	Unqualified successful completion
01	Warning
02	No data
07	Dynamic SQL error
08	Connection exception
09	Triggered action exception
0A	Feature not supported
0F	Invalid token
21	Cardinality violation
22	Data exception
23	Constraint violation
24	Invalid cursor state
25	Invalid transaction state
26	Invalid SQL statement identifier
28	Invalid authorization specification
2D	Invalid transaction termination
2E	Invalid connection name
34	Invalid cursor name
38	External function exception
39	External function call exception

Table 25. SQLState Class Codes (continued)

Code	Class
40	Transaction rollback
42	Syntax error or access rule violation
44	With check option violation
46	Java DDL
51	Invalid application state
54	SQL or product limit exceeded
55	Object not in prerequisite state
56	Miscellaneous SQL or product error
57	Resource not available or operator intervention
58	System ErrorResource
59	DB2 Everyplace Administrator error
HY	Generated by the DB2 CLI or ODBC driver
IM	Generated by the ODBC driver manager

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "SQLState listing" on page 170
- "DB2 Everyplace supported parameter markers" on page 72

SQLState messages reported by SQL

Table 26 lists all of the SQLSTATEs for the SQL statements reported by the DB2 Everyplace SQL engine. The SQLSTATEs reported by DB2 CLI are listed under each DB2 CLI function description in "DB2 CLI function summary" on page 182.

Table 26. SQLSTATE messages reported by SQL

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Value was truncated.	The value was truncated by a system cast or adjustment function.
01550	The index was not created.	The index was not created, because an index with the specified description already exists.
02000	No row was found.	No row was found during the execution of a FETCH, DELETE, or UPDATE statement.
07001	Wrong number of parameters.	A parameter marker has not been bound.
07005	Invalid parameter.	The statement name of the cursor identifies a prepared statement that cannot be associated with a cursor.
07006	Invalid variable.	An input host variable cannot be used because of its data type.
08002	Connection already exists.	A connection already exists.
22001	Value requires truncation.	A value requires truncation by a system cast or adjustment function.
22002	No null indicator provided.	A NULL value cannot be assigned because no storage is provided.
22003	Numeric value out of range.	A numeric value is not within the range of its target column.

Table 26. SQLSTATE messages reported by SQL (continued)

SQLSTATE	Description	Explanation
22007	Invalid datetime format.	The syntax of the string representation of a datetime value is incorrect.
22008	Datetime value out of range.	The string representation of a datetime value is out of range.
22012	Divide by zero.	A divide by zero operation was attempted.
22504	Fragmented MBCS character.	The data contains an improperly formed multi-byte character.
23502	Null value not allowed.	The assignment of a NULL value to a NOT NULL column is not allowed.
23505	Values are not unique.	The operation was not valid because it would produce duplicate keys.
23513	Invalid value.	The resulting row of the INSERT or UPDATE statement does not conform to the check constraint definition.
23515	More than one primary key clause is specified.	More than one primary key clause is specified.
24000	Invalid cursor state.	The *StatementHandle* was in an executed state but no result set was associated with the *StatementHandle*.
24501	Cursor not open.	A FETCH is not valid because no result set has been generated.
24505	Cursor not positioned.	A FETCH is not valid because the cursor is not positioned on a row.
34000	Cursor name is invalid.	Cursor name is invalid.
42501*	Authorization ID not permitted to perform specified operation on identified object	The current user is trying to remove a privilege from a non-existing user.
42502*	Authorization ID not permitted to perform operation as specified	The current user does not have an authenticated connection. When an application (which does not have the encryption library or the CryptoPlugin.dll) executes an encryption related SQL commands (GRANT, REVOKE and CREATE TABLE) an error of ″42502″ will be returned. This is to prevent applications from crashing.
42505*	Connection authorization failure occurred.	A registered user attempts to connect and cannot be authenticated.
42506*	Owner authorization failure.	The connected user could not be authenticated. (Wrong password.)
42601	Syntax error.	A syntax error in the SQL statement was detected.
42603	String constant does not have an ending delimiter.	A string constant or delimited identifier does not have an ending delimiter.
42610	Invalid use of a parameter marker.	The statement contains a parameter marker that is not valid. See Table 14 on page 72 for valid usage of parameter markers.
42611	Invalid length specification.	A length specification exceeds the limit.
42614	A duplicate keyword is invalid.	A duplicate keyword is invalid.
42621	The check constraint is invalid.	The check constraint is invalid.
42622	Name is too long.	The name of an identifier is too long.
42702	Ambiguous column name reference.	There is more than one possible column being referenced.
42703	Undefined column name.	A column name is not in the referenced tables.
42704	Undefined object.	The table does not exist.
42710	Named object already exists.	A table with the same name already exists.

Table 26. SQLSTATE messages reported by SQL (continued)

SQLSTATE	Description	Explanation
42711	Duplicated column name.	The same column name is specified more than once.
42802	Number of values does not match the number of columns.	The number of values assigned is not the same as the number of columns specified or implied.
42803	Column reference in SELECT list is not specified in the GROUP BY clause.	A column name and an aggregation function are contained in the select list, but there is no GROUP BY clause.
42818	Incompatible data types of operands.	The data types of the operands of an operation are not compatible.
42820	Literal value out of range.	The specified numeric value is not in the acceptable range.
42821	Incompatible data types.	A value is not compatible with the data type of a target column.
42822	Invalid ORDER BY item.	The ORDER BY item is not in the select list.
42824	Invalid LIKE operand.	An operand of LIKE is not a string, or the first operand is not a column.
42829	FOR UPDATE OF is invalid.	FOR UPDATE OF is invalid because the result table designated by the cursor cannot be modified.
42830	The foreign key does not conform to the description of the parent key.	The foreign key does not conform to the description of the parent key.
42831	Nullable columns in primary key.	A column specified in the primary key clause cannot be nullable.
42832*	Unauthorized access to system objects.	The operation is not allowed on system objects.
42884	Unknown function name.	No function or procedure was found with the specified name and compatible arguments.
42887	Unsupported feature	The feature is not supported in the current release.
42894	The DEFAULT value is invalid.	The DEFAULT value is invalid.
42902	Duplicate object table reference.	The object table of the INSERT statement is also identified in a FROM clause.
42903	A WHERE clause or SET clause includes an invalid reference.	A WHERE clause or SET clause includes an invalid reference, such as a column function.
42962	LOB column cannot be used as a key.	A LOB column cannot be used as the primary key.
54001	Statement too long.	The query statement is too long.
54008	Key is too long.	Too many columns in a primary key, foreign key, or index.
54010	Table record length is too long.	The record length of the table is too long.
55002	DB2ePLANTABLE not defined properly.	EXPLAIN cannot be executed with an incorrect declaration of DB2ePLANTABLE.
55009	File is read-only.	The file is read-only. In a read-only environment, only SELECT queries can be executed.
57001	Table not available.	REORG cannot be executed on a table that is under a transaction scope.
57011	Out of memory.	The system is not able to allocate dynamic memory.
57014	Processing was cancelled due to an interrupt.	The execution of a query is canceled due to user interruption.
58004	Internal system error (continue).	A non-severe system error occurred.

Table 26. SQLSTATE messages reported by SQL (continued)

SQLSTATE	Description	Explanation
58005	Internal system error (stop).	A severe system error occurred.

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "SQLState listing" on page 170
- "DB2 Everyplace supported parameter markers" on page 72
- "Summary of SQLState class codes" on page 170

SQLState messages reported by CLI

Table 27. SQLState messages reported by CLI

SQLSTATE	CLI function name	Description	Explanation
01000	SQLAllocHandle	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01000	SQLFreeHandle	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01002	SQLDisconnect	Disconnect error.	An error occurred during the disconnect. However, the disconnect succeeded. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	SQLDescribeCol	Data truncated.	The column name returned in the argument *ColumnName* was longer than the value specified in the argument *BufferLength*. The argument *NameLengthPtr* contains the length of the full column name. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	SQLFetch	Data truncated.	The data returned for one or more columns was truncated. String values or numeric values are right truncated. (SQL_SUCCESS_WITH_INFO is returned if no error occurred.)
01004	SQLGetData	Data truncated.	The data returned for the specified column (*ColumnNumber*) was truncated. String or numeric values are right truncated. (SQL_SUCCESS_WITH_INFO is returned.)
01S06*	SQLFetchScroll	Attempted to fetch before the result set returned the first row set.	The requested row set overlapped the start of the result set when the current position was beyond the first row, and either *FetchOrientation* was SQL_PRIOR, or *FetchOrientation* was SQL_RELATIVE with a negative F*etchOffset* whose absolute value was less than or equal to the current SQL_ATTR_ROW_ARRAY_SIZE. (Function returns SQL_SUCCESS_WITH_INFO.)
07005	SQLDescribeCol	The statement did not return a result set.	The statement associated with the *StatementHandle* did not return a result set. There were no columns to describe. (Call `SQLNumResultCols()` first to determine if there are any rows in the result set.)

Table 27. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
07006	SQLBindParameter	Invalid conversion.	The conversion from the data type identified by the *ValueType* argument to the data type identified by the *ParameterType* argument is not a meaningful conversion. (For example, conversion from SQL_C_DATE to SQL_DOUBLE.)
07006	SQLFetch	Invalid conversion.	The data type could not be converted in a meaningful manner to the data type specified by *fCType* in `SQLBindCol()` .
07006	SQLGetData	Invalid conversion.	The data type cannot be converted to the C data type specified by the argument *TargetType* . The function was called before for the same *ColumnNumber* value but with a different *TargetType* value.
07009	SQLBindCol	Invalid descriptor index.	The value specified for the argument *ColumnNumber* exceeded the maximum number of columns in the result set.
07009	SQLDescribeCol	Invalid descriptor index	The value specified for *ColumnNumber* was equal to or less than 0. The value specified for the argument *ColumnNumber* was greater than the number of columns in the result set.
08001	SQLConnect	Unable to connect to data source.	DB2 CLI was unable to establish a connection with the data source (server).
08002	SQLConnect	Connection in use.	The specified *ConnectionHandle* was already used to establish a connection with a data source and the connection is still open.
08003	SQLAllocHandle	Connection is closed.	The *HandleType* argument was SQL_HANDLE_STMT, but the connection specified by the *InputHandle* argument was not open. The connection process must be completed successfully (and the connection must be open) for DB2 CLI to allocate a statement handle.
08003	SQLDisconnect	Connection is closed.	The connection specified in the argument *ConnectionHandle* was not open.
08004	SQLConnect	The application server rejected establishment of the connection.	The data source (server) rejected the establishment of the connection.
08S01	SQLFreeHandle	Communication link failure.	The *HandleType* argument was SQL_HANDLE_DBC, and the communication link between DB2 CLI and the data source to which it was trying to connect failed before the function completed processing.
22002	SQLFetch	Invalid output or indicator buffer specified.	The pointer value specified for the argument *pcbValue* in `SQLBindCol()` was a null pointer and the value of the corresponding column is null. There is no means to report SQL_NULL_DATA.
22002	SQLGetData	Invalid output or indicator buffer specified.	The pointer value specified for the argument *StrLen_or_IndPtr* was a null pointer and the value of the column is null. There is no means to report SQL_NULL_DATA.

Table 27. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
22003	SQLExecDirect	Numeric value out of range.	A numeric value assigned to a numeric type column caused truncation of the whole part of the number, either at the time of assignment or in computing an intermediate result.
22005	SQLGetData	Error in assignment.	A returned value was incompatible with the data type denoted by the argument *TargetType*
39001 *	SQLExecute	A user-defined function returned an invalid SQLSTATE.	A user-defined function returned an invalid SQLSTATE.
40003 **08**S01	SQLBindCol	Communication link error.	The communication link between the application and data source failed before the function completed.
40003 **08**S01	SQLBindParameter	Communication link failure.	The communication link between the application and data source failed before the function completed.
40003 **08**S01	SQLDescribeCol	Communication link failure.	The communication link between the application and data source failed before the function completed.
40003 **08**S01	SQLFreeStmt	Communication link failure.	The communication link between the application and data source failed before the function completed.
40003 **08**S01	SQLGetData	Communication link failure.	The communication link between the application and data source failed before the function completed.
40003 **08**S01	SQLNumResultCols	Communication link failure.	The communication link between the application and data source failed before the function completed.
40003 **08**S01	SQLRowCount	Communication link failure.	The communication link between the application and data source failed before the function completed.
42nnn*	SQLPrepare	Syntax Error.	**42**nnn SQLSTATES indicate a variety of syntax or access problems with the statement. The characters nnn refer to any SQLSTATE with that class code. Example: **42**nnn refers to any SQLSTATE in the **42** class.
42xxx	SQLExecDirect	Syntax error or access rule violation.	**42**xxx SQLSTATES indicate a variety of syntax or access problems with the statement. xxx refers to any SQLSTATE with that class code. Example: **42**xxx refers to any SQLSTATE in the **42** class.
42xxx	SQLNumResultCols	Syntax Error.	**42**xxx SQLSTATES indicate a variety of syntax or access problems with the statement. xxx refers to any SQLSTATE with that class code. Example: **42**xxx refers to any SQLSTATE in the **42** class.
58004	SQLBindCol	Unexpected system failure.	Unrecoverable system error.
58004	SQLBindParameter	Unexpected system failure.	Unrecoverable system error.
58004	SQLConnect	Unexpected system failure.	Unrecoverable system error.

Table 27. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
58004	SQLDescribeCol	Unexpected system failure.	Unrecoverable system error.
58004	SQLDisconnect	Unexpected system failure.	Unrecoverable system error.
58004	SQLExecDirect	Unexpected system failure.	Unrecoverable system error.
58004	SQLFetch	Unexpected system failure.	Unrecoverable system error.
58004	SQLFreeStmt	Unexpected system failure.	Unrecoverable system error.
58004	SQLGetData	Unexpected system failure.	Unrecoverable system error.
58004	SQLPrepare	Unexpected system failure.	Unrecoverable system error.
58004	SQLNumResultCols	Unexpected system failure.	Unrecoverable system error.
58004	SQLRowCount	Unexpected system failure.	Unrecoverable system error.
59101*	SQLExecute	User not defined.	User is not defined in the Mobile Devices Administration Center control database.
59102*	SQLExecute	Incorrect password.	User password does not match the password defined in the Mobile Devices Administration Center.
59103*	SQLExecute	Group not defined.	Group is not defined in the Mobile Devices Administration Center.
59104*	SQLExecute	Application not defined.	Application is not defined in the Mobile Devices Administration Center.
59105*	SQLExecute	Subscription not defined.	Subscription with AgentAdapter is not defined in the Mobile Devices Administration Center.
59106*	SQLExecute	Subscription not complete.	The subscription does not have all the required information to invoke a remote stored procedure.
59120*	SQLExecute	XML conversion error.	AgentAdapter failed at converting user input data to XML document.
59121*	SQLExecute	General AgentAdapter error.	General AgentAdapter error.
59122*	SQLExecute	Loading library failed.	Some required libraries cannot be found on the system.
HY000	SQLAllocHandle	General error.	An error occurred for which there is no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY000	SQLFreeHandle	General error.	An error occurred for which there is no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	SQLAllocHandle	Memory allocation error.	DB2 CLI is unable to allocate memory for the specified handle.

Table 27. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
HY001	SQLBindCol	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLBindParameter	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLConnect	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLDescribeCol	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLDisconnect	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLExecDirect	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLFetch	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLFreeHandle	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLFreeStmt	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLGetData	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLPrepare	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLNumResultCols	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY001	SQLRowCount	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY002	SQLBindCol	Invalid column number.	The value specified for the argument *ColumnNumber* is less than 0. The value specified for the argument *ColumnNumber* exceeded the maximum number of columns supported by the data source.
HY002	SQLDescribeCol	Invalid column number.	The value specified for the argument *ColumnNumber* is less than 1. The value specified for the argument *ColumnNumber* is greater than the number of columns in the result set.
HY002	SQLGetData	Invalid column number.	The specified column is less than 0 or greater than the number of result columns.
HY003	SQLBindCol	Program type out of range.	*TargetType* is not a valid data type or SQL_C_DEFAULT.
HY003	SQLBindParameter	Program type out of range.	The value specified by the argument *ParameterNumber* is not a valid data type or SQL_C_DEFAULT.
HY003	SQLGetData	Program type out of range.	*TargetType* is not a valid data type or SQL_C_DEFAULT.
HY004	SQLBindParameter	SQL data type out of range.	The value specified for the argument *ParameterType* is not a valid SQL data type.

Table 27. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
HY009	SQLBindParameter	Invalid argument value.	The argument *ParameterValuePtr* is a null pointer, and the argument *StrLen_or_IndPtr* is a null pointer, and *InputOutputType* is not SQL_PARAM_OUTPUT.
HY009	SQLExecDirect	Invalid argument value.	*StatementText* is a null pointer.
HY009	SQLNumResultCols	Invalid argument value.	*StatementText* is a null pointer.
HY010	SQLDescribeCol	Function sequence error.	The function is called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle* .
HY010	SQLExecute	Function sequence error.	The specified *StatementHandle* is not in a prepared state. `SQLExecute()` is called without first calling `SQLPrepare()`.
HY010	SQLFetch	Function sequence error.	The function is called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle*.
HY010	SQLFreeHandle	Function sequence error.	The *HandleType* argument is SQL_HANDLE_ENV, and at least one connection is in an allocated or connected state. `SQLDisconnect()` and `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_DBC must be called for each connection before calling `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_ENV. The *HandleType* argument is SQL_HANDLE_DBC, and the function is called before calling `SQLDisconnect()` for the connection. The *HandleType* argument is SQL_HANDLE_STMT; `SQLExecute()` or `SQLExecDirect()` is called with the statement handle, and returned SQL_NEED_DATA. (DM) All subsidiary handles and other resources were not released before `SQLFreeHandle()` is called.
HY010	SQLGetData	Function sequence error.	The function is called without first calling `SQLFetch()`.
HY010	SQLNumResultCols	Function sequence error.	The function is called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle*.
HY010	SQLRowCount	Function sequence error.	The function is called prior to calling `SQLExecute()` or `SQLExecDirect()` for the *StatementHandle*.
HY013	SQLAllocHandle	Unexpected memory handling error.	The *HandleType* argument is SQL_HANDLE_DBC, or SQL_HANDLE_STMT; and the function call cannot be processed because the underlying memory objects cannot be accessed, possibly because of low memory conditions.
HY013	SQLBindCol	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY013	SQLBindParameter	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY013	SQLConnect	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY013	SQLDescribeCol	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY013	SQLDisconnect	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY013	SQLExecDirect	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.

Table 27. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
HY013	SQLFetch	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY013	SQLFreeHandle	Unexpected memory handling error.	The *HandleType* argument is SQL_HANDLE_STMT, and the function call cannot be processed because the underlying memory objects cannot be accessed, possibly because of low memory conditions.
HY013	SQLGetData	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY013	SQLNumResultCols	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY013	SQLNumResultCols	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY013	SQLRowCount	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY014	SQLAllocHandle	No more handles.	The limit for the number of handles that can be allocated for the type of handle indicated by the *HandleType* argument is reached.
HY014	SQLExecDirect	No more handles.	DB2 CLI is unable to allocate a handle due to internal resources.
HY014	SQLNumResultCols	No more handles.	DB2 CLI is unable to allocate a handle due to internal resources.
HY017	SQLFreeHandle	Invalid use of an automatically allocated descriptor handle.	The *Handle* argument is set to the handle for an automatically allocated descriptor or an implementation descriptor.
HY024	SQLSetStmtAttr	Invalid attribute value.	Given the specified *Attribute* value, an invalid value is specified in *ValuePtr*.
HY090	SQLBindCol	Invalid string or buffer length.	The value specified for the argument *BufferLength* is less than 1, and the argument *TargetType* is either SQL_C_CHAR, SQL_C_BINARY or SQL_C_DEFAULT.
HY090	SQLBindParameter	Invalid string or buffer length.	The value specified for the argument *BufferLength* is less than 0.
HY090	SQLDescribeCol	Invalid string or buffer length.	The length specified in argument *BufferLength* is less than 1.
HY090	SQLExecDirect	Invalid string or buffer length.	The argument *TextLength* is less than 1 but not equal to SQL_NTS.
HY090	SQLGetData	Invalid string or buffer length.	The value of the argument *BufferLength* is less than 0 and the argument *TargetType* is SQL_C_CHAR or SQL_C_BINARY; or *TargetType* is SQL_C_DEFAULT and the default type is one of SQL_C_CHAR, SQL_C_BINARY, or SQL_C_DBCHAR.
HY090	SQLNumResultCols	Invalid string or buffer length	The argument *TextLength* is less than 1, but not equal to SQL_NTS.
HY092	SQLAllocHandle	Option type out of range.	The *HandleType* argument is not: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT

Table 27. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
HY092	SQLFreeStmt	Option type out of range.	The value specified for the argument *Option* is not SQL_DROP or SQL_RESET_PARAMS.
HY093	SQLBindParameter	Invalid parameter number.	The value specified for the argument *ValueType* is less than 1 or greater than the maximum number of parameters supported by the server.
HY094	SQLBindParameter	Invalid scale value.	The value specified for *ParameterType* is either SQL_DECIMAL or SQL_NUMERIC, and the value specified for *DecimalDigits* is less than 0 or greater than the value for the argument *ParamDef* (precision).
HY104	SQLBindParameter	Invalid precision value.	The value specified for *ParameterType* is either SQL_DECIMAL or SQL_NUMERIC, and the value specified for *ParamDef* is less than 1.
HY105	SQLBindParameter	Invalid parameter type.	*InputOutputType* is not SQL_PARAM_INPUT.
HY106	SQLFetchScroll	Fetch type out of range.	The value specified for the argument *FetchOrientation* is not valid. The value of the SQL_CURSOR_TYPE statement attribute is SQL_CURSOR_FORWARD_ONLY and the value of argument *FetchOrientation* is not SQL_FETCH_NEXT.
HY107	SQLFetchScroll	Row value out of range.	The value specified with the SQL_ATTR_CURSOR_TYPE statement attribute is SQL_CURSOR_KEYSET_DRIVEN, but the value specified with the SQL_ATTR_KEYSET_SIZE statement attribute is greater than 0 and less than the value specified with the SQL_ATTR_ROW_ARRAY_SIZE statement attribute.
HY501	SQLConnect	Invalid *DataSource* name.	The specified *DataSource* name is not valid.
HYC00	SQLBindCol	Driver not capable.	DB2 CLI recognizes, but does not support the data type specified in the argument *TargetType* .
HYC00	SQLBindParameter	Driver not capable.	DB2 CLI or data source does not support the conversion specified by the combination of the value specified for the argument *ValueType* and the value specified for the argument *ParameterType*. The value specified for the argument *ParameterType* is not supported by either DB2 CLI or the data source.
HYC00	SQLDescribeCol	Driver not capable.	The SQL data type of column *ColumnNumber* is not recognized by DB2 CLI.
HYC00	SQLGetData	Driver not capable.	The SQL data type for the specified data type is recognized but not supported by DB2 CLI. The requested conversion from the SQL data type to the application data *TargetType* cannot be performed by DB2 CLI or the data source.

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129
- "Data type compatibility for assignments and comparisons" on page 166
- "SQLState listing" on page 170

- "DB2 Everyplace supported parameter markers" on page 72
- "Summary of SQLState class codes" on page 170

SQLState messages reported by JDBC

Table 28. SQLState messages reported by JDBC

SQLSTATE	Description	Explanation
0100C	One or more adhoc result sets were returned.	DB2 Everyplace does not support ResultSet.CONCUR_UPDATABLE for the concurrency mode of a *ResultSet* object. ResultSet.CONCUR_READ_ONLY is used instead.
0641E	There is a SELECT statement in the batch.	A SELECT statement is not allowed in the batch.
0643E	There is no statement in the batch.	The batch does not have any statement.
22005	Error in assignment.	A parameter type is incompatible with the target data type.
22011	A substring error occurred.	Invalid ordinal position for the first byte in the BLOB value to be extracted.
S1010	Function sequence error.	`CallableStatement` get method is called without first calling `registerOutParameter`.

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267

Supported DB2 CLI functions

This chapter describes the DB2 Call Level Interface (DB2 CLI) functions supported by DB2 Everyplace.

- "DB2 CLI function summary" provides a brief description of the purpose of each function and a brief summary of the differences between the DB2 CLI functions supported by DB2 Everyplace and the standard DB2 CLI functions.
- "Key to DB2 CLI function descriptions" on page 186 provides an explanation of the function descriptions for each CLI function.
- "Data conversion by DB2 CLI functions" on page 266 contains a table listing the supported data conversions between C and SQL data types

DB2 CLI function summary

Table 29 gives a summary of the DB2 CLI functions supported by DB2 Everyplace, including the purpose of each function and a summary of the differences between the DB2 CLI functions supported by DB2 Everyplace and the standard DB2 CLI functions.

Table 29. DB2 CLI function list

Function name	Purpose	Summary of differences
SQLAllocConnect	Obtains a connection handle.	
SQLAllocEnv	Obtains an environment handle.	
SQLAllocHandle	Obtains a handle.	

Table 29. DB2 CLI function list (continued)

Function name	Purpose	Summary of differences
SQLAllocStmt	Allocates a statement handle.	
SQLBindCol	Assigns storage for a result column and specifies the data type.	The target type is restricted to the supported data types. LOB locator is not supported.
SQLBindParameter	Assigns storage for a parameter in an SQL statement.	Does not support binding to arrays of application variables or LOB locators. Does not support SQLPutData(), so the application should put the value of the parameter in *ParameterValuePtr* before calling SQLExecute(). The parameter type is limited to only INPUT because stored procedures are not supported.
SQLColumns	Returns the list of column names in specified tables.	*CatalogName*, *NameLength1*, *SchemaName*, *NameLength2* are ignored. Columns 2, 12, and 15 of the returned result set are always NULL. The return code SQL_STILL_EXECUTING is not supported.
SQLConnect	Connects to a specific driver by data source name, user ID, and password.	
SQLDescribeCol	Describes a column in the result set.	The column information is limited by the supported column data types.
SQLDisconnect	Closes the connection.	
SQLEndTran	Requests a COMMIT or ROLLBACK for all operations on all statements associated with a connection.	Connection attribute SQL_ATTR_AUTOCOMMIT must be set to SQL_AUTOCOMMIT_OFF before calling SQLEndTran().
SQLError	Returns additional error or status information.	
SQLExecDirect	Executes a statement.	The return codes, SQL_STILL_EXECUTING and SQL_NEED_DATA, are not supported. Asynchronous CLI calls are not supported.
SQLExecute	Executes a prepared statement.	All parameters must be bound before calling SQLExecute(). Asynchronous execution of SQL calls is not supported.

Table 29. DB2 CLI function list (continued)

Function name	Purpose	Summary of differences
SQLFetch	Returns a result row.	The result is fetched one row at a time, not by row sets. Statement descriptors are not supported. The return code SQL_STILL_EXECUTING is not supported.
SQLFetchScroll	Returns a result row set.	The result is fetched by row sets. The return code SQL_STILL_EXECUTING is not supported.
SQLForeignKeys	Returns information about foreign keys for the specified table.	*PKCatalogName*, *NameLength1*, *PKSchemaName*, *NameLength2*, *FKCatalogName*, *NameLength4*, *FKSchemaName*, *NameLength5* are ignored. Columns 1, 2, 5, 6, 12, and 13 of the returned result set are always a zero length string. Columns 10, 11, and 14 of the returned result set are always zero. The return code SQL_STILL_EXECUTING is not supported.
SQLFreeConnect	Releases the connection handle.	
SQLFreeEnv	Releases the environment handle.	
SQLFreeHandle	Frees handle resources.	
SQLFreeStmt	Ends statement processing, discards pending results, and, optionally, frees all resources associated with the statement handle.	Only the SQL_DROP and SQL_RESET_PARAMS options are supported.
SQLGetConnectAttr	Returns the current setting of a connection attribute.	DB2 Everyplace supports a subset of connection attributes supported by DB2. DB2 Everyplace also supports some connection attributes not supported by DB2.
SQLGetCursorName	Returns the cursor name associated with a statement handle.	The internally generated cursor name always begin with CUR.
SQLGetData	Returns part or all of one column of one row of a result set.	The target type is restricted to the supported data types. LOB locator is not supported. The return code SQL_STILL_EXECUTING is not supported.

Table 29. DB2 CLI function list (continued)

Function name	Purpose	Summary of differences
SQLGetDiagRec	Gets multiple fields of diagnostic data.	Only diagnostic records associated with a statement handle or connection handle are supported. Only single diagnostic records are supported.
SQLGetInfo	Returns information about a specific driver and data source.	DB2 Everyplace supports a subset of the information types supported by DB2.
SQLGetStmtAttr	Returns the current setting of a statement attribute.	DB2 Everyplace supports a subset of statement attributes supported by DB2. DB2 Everyplace also supports some statement attributes not supported by DB2.
SQLNumParams	Returns the number of parameter markers in an SQL statement.	The return code SQL_STILL_EXECUTING is not supported.
SQLNumResultCols	Returns the number of columns in the result set.	
SQLPrepare	Prepares an SQL statement for later execution.	
SQLPrimaryKeys	Returns a list of column names that comprise the primary key for a table.	*CatalogName*, *NameLength1*, *SchemaName*, *NameLength2* are ignored. Columns 1, 2, and 6 of the returned result set are always a zero length string. The return code SQL_STILL_EXECUTING is not supported.
SQLRowCount	Returns the number of rows affected by an insert, update, or delete request.	
SQLSetConnectAttr	Sets options related to a connection.	DB2 Everyplace supports a subset of connection attributes supported by DB2. DB2 Everyplace also supports some connection attributes not supported by DB2.
SQLSetStmtAttr	Sets options related to a statement.	DB2 Everyplace supports a subset of statement attributes supported by DB2. DB2 Everyplace also supports some statement attributes not supported by DB2.
SQLTables	Returns the list of table names stored in a specific data source.	*CatalogName*, *NameLength1*, *SchemaName*, *NameLength2*, *TableType*, *NameLength4* are ignored. DB2 Everyplace only supports type "TABLE." The return code SQL_STILL_EXECUTING is not supported.

Related reference:

- "Data conversion by DB2 CLI functions" on page 266
- "Key to DB2 CLI function descriptions"

Key to DB2 CLI function descriptions

Each function description contains the following sections:

Purpose

This section gives a brief overview of what the function does. It also indicates if any functions should be called before and after calling the function being described.

Each function also has a table that indicates to which specification or standard the function conforms.

This table indicates support of the function. Some functions use a set of options that do not apply to all specifications or standards. Any significant differences are identified in the restrictions section for the function.

Syntax

This section contains the generic 'C' prototype. The generic prototype is used for all environments, including Windows.

All function arguments that are pointers are defined using the macro `FAR`, this macro is defined out (set to a blank) for all platforms except Windows. On Windows `FAR` is used to define pointer arguments as far pointers.

Arguments

This section lists each function argument, along with its data type, a description, and whether it is an input or output argument.

Some functions contain input or output arguments, which are known as *deferred* or *bound* arguments.

These arguments are pointers to buffers allocated by the application, and are associated with (or bound to) either a parameter in an SQL statement, or a column in a result set. The data areas specified by the function are accessed by DB2 CLI at a later time. These deferred data areas must still be valid at the time DB2 CLI accesses them.

Usage This section provides information about how to use the function and any special considerations. Possible error conditions are not discussed here, but are listed in the diagnostics section instead.

Return codes

This section lists all the possible function return codes. When SQL_ERROR or SQL_SUCCESS_WITH_INFO is returned, error information can be obtained by calling `SQLError()` or `SQLGetDiagRec()`.

Diagnostics

This section contains a table that lists the SQLSTATEs explicitly returned by DB2 CLI (SQLSTATEs generated by the DBMS might also be returned) and indicates the cause of the error. These values are obtained by calling `SQLError()` or `SQLGetDiagRec()` after the function returns an SQL_ERROR or SQL_SUCCESS_WITH_INFO.

Restrictions

This section indicates any differences or limitations between DB2 Everyplace CLI and ODBC that might affect an application.

See the *IBM DB2 Universal Database Call Level Interface Guide and Reference* for more information about DB2 CLI, including information about return codes, diagnostics, examples, setting up the CLI environment, and accessing the sample applications.

Related reference:

- "Data conversion by DB2 CLI functions" on page 266
- "DB2 CLI function summary" on page 182

SQLAllocConnect—Allocate connection handle

In ODBC Version 3, `SQLAllocConnect()` was deprecated and replaced with `SQLAllocHandle()`; see "SQLAllocHandle—Allocate handle" for more information.

Recommendation: Although this version of DB2 CLI continues to support `SQLAllocConnect()`, use `SQLAllocHandle()` in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLAllocConnect(henv, hdbc);
```

for example, would be rewritten using the new function as:

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

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLAllocEnv—Allocate environment handle

In ODBC Version 3, `SQLAllocEnv()` was deprecated and replaced with `SQLAllocHandle()`; see "SQLAllocHandle—Allocate handle" for more information.

Recommendation: Although this version of DB2 CLI continues to support `SQLAllocEnv()`, use `SQLAllocHandle()` in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLAllocEnv(henv);
```

for example, would be rewritten using the new function as:

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

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLAllocHandle—Allocate handle

Purpose:

Specification:	**DB2 CLI** 5.0	**ODBC** 3.0	**ISO CLI**

SQLAllocHandle() allocates environment, connection, or statement handles.

This function is a generic function for allocating handles that replaces the deprecated Version 2 functions SQLAllocConnect(), SQLAllocEnv(), and SQLAllocStmt().

Syntax:

```
SQLRETURN   SQLAllocHandle   (SQLSMALLINT       HandleType,
                              SQLHANDLE         InputHandle,
                              SQLHANDLE         *OutputHandlePtr);
```

Function arguments:

Table 30. SQLAllocHandle arguments

Data type	Argument	Use	Description
SQLSMALLINT	*HandleType*	input	The type of handle to be allocated by SQLAllocHandle(). Must be one of the following values: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT
SQLHANDLE	*InputHandle*	input	Existing handle to use as a context for the new handle being allocated. If *HandleType* is SQL_HANDLE_ENV, this is SQL_NULL_HANDLE. If *HandleType* is SQL_HANDLE_DBC, this must be an environment handle; and if it is SQL_HANDLE_STMT, it must be a connection handle.
SQLHANDLE	*OutputHandlePtr*	output	Pointer to a buffer in which to return the handle to the newly allocated data structure.

Usage:

SQLAllocHandle() is used to allocate environment, connection, and statement handles, as described below.

Multiple statement handles can be allocated by an application at one time.

If the application calls SQLAllocHandle() with **OutputHandlePtr* set to an environment, connection, statement, or descriptor handle that already exists, DB2 CLI overwrites the information associated with the handle. DB2 CLI does not check to see whether the handle entered in **OutputHandlePtr* is already in use, nor does it check the previous contents of a handle before overwriting them.

For DB2 Everyplace, all handles except the statement handle are dummy handles and do not carry usable information.

A statement handle provides access to statement information, such as error messages, and status information for SQL statement processing. To request a statement handle, an application connects to a data source, and then calls SQLAllocHandle() prior to submitting SQL statements. In this call, *HandleType* should be set to SQL_HANDLE_STMT and *InputHandle* should be set to the connection handle that was returned by the call to SQLAllocHandle() that allocated that handle. DB2 CLI allocates the statement handle, associates the statement handle with the connection specified, and passes the value of the associated handle

back in **OutputHandlePtr*. The application passes the **OutputHandlePtr* value in all subsequent calls that require a statement handle.

When an application exits, all DB2 Everyplace resources allocated for the application are released, so handles that the application uses are no longer valid.

For DB2 Everyplace, no descriptor is associated with a statement handle with attributes that can be changed by an application.

When using DB2 Everyplace with Visual Basic and the DB2 Everyplace CLI/ODBC interface, you must explicitly call the mapped/underlying CLI functions defined in sqlcli.h. For example, calling `SQLAllocHandle()` fails. Calling `SQLAllocHandleVer(SQL_HANDLE_STMT, hdbc, hstmt, DB2eVersion)` is successful.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

When allocating a handle other than an environment handle, if `SQLAllocHandle()` returns SQL_ERROR, it sets *OutputHandlePtr* to SQL_NULL_HENV, SQL_NULL_HDBC, or SQL_NULL_HSTMT, depending on the value of *HandleType*, unless the output argument is a null pointer. The application can then obtain additional information from the diagnostic data structure associated with the handle in the *InputHandle* argument.

Diagnostics:

Table 31. SQLAllocHandle SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection is closed.	The *HandleType* argument is SQL_HANDLE_STMT, but the connection specified by the *InputHandle* argument is not open. The connection process must be completed successfully (and the connection must be open) for DB2 CLI to allocate a statement handle.
HY000	General error.	An error occurred for which there is no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation error.	DB2 CLI is unable to allocate memory for the specified handle.
HY013	Unexpected memory handling error.	The *HandleType* argument is SQL_HANDLE_DBC, or SQL_HANDLE_STMT; and the function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY014	No more handles.	The limit for the number of handles that can be allocated for the type of handle indicated by the *HandleType* argument is reached.

Table 31. SQLAllocHandle SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY092	Option type out of range.	The *HandleType* argument is not: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLExecDirect—Execute a statement directly" on page 210
- "SQLFreeHandle—Free handle resources" on page 226

SQLAllocStmt—Allocate a statement handle

In ODBC Version 3, `SQLAllocStmt()` was deprecated and replaced with `SQLAllocHandle()`; see "SQLAllocHandle—Allocate handle" on page 187 for more information.

Recommendation: Although this version of DB2 CLI continues to support `SQLAllocStmt()`, use `SQLAllocHandle()` in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLAllocStmt(hdbc, hstmt);
```

for example, would be rewritten using the new function as:

```
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, hstmt);
```

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLBindCol—Bind a column to an application variable

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLBindCol()` is used to associate (bind) columns in a result set to application variables, for all C data types. Data is transferred from the DBMS to the application when `SQLFetch()` is called. Data conversion might occur when the data is transferred.

`SQLBindCol()` is called once for each column in the result set that the application needs to retrieve.

In general, `SQLPrepare()` or `SQLExecDirect()` is called before this function, and `SQLFetch()` is called after. Column attributes might also be needed before calling `SQLBindCol()`, and can be obtained using `SQLDescribeCol()`.

Syntax:

```
SQLRETURN   SQLBindCol       (SQLHSTMT          StatementHandle,   /* hstmt */
                             SQLUSMALLINT       ColumnNumber,      /* icol */
                             SQLSMALLINT        TargetType,        /* fCType */
                             SQLPOINTER         TargetValuePtr,    /* rgbValue */
                             SQLINTEGER         BufferLength,      /* cbValueMax */
                             SQLINTEGER   *FAR StrLen_or_IndPtr); /* pcbValue */
```

Function arguments:

Table 32. SQLBindCol arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLUSMALLINT	*ColumnNumber*	input	Number identifying the column. Columns are numbered sequentially, from left to right. Column numbers start at 1.
SQLSMALLINT	*TargetType*	input	The C data type for column number *ColumnNumber* in the result set. The following types are supported: SQL_C_BINARY SQL_C_BIT SQL_C_CHAR SQL_C_DOUBLE SQL_C_FLOAT SQL_C_LONG SQL_C_SHORT SQL_C_TYPE_DATE SQL_C_TYPE_TIME SQL_C_TYPE_TIMESTAMP SQL_C_TINYINT Specifying SQL_C_DEFAULT causes data to be transferred to its default C data type.
SQLPOINTER	*TargetValuePtr*	input/output (deferred)	Pointer to the buffer where DB2 CLI is to store the column data when the fetch occurs. If *TargetValuePtr* is null, the column is unbound.
SQLINTEGER	*BufferLength*	input	Size of *TargetValuePtr* buffer in bytes available to store the column data. If *TargetType* denotes a binary or character string or is SQL_C_DEFAULT, then *BufferLength* must be > 0, or an error returns. Otherwise, this argument is ignored.
SQLINTEGER *	*StrLen_or_IndPtr*	input/output (deferred)	Pointer to value that indicates the number of bytes that DB2 CLI has available to return in the *TargetValuePtr* buffer. `SQLFetch()` returns SQL_NULL_DATA in this argument if the data value of the column is null. SQL_NO_LENGTH might also be returned. Refer to the usage section for more information.

For this function, both *TargetValuePtr* and *StrLen_or_Ind* are deferred outputs, meaning that the storage locations these pointers point to do not get updated until a result set row is fetched. As a result, the locations referenced by these pointers must remain valid until `SQLFetch()` is called. For example, if `SQLBindCol()` is called within a local function, `SQLFetch()` must be called from within the same scope of the function or the *TargetValuePtr* buffer must be allocated or declared as static or global.

Usage:

The application calls `SQLBindCol()` one time for each column in the result set for which it wants to retrieve the data. Result sets are generated either by calling `SQLExecute()` or `SQLExecDirect()`. When `SQLFetch()` is called, the data in each of these *bound* columns is placed into the assigned location (given by the pointers *TargetValuePtr* and *StrLen_or_Ind*).

Columns are identified by a number, assigned sequentially from left to right. Column numbers start at one.

The number of columns in the result set can be determined by calling `SQLNumResultCols()`.

The application can query the attributes (such as data type and length) of the column by first calling `SQLDescribeCol()`. 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.

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 are fetched for the current row.

In subsequent fetches, the application can change the binding of these columns or bind previously unbound columns by calling `SQLBindCol()`. The new binding does not apply to data already fetched, it is used on the next fetch. To unbind a single column, call `SQLBindCol()` with the *TargetValuePtr* pointer set to NULL. To unbind all the columns, the application should call `SQLFreeStmt()`.

The application must ensure that enough storage is allocated for the data to be retrieved. If the buffer is to contain variable length data, the application must allocate as much storage as the maximum length of the bound column requires; otherwise, the data might be truncated. If the buffer is to contain fixed length data, DB2 CLI assumes that the size of the buffer is the length of the C data type. If data conversion is specified, the required size might be affected.

If string truncation occurs, SQL_SUCCESS_WITH_INFO is returned and *StrLen_or_IndPtr* is set to the actual size of *TargetValuePtr* available for return to the application.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 33. SQLBindCol SQLSTATEs

SQLSTATE	Description	Explanation
07009	Invalid descriptor index.	The value specified for the argument *ColumnNumber* exceeded the maximum number of columns in the result set.

Table 33. SQLBindCol SQLSTATEs (continued)

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link error.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY002	Invalid column number.	The value specified for the argument *ColumnNumber* is less than 0. The value specified for the argument *ColumnNumber* exceeded the maximum number of columns supported by the data source.
HY003	Program type out of range.	*TargetType* is not a valid data type or SQL_C_DEFAULT.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* is less than one, and the argument *TargetType* is either SQL_C_CHAR, SQL_C_BINARY, or SQL_C_DEFAULT.
HYC00	Driver not capable.	DB2 CLI recognizes, but does not support the data type specified in the argument *TargetType*.

Additional diagnostic messages relating to the bound columns might be reported at fetch time.

Restrictions:

Output buffers need to be word-aligned (even). Many processors such as the Motorola 68000 have word-alignment rules, and for non-character data types, the application should align the buffer properly.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLBindParameter—Bind a parameter marker to a buffer

Purpose:

Specification:	DB2 CLI 2.1	ODBC 2.0	

`SQLBindParameter()` is used to associate (bind) parameter markers in an SQL statement to application variables, for all C data types. In this case data is transferred from the application to the DBMS when `SQLExecute()` or `SQLExecDirect()` is called. Data conversion might occur when the data is transferred.

Syntax:

SQLBindParameter

```
SQLRETURN  SQL_API SQLBindParameter(
                   SQLHSTMT         StatementHandle,  /* hstmt */
                   SQLUSMALLINT     ParameterNumber,  /* ipar */
                   SQLSMALLINT      InputOutputType,  /* fParamType */
                   SQLSMALLINT      ValueType,        /* fCType */
                   SQLSMALLINT      ParameterType,    /* fSqlType */
                   SQLUINTEGER      ColumnSize,       /* cbColDef */
                   SQLSMALLINT      DecimalDigits,    /* ibScale */
                   SQLPOINTER       ParameterValuePtr,/* rgbValue */
                   SQLINTEGER       BufferLength,     /* cbValueMax */
                   SQLINTEGER *FAR  StrLen_or_IndPtr);/* pcbValue */
```

Function arguments:

Table 34. SQLBindParameter arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLUSMALLINT	*ParameterNumber*	input	Parameter marker number, ordered sequentially left to right, starting at one.
SQLSMALLINT	*InputOutputType*	input	The type of parameter. The supported type is: • SQL_PARAM_INPUT: When the statement is executed, the actual data value for the parameter is sent to the server; the *ParameterValuePtr* buffer must contain valid input data values and the *StrLen_or_IndPtr* buffer must contain the corresponding length value or SQL_NTS, or SQL_NULL_DATA. DB2 Everyplace does not support `SQLPutData()`, so you should not store the parameter value in the *ParameterValuePtr* buffer. • SQL_PARAM_INPUT_OUTPUT: The parameter marker is associated with an input/output parameter of the called stored procedure. When the statement is executed, actual data values for the parameter are sent to the server. The *ParameterValuePtr* buffer must contain valid input data values; the *StrLen_or_IndPtr* buffer must contain the corresponding length value or SQL_NTS, SQL_NULL_DATA. • SQL_PARAM_OUTPUT: The parameter marker is associated with an output parameter of the called stored procedure or the return value of the stored procedure. After the statement is executed, data for the output parameter is returned to the application buffer specified by *ParameterValuePtr* and *StrLen_or_IndPtr*, unless both are NULL pointers, in which case the output data is discarded. If an output parameter does not have a return value then *StrLen_or_IndPtr* is set to SQL_NULL_DATA.
SQLSMALLINT	*ValueType*	input	C data type of the parameter. The following types are supported: • SQL_C_BINARY • SQL_C_BIT • SQL_C_CHAR • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_SHORT • SQL_C_TYPE_DATE • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TINYINT Specifying SQL_C_DEFAULT causes data to be transferred from its default C data type to the type indicated in *ParameterType*.
SQLSMALLINT	*ParameterType*	input	SQL data type of the parameter. The supported types are: • SQL_BLOB • SQL_CHAR • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARCHAR

Table 34. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLUINTEGER	*ColumnSize*	input	Precision of the corresponding parameter marker. • If *ParameterType* denotes a binary or single-byte character string (such as SQL_CHAR, SQL_BLOB), this is the maximum length in bytes for this parameter marker. • If not, this argument is ignored.
SQLSMALLINT	*DecimalDigits*	input	Scale of the corresponding parameter if *ParameterType* is SQL_DECIMAL.
SQLPOINTER	*ParameterValuePtr*	input (deferred), output (deferred), or both	• On input (*InputOutputType* set to SQL_PARAM_INPUT or SQL_PARAM_INPUT_OUTPUT): At execution time, if *StrLen_or_IndPtr* does not contain SQL_NULL_DATA, *ParameterValuePtr* points to a buffer that contains the actual data for the parameter. • On output (InputOutputType set to SQL_PARAM_OUTPUT or SQL_PARAM_INPUT_OUTPUT): *ParameterValuePtr* points to the buffer where the output parameter value of the stored procedure is stored. • A null *ParameterValuePtr* indicates unbinding the parameter.
SQLINTEGER	*BufferLength*	input	For character and binary data, *BufferLength* specifies the length of the *ParameterValuePtr* buffer. For non-character and non-binary data, this argument is ignored and the length of the *ParameterValuePtr* buffer is assumed to be the length associated with the C data type. For output parameters, *BufferLength* is used to determine whether to truncate data.
SQLINTEGER *	*StrLen_or_IndPtr*	input (deferred), output (deferred), or both	• If this is an input or input/output parameter: This is the pointer to the location that contains (when the statement is executed) the length of the parameter marker value stored at *ParameterValuePtr*. To specify a null value for a parameter marker, this storage location must contain SQL_NULL_DATA. If *ValueType* is SQL_C_CHAR, this storage location must contain either the exact length of the data stored at *ParameterValuePtr*, or SQL_NTS if the contents at *ParameterValuePtr* is null-terminated. If it contains the exact length, no null character is allowed in the data stored at ParameterValuePtr. If *ValueType* indicates character data (explicitly, or implicitly using SQL_C_DEFAULT), and this pointer is set to NULL, the application must provide a null-terminated string in *ParameterValuePtr*. This also implies that this parameter marker never has a null value. • If this is an output parameter (*InputOutputType* is set to SQL_PARAM_OUTPUT): This must be an output parameter or return value of a stored procedure CALL and points to one of the following, after the execution of the stored procedure: – Number of bytes available to return in *ParameterValuePtr*, excluding the null-termination character. – SQL_NULL_DATA

Usage:

A parameter marker is represented by a ? character in an SQL statement and is used to indicate a position in the statement where an application-supplied value is to be substituted when the statement is executed. This value can be obtained from an application variable. `SQLBindParameter()` is used to bind the application storage area to the parameter marker.

The application must bind a variable to each parameter marker in the SQL statement before executing the SQL statement. For this function, *ParameterValuePtr* and *StrLen_or_IndPtr* are deferred arguments. The storage locations must be valid and contain input data values when the statement is executed. This means that either the `SQLExecDirect()` or `SQLExecute()` call must be kept in the same procedure scope as the `SQLBindParameter()` calls, or these storage locations must be dynamically allocated or declared statically or globally.

Parameter markers are referenced by number (*ColumnNumber*) and are numbered sequentially from left to right, starting at one.

All parameters bound by this function remain in effect until one of the following functions is called:

- `SQLFreeStmt()` is called with the SQL_RESET_PARAMS option.
- `SQLFreeHandle()` is called with *HandleType* set to SQL_HANDLE_STMT.
- `SQLBindParameter()` is called again for the same parameter *ParameterNumber* number.

After the SQL statement is executed and the results processed, the application might want 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()` must be called with SQL_RESET_PARAMS to reset or clear the parameter bindings.

The C buffer data type given by *ValueType* must be compatible with the SQL data type indicated by *ParameterType*, or an error occurs.

Because the data in the variables referenced by *ParameterValuePtr* and *StrLen_or_IndPtr* is not verified until the statement is executed, data content or format errors are not detected or reported until `SQLExecute()` or `SQLExecDirect()` is called.

For this function, *ParameterValuePtr* and *StrLen_or_IndPtr* are deferred arguments. In the case where *InputOutputType* is set to SQL_PARAM_INPUT, the storage locations must be valid and contain input data values when the statement is executed. This means that either the `SQLExecDirect()` or `SQLExecute()` call must be kept in the same procedure scope as the `SQLBindParameter()` calls, or these storage locations must be dynamically allocated or declared statically or globally.

DB2 Everyplace supports SQL_PARAM_INPUT, SQL_PARAM_INPUT_OUTPUT, and SQL_PARAM_OUTPUT. DB2 Everyplace does not support `SQLPutData()`, so you should not store the parameter value in the *ParameterValuePtr* buffer.

For character and binary C data, the *BufferLength* argument specifies the length of the *ParameterValuePtr* buffer. For all other types of C data, the *BufferLength* argument is ignored.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 35. SQLBindParameter SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The conversion from the data value identified by the *ValueType* argument to the data type identified by the *ParameterType* argument is not a meaningful conversion. (For example, conversion from SQL_C_DATE to SQL_DOUBLE.)
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.

Table 35. SQLBindParameter SQLSTATEs (continued)

SQLSTATE	Description	Explanation
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY003	Program type out of range.	The value specified by the argument *ParameterNumber* is not a valid data type or SQL_C_DEFAULT.
HY004	SQL data type out of range.	The value specified for the argument *ParameterType* is not a valid SQL data type.
HY009	Invalid argument value.	The argument *ParameterValuePtr* is a null pointer, and the argument *StrLen_or_IndPtr* is a null pointer, and *InputOutputType* is not SQL_PARAM_OUTPUT.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* is less than 0.
HY093	Invalid parameter number.	The value specified for the argument *ValueType* is less than one or greater than the maximum number of parameters supported by the server.
HY094	Invalid scale value.	The value specified for *ParameterType* is either SQL_DECIMAL or SQL_NUMERIC, and the value specified for *DecimalDigits* is less than 0 or greater than the value for the argument *ParamDef* (precision).
HY104	Invalid precision value.	The value specified for *ParameterType* is either SQL_DECIMAL or SQL_NUMERIC, and the value specified for *ParamDef* is less than one.
HY105	Invalid parameter type.	*InputOutputType* is not SQL_PARAM_INPUT.
HYC00	Driver not capable.	DB2 CLI or data source does not support the conversion specified by the combination of the value specified for the argument *ValueType* and the value specified for the argument *ParameterType*. The value specified for the argument *ParameterType* is not supported by either DB2 CLI or the data source.

Related reference:

- "Data type compatibility for assignments and comparisons" on page 166
- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLExecDirect—Execute a statement directly" on page 210
- "SQLExecute—Execute a statement" on page 212

SQLConnect—Connect to a data source

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

`SQLConnect()` establishes a connection to the target database.

A connection handle must be allocated using `SQLAllocHandle()` before calling this function.

This function must be called before allocating a statement handle using `SQLAllocHandle()`.

Syntax:

```
SQLRETURN   SQLConnect      (
                SQLHDBC          ConnectionHandle,  /* hdbc */
                SQLCHAR      *FAR ServerName,       /* szDSN */
                SQLSMALLINT      NameLength1,       /* cbDSN */
                SQLCHAR      *FAR UserName,         /* szUID */
                SQLSMALLINT      NameLength2,       /* cbUID */
                SQLCHAR      *FAR Authentication,   /* szAuthStr */
                SQLSMALLINT      NameLength3);      /* cbAuthStr */
```

Function arguments:

Table 36. SQLConnect arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle.
SQLCHAR *	*ServerName*	input	Location and name of the database. The name is optional. The name is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength1*	input	Length of contents of *ServerName* argument.
SQLCHAR *	*UserName*	input	Authorization-name (user identifier). This string is used with encryption; otherwise it is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength2*	input	Length of contents of *UserName* argument.
SQLCHAR *	*Authentication*	input	Authentication-string (password). This string is used with encryption; otherwise, it is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength3*	input	Length of contents of *Authentication* argument.

Notes:

A *non-registered* user (someone who doesn't exist in the DB2eSYSUSERS table) will receive the warning message, 42704 (object not defined), when attempting to connect to an encrypted DB2 Everyplace database during a call to the SQLGetDiagRec() CLI function. A *registered* user will not receive this warning. In contrast, both a non-registered and registered user is able to connect to the database during the SQLConnect() function call and will not receive a warning message.

Usage:

`SQLConnect()` can be used to connect to data sources in different locations.

To access a data source on the local device, the *ServerName* argument is set to a data source name. The data source name is ignored by DB2 Everyplace and the local data source is accessed.

For applications using secondary storage devices, the *ServerName* argument accepts a string pointing to the location of a *DataSource* that exists locally or on a supported secondary storage device such as the IBM Microdrive, Sony Memory Stick, Compact Flash, SD Memory Card, or MultiMediaCard. The *ServerName* string format is:

```
ServerName=Device:/Path/DataSource
```

Device This is the name of the device on which the *DataSource* is stored. The reserved character # is used to access any Compact Flash (CF) Type II storage device (on Palm OS devices with CF support). The secondary storage is addressed using the reserved characters #. #0 and #1 specify which secondary storage slot to access. # is equivalent to #0. For example:

```
ServerName=#:/storage/
```

DB2 Everyplace connects to the *DataSource* in the `storage` directory of the IBM Microdrive in the first CF slot.

Path This is the path to the *DataSource* on the *Device*. When *Path* is specified without a *Device*:/, the local file system path relative to the application location is used. Files should not be written to the root directory of a volume. Files in the root directory are not supported by some media types. For example:

```
ServerName=dir1/dir2/DATA1
```

Note: There is no path length limit in DB2 Everyplace.
If the application is located in `/myapp` on the local file system, DB2 Everyplace connects to the *DataSource* located in `/myapp/dir1/dir2/`. The *DataSource* name DATA1 is ignored.

DataSource
Optional: The name of the data source to connect to. This name is ignored by DB2 Everyplace.

To access a remote stored procedure using the Remote Query and Stored Procedure adapter, the *ServerName* argument is used to identify the location and name of the database. For applications using the Remote Query and Stored Procedure adapter to access remote databases, the *ServerName* argument accepts a URL format:

```
http://IPAddress:portNumber/path?DB=DataSource
```

IPAddress and *Authentication* are required.

If using Windows CE object storage rather then a conventional file system, either:

- set the path parameter of CLI function `SQLConnect` to `"@:\"`

or

- in CLP, execute `'connect to @:\'`

There is no ″directory″ concept in Windows CE object storage. When using object storage, the user cannot specify the directory, or path, in which tables are created. All tables in object storage are created in the same namespace. Because of this

limitation, multiple simultaneous connections to object storage cannot be established. The lock file for connection serialization purposes is created in the root path of the file system.

When using object storage, DB2 Everyplace files cannot be deleted manually, unlike when using a conventional file system.

Examples:

Connect to the data source locally at `c:\dir1\dir2\`. The data source name `DS1` is ignored:

```
ServerName=c:/dir1/dir2/DS1
```

Connect to the data source locally at `/dir1/dir2/` using UNIX file system notation:

```
ServerName=/dir1/dir2/
```

Connect to the data source locally in the `dir1\` directory relative to the application path. If the application is located in `c:\myapp\`, the `c:\myapp\dir1\` data source is accessed:

```
ServerName=dir1\
```

Connect to the data source in the `/dir1/` directory on the storage memory in secondary storage slot 1:

```
ServerName=#1:/dir1/
```

Connect to the DB2 Everyplace Sync Server `192.168.0.1` on port `8080` and database `mysample` using the remote query and stored procedure adapter.

```
ServerName=
 http://192.168.0.1:8080/db2e/servlet/com.ibm.mobileservices.adapter
                .agent.AgentServlet?DB=mysample
```

Connect to the data source using Windows CE storage.

```
ServerName=@:\
```

Connection Serialization:

See “Connection serialization” on page 63 for information about connection serialization.

Connection Authentication:

Database encryption requires rudimentary user authentication. DB2 Everyplace uses the UserName and Authentication to authenticate the user at connect time.

The authentication works as follows: If the DB2eSYSUSERS catalog table does not exist in the database that SQLConnect connects to, then the UserName and Authentication information is ignored. DB2 Everyplace distinguish between *registered* and *non-registered* users. A registered user is a user that is listed in the DB2eSYSUSERS table added through the GRANT SQL statement. At connect time, if there is a DB2eSYSUSERS table and the UserName belongs to a registered user, authentication is attempted. If the password given in the Authentication parameter is not correct, an error (42505) is returned. If the UserName is non-registered, then the SQLConnect function will succeed. However, a subsequent call to SQLGetDiagRec will return the warning 42704 (object not defined). This allows applications to distinguish between the case of a registered user successfully

connecting and a non-registered user who is successfully connected. For more information, see "Overview of local data encryption" on page 77, 334, and "GRANT" on page 147.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 37. SQLConnect SQLSTATEs

SQLSTATE	Description	Explanation
08001	Unable to connect to data source.	DB2 CLI is unable to establish a connection with the data source (server).
08002	Connection in use.	The specified *ConnectionHandle* has already been used to establish a connection with a data source and the connection is still open.
08004	The application server rejected establishment of the connection.	The data source (server) rejected the establishment of the connection.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY501	Invalid *DataSource* name.	The specified *DataSource* name is not valid.
HYT00	Connection timeout expired.	The timeout period expired before the application was able to connect to the data source. The timeout period can be set using the SQL_ATTR_LOGIN_TIMEOUT attribute for *SQLSetConnectAttr()*.This error is returned when the database is in use by another application.

Restrictions:

`SQLConnect()` must be called before any SQL statements can be executed.

Related concepts:

- "Connection serialization" on page 63
- "SQLAllocHandle—Allocate handle" on page 187
- "SQLDisconnect—Disconnect from a data source" on page 207

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLAllocHandle—Allocate handle" on page 187
- "SQLDisconnect—Disconnect from a data source" on page 207

SQLColumns - Get Column Information for a Table

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

SQLColumns() returns a list of columns in the specified tables. The information is returned in an SQL result set, which can be retrieved using the same functions that are used to fetch a result set generated by a query.

Syntax:

```
SQLRETURN   SQLColumns       (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR      FAR  *CatalogName,      /* szCatalogName */
               SQLSMALLINT       NameLength1,       /* cbCatalogName */
               SQLCHAR      FAR  *SchemaName,       /* szSchemaName */
               SQLSMALLINT       NameLength2,       /* cbSchemaName */
               SQLCHAR      FAR  *TableName,        /* szTableName */
               SQLSMALLINT       NameLength3,       /* cbTableName */
               SQLCHAR      FAR  *ColumnName,       /* szColumnName */
               SQLSMALLINT       NameLength4);      /* cbColumnName */
```

Function arguments:

Table 38. SQLColumns arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLCHAR	*CatalogName*	Input	Buffer that may contain a *pattern-value* to qualify the result set. *Catalog* is the first part of a 3 part table name. This argument is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength1*	Input	Length of *CatalogName*. This argument is ignored by DB2 Everyplace.
SQLCHAR	*SchemaName*	Input	Buffer that may contain a *pattern-value* to qualify the result set by schema name. This argument is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength2*	Input	Length of *SchemaName*. This argument is ignored by DB2 Everyplace.
SQLCHAR	*TableName*	Input	Buffer that may contain a *pattern-value* to qualify the result set by table name.
SQLSMALLINT	*NameLength3*	Input	Length of *TableName*.
SQLCHAR	*ColumnName*	Input	Buffer that may contain a *pattern-value* to qualify the result set by column name.
SQLSMALLINT	*NameLength4*	Input	Length of *ColumnName*.

Usage:

This function is called to retrieve information about the columns of either a table or a set of tables. A typical application may wish to call this function after a call to SQLTables() to determine the columns of a table. The application should use the character strings returned in the TABLE_NAME of the SQLTables() result set as input to this function.

SQLColumns() returns a standard result set, ordered by TABLE_NAME, and ORDINAL_POSITION. 203 lists the columns in the result set.

The *TableName*, and *ColumnName* arguments accept search patterns.

This function does not return information on the columns in a result set. `SQLDescribeCol()` or `SQLColAttribute()` should be used instead.

Calls to `SQLColumns()` should be used sparingly, because in many cases they map to a complex and thus expensive query against the system catalog. The results should be saved rather than repeating calls.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside 128 characters (plus the null-terminator) for the output buffer, or alternatively, call `SQLGetInfo()` with the SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN to determine respectively the actual lengths of the TABLE_NAME, and COLUMN_NAME columns supported by the connected DBMS.

Although new columns may be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

Columns Returned By SQLColumns:

Column 1 TABLE_CAT (VARCHAR(128))
This is always NULL.

Column 2 TABLE_SCHEM (VARCHAR(128))
This is always NULL.

Column 3 TABLE_NAME (VARCHAR(128) not NULL)
Name of the table.

Column 4 COLUMN_NAME (VARCHAR(128) not NULL)
Column identifier. Name of the column of the specified table, view, alias, or synonym.

Column 5 DATA_TYPE (SMALLINT not NULL)
SQL data type of column identified by COLUMN_NAME. This is one of the values in the Symbolic SQL Data Type column in "SQL symbolic and default data types" on page 167.

Column 6 TYPE_NAME (VARCHAR(128) not NULL)
Character string representing the name of the data type corresponding to DATA_TYPE.

Column 7 COLUMN_SIZE (INTEGER)

If the DATA_TYPE column value denotes a character or binary string, then this column contains the maximum length in characters for the column.

For DATE, TIME, or TIMESTAMP data types, this is the total number of characters required to display the value when converted to character.

For numeric data types, this is the total number of digits allowed in the column.

See also, "Data type attributes" on page 167.

Column 8 BUFFER_LENGTH (INTEGER)
The maximum number of bytes for the associated C buffer to store data from this column if SQL_C_DEFAULT were specified on the `SQLBindCol()`, `SQLGetData()` and `SQLBindParameter()` calls. This length does not include

any null-terminator. For exact numeric data types, the length accounts for the decimal and the sign. See also, "Data type attributes" on page 167

Column 9 DECIMAL_DIGITS (SMALLINT)
The scale of the column. NULL is returned for data types where scale is not applicable. See also, "Data type attributes" on page 167

Column 10 NUM_PREC_RADIX (SMALLINT)

Either 10 or NULL.

If DATA_TYPE is an exact numeric data type, this column contains the value 10 and the COLUMN_SIZE contains the number of decimal digits allowed for the column.

For numeric data types, the DBMS returns a NUM_PREC_RADIX of 10.

NULL is returned for data types where radix is not applicable.

Column 11 NULLABLE (SMALLINT not NULL)

SQL_NO_NULLS if the column does not accept NULL values.

SQL_NULLABLE if the column accepts NULL values.

Column 12 REMARKS (VARCHAR(254))
This is always NULL.

Column 13 COLUMN_DEF (VARCHAR(254))

The column's default value. If the default value is a numeric literal, then this column contains the character representation of the numeric literal with no enclosing single quotes. If the default value is a character string, then this column is that string enclosed in single quotes. If the default value a pseudo-literal, such as for DATE, TIME, and TIMESTAMP columns, then this column contains the keyword of the pseudo-literal (e.g. CURRENT DATE) with no enclosing quotes.

If NULL was specified as the default value, then this column returns the word NULL, not enclosed in quotes. If no default value was specified, then this column is NULL.

Column 14 SQL_DATA_TYPE (SMALLINT not NULL)
This column is the same as the DATA_TYPE column.

Column 15 SQL_DATETIME_SUB (SMALLINT)
This column is always NULL.

Column 16 CHAR_OCTET_LENGTH (INTEGER)
Contains the maximum length in octets for a character data type column. For Single Byte character sets, this is the same as COLUMN_SIZE. For all other data types it is NULL.

Column 17 ORDINAL_POSITION (INTEGER not NULL)
The ordinal position of the column in the table. The first column in the table is number 1.

Column 18 IS_NULLABLE (VARCHAR(254))

Contains the string 'NO' if the column is known to be not nullable; and 'YES' otherwise.

This result set is identical to the X/Open CLI *Columns()* result set specification, which is an extended version of the *SQLColumns()* result set specified in ODBC V2. The ODBC *SQLColumns()* result set includes every column in the same position.

Note: This result set is identical to the X/Open CLI `Columns()` result set specification, which is an extended version of the `SQLColumns()` result set specified in ODBC V2. The ODBC `SQLColumns()` result set includes every column in the same position.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 39. SQLColumns SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to internal resources.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal SQL_NTS.

Restrictions:

None.

Related reference:
- "SQLTables - Get Table Information" on page 263

SQLDescribeCol—Return a set of attributes for a column

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLDescribeCol()` returns a set of commonly used descriptor information (column name, type, precision, scale, nullability) for the indicated column in the result set generated by a query.

Either `SQLPrepare()` or `SQLExecDirect()` must be called before calling this function.

This function is usually called before a bind column function (`SQLBindCol()`) to determine the attributes of a column before binding it to an application variable.

Syntax:

```
SQLRETURN   SQLDescribeCol   (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLCHAR      *FAR ColumnName,        /* szColName */
                SQLSMALLINT       BufferLength,      /* cbColNameMax */
                SQLSMALLINT  *FAR NameLengthPtr,     /* pcbColName */
                SQLSMALLINT  *FAR DataTypePtr,       /* pfSqlType */
                SQLUINTEGER  *FAR ColumnSizePtr,     /* pcbColDef */
                SQLSMALLINT  *FAR DecimalDigitsPtr,  /* pibScale */
                SQLSMALLINT  *FAR NullablePtr);      /* pfNullable */
```

Function arguments:

Table 40. SQLDescribeCol arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLUSMALLINT	*ColumnNumber*	input	Column number to be described. Columns are numbered sequentially from left to right, starting at one.
SQLCHAR *	*ColumnName*	output	Pointer to column name buffer. This is set to NULL if the column name cannot be determined.
SQLSMALLINT	*BufferLength*	input	Size of *ColumnName* buffer.
SQLSMALLINT *	*NameLengthPtr*	output	Bytes available to return for *ColumnName* argument. Truncation of column name (*ColumnName*) to *BufferLength - 1* bytes occurs if *NameLengthPtr* is greater than or equal to *BufferLength*.
SQLSMALLINT *	*DataTypePtr*	output	Base SQL data type of column.
SQLUINTEGER *	*ColumnSizePtr*	output	Precision of column as defined in the database.
SQLSMALLINT *	*DecimalDigitsPtr*	output	Scale of column as defined in the database (applies only to SQL_DECIMAL).
SQLSMALLINT *	*NullablePtr*	output	Indicates whether NULLs are allowed for this column. Either: SQL_NO_NULLS SQL_NULLABLE

Usage:

Columns are identified by a number, are numbered sequentially from left to right, and might be described in any order. Column numbers start at one.

If a null pointer is specified for any of the pointer arguments, DB2 CLI assumes that the information is not needed by the application, and nothing is returned.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

If `SQLDescribeCol()` returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, one of the following SQLSTATEs might be obtained by calling the `SQLError()` function.

Table 41. SQLDescribeCol SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The column name returned in the argument *ColumnName* is longer than the value specified in the argument *BufferLength*. The argument *NameLengthPtr* contains the length of the full column name. (Function returns SQL_SUCCESS_WITH_INFO.)
07005	The statement did not return a result set.	The statement associated with the *StatementHandle* did not return a result set. There were no columns to describe. (Call `SQLNumResultCols()` first to determine if there are any rows in the result set.)

Table 41. SQLDescribeCol SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07009	Invalid descriptor index	The value specified for *ColumnNumber* is equal to or less than 0. The value specified for the argument *ColumnNumber* is greater than the number of columns in the result set.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY002	Invalid column number.	The value specified for the argument *ColumnNumber* is less than one, or the value specified for the argument *ColumnNumber* is greater than the number of columns in the result set.
HY090	Invalid string or buffer length.	The length specified in argument *BufferLength* is less than one.
HY010	Function sequence error.	The function is called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle*.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HYC00	Driver not capable.	The SQL data type of column *ColumnNumber* is not recognized by DB2 CLI.

Restrictions:

DB2 Everyplace supports only the following ODBC defined data types:

- SQL_BLOB
- SQL_CHAR
- SQL_DECIMAL
- SQL_INTEGER
- SQL_SMALLINT
- SQL_TYPE_DATE
- SQL_TYPE_TIME
- SQL_TYPE_TIMESTAMP
- SQL_VARCHAR

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLExecDirect—Execute a statement directly" on page 210
- "SQLNumResultCols—Get number of result columns" on page 247

SQLDisconnect—Disconnect from a data source

Purpose:

Specification:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

`SQLDisconnect()` closes the connection associated with the database connection handle.

After calling this function, either call `SQLConnect()` to connect to another database, or call `SQLFreeHandle()`.

Syntax:

```
SQLRETURN   SQLDisconnect    (SQLHDBC          ConnectionHandle;) /* hdbc */
```

Function arguments:

Table 42. SQLDisconnect arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle.

Usage:

If an application calls `SQLDisconnect()` before it frees all the statement handles associated with the connection, DB2 CLI frees them after it successfully disconnects from the database.

If SQL_SUCCESS_WITH_INFO is returned, it means that the disconnect from the database is successful, but additional error or implementation-specific information is available. For example, a problem is encountered during processing subsequent to disconnecting the connection, or if there is no current connection because of an event that occurred independently of the application (such as communication failure).

After a successful `SQLDisconnect()` call, the application can reuse *ConnectionHandle* to make another `SQLConnect()` or `SQLDriverConnect()` request.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 43. SQLDisconnect SQLSTATEs

SQLSTATE	Description	Explanation
01002	Disconnect error.	An error occurred during the disconnection. However, the disconnection succeeded. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection is closed.	The connection specified in the argument *ConnectionHandle* is not open.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLAllocHandle—Allocate handle" on page 187
- "SQLConnect—Connect to a data source" on page 198
- "SQLFreeHandle—Free handle resources" on page 226

SQLEndTran—Request a COMMIT or ROLLBACK

Purpose:

Specification:	DB2 CLI	ODBC	ISO CLI

`SQLEndTran()` requests a COMMIT or ROLLBACK operation for all active operations on all statements associated with a connection.

Syntax:

```
SQLRETURN   SQLEndTran      (SQLSMALLINT       HandleType,
                             SQLHANDLE         Handle,
                             SQLSMALLINT       Completion Type);
```

Function arguments:

Table 44. SQLEndTran arguments

Data type	Argument	Use	Description
SQLSMALLINT	*HandleType*	input	Handle Type.
SQLHANDLE	*Handle*	input	Connection handle.
SQLSMALLINT	*CompletionType*	input	How to complete the active operations associated with a connection.

Usage:

HandleType
Handle type identifier. Only SQL_HANDLE_DBC (a connection handle) is allowed.

Handle The handle, of the type indicated by *HandleType*.

CompletionType
One of the following two values:

- SQL_COMMIT
- SQL_ROLLBACK

In manual-commit mode, `SQLEndTran()` must be called before calling `SQLDisconnect()`. If `SQLEndTran()` is *not* called before `SQLDisconnect()`, the operations that updated the database (since the last transaction started) are rolled back.

When a ROLLBACK is performed, all the statement handles are cleared.

If the application crashes or terminates prematurely during use in manual mode, the updates since the last COMMIT are lost. `SQLEndTran()` must be called before calling disconnect.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 45. SQLEndTran SQLSTATEs

SQLSTATE	Description	Explanation
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI is unable to allocate a handle due to internal resources.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLSetConnectAttr—Set options related to a connection" on page 254

SQLError—Retrieve error information

In ODBC Version 3, `SQLError()` was deprecated and replaced with `SQLGetDiagRec()` and `SQLGetDiagField()`; see "SQLGetDiagRec—Get multiple fields settings of diagnostic record" on page 238 for more information.

Recommendation: Although this version of DB2 CLI continues to support `SQLError()`, use `SQLGetDiagRec()` in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

For example, to get diagnostic information associated with a particular statement handle, the statement:

```
SQLError(henv, hdbc, hstmt, szSqlState, pfNativeError, szErrorMsg,
    cbErrorMsgMax, pcbErrorMsg);
```

would be rewritten using the new function as:

```
SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, szSqlState, pfNativeError,
    szErrorMsg, cbErrorMsgMax, pcbErrorMsg);
```

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLExecDirect—Execute a statement directly

Purpose:

Specification:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

`SQLExecDirect()` directly executes the specified SQL statement. The statement can be executed only once.

Syntax:

```
SQLRETURN   SQLExecDirect    (SQLHSTMT         StatementHandle,   /* hstmt */
                              SQLCHAR      *FAR StatementText,     /* szSqlStr */
                              SQLINTEGER        TextLength);       /* cbSqlStr */
```

Function arguments:

Table 46. SQLExecDirect arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLCHAR *	*StatementText*	input	SQL statement string.
SQLINTEGER	*TextLength*	input	Length of the contents of *StatementText* argument. The length must be set to either the exact length of the statement, or if the statement is null-terminated, set to SQL_NTS.

Usage:

The SQL statement string cannot contain parameter markers.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL_NO_DATA_FOUND is returned if the SQL statement is a Searched UPDATE or Searched DELETE and no rows satisfy the search condition.

Diagnostics:

Table 47. SQLExecDirect SQLSTATEs

SQLSTATE	Description	Explanation
22003	Numeric value out of range.	A numeric value assigned to a numeric type column caused truncation of the whole part of the number, either at the time of assignment or in computing an intermediate result.
42xxx	Syntax error or access rule violation.	**42**xxx SQLSTATEs indicate a variety of syntax or access problems with the statement. xxx refers to any SQLSTATE with that class code. Example: **42**xxx refers to any SQLSTATE in the **42** class.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY009	Invalid argument value.	*StatementText* is a null pointer.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI is unable to allocate a handle due to internal resources.

Table 47. SQLExecDirect SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	The argument *TextLength* is less than one but not equal to SQL_NTS.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLBindCol—Bind a column to an application variable" on page 190

SQLExecute—Execute a statement

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLExecute()` executes a statement, that is successfully prepared using `SQLPrepare()`, one or multiple times. The statement is executed using the current value of any application variables that were bound to parameter markers by `SQLBindParameter()`.

Syntax:

```
SQLRETURN   SQLExecute      (SQLHSTMT          StatementHandle);  /* hstmt */
```

Function arguments:

Table 48. SQLExecute arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.

Usage:

The SQL statement string might contain parameter markers. A parameter marker is represented by a ? character, and is used to indicate a position in the statement where an application-supplied value is to be substituted when `SQLExecute()` is called. This value can be obtained from an application variable. `SQLBindParameter()` is used to bind the application storage area to the parameter marker.

All parameters must be bound before calling `SQLExecute()`.

After the application processes the results from the `SQLExecute()` call, it can execute the statement again with new (or the same) parameter values.

A statement executed by `SQLExecDirect()` cannot be re-executed by calling `SQLExecute()`; `SQLPrepare()` must be called first.

If a result set is generated, `SQLFetch()` retrieves the next row of data into bound variables. Data can also be retrieved by calling `SQLGetData()` for any column that is not bound.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL_NO_DATA_FOUND is returned if the SQL statement is a Searched UPDATE or Searched DELETE and no rows satisfy the search condition.

Diagnostics:

The SQLSTATEs for `SQLExecute()` include all those for `SQLExecDirect()` (refer to Table 47 on page 211), except for **HY**009 and **HY**090, and also include the SQLSTATEs in Table 49.

Table 49. SQLExecute SQLSTATEs

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The specified *StatementHandle* is not in the prepared state. `SQLExecute()` is called without first calling `SQLPrepare()`.
08004	The application server rejected the connection.	The user name or password used to connect to the data source is not correct.
08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
39001	A user-defined function has returned an invalid SQLSTATE.	A user-defined function returned an SQLSTATE that is not valid.
59101	User not defined.	User is not defined in the Mobile Devices Administration Center control database.
59102	Incorrect password.	User password does not match the password defined in the Mobile Devices Administration Center.
59103	Group not defined.	Group is not defined in the Mobile Devices Administration Center.
59104	Application not defined.	Application is not defined in the Mobile Devices Administration Center.
59105	Subscription not defined.	Subscription with "AgentAdapter" is not defined in the Mobile Devices Administration Center.
59106	Subscription not complete.	The subscription does not have all the required information to invoke a remote stored procedure.
59120	XML conversion error.	AgentAdapter failed at converting user input data to XML document.
59121	General AgentAdapter error.	General AgentAdapter error.

Table 49. SQLExecute SQLSTATEs (continued)

SQLSTATE	Description	Explanation
59122	Loading library failed.	Some required libraries can not be found on the system.
HY501	Invalid DataSource name.	The specified data source name is not valid.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLBindParameter—Bind a parameter marker to a buffer" on page 193
- "SQLBindCol—Bind a column to an application variable" on page 190
- "SQLExecDirect—Execute a statement directly" on page 210
- "SQLPrepare—Prepare a statement" on page 248
- "SQLFetch—Fetch next row"

SQLFetch—Fetch next row

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	

`SQLFetch()` advances the cursor to the next row of the result set and retrieves any bound columns.

Columns can be bound to application storage.

When `SQLFetch()` is called, the appropriate data transfer is performed, along with any data conversion if conversion is indicated when the column is bound. The columns can also be received individually after the fetch, by calling `SQLGetData()`.

`SQLFetch()` can be called only after a result set is generated (using the same statement handle) by executing a query.

Syntax:

```
SQLRETURN   SQLFetch         (SQLHSTMT          StatementHandle);  /* hstmt */
```

Function arguments:

Table 50. SQLFetch arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.

Usage:

SQLFetch() can be called only after a result set is generated on the same statement handle. Before SQLFetch() is called the first time, the cursor is positioned before the start of the result set.

The number of application variables bound with SQLBindCol() must not exceed the number of columns in the result set, or SQLFetch() fails.

If SQLBindCol() has not been called to bind any columns, then SQLFetch() does not return data to the application, but just advances the cursor. In this case SQLGetData() could be called to obtain all of the columns individually. Data in unbound columns is discarded when SQLFetch() advances the cursor to the next row.

Columns can be bound to application storage. SQLBindCol() is used to bind application storage to the column. Data is transferred from the database to the application at fetch time. The length of the available data to return is also set.

If any bound storage buffer is not large enough to hold the data returned by SQLFetch(), the data is truncated. If character data is truncated, SQL_SUCCESS_WITH_INFO is returned, and an SQLSTATE is generated indicating truncation. The SQLBindCol() deferred output argument *pcbValue* contains the actual length of the column data retrieved from the server. The application should compare the actual output length to the input buffer length (*pcbValue* and *cbValueMax* arguments from SQLBindCol()) to determine which character columns were truncated.

Truncation of numeric data types is reported as a warning if the truncation involves digits to the right of the decimal point. If truncation occurs to the left of the decimal point, an error is returned (refer to the diagnostics section).

When all the rows are retrieved from the result set, or the remaining rows are not needed, call SQLFreeStmt() to close the cursor and discard the remaining data and associated resources.

DB2 Everyplace fetches at most one row at a time, instead of using a row set. DB2 Everyplace does not support statement descriptors.

SQLFetch() determines whether the application specified separate length and indicator buffers. In this case, when the data is not NULL, SQLFetch() sets the indicator buffer to 0 and returns the length in the length buffer. When the data is NULL, SQLFetch() sets the indicator buffer to SQL_NULL_DATA and does not modify the length buffer.

Positioning the cursor:

When the result set is created, the cursor is positioned before the start of the result set. SQLFetch() fetches the next row.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL_NO_DATA_FOUND is returned if there are no rows in the result set, or previous SQLFetch() calls have fetched all the rows from the result set.

If all the rows have been fetched, the cursor is positioned after the end of the result set.

Diagnostics:

Table 51. SQLFetch SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The data returned for one or more columns is truncated. String values or numeric values are right truncated. (SQL_SUCCESS_WITH_INFO is returned if no error occurred.)
07006	Invalid conversion.	The data value could not be converted in a meaningful manner to the data type specified by *fCType* in SQLBindCol().
22002	Invalid output or indicator buffer specified.	The pointer value specified for the argument *pcbValue* in SQLBindCol() is a null pointer and the value of the corresponding column is null. There is no means to report SQL_NULL_DATA.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error.	The function is called prior to calling SQLPrepare() or SQLExecDirect() for the *StatementHandle*.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLBindCol—Bind a column to an application variable" on page 190
- "SQLExecDirect—Execute a statement directly" on page 210
- "SQLGetData—Get data from a column" on page 234

SQLFetchScroll—Fetch row set and return data for all bound columns

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	

SQLFetchScroll() fetches the specified row set of data from the result set and returns data for all bound columns. Row sets can be specified at an absolute or relative position.

Syntax:

```
SQLRETURN   SQLFetchScroll  (
                  SQLHSTMT          StatementHandle,
                  SQLSMALLINT       FetchOrientation,
                  SQLINTEGER        FetchOffset);
```

Function arguments:

Table 52. SQLFetchScroll arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLSMALLINT	*FetchOrientation*	input	Type of fetch: • SQL_FETCH_NEXT • SQL_FETCH_PRIOR • SQL_FETCH_FIRST • SQL_FETCH_LAST • SQL_FETCH_ABSOLUTE • SQL_FETCH_RELATIVE
SQLINTEGER	*FetchOffset*	input	Number of the row to fetch. The interpretation of this argument depends on the value of the *FetchOrientation* argument.

Usage:

`SQLFetchScroll()` returns a specified row set from the result set. Row sets can be specified by absolute or relative position. `SQLFetchScroll()` can be called only while a result set exists—that is, after a call that creates a result set and before the cursor over that result set is closed. If any columns are bound, it returns the data in those columns. If the application has specified a pointer to a row status array or a buffer in which to return the number of rows fetched, `SQLFetchScroll()` returns this information as well. Calls to `SQLFetchScroll()` can be mixed with calls to `SQLFetch()`.

Positioning the cursor:

When the result set is created, the cursor is positioned before the start of the result set. `SQLFetchScroll()` positions the block cursor based on the values of the *FetchOrientation* and *FetchOffset* arguments as shown in the following list. The exact rules for determining the start of the new row set are shown in the next section.

FetchOrientation	Meaning
SQL_FETCH_NEXT	Return the next row set. This is equivalent to calling `SQLFetch()`. `SQLFetchScroll()` ignores the value of *FetchOffset*.
SQL_FETCH_PRIOR	Return the prior row set. `SQLFetchScroll()` ignores the value of *FetchOffset*.
SQL_FETCH_RELATIVE	Return the row set *FetchOffset* from the start of the current row set.
SQL_FETCH_ABSOLUTE	Return the row set starting at row *FetchOffset*.
SQL_FETCH_FIRST	Return the first row set in the result set. `SQLFetchScroll()` ignores the value of *FetchOffset*.
SQL_FETCH_LAST	Return the last complete row set in the result set. `SQLFetchScroll()` ignores the value of *FetchOffset*.

The SQL_ATTR_ROW_ARRAY_SIZE statement attribute specifies the number of rows in the row set. If the row set being fetched by `SQLFetchScroll()` overlaps the end of the result set, `SQLFetchScroll()` returns a partial row set. That is, if S + R - 1 is greater than L, where S is the starting row of the row set being fetched, R is

the row set size, and L is the last row in the result set, then only the first L - S + 1 rows of the row set are valid. The remaining rows are empty and have a status of SQL_ROW_NOROW.

After `SQLFetchScroll()` returns, the row set cursor is positioned on the first row of the result set.

Cursor positioning rules:

The following sections describe the exact rules for each value of *FetchOrientation*. These rules use the following notation:

Before start
The block cursor is positioned before the start of the result set. If the first row of the new row set is before the start of the result set, `SQLFetchScroll()` returns SQL_NO_DATA.

After end
The block cursor is positioned after the end of the result set. If the first row of the new row set is after the end of the result set, `SQLFetchScroll()` returns SQL_NO_DATA.

CurrRowsetStart
This is the number of the first row in the current row set.

LastResultRow
This is the number of the last row in the result set.

RowsetSize
This is the row set size.

FetchOffset
This is the value of the *FetchOffset* argument.

SQL_FETCH_NEXT rules:

Table 53. SQL_FETCH_NEXT rules:

Condition	First row of new row set
Before start	1
CurrRowsetStart + RowsetSize <= LastResultRow	CurrRowsetStart + RowsetSize
CurrRowsetStart + RowsetSize > LastResultRow	After end
After end	After end

SQL_FETCH_PRIOR rules:

Table 54. SQL_FETCH_PRIOR rules:

Condition	First row of new row set
Before start	Before start
CurrRowsetStart = 1	Before start
1 < CurrRowsetStart <= RowsetSize	1[a]
CurrRowsetStart > RowsetSize	CurrRowsetStart - RowsetSize
After end AND LastResultRow < RowsetSize	1[a]
After end AND LastResultRow >= RowsetSize	LastResult - RowRowsetSize + 1

a SQLFetchScroll() returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first row set) and SQL_SUCCESS_WITH_INFO.

SQL_FETCH_RELATIVE rules:

Table 55. SQL_FETCH_RELATIVE rules:

Condition	First row of new row set
(Before start AND *FetchOffset* > 0) OR (After end AND *FetchOffset* < 0)	--[a]
Before start AND *FetchOffset* <= 0	Before start
CurrRowsetStart = 1 AND *FetchOffset* < 0	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| > RowsetSize	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| <= RowsetSize	1[b]
1 <= CurrRowsetStart + *FetchOffset* <= LastResultRow	CurrRowsetStart + *FetchOffset*
CurrRowsetStart + *FetchOffset* > LastResultRow	After end
After end AND *FetchOffset* >= 0	After end

a SQLFetchScroll() returns the same row set as if it is called with *FetchOrientation* set to SQL_FETCH_ABSOLUTE. For more information, see the SQL_FETCH_ABSOLUTE section.

b SQLFetchScroll() returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first row set.) and SQL_SUCCESS_WITH_INFO.

SQL_FETCH_ABSOLUTE rules:

Table 56. SQL_FETCH_ABSOLUTE rules:

Condition	First row of new row set
FetchOffset < 0 AND |*FetchOffset*| <= LastResultRow	LastResultRow + *FetchOffset* + 1
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow AND |*FetchOffset*| > RowsetSize	Before start
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow AND |*FetchOffset*| <= RowsetSize	1[a]
FetchOffset = 0	Before start
1 <= *FetchOffset* <= LastResultRow	*FetchOffset*
FetchOffset > LastResultRow	After end

a SQLFetchScroll() returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first row set.) and SQL_SUCCESS_WITH_INFO.

SQL_FETCH_FIRST rules::

Table 57. SQL_FETCH_FIRST rules:

Condition	First row of new row set
Any	1

SQL_FETCH_LAST rules:

Table 58. SQL_FETCH_LAST rules:

Condition	First row of new row set
RowsetSize <= LastResultRow	LastResultRow - RowsetSize + 1
RowsetSize > LastResultRow	1

Returning data in bound columns:

`SQLFetchScroll()` returns data in bound columns in the same way as `SQLFetch()`. For more information see "SQLFetch—Fetch next row" on page 214.

If no columns are bound, `SQLFetchScroll()` does not return data but does move the block cursor to the specified position. As with `SQLFetch()`, you can use `SQLGetData()` to retrieve the information in this case.

Buffer addresses:

`SQLFetchScroll()` uses the same formula to determine the address of data and length/indicator buffers as `SQLFetch()`. For more information, see "SQLBindCol—Bind a column to an application variable" on page 190.

Row status array:

The row status array is used to return the status of each row in the row set. The address of this array is specified with the SQL_ATTR_ROW_STATUS_PTR statement attribute. The array is allocated by the application and must have as many elements as are specified by the SQL_ATTR_ROW_ARRAY_SIZE statement attribute. Its values are set by `SQLFetch()` and `SQLFetchScroll()`. If the value of the SQL_ATTR_ROW_STATUS_PTR statement attribute is a null pointer, these functions do not return the row status.

The contents of the row status array buffer are undefined if `SQLFetch()` or `SQLFetchScroll()` does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO.

The following values are returned in the row status array.

Row status array value	Description
SQL_ROW_SUCCESS	The row is successfully fetched.
SQL_ROW_SUCCESS_WITH_INFO	The row is successfully fetched. However, a warning is returned about the row.
SQL_ROW_ERROR	An error occurred while fetching the row.
SQL_ROW_NOROW	The row set overlapped the end of the result set, and no row returns that corresponds to this element of the row status array.

Rows fetched buffer:

The rows fetched buffer is used to return the number of rows fetched, including those rows for which no data returns because an error occurred while they were being fetched. It is the number of rows for which the value in the row status array is not SQL_ROW_NOROW. The address of this buffer is specified with the SQL_ATTR_ROWS_FETCHED_PTR statement attribute. The buffer is allocated by

the application. It is set by SQLFetch() and SQLFetchScroll(). If the value of the SQL_ATTR_ROWS_FETCHED_PTR statement attribute is a null pointer, these functions do not return the number of rows fetched. To determine the number of the current row in the result set, an application can call SQLGetStmtAttr() with the SQL_ATTR_ROW_NUMBER attribute.

The contents of the rows fetched buffer are undefined if SQLFetch() or SQLFetchScroll() does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, except when SQL_NO_DATA is returned, in which case the value in the rows fetched buffer is set to 0.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 59. SQLFetchScroll SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The data returned for one or more columns is truncated. String values or numeric values are right truncated. (SQL_SUCCESS_WITH_INFO is returned if no error occurred.)
01S06	Attempted to fetch before the result set returned the first row set.	The requested row set overlapped the start of the result set when the current position is beyond the first row, and either FetchOrientation is SQL_PRIOR, or FetchOrientation is SQL_RELATIVE with a negative FetchOffset whose absolute value is less than or equal to the current SQL_ATTR_ROW_ARRAY_SIZE. (Function returns SQL_SUCCESS_WITH_INFO.)
07006	Invalid conversion.	The data value could not be converted in a meaningful manner to the data type specified by *fCType* in SQLBindCol().
22002	Invalid output or indicator buffer specified.	The pointer value specified for the argument *pcbValue* in SQLBindCol() is a null pointer and the value of the corresponding column is null. There is no means to report SQL_NULL_DATA.
22003	Numeric value out of range.	Returning the numeric value (as numeric or string) for one or more bound columns would have caused the whole (as opposed to fractional) part of the number to be truncated.
24000	Invalid cursor state.	The *StatementHandle* is in an executed state, but no result set is associated with the *StatementHandle*.
HY000	General error.	An error occurred for which there is no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.

Table 59. SQLFetchScroll SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The function is called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle*.
HY106	Fetch type out of range.	The value specified for the argument *FetchOrientation* is not valid. The value of the SQL_CURSOR_TYPE statement attribute is SQL_CURSOR_FORWARD_ONLY and the value of argument *FetchOrientation* is not SQL_FETCH_NEXT.
HY107	Row value out of range.	The value specified with the SQL_ATTR_CURSOR_TYPE statement attribute is SQL_CURSOR_KEYSET_DRIVEN, but the value specified with the SQL_ATTR_KEYSET_SIZE statement attribute is greater than 0 and less than the value specified with the SQL_ATTR_ROW_ARRAY_SIZE statement attribute.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLBindCol—Bind a column to an application variable" on page 190
- "SQLDescribeCol—Return a set of attributes for a column" on page 205
- "SQLExecDirect—Execute a statement directly" on page 210
- "SQLFetch—Fetch next row" on page 214
- "SQLExecute—Execute a statement" on page 212
- "SQLNumResultCols—Get number of result columns" on page 247
- "SQLSetStmtAttr—Set options related to a statement" on page 257

SQLForeignKeys—Get the list of foreign key columns

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLForeignKeys()` returns information about foreign keys for the specified table. The information is returned in a SQL result set that can be processed using the same functions that are used to retrieve a result set generated by a query. *PKCatalogName*, *NameLength1*, *PKSchemaName*, *NameLength2*, *FKCatalogName*, *NameLength4*, *FKSchemaName* and *NameLength5* are ignored. Columns 1, 2, 5, 6, 12, and 13 of the returned result set are always a zero-length string. Columns 10, 11, and 14 of the returned result set are always zero.

Syntax:

```
SQLRETURN   SQLForeignKeys  (
                  SQLHSTMT         StatementHandle,  /* hstmt */
                  SQLCHAR      *FAR PKCatalogName,   /* szPkCatalogName */
                  SQLSMALLINT      NameLength1,      /* cbPkCatalogName */
                  SQLCHAR      *FAR PKSchemaName,    /* szPkSchemaName */
                  SQLSMALLINT      NameLength2,      /* cbPkSchemaName */
                  SQLCHAR      *FAR PKTableName,     /* szPkTableName */
                  SQLSMALLINT      NameLength3,      /* cbPkTableName */
                  SQLCHAR      *FAR FKCatalogName,   /* szFkCatalogName */
                  SQLSMALLINT      NameLength4       /* cbFkCatalogName */
                  SQLCHAR      *FAR FKSchemaName,    /* szFkSchemaName */
                  SQLSMALLINT      NameLength5,      /* cbFkSchemaName */
                  SQLCHAR      *FAR FKTableName,     /* szFkTableName */
                  SQLSMALLINT      NameLength6);     /* cbFkTableName */
```

Function arguments:

Table 60. SQLForeignKeys arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLCHAR*	*PKCatalogName*	input	Catalog qualifier of the primary key table. This field is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength1*	input	Length of *PKCatalogName*. This field is ignored by DB2 Everyplace.
SQLCHAR*	*PKSchemaName*	input	Schema qualifier of primary key table. This field is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength2*	input	Length of *PKSchemaName*. This field is ignored by DB2 Everyplace.
SQLCHAR*	*PKTableName*	input	Name of the table containing the primary key.
SQLSMALLINT	*NameLength3*	input	Length of *PKTableName*.
SQLCHAR*	*FKCatalogName*	input	Catalog qualifier of the table containing the foreign key. This field is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength4*	input	Length of *FKCatalogName*. This field is ignored by DB2 Everyplace.
SQLCHAR*	*FKSchemaName*	input	Schema qualifier of the table containing the foreign key. This field is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength5*	input	Length of *FKSchemaName*. This field is ignored by DB2 Everyplace.
SQLCHAR*	*FKTableName*	input	Name of the table containing the foreign key.
SQLSMALLINT	*NameLength6*	input	Length of *FKTableName*.

Usage:

If *PKTableName* contains a table name, and *FKTableName* is an empty string, `SQLForeignKeys()` returns a result set containing the primary key of the specified table and all of the foreign keys (in other tables) that refer to it.

If *FKTableName* contains a table name, and *PKTableName* is an empty string, `SQLForeignKeys()` returns a result set containing all of the foreign keys in the specified table and the primary keys (in other tables) to which they refer.

If both *PKTableName* and *FKTableName* contain table names, `SQLForeignKeys()` returns the foreign keys in the table specified in *FKTableName* that refer to the primary key of the table specified in *PKTableName*. This should be one key at the most.

If the foreign keys associated with a primary key are requested, the result set is ordered by FKTABLE_NAME and ORDINAL_POSITION. If the primary keys associated with a foreign key are requested, the result set is ordered by PKTABLE_NAME and ORDINAL_POSITION.

The VARCHAR columns of the catalog functions result set are declared with a maximum length attribute of 128 to be consistent with SQL92 limits.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns does not change.

The result set contains these columns::

Column 1 PKTABLE_CAT (VARCHAR(128))
This is always a zero-length string.

Column 2 PKTABLE_SCHEM (VARCHAR(128))
This is always a zero-length string.

Column 3 PKTABLE_NAME (VARCHAR(128) not NULL)
Name of the table containing the primary key.

Column 4 PKCOLUMN_NAME (VARCHAR(128) not NULL)
Primary key column name.

Column 5 FKTABLE_CAT (VARCHAR(128))
This is always a zero-length string.

Column 6 FKTABLE_SCHEM (VARCHAR(128))
This is always a zero-length string.

Column 7 FKTABLE_NAME (VARCHAR(128) not NULL)
Name of the table containing the foreign key.

Column 8 FKCOLUMN_NAME (VARCHAR(128) not NULL)
Foreign key column name.

Column 9 ORDINAL_POSITION (SMALLINT not NULL)
Ordinal position of the column in the key, starting at 1.

Column 10 UPDATE_RULE (SMALLINT)
This is always a zero.

Column 11 DELETE_RULE (SMALLINT)
This is always a zero.

Column 12 FK_NAME (VARCHAR(128))
This is always a zero-length string.

Column 13 PK_NAME (VARCHAR(128))
This is always a zero-length string.

Column 14 DEFERRABILITY (SMALLINT)
This is always a zero.

The column names used by DB2 CLI follow the X/Open CLI CAE specification style. The column types, contents, and order are identical to those defined for the `SQLForeignKeys()` result set in ODBC.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 61. SQLForeign SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor is already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY009	Invalid argument value.	The arguments *PKTableName* and *FKTableName* were both NULL pointers.
HY010	Function sequence error.	The function is called while in a data-at-execute (`SQLPrepare()` or `SQLExecDirect()`) operation.
HY014	No more handles.	DB2 CLI is unable to allocate a handle due to internal resources.
HY090	Invalid string or buffer length.	The value of one of the name length arguments is less than 0, but not equal SQL_NTS

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLPrimaryKeys—Get primary key columns of a table" on page 250

SQLFreeConnect—Free connection handle

In ODBC Version 3, `SQLFreeConnect()` was deprecated and replaced with `SQLFreeHandle()`; see "SQLFreeHandle—Free handle resources" on page 226 for more information.

Recommendation: Although this version of DB2 CLI continues to support `SQLFreeConnect()`, use `SQLFreeHandle()` in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLFreeConnect(hdbc);
```

for example, would be rewritten using the new function as:

```
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
```

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLFreeEnv—Free environment handle

In ODBC Version 3, SQLFreeEnv() was deprecated and replaced with SQLFreeHandle(); see "SQLFreeHandle—Free handle resources" for more information.

Recommendation: Although this version of DB2 CLI continues to support SQLFreeEnv(), use SQLFreeHandle() in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLFreeEnv(henv);
```

for example, would be rewritten using the new function as:

```
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLFreeHandle—Free handle resources

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLFreeHandle() frees resources associated with a specific environment, connection, or statement handle.

This function is a generic function for freeing resources. It replaces SQLFreeConnect (for freeing a connection handle) and SQLFreeEnv() (for freeing an environment handle). SQLFreeHandle() also replaces SQLFreeStmt() (with the SQL_DROP Option) for freeing a statement handle.

Syntax:

```
SQLRETURN   SQLFreeHandle    (SQLSMALLINT      HandleType,
                              SQLHANDLE        Handle);
```

Function arguments:

Table 62. SQLFreeHandle arguments

Data type	Argument	Use	Description
SQLSMALLINT	*HandleType*	input	The type of handle to be freed by SQLFreeHandle(). Must be one of the following values: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT If *HandleType* is not one of the above values, SQLFreeHandle() returns SQL_INVALID_HANDLE.
SQLHANDLE	*Handle*	input	The name of the handle to be freed.

Usage:

SQLFreeHandle() is used to free handles for environments, connections, and statements.

An application should not use a handle after the handle is freed; DB2 CLI does not check the validity of a handle in a function call.

Freeing an 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 and any active connection remains valid.

Freeing a connection handle:

Prior to calling SQLFreeHandle() with a *HandleType* of SQL_HANDLE_DBC, an application must call SQLDisconnect() for the connection. Otherwise, the call to SQLFreeHandle() returns SQL_ERROR and the connection remains valid.

Freeing a 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 deleted. If there are results pending when SQLFreeHandle() is called, the result sets are discarded.

SQLDisconnect() automatically drops any statements open on the connection.

Return codes:

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

If SQLFreeHandle() returns SQL_ERROR, the handle is still valid.

Diagnostics:

Table 63. SQLFreeHandle SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure.	The *HandleType* argument is SQL_HANDLE_DBC, and the communication link between DB2 CLI and the data source to which it is trying to connect failed before the function completed processing.
HY000	General error.	An error occurred for which there is no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.

Table 63. SQLFreeHandle SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The *HandleType* argument is SQL_HANDLE_ENV, and at least one connection is in an allocated or connected state. `SQLDisconnect()` and `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_DBC must be called for each connection before calling `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_ENV. The *HandleType* argument is SQL_HANDLE_DBC, and the function is called before calling `SQLDisconnect()` for the connection. The *HandleType* argument is SQL_HANDLE_STMT; `SQLExecute()` or `SQLExecDirect()` is called with the statement handle, and returned SQL_NEED_DATA. (DM) All subsidiary handles and other resources were not released before `SQLFreeHandle()` is called.
HY013	Unexpected memory handling error.	The *HandleType* argument is SQL_HANDLE_STMT and the function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY017	Invalid use of an automatically allocated descriptor handle.	The *Handle* argument is set to the handle for an automatically allocated descriptor or an implementation descriptor.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLAllocHandle—Allocate handle" on page 187

SQLFreeStmt—Free (or reset) a statement handle

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLFreeStmt()` ends processing on the statement referenced by the statement handle. Use this function to:

- Disassociate (reset) parameters from application variables.
- Drop the statement handle and free the DB2 CLI resources associated with the statement handle.

`SQLFreeStmt()` is called after executing an SQL statement and processing the results.

Syntax:

```
SQLRETURN   SQLFreeStmt      (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLUSMALLINT      Option);           /* fOption */
```

Function arguments:

Table 64. SQLFreeStmt arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLUSMALLINT	*Option*	input	Option that specifies the manner of freeing the statement handle. The option must have one of the following values: SQL_DROP or SQL_RESET_PARAMS.

Usage:

`SQLFreeStmt()` can be called with the following options:

SQL_DROP
DB2 CLI resources associated with the input statement handle are freed, and the handle is invalidated. All pending results are discarded.

This option is replaced with a call to `SQLFreeHandle()` with the *HandleType* set to SQL_HANDLE_STMT.

Recommendation: Although this version of DB2 CLI continues to support this option, use `SQLFreeHandle()` in your DB2 CLI programs so that they conform to the latest standards.

SQL_RESET_PARAMS
Releases all parameter buffers set by `SQLBindParameter()` for the *StatementHandle*.

Alternatively you can drop the statement handle and allocate a new one.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

SQL_SUCCESS_WITH_INFO is not returned if *Option* is set to SQL_DROP, because there would be no statement handle to use when `SQLError()` is called.

Diagnostics:

Table 65. SQLFreeStmt SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.

Table 65. SQLFreeStmt SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY092	Option type out of range.	The value specified for the argument *Option* is not SQL_DROP or SQL_RESET_PARAMS.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLAllocHandle—Allocate handle" on page 187
- "SQLBindCol—Bind a column to an application variable" on page 190

SQLGetConnectAttr—Get current setting of a connection attribute

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetConnectAttr() returns the current setting of a connection attribute.

Syntax:

```
SQLRETURN   SQLGetConnectAttr(SQLHDBC          ConnectionHandle,
                              SQLINTEGER       Attribute,
                              SQLPOINTER       ValuePtr,
                              SQLINTEGER       BufferLength,
                              SQLINTEGER       *StringLengthPtr);
```

Function arguments:

Table 66. SQLGetConnectAttr arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle.
SQLINTEGER	*Attribute*	input	Attribute to retrieve.
SQLPOINTER	*ValuePtr*	output	A pointer to memory in which to return the current value of the attribute specified by *Attribute*.
SQLINTEGER	*BufferLength*	input	• If *ValuePtr* points to a character string, this argument should be the length of **ValuePtr*. • If *ValuePtr* is a pointer, but not to a string, then *BufferLength* should have the value *SQL_IS_POINTER*. • If the value in **ValuePtr* is a Unicode string, the *BufferLength* argument must be an even number.

Table 66. SQLGetConnectAttr arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	*StringLengthPtr*	output	A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **ValuePtr*. • If *ValuePtr* is a null pointer, no length is returned. • If the attribute value is a character string, and the number of bytes available to return is greater than *BufferLength* minus the length of the null-termination character, the data in **ValuePtr* is truncated to *BufferLength* minus the length of the null-termination character and is null-terminated by DB2 CLI.

Usage:

A call to `SQLGetConnectAttr()` returns in **ValuePtr* the value of the connection attribute specified in *Attribute*. In DB2 Everyplace, that value is a 32-bit value and the *BufferLength* and *StringLengthPtr* arguments are not used.

The following connection attributes can be retrieved by `SQLGetConnectAttr()`. For a description of the attributes, see SQLSetConnectAttr--Set options related to a connection.

- SQL_ATTR_AUTOCOMMIT (DB2 CLI/ODBC)
- SQL_ATTR_CONNECTION_DEAD (DB2 CLI/ODBC)
- SQL_ATTR_LOGIN_TIMEOUT (DB2 CLI/ODBC)
- SQL_ATTR_FILENAME_FORMAT (DB2 Everyplace)

Depending on the attribute, an application does not need to establish a connection prior to calling `SQLGetConnectAttr()`.

Return Codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 67. SQLGetConnectAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The data returned in **ValuePtr* was truncated to be *BufferLength* minus the length of a null termination character. The length of the untruncated string value is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)

Table 67. SQLGetConnectAttr SQLSTATEs (continued)

SQLSTATE	Description	Explanation
08003	Connection is closed.	An *Attribute* value was specified that required an open connection.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* was less than 0.
HY092	Option type out of range.	The value specified for the argument *Attribute* was not valid.
HYC00	Driver not capable.	The value specified for the argument Attribute was a valid connection or statement attribute for the version of the DB2 CLI driver, but was not supported by the data source.

Restrictions:

None.

Related reference:

• "SQLSetConnectAttr—Set options related to a connection" on page 254

SQLGetCursorName—Get cursor name

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLGetCursorName()` returns the cursor name associated with the input statement handle. If a cursor name is explicitly set by calling `SQLSetCursorName()`, this name returns; otherwise, an implicitly generated name returns.

Syntax:

```
SQLRETURN   SQLGetCursorName (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      *FAR CursorName,        /* szCursor */
                SQLSMALLINT       BufferLength,      /* cbCursorMax */
                SQLSMALLINT  *FAR NameLengthPtr);    /* pcbCursor */
```

Function arguments:

Table 68. SQLGetCursorName Arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLCHAR *	*CursorName*	output	Cursor name
SQLSMALLINT	*BufferLength*	input	Length of buffer *CursorName*
SQLSMALLINT *	*NameLengthPtr*	output	Number of bytes available to return for *CursorName*

Usage:

`SQLGetCursorName()` returns the cursor name set explicitly with `SQLSetCursorName()`, or if no name is set, it returns the cursor name internally generated by DB2 CLI.

If a name is set explicitly using `SQLSetCursorName()`, this name returns until the statement is dropped, or until another explicit name is set.

Internally generated cursor names always begin with SQLCUR or SQL_CUR. Cursor names are always 18 characters or less and are always unique within a connection.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 69. SQLGetCursorName SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The cursor name returned in *CursorName* is longer than the value in *BufferLength*, and is truncated to *BufferLength* - 1 bytes. The argument *NameLengthPtr* contains the length of the full cursor name available for return. The function returns SQL_SUCCESS_WITH_INFO.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error.	The function is called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.

Table 69. SQLGetCursorName SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* is less than 0.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLExecDirect—Execute a statement directly" on page 210

SQLGetData—Get data from a column

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLGetData()` retrieves data for a single column in the current row of the result set. This is an alternative to `SQLBindCol()`, which is used to transfer data directly into application variables on each `SQLFetch()` call.

`SQLFetch()` must be called before `SQLGetData()`.

After calling `SQLGetData()` for each necessary column, `SQLFetch()` is called to retrieve the next row.

Syntax:

```
SQLRETURN   SQLGetData       (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLSMALLINT       TargetType,        /* fCType */
                SQLPOINTER        TargetValuePtr,    /* rgbValue */
                SQLINTEGER        BufferLength,      /* cbValueMax */
                SQLINTEGER   *FAR StrLen_or_IndPtr); /* pcbValue */
```

Function arguments:

Table 70. SQLGetData arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLUSMALLINT	*ColumnNumber*	input	Column number for which the data retrieval is requested. Result set columns are numbered sequentially. Column numbers start at one.

Table 70. SQLGetData arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*TargetType*	input	The C data type of the column identified by *ColumnNumber*. The following types are supported: SQL_C_BINARY SQL_C_BIT SQL_C_CHAR SQL_C_DOUBLE SQL_C_FLOAT SQL_C_LONG SQL_C_SHORT SQL_C_TYPE_DATE SQL_C_TYPE_TIME SQL_C_TYPE_TIMESTAMP SQL_C_TINYINT Specifying SQL_C_DEFAULT results in the data being converted to its default C data type.
SQLPOINTER	*TargetValuePtr*	output	Pointer to the buffer where the retrieved column data is to be stored. The output buffer needs to be word-aligned (even). Many processors such as the Motorola 68000 have word-alignment rules, and for non-character data types, the application should align the buffer properly.
SQLINTEGER	*BufferLength*	input	Maximum size of the buffer pointed to by *TargetValuePtr*. If *TargetType* denotes a binary or character string, then BufferLength must be > 0, or an error returns. Otherwise, the argument is ignored.

Table 70. SQLGetData arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	*StrLen_or_IndPtr*	output	Pointer to the value that indicates the number of bytes that DB2 CLI has available to return in the *TargetValuePtr* buffer. If data truncation occurs, this contains the total number of bytes required to retrieve the whole column. For binary and character data types, the application can alternatively choose the piecemeal retrieval mode to retrieve large data piece by piece. In this mode, the *StrLen_or_IndPtr* argument contains the number of bytes *left* in the column. The value is SQL_NULL_DATA if the data value of the column is null. If this pointer is NULL and `SQLFetch()` obtained a column containing null data, then this function fails because it has no means of reporting this. If `SQLFetch()` fetched a column containing binary data, then the pointer to *StrLen_or_IndPtr* must not be NULL or this function fails because it has no other means of informing the application about the length of the data retrieved in the *TargetValuePtr* buffer.

Usage:

`SQLGetData()` can be used with `SQLBindCol()` for the same result set if `SQLFetch()` is used. The general steps are:

1. `SQLFetch()` advances to the first row, retrieves the first row, and transfers data for bound columns.
2. `SQLGetData()` transfers data for the specified column.
3. `SQLGetData()` repeats step 2 for each column needed.
4. `SQLFetch()` advances to the next row, retrieves the next row, and transfers data for bound columns.
5. Steps 2, 3 and 4 are repeated for each row in the result set, or until the result set is no longer needed.

To discard the column data part way through the retrieval, the application can call `SQLGetData()` with *ColumnNumber* set to the next column position of interest. To discard data that has not been retrieved for the entire row, the application should call `SQLFetch()` to advance to the next row; or, if no more data from the result set is needed, calls `SQLFreeStmt()`.

The *TargetType* input argument determines the type of data conversion (if any) needed before the column data is placed into the storage area pointed to by *TargetValuePtr*.

The value returned in *TargetValuePtr* is null-terminated unless the column data to be retrieved is binary.

Truncation of numeric data types is reported as a warning if the truncation involves digits to the right of the decimal point. If truncation occurs to the left of the decimal point, an error returns.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL_SUCCESS returns if a zero-length string is retrieved by `SQLGetData()`. If this is the case, *StrLen_or_IndPtr* contains 0, and *TargetValuePtr* contains a null terminator.

If the preceding call to `SQLFetch()` fails, do not call `SQLGetData()` because the result is undefined.

Diagnostics:

Table 71. SQLGetData SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	Data returned for the specified column (*ColumnNumber*) is truncated. String or numeric values are right truncated. SQL_SUCCESS_WITH_INFO is returned.
07006	Invalid conversion.	The data value cannot be converted to the C data type specified by the argument *TargetType*. The function is called before for the same *ColumnNumber* value but with a different *TargetType* value.
22002	Invalid output or indicator buffer specified.	The pointer value specified for the argument *StrLen_or_IndPtr* is a null pointer and the value of the column is null. There is no means to report SQL_NULL_DATA.
22005	Error in assignment.	A returned value is incompatible with the data type denoted by the argument *TargetType*.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY002	Invalid column number.	The specified column is less than 0 or greater than the number of result columns.
HY003	Program type out of range.	*TargetType* is not a valid data type or SQL_C_DEFAULT.
HY010	Function sequence error.	The function is called without first calling `SQLFetch()`.

Table 71. SQLGetData SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value of the argument *BufferLength* is less than 0 and the argument *TargetType* is SQL_C_CHAR or SQL_C_BINARY, or *TargetType* is SQL_C_DEFAULT and the default type is one of SQL_C_CHAR, SQL_C_BINARY, or SQL_C_DBCHAR.
HYC00	Driver not capable.	The SQL data type for the specified data type is recognized but not supported by DB2 CLI. The requested conversion from the SQL data type to the application data *TargetType* cannot be performed by DB2 CLI or the data source.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLBindCol—Bind a column to an application variable" on page 190

SQLGetDiagRec—Get multiple fields settings of diagnostic record

Purpose:

Specification:	**DB2 CLI** 5.0	**ODBC** 3.0	

`SQLGetDiagRec()` returns the current value of the SQLSTATE field of a diagnostic record that contains error, warning, and status information.

A connection handle must be allocated using `SQLAllocHandle()` before calling this function.

Syntax:

```
SQLRETURN   SQLGetDiagRec    (SQLSMALLINT       HandleType,
                              SQLHANDLE         Handle,
                              SQLSMALLINT       RecNumber,
                              SQLCHAR           *SQLState,
                              SQLINTEGER        *NativeErrorPtr,
                              SQLCHAR           *MessageText,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *TextLengthPtr);
```

Function arguments:

Table 72. SQLGetDiagRec arguments

Data type	Argument	Use	Description
SQLSMALLINT	*HandleType*	input	A handle-type identifier that describes the type of handle for which diagnostics are desired. Can be SQL_HANDLE_STMT or SQL_HANDLE_DBC.
SQLHANDLE	*Handle*	input	A handle for the diagnostic data structure, of the type indicated by *HandleType*.
SQLSMALLINT	*RecNumber*	input	Indicates the status record from which the application seeks information. Status records must be 1.
SQLCHAR	*SQLState*	output	Pointer to a buffer in which to return a 5 character SQLSTATE code pertaining to the diagnostic record *RecNumber*. The first two characters indicate the class; the next three indicate the subclass.
SQLINTEGER	*NativeErrorPtr*	output	Pointer to a buffer in which to return the native error code, specific to the data source.
SQLCHAR	*MessageText*	output	Pointer to a buffer in which to return the error message text. The fields returned by `SQLGetDiagRec()` are contained in a text string.
SQLINTEGER	*BufferLength*	input	Length (in bytes) of the *MessageText* buffer.
SQLSMALLINT	*TextLengthPtr*	output	Pointer to a buffer in which to return the total number of bytes (excluding the number of bytes required for the null termination character) available to return in *MessageText*. If the number of bytes available to return is greater than *BufferLength*, then the error message text in *MessageText* is truncated to *BufferLength* minus the length of the null termination character.

Usage:

An application typically calls `SQLGetDiagRec()` when a previous call to a DB2 CLI function returns anything other than SQL_SUCCESS.

`SQLGetDiagRec()` returns a character string containing multiple fields of the diagnostic data structure record.

The functionality of `SQLGetDiagRec()` is extended in Version 8.1 of DB2 Everyplace. The following SQLSTATEs can now be returned : 57011, HY024, HY092, HY000, HY012. See “SQLState listing” on page 170 for more information about these SQLSTATEs.

`SQLGetDiagRec()` retrieves only the diagnostic information most recently associated with the handle specified in the *Handle* argument. If the application calls any function, except `SQLGetDiagRec()`, any diagnostic information from the previous calls on the same handle is lost.

***HandleType* argument**

Each handle type can have diagnostic information associated with it. The *HandleType* argument denotes the handle type of *Handle*. DB2 Everyplace supports statement handles and connection handles.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

`SQLGetDiagRec()` does not post error values for itself. It uses the following return values to report the outcome of its own execution:

SQL_SUCCESS
The function successfully returned diagnostic information.

SQL_SUCCESS_WITH_INFO
The *MessageText* buffer is too small to hold the requested diagnostic message. No diagnostic records are generated. To determine that a truncation occurred, the application must compare *BufferLength* to the actual number of bytes available, which is written to *StringLengthPtr*.

SQL_INVALID_HANDLE
The handle indicated by *HandleType* and *Handle* is not a valid handle.

SQL_ERROR
One of the following situations occurred:
- *RecNumber* is negative or 0.
- *BufferLength* is less than zero.

SQL_NO_DATA
RecNumber is greater than the number of diagnostic records that existed for the handle specified in *Handle*. The function also returns SQL_NO_DATA for any positive *RecNumber* if there are no diagnostic records for *Handle*.

Restrictions:

None.

Related reference:
- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLGetInfo—Get general information

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLGetInfo()` returns general information (including supported data conversions) about the DBMS to which the application is connected.

Syntax:

```
SQLRETURN   SQLGetInfo      (
              SQLHDBC          ConnectionHandle,   /* hdbc */
              SQLUSMALLINT      InfoType,      /* fInfoType */
              SQLPOINTER        InfoValuePtr,    /* rgbInfoValue */
              SQLSMALLINT       BufferLength,        /* cbInfoValueMax */
              SQLSMALLINT     *FAR StringLengthPtr,       /* pcbInfoValue */
```

Function arguments:

Table 73. SQLGetInfo arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Database connection handle
SQLUSMALLINT	*InfoType*	output	The type of information desired. The argument must be one of the values in the first column of the tables in Data Types and Data Conversion.

Table 73. SQLGetInfo arguments (continued)

Data type	Argument	Use	Description
SQLPOINTER	*InfoValuePtr*	output (also input)	Pointer to buffer where this function stores the necessary information. Depending on the type of information being retrieved, 5 types of information can be returned: 16-bit integer value 32-bit integer value 32-bit binary value 32-bit mask Null-terminated character string
SQLSMALLINT	*BufferLength*	input	Maximum size of the buffer pointed to by *InfoValuePtr*.
SQLSMALLINT *	*StrLen_or_IndPtr*	output	Pointer to location where this function returns the total number of bytes available to return the desired information. In the case of string output, this size does not include the null terminating character. If the value in the location pointed to by *StringLengthPtr* is greater than the size of the *InfoValuePtr* buffer as specified in *BufferLength*, then the string output information would be truncated to *BufferLength* - 1 bytes and the function would return with SQL_SUCCESS_WITH_INFO.

Usage:

Refer to Information Returned By SQLGetInfo for a list of the possible values of *InfoType* and a description of the information that `SQLGetInfo()` would return for that value.

DB2 CLI returns a value for each *InfoType* in this table. If the *InfoType* does not apply or is not supported, the result is dependent on the return type:

- If the return type is a character string containing 'Y' or 'N', "N" is returned.
- If the return type is a character string containing a value other than just 'Y' or 'N', an empty string is returned.
- If the return type is a 16-bit integer, 0 (zero) is returned.
- If the return type is a 32-bit integer, 0 (zero) is returned.
- If the return type is a 32-bit mask, 0 (zero) is returned.

Information Returned By SQLGetInfo

SQL_DBMS_NAME (string)
Tthe name of the DBMS product being accessed. For example: "DB2 Everyplace".

SQL_DBMS_VER (string)
Tthe version of DB2 Everyplace DBMS product. The information returned is string of the form: `DB2 Everyplace Vm.v.r Build yyyy-mm-dd`, where `m` is the major version, `v` is the minor version, `r` is the release, and `yyyy-mm-dd` is the date of the build in ISO format.

For example:

```
'DB2 Everyplace V8.1.2 Build 2003-04-01'
is DB2 Everyplace Version 8.1.2 built on April 01, 2003
```

Note: Applications require a buffer that can contain at least 39 characters (BUFSIZE). For example:

```
rc = SQLGetInfo(hdbc, SQL_DBMS_VER, buf, BUFSIZE, &len);
```

SQL_IDENTIFIER_QUOTE_CHAR (string)
Indicates the character used to surround a delimited identifier.

SQL_MAX_BINARY_LITERAL_LEN (32-bit unsigned integer)
A 32-bit unsigned integer value specifying the maximum length of a hexadecimal literal in a SQL statement.

SQL_MAX_CHAR_LITERAL_LEN (32-bit unsigned integer)
The maximum length of a character literal in an SQL statement (in bytes).

SQL_MAX_COLUMN_NAME_LEN (16-bit integer)
The maximum length of a column name (in bytes).

SQL_MAX_COLUMNS_IN_GROUP_BY (16-bit integer)
Indicates the maximum number of columns that the server supports in a GROUP BY clause. Zero if no limit.

SQL_MAX_COLUMNS_IN_INDEX (16-bit integer)
Indicates the maximum number of columns that the server supports in an index. Zero if no limit.

SQL_MAX_COLUMNS_IN_ORDER_BY (16-bit integer)
Indicates the maximum number of columns that the server supports in an ORDER BY clause. Zero if no limit.

SQL_MAX_COLUMNS_IN_SELECT (16-bit integer)
Indicates the maximum number of columns that the server supports in a select list. Zero if no limit.

SQL_MAX_CONCURRENT_ACTIVITIES (16-bit integer)
The maximum number of active environments that the DB2 Everyplace CLI driver can support. If there is no specified limit or the limit is unknown, this value is set to zero.

SQL_MAX_DRIVER_CONNECTIONS (16-bit integer)
The maximum number of active connections supported per application.

SQL_MAX_INDEX_SIZE (32-bit unsigned integer)
Indicates the maximum size in bytes that the server supports for the combined columns in an index. Zero if no limit.

SQL_MAX_ROW_SIZE (32-bit unsigned integer)
Specifies the maximum length in bytes that the server supports in single row of a base table. Zero if no limit.

SQL_MAX_STATEMENT_LEN (32-bit unsigned integer)
Indicates the maximum length of an SQL statement string in bytes, including the number of white spaces in the statement.

SQL_MAX_TABLE_NAME_LEN (16-bit integer)
The maximum length of a table name (in bytes).

SQL_MAX_TABLES_IN_SELECT (16-bit integer)
Indicates the maximum number of table names allowed in a FROM clause in a query specification.

SQL_MAX_USER_NAME_LEN (16-bit integer)
Indicates the maximum size allowed for a user identifier (in bytes).

SQL_SEARCH_PATTERN_ESCAPE (string)
Used to specify what the driver supports as an escape character for catalog functions such as `(SQLTables(), SQLColumns())`.

SQL_TXN_CAPABLE (16-bit integer)
Indicates whether transactions can contain DDL or DML or both.

- SQL_TC_NONE = transactions not supported.
- SQL_TC_DML = transactions can only contain DML statements (SELECT, INSERT, UPDATE, DELETE, etc.) DDL statements (CREATE TABLE, DROP INDEX, etc.) encountered in a transaction cause an error.
- SQL_TC_DDL_COMMIT = transactions can only contain DML statements. DDL statements encountered in a transaction cause the transaction to be committed.
- SQL_TC_DDL_IGNORE = transactions can only contain DML statements. DDL statements encountered in a transaction are ignored.
- SQL_TC_ALL = transactions can contain DDL and DML statements in any order.

SQL_USER_NAME (string)
The user name used in a particular database. This is the identifier specified on the `SQLConnect()` call.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLGetStmtAttr—Get current setting of a statement attribute

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLGetStmtAttr()` returns the current setting of a statement attribute.

Syntax:

```
SQLRETURN   SQLGetStmtAttr      (
                SQLHSTMT            StatementHandle,
                SQLINTEGER          Attribute,
                SQLPOINTER          ValuePtr,
                SQLINTEGER          BufferLength,
                SQLINTEGER         *StringLengthPtr);
```

Function arguments:

Table 74. SQLGetStmtAttr arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLINTEGER	*Attribute*	input	Attribute to retrieve.
SQLPOINTER	*ValuePtr*	output	Pointer to a buffer in which to return the value of the attribute specified in *Attribute*.
SQLINTEGER	*BufferLength*	input	If *Attribute* is an ODBC-defined attribute and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *Attribute* is an ODBC-defined attribute and **ValuePtr* is an integer, *BufferLength* is ignored. If *Attribute* is a DB2 CLI attribute, the application indicates the nature of the attribute by setting the *BufferLength* argument. *BufferLength* can have the following values: • If **ValuePtr* is a pointer to a character string, then *BufferLength* is the length of the string or SQL_NTS. • If **ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in *BufferLength*. • If **ValuePtr* is a pointer to a value other than a character string or binary string, then *BufferLength* should have the value SQL_IS_POINTER. • If **ValuePtr* contains a fixed-length data type, then *BufferLength* is either SQL_IS_INTEGER or SQL_IS_UINTEGER, as appropriate.
SQLSMALLINT	**StringLengthPtr*	output	A pointer to a buffer in which to return the total number of bytes (excluding the null termination character) available to return in **ValuePtr*. If this is a null pointer, no length is returned. If the attribute value is a character string, and the number of bytes available to return is greater than or equal to *BufferLength*, the data in **ValuePtr* is truncated to *BufferLength* minus the length of a null termination character and is null terminated by the DB2 CLI.

Usage:

A call to `SQLGetStmtAttr()` returns in **ValuePtr* the value of the statement attribute specified in *Attribute*. In DB2 Everyplace, that value is a 32-bit value and the *BufferLength* and *StringLengthPtr* arguments are not used.

The following statement attributes can be retrieved by `SQLGetStmtAttr()`. For a description of the attributes, see "SQLSetStmtAttr—Set options related to a statement" on page 257.

- SQL_ATTR_CURSOR_SCROLLABLE (DB2 CLI/ODBC)
- SQL_ATTR_CURSOR_SENSITIVITY (DB2 CLI/ODBC)
- SQL_ATTR_CURSOR_TYPE (DB2 CLI/ODBC)
- SQL_ATTR_ROW_ARRAY_SIZE (DB2 CLI/ODBC)
- SQL_ATTR_ROW_BIND_TYPE (DB2 CLI/ODBC)
- SQL_ATTR_ROW_NUMBER (DB2 CLI/ODBC)
- SQL_ATTR_DELETE_MODE (DB2 Everyplace)
- SQL_ATTR_DIRTYBIT_SET_MODE (DB2 Everyplace)
- SQL_ATTR_READ_MODE (DB2 Everyplace)
- SQL_ATTR_REORG_MODE (DB2 Everyplace)

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 75. SQLGetStmtAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The data returned in **ValuePtr* is truncated to be *BufferLength* minus the length of a null termination character. The length of the untruncated string value is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state.	The argument *Attribute* is SQL_ATTR_ROW_NUMBER and the cursor is not open, or the cursor is positioned before the start of the result set or after the end of the result set.
HY000	General error.	An error occurred for which there is no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error.	An asynchronously executing function is called for the *StatementHandle* and is still executing when this function is called. `SQLExecute()` or `SQLExecDirect()` is called for the *StatementHandle* and returned SQL_NEED_DATA. This function is called before data is sent for all data-at-execution parameters or columns.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.

Table 75. SQLGetStmtAttr SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	The value specified for argument *BufferLength* is less than 0.
HY092	Option type out of range.	The value specified for the argument *Attribute* is not valid for this version of DB2 CLI
HY109	Invalid cursor position.	The *Attribute* argument is SQL_ATTR_ROW_NUMBER and the row had been deleted or could not be fetched.
HYC00	Driver not capable.	The value specified for the argument *Attribute* is a valid DB2 CLI attribute for the version of DB2 CLI, but is not supported by the data source.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLSetConnectAttr—Set options related to a connection" on page 254
- "SQLSetStmtAttr—Set options related to a statement" on page 257

SQLNumParams - Get Number of Parameters in A SQL Statement

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLNumParams()` returns the number of parameter markers in an SQL statement.

Syntax:

```
SQLRETURN   SQLNumParams     (SQLHSTMT          StatementHandle,
                              SQLSMALLINT  FAR  *ParameterCountPtr);
```

Function arguments:

Table 76. SQLNumParams arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLSMALLINT	*ParameterCountPtr*	Output	Number of parameters in the statement.

Usage:

This function can only be called after the statement associated with *StatementHandle* has been prepared. If the statement does not contain any parameter markers, *ParameterCountPtr* is set to 0.

An application can call this function to determine how many SQLBindParameter() calls are necessary for the SQL statement associated with the statement handle.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 77. SQLNumParams SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error.	This function was called before SQLPrepare() was called for the specified *StatementHandle*.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

Restrictions:

None.

Related reference:
- "SQLBindParameter—Bind a parameter marker to a buffer" on page 193
- "SQLPrepare—Prepare a statement" on page 248

SQLNumResultCols—Get number of result columns

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	

SQLNumResultCols() returns the number of columns in the result set associated with the input statement handle.

SQLPrepare() or SQLExecDirect() must be called before calling this function.

After calling this function, you can call SQLColAttribute() or one of the bind column functions.

Syntax:

```
SQLRETURN   SQLNumResultCols (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLSMALLINT  FAR  *ColumnCountPtr);  /* pccol */
```

Function arguments:

Table 78. SQLNumResultCols arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.

Table 78. SQLNumResultCols arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	*ColumnCountPtr*	output	Number of columns in the result set.

Usage:

The function sets the output argument to zero if the last statement or function executed on the input statement handle did not generate a result set.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 79. SQLNumResultCols SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error.	The function is called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle*.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.

Restrictions:

None.

Related reference:
- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLBindCol—Bind a column to an application variable" on page 190
- "SQLDescribeCol—Return a set of attributes for a column" on page 205
- "SQLExecDirect—Execute a statement directly" on page 210
- "SQLGetData—Get data from a column" on page 234

SQLPrepare—Prepare a statement

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLPrepare() associates an SQL statement with the input statement handle and sends the statement to the DBMS to be prepared. The application can reference this prepared statement by passing the statement handle to other functions.

If the statement handle is previously used with a query statement (or any function that returns a result set), SQLFreeStmt() must be called before calling SQLPrepare().

Syntax:

```
SQLRETURN   SQLPrepare      (SQLHSTMT          StatementHandle,   /* hstmt */
                             SQLCHAR      FAR  *StatementText,    /* szSqlStr */
                             SQLINTEGER        TextLength);       /* cbSqlStr */
```

Function arguments:

Table 80. SQLPrepare arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLCHAR	*StatementText*	input	SQL statement string
SQLINTEGER	*TextLength*	input	Length of contents of *StatementText* argument. This must be set to either the exact length of the SQL statement in *szSqlstr*, or to SQL_NTS if the statement text is null-terminated.

Usage:

After a statement is prepared using SQLPrepare(), the application can request information about the format of the result set (if the statement is a query) by calling either:
- SQLNumResultCols()
- SQLDescribeCol()

The SQL statement string might contain parameter markers. A parameter marker is represented by a ? character and is used to indicate a position in the statement in which an application-supplied value is to be substituted when SQLExecute() is called. The bind parameter function, SQLBindParameter(), binds (associates) application values with each parameter marker and indicates if any data conversion should be performed at the time the data is transferred.

All parameters must be bound before calling SQLExecute(). For more information, refer to "SQLExecute—Execute a statement" on page 212.

Refer to the section on the PREPARE statement in the *DB2 Universal Database SQL Reference* for information on rules related to parameter markers.

After the application processes the results from the SQLExecute() call, the application can execute the statement again with new (or the same) parameter values.

Return codes:
- SQL_SUCCESS

- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 81. SQLPrepare SQLSTATEs

SQLSTATE	Description	Explanation
42nnn	Syntax Error.	**42**nnn SQLSTATEs indicate a variety of syntax or access problems with the statement. The characters nnn refer to any SQLSTATE with that class code. Example: **42**nnn refers to any SQLSTATE in the **42** class.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY009	Invalid argument value.	*StatementText* is a null pointer.
HY013	Unexpected memory handling error.	DB2 CLI is unable to access memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI is unable to allocate a handle due to internal resources.
HY090	Invalid string or buffer length.	The argument *TextLength* is less than one, but not equal to SQL_NTS.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLBindParameter—Bind a parameter marker to a buffer" on page 193
- "SQLDescribeCol—Return a set of attributes for a column" on page 205
- "SQLExecDirect—Execute a statement directly" on page 210
- "SQLExecute—Execute a statement" on page 212
- "SQLNumResultCols—Get number of result columns" on page 247

SQLPrimaryKeys—Get primary key columns of a table

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

`SQLPrimaryKeys()` returns a list of column names that comprise the primary key for a table. The information is returned in an SQL result set, which can be retrieved using the same functions that are used to process a result set generated by a query. *CatalogName*, *NameLength1*, *SchemaName*, and *NameLength2* are ignored. Columns 1, 2, and 6 of the returned result set are always a zero length string.

Syntax:

```
SQLRETURN   SQLPrimaryKeys  (
                  SQLHSTMT         StatementHandle,  /* hstmt */
                  SQLCHAR      FAR *CatalogName,     /* szCatalogName */
                  SQLSMALLINT      NameLength1,      /* cbCatalogName */
                  SQLCHAR      FAR *SchemaName,      /* szSchemaName */
                  SQLSMALLINT      NameLength2,      /* cbSchemaName */
                  SQLCHAR      FAR *TableName,       /* szTableName */
                  SQLSMALLINT      NameLength3);     /* cbTableName */
```

Function arguments:

Table 82. SQLPrimaryKeys arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLCHAR*	*CatalogName*	input	Catalog qualifier of a three–part table name. This field is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength1*	input	Length of *CatalogName*. This field is ignored by DB2 Everyplace.
SQLCHAR*	*SchemaName*	input	Schema qualifier of table name. This field is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength2*	input	Length of *SchemaName*. This field is ignored by DB2 Everyplace.
SQLCHAR*	*TableName*	input	Table name.
SQLSMALLINT	*NameLength3*	input	Length of *TableName*.

Usage:

`SQLPrimaryKeys()` returns the primary key columns from a single table. Search patterns cannot be used to specify the table name.

If the specified table does not contain a primary key, an empty result set is returned.

Calls to `SQLPrimaryKeys()` in many cases map to complex and, thus, expensive queries against the system catalog.

Although new columns can be added and the names of the existing columns changed in future releases, the position of the current columns does not change.

The result set contains these columns, ordered by TABLE_NAME, and ORDINAL_POSITION:

Column 1 TABLE_CAT (VARCHAR(128))
This is always a zero-length string.

Column 2 TABLE_SCHEM (VARCHAR(128))
This is always a zero-length string.

Column 3 TABLE_NAME (VARCHAR(128) not NULL)
Name of the specified table.

Column 4 COLUMN_NAME (VARCHAR(128) not NULL)
Primary key column name.

Column 5 ORDINAL_POSITION (SMALLINT not NULL)
Column sequence number in the primary key, starting with one.

Column 6 PK_NAME (VARCHAR(128))
This is always a zero-length string.

The column names used by DB2 CLI/ODBC follow the X/Open CLI CAE specification style.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 83. SQLPrimaryKey SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor is already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error.	The function is called while in a data-at-execute (`SQLPrepare()` or `SQLExecDirect()`) operation.
HY014	No more handles.	DB2 CLI is unable to allocate a handle due to internal resources.
HY090	Invalid string or buffer length.	The value of one of the name length arguments is less than 0, but not equal SQL_NTS.

Restrictions:

Use calls to `SQLPrimaryKeys()` sparingly, and save the results rather than repeating calls.

Related reference:
- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLForeignKeys—Get the list of foreign key columns" on page 222

SQLRowCount—Get row count

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLRowCount()` returns the number of rows in a table that were affected by an UPDATE, INSERT, DELETE, or SELECT with scrollable cursor statement executed against the table.

`SQLExecute()` or `SQLExecDirect()` must be called before calling this function.

Syntax:

```
SQLRETURN   SQLRowCount      (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLINTEGER   FAR  *RowCountPtr);     /* pcrow */
```

Function arguments:

Table 84. SQLRowCount arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLINTEGER	*RowCountPtr*	output	Pointer to location where the number of rows affected is stored.

Usage:

If the last executed statement referenced by the input statement handle is not an UPDATE, INSERT, or DELETE statement, or if it did not execute successfully, then the function sets the contents of *RowCountPtr* to -1.

Any rows in other tables that might have been affected by the statement are not included in the count.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 85. SQLRowCount SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function is called prior to calling `SQLExecute()` or `SQLExecDirect()` for the *StatementHandle*.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

Restrictions:

None.

Related reference:
- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

- "SQLExecDirect—Execute a statement directly" on page 210
- "SQLExecute—Execute a statement" on page 212
- "SQLNumResultCols—Get number of result columns" on page 247

SQLSetConnectAttr—Set options related to a connection

Purpose:

Specification:	DB2 CLI	ODBC 1.0	ISO CLI

`SQLSetConnectAttr()` sets options related to a connection.

Syntax:

```
SQLRETURN   SQLSetConnectAttr   (SQLHDBC        ConnectionHandle,
                                 SQLINTEGER     Attribute,
                                 SQLPOINTER     ValuePtr,
                                 SQLINTEGER     StringLength);
```

Function arguments:

Table 86. SQLSetConnectAttr arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle.
SQLINTEGER	*Attribute*	input	Option to set.
SQLPOINTER	*ValuePtr*	input	If *Attribute* is an ODBC-defined attribute and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of *ValuePtr*. If *Attribute* is an ODBC-defined attribute and ValuePtr is an integer, *StringLength* is ignored. If *Attribute* is a DB2 CLI attribute, the application indicates the nature of the attribute by setting the *StringLength* argument. *StringLength* can have the following values: • If *ValuePtr* is a pointer to a character string, *StringLength* is the length of the string or SQL_NTS. • If *ValuePtr* is a pointer to a binary buffer, the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in *StringLength*. This places a negative value in *StringLength*. • If *ValuePtr* is a pointer to a value other than a character string or a binary string, *StringLength* should have the value SQL_IS_POINTER. • If *ValuePtr* contains a fixed-length value, *StringLength* is either SQL_IS_INTEGER or SQL_IS_UINTEGER, as appropriate.

Table 86. SQLSetConnectAttr arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	*StringLength*	input	If *ValuePtr* points to a character string or a binary buffer, this argument should be the length of *ValuePtr*. If *ValuePtr* is a pointer, but not to a string or binary buffer, *StringLength* should have the value SQL_IS_POINTER. If *ValuePtr* is not a pointer, *StringLength* should have the value SQL_IS_NOT_POINTER.

Usage:

Connection attributes for a connection remain in effect until they are changed by another call to `SQLSetConnectAttr()` or the connection is dropped by calling `SQLDisconnect()`.

`SQLSetConnectAttr()` accepts attribute information in one of two different formats: a null-terminated character string or a 32-bit integer value. The format of each is noted in the attribute's description. Character strings pointed to by the *ValuePtr* argument of `SQLSetConnectAttr()` have a length of *StringLength*.

Connection attributes:

The currently defined attributes are shown below.

SQL_ATTR_AUTOCOMMIT (DB2 CLI/ODBC)

A 32-bit integer value that specifies the mode type. The supported values are:

- SQL_AUTOCOMMIT_ON = Each statement is automatically committed. This is the default.

  In auto-commit mode, all updates performed by a statement are made persistent automatically after the statement is executed. Auto-commit mode is the default behavior. By default, transaction support is not enabled and furthermore, statement level atomicity is not guaranteed. For example, the following UPDATE statement might fail during processing and only a subset of rows might be updated:

  ```
  UPDATE T SET A = A + 1
  ```

  There might be many reasons why the update/delete/insert operations fail. For example, a check constraint can be violated during an update. As a result, a portion of the table can be updated properly while the rest of the table cannot, and the changes cannot be rolled back.
- SQL_AUTOCOMMIT_OFF = The application must manually, explicitly commit or rollback a transaction. Committing or rolling back a transaction is accomplished by calling `SQLEndTran()`. For more information about using `SQLEndTran()`, see "SQLEndTran—Request a COMMIT or ROLLBACK" on page 209.

  In manual-commit mode, transactions are started implicitly with the first access to the database using SQLPrepare() and SQLExecDirect(). At this point a transaction has begun, even if the call failed. The transaction ends when you use SQLEndTran() to either ROLLBACK or COMMIT the transaction.

In manual-commit mode, transactions can issue any SQL statements, including DDL and DML (for example, CREATE TABLE or UPDATE statements).

SQL_ATTR_CONNECTION_DEAD (DB2 CLI/ODBC)
A READ ONLY 32-bit integer value that indicates whether or not the connection is still active. DB2 CLI will return one of the following values:

- SQL_CD_FALSE - the connection is still active.
- SQL_CD_TRUE - the connection is dead.

SQL_ATTR_LOGIN_TIMEOUT (DB2 CLI/ODBC)
A 32-bit integer value corresponding to the number of seconds to wait for a login request to complete before returning control to the application.

SQL_ATTR_FILENAME_FORMAT (DB2 Everyplace)
A 32-bit integer specifies whether DB2e database engine should create filenames in long or 8.3 format. Applications are allowed to change filename format only if no catalog files exist in the path connected when SQLSetConnectAttr is invoked. SQL_ERROR with SQLState HY000 will be returned if changing filename format is denied due to pre-existing catalog files. For example, if an application connects to a path where DB2 Everyplace catalog files already exist, any attempts to change filename format will fail. If an application connects to a path where no DB2 Everyplace catalog files exist and it attempts to change filename format after the first CREATE TABLE statement, SQLSetConnectAttr will return SQL_ERROR too. This is because catalog files are created during the very first CREATE TABLE statement, and it is not allowed to change filename format after the creation of catalog files. The default filename format is platform dependent. SQL_FILENAME_FORMAT_LONG is currently the default for all platforms supported.

Attribute values:

```
SQL_FILENAME_FORMAT_LONG - files will be created in long filename format.
SQL_FILENAME_FORMAT_83- files will be created in 8.3 filename format.
```

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 87. SQLSetConnectAttr SQLSTATEs

SQLSTATE	Description	Explanation
HY000	General error.	Filename format cannot be changed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to internal resources.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS.

Restrictions:

- The number of tables that can be updated inside a transaction is limited. DB2 Everyplace permits a maximum of 256 open files inside a transaction, assuming that the operating system also permits this many open files. This typically means that approximately 100 tables can be updated. The number of tables depends on index usage and number of statement handles. As the number of active statement handles increases, fewer tables can potentially be updated. Each table is counted only one time even if it is accessed or updated several times inside a transaction.
- Transactions were added to DB2 Everyplace to allow consistent updating and insertion of several related records in a number of tables. Changes are written to the data tables after the application commits the transaction.
- If the application terminated prematurely without committing the current transaction, the updates within that transaction are rolled back automatically.
- After the SQLEndTran returns, the transaction is either committed or rolled back.
- When an application connects to a database that terminated prematurely (during an active transaction) then the transaction is recovered. The database recovers transactions using the following logic:
  - If the transaction is not complete, the database is *not* be updated.
  - If the transaction is complete, the database *is* updated with the information from that transaction.
  - If the recovery is interrupted, the appropriate action is performed at the next connect.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182
- "SQLEndTran—Request a COMMIT or ROLLBACK" on page 209

SQLSetStmtAttr—Set options related to a statement

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

`SQLSetStmtAttr()` sets options related to a statement.

Syntax:

```
SQLRETURN   SQLSetStmtAttr   (SQLHSTMT          StatementHandle,
                              SQLINTEGER        Attribute,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        StringLength);
```

Function arguments:

Table 88. SQLSetStmtAttr arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLINTEGER	*Attribute*	input	Option to set.

Table 88. SQLSetStmtAttr arguments (continued)

Data type	Argument	Use	Description
SQLPOINTER	*ValuePtr*	input	If *Attribute* is an ODBC-defined attribute and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *Attribute* is an ODBC-defined attribute and ValuePtr is an integer, *StringLength* is ignored. If *Attribute* is a DB2 CLI attribute, the application indicates the nature of the attribute by setting the *StringLength* argument. *StringLength* can have the following values: • If *ValuePtr* is a pointer to a character string, then *StringLength* is the length of the string or SQL_NTS. • If *ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in *StringLength*. This places a negative value in *StringLength*. • If *ValuePtr* is a pointer to a value other than a character string or a binary string, then *StringLength* should have the value SQL_IS_POINTER. • If *ValuePtr* contains a fixed-length value, then *StringLength* is either SQL_IS_INTEGER or SQL_UINTEGER.
SQLINTEGER	*StringLength*	input	If *ValuePtr* points to a character string or a binary buffer, this argument should be the length of *ValuePtr*. If *ValuePtr* is a pointer, but does not point to a string or binary buffer, then *StringLength* should have the value SQL_IS_POINTER. If *ValuePtr* is not a pointer, then *StringLength* should have the value SQL_IS_NOT_POINTER.

Usage:

Statement attributes for a statement remain in effect until they are changed by another call to `SQLSetStmtAttr()` or the statement is dropped by calling `SQLFreeHandle()`. Calling `SQLFreeStmt()` with the SQL_CLOSE, SQL_UNBIND, or SQL_RESET_PARAMS options does not reset statement attributes.

Some statement attributes support substitution of a similar value if the data source does not support the value specified in *ValuePtr*. In such cases, DB2 CLI returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed). For example, if Attribute is SQL_ATTR_CONCURRENCY, *ValuePtr* is SQL_CONCUR_ROWVER, and the data source does not support this, DB2 CLI substitutes SQL_CONCUR_VALUES and returns SQL_SUCCESS_WITH_INFO. To determine the substituted value, an application calls `SQLGetStmtAttr()`. The format of information set with *ValuePtr* depends on the specified Attribute.

SQLSetStmtAttr() accepts attribute information in one of two different formats: a null-terminated character string or a 32-bit integer value. The format of each is noted in the attribute's description. This format applies to the information returned for each attribute in SQLGetStmtAttr(). Character strings pointed to by the *ValuePtr* argument of SQLSetStmtAttr() have a length of *StringLength*.

The dirty bit:

DB2 Everyplace uses the dirty bit to track changes made to a record. The behavior of the dirty bit is affected by the SQL_ATTR_DELETE_MODE, SQL_ATTR_READ_MODE, and SQL_ATTR_DIRTYBIT_SET_MODE statement attributes. The following table shows the states of the dirty bit after certain database operations are performed on a record. The table assumes that the SQL_ATTR_DIRTYBIT_SET_MODE parameter is set to SQL_DIRTYBIT_SET_BY_SYSTEM with the dirty bit maintained by the system.

Table 89. DB2 Everyplace dirty bit states

Actions on a record	Dirty bit state
clean state (0) then INSERT	INSERT
clean state (0) then DELETE	DELETE
clean state (0) then UPDATE	UPDATE
DELETE then INSERT	UPDATE
DELETE then DELETE	Not applicable
DELETE then UPDATE	Not applicable
INSERT then INSERT	Not applicable
INSERT then DELETE	Physical removal of record
INSERT then UPDATE	INSERT
UPDATE then INSERT	Not applicable
UPDATE then DELETE	DELETE
UPDATE then UPDATE	UPDATE

The value of the dirty bit can be obtained by querying the $dirty column of a table. For example, the following statement returns the dirty bit and the NAME column of the PHONEBOOK table:

```
SELECT $dirty, NAME from PHONEBOOK
```

The dirty bit can have the following values.

Table 90. DB2 Everyplace dirty bit values

Description	Dirty bit value
Record unchanged (CLEAN)	0
Record deleted (DELETE)	1
Record inserted (INSERT)	2
Record updated (UPDATE)	3

Statement attributes:

The currently defined attributes are shown below.

SQL_ATTR_CURSOR_SCROLLABLE (DB2 CLI)
A 32-bit integer that specifies the level of support that the application requires. Setting this attribute affects subsequent calls to `SQLExecDirect()` and `SQLExecute()`. The supported values are:

- SQL_NONSCROLLABLE

  Scrollable cursors are not required on the statement handle. If the application calls `SQLFetchScroll()` on this handle, the only valid value of `FetchOrientation()` is SQL_FETCH_NEXT. This is the default.

- SQL_SCROLLABLE

  Scrollable cursors are required on the statement handle. When calling `SQLFetchScroll()`, the application might specify any valid value of `FetchOrientation` so that the cursor can be positioned in modes other than the sequential mode.

SQL_ATTR_CURSOR_SENSITIVITY (DB2 CLI)
A 32-bit integer value that specifies whether a cursor is sensitive to the write activity of another cursor. The supported values are:

- SQL_UNSPECIFIED

  The write activity of other cursors has an undefined impact on the current cursor. This is the default.

- SQL_INSENSITIVE

  The write activity of other cursors has no impact on the current cursor.

  Note: Use this attribute value sparingly because it can affect performance.

SQL_ATTR_CURSOR_TYPE (DB2 CLI)
A 32-bit integer value that specifies the cursor type. The supported values are:

- SQL_CURSOR_FORWARD_ONLY = The cursor scrolls forward only. This is the default.
- SQL_CURSOR_STATIC = The data in the result set is static.

This option cannot be specified for an open cursor.

SQL_ATTR_ROW_ARRAY_SIZE (DB2 CLI)
A 32-bit integer value that specifies the number of rows in the row set. This is the number of rows returned by each call to `SQLFetch()` or `SQLFetchScroll()`. The default value is 1. If the specified row set size exceeds the maximum row set size supported by the data source, DB2 CLI substitutes that value and returns SQLSTATE 01S02 (Option value changed). This option can be specified for an open cursor.

SQL_ATTR_ROW_BIND_TYPE (DB2 CLI)
A 32-bit integer value that sets the binding orientation to be used when `SQLFetch()` or `SQLFetchScroll()` is called on the associated statement. Column-wise binding is selected by supplying the defined constant SQL_BIND_BY_COLUMN in *ValuePtr*. The length specified in *ValuePtr* must include space for all of the bound columns and any padding of the structure or buffer to ensure that, when the address of a bound column is incremented with the specified length, the result points to the beginning of the same column in the next row. When using the **sizeof** operator with structures or unions in ANSI C, this behavior is guaranteed. Column-wise binding is the default binding orientation for SQLFetchScroll().

SQL_ATTR_ROW_NUMBER (DB2 CLI)
A 32-bit integer value that is the number of the current row in the entire result set. If the number of the current row cannot be determined or there is no current row, DB2 CLI returns 0. This attribute can be retrieved by a call to `SQLGetStmtAttr()`, but not set by a call to `SQLSetStmtAttr()`.

SQL_ATTR_ROW_STATUS_PTR (DB2 CLI)
A 16-bit unsigned integer value that points to an array of UWORD values containing row status values after a call to `SQLFetch()` or `SQLFetchScroll()`. The array has as many elements as there are rows in the row set. This statement attribute can be set to a null pointer, in which case DB2 CLI does not return row status values. This attribute can be set at any time, but the new value is not used until the next time `SQLFetch()` or `SQLFetchScroll()` is called.

SQL_ATTR_ROWS_FETCHED_PTR (DB2 CLI)
A 32-bit unsigned integer value that points to a buffer in which to return the number of rows fetched after a call to `SQLFetch()` or `SQLFetchScroll()`.

SQL_ATTR_DELETE_MODE (DB2 Everyplace)
The supported values are:

- SQL_DELETE_MARK_ONLY

  This is the system default. When a delete SQL statement is executed, records are only marked as ″delete″. The record contents can still be read if the SQL_READ_INCLUDE_MARKED_DELETE is set.

- SQL_DELETE_PHYSICAL_REMOVE

  A delete SQL statement physically removes the records meeting the WHERE clause condition, regardless of its dirty bit.

For example, use the following syntax to physically remove some records ignoring the status of the dirty bits:

```
SQLSetStmtAttr (stmt, SQL_ATTR_DELETE_MODE, SQL_DELETE_PHYSICAL_REMOVE, 0)
```

Next execute the following SQL statement to delete all records from table T where X is not equal to 0:

```
DELETE T WHERE X<>0
```

SQL_ATTR_DIRTYBIT_SET_MODE (DB2 Everyplace)
A 32-bit integer value that specifies the cursor type. The supported values are:

- SQL_DIRTYBIT_SET_BY_SYSTEM

  This is the system default. A record that is inserted, updated, or deleted has a dirty bit that is set to INSERT, UPDATE, or DELETE, respectively. No UPDATE of the $dirty column is allowed when the SQL_DIRTYBIT_SET_BY_SYSTEM is set.

- SQL_DIRTYBIT_SET_BY_APPLICATION

  The application is responsible for setting the dirty bit when inserting, updating, or deleting records. The semantics for each operation are:

  UPDATE
  The system sets the dirty bit exactly as specified by the application. For example, if an application executes the following statement then all records in the table are reset to 0 (CLEAN):

  ```
  UPDATE T SET $dirty=0 WHERE $dirty>0
  ```

  INSERT
  The dirty bit of the newly inserted record is set to CLEAN.

DELETE
If SQL_DELETE_PHYSICAL_REMOVE is set, DELETE physically removes records from the database. Otherwise, the values of the $dirty column are set to DELETE and the records remain in the database.

For example, to clean the dirty bit of a record use the following statement:

```
SQLSetStmtAttr (stmt, SQL_ATTR_DIRTYBIT_SET_MODE,
                SQL_DIRTYBIT_SET_BY_APPLICATION, 0)
```

Then execute the following SQL statement:

```
UPDATE T SET $DIRTY=0 WHERE $DIRTY>0
```

In general, applications can set SQL_DIRTYBIT_SET_BY_APPLICATION when the dirty bits are not needed for tracking database updates by end-users.

SQL_ATTR_READ_MODE (DB2 Everyplace)
A 32-bit integer value that specifies the cursor type. The supported values are:

- SQL_READ_EXCLUDE_MARKED_DELETE

  This is the system default. All records with the dirty bit set to ″delete″ are hidden from SQL.
- SQL_READ_INCLUDE_MARKED_DELETE

  Once set, the records with the dirty bit set to DELETE are visible from SQL SELECT statement. Applications can distinguish those deleted records from other records by examining the dirty bit for a record.

For example, use the following statement to read all records with the dirty bit set, including those with dirty bits marked as DELETE:

```
SQLSetStmtAttr (stmt, SQL_ATTR_READ_MODE, SQL_READ_INCLUDE_MARKED_DELETE, 0)
```

then execute the following SQL statement to retrieve all records:

```
SELECT * FROM T WHERE $dirty<>0
```

SQL_ATTR_REORG_MODE (DB2 Everyplace)
A 32-bit integer value that specifies whether automatic database reorganization is performed on user created tables and whether explicit REORG SQL statements are allowed. The supported values are:

- SQL_REORG_ENABLED - This is the system default. Database reorganization can be performed by DB2 Everyplace or explicitly by the user with a REORG SQL statement.
- SQL_REORG_DISABLED - REORG SQL statements are restricted and automatic database reorganization of user-created tables is disabled.

This option cannot be specified for an open cursor.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 91. SQLSetStmtAttr SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor is already opened on the statement handle.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error.	The function is called while in a data-at-execute (`SQLPrepare()` or `SQLExecDirect()`) operation. The function is called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY014	No more handles.	DB2 CLI is unable to allocate a handle due to internal resources.
HY090	Invalid string or buffer length.	The value of one of the name length arguments is less than 0, but not equal SQL_NTS.

Restrictions:

None.

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

SQLTables - Get Table Information

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

`SQLTables()` returns a list of table names and associated information stored in the system catalog of the connected data source. The list of table names is returned as a result set, which can be retrieved using the same functions that are used to process a result set generated by a query.

Syntax:

```
SQLRETURN   SQLTables        (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      FAR  *CatalogName,      /* szCatalogName */
                SQLSMALLINT       NameLength1,       /* cbCatalogName */
                SQLCHAR      FAR  *SchemaName,       /* szSchemaName */
                SQLSMALLINT       NameLength2,       /* cbSchemaName */
                SQLCHAR      FAR  *TableName,        /* szTableName */
                SQLSMALLINT       NameLength3,       /* cbTableName */
                SQLCHAR      FAR  *TableType,        /* szTableType */
                SQLSMALLINT       NameLength4);      /* cbTableType */
```

Function arguments:

Table 92. SQLTables arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLCHAR	*CatalogName*	Input	Buffer that may contain a *pattern-value* to qualify the result set. *Catalog* is the first part of a 3 part table name. This field is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength1*	Input	Length of *CatalogName*. This field is ignored by DB2 Everyplace.
SQLCHAR	*SchemaName*	Input	Buffer that may contain a *pattern-value* to qualify the result set by schema name. This field is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength2*	Input	Length of *SchemaName*. This field is ignored by DB2 Everyplace.
SQLCHAR	*TableName*	Input	Buffer that may contain a *pattern-value* to qualify the result set by table name.
SQLSMALLINT	*NameLength3*	Input	Length of *TableName*.
SQLCHAR	*TableType*	Input	DB2 Everyplace only supports type TABLE. This field is ignored by DB2 Everyplace.
SQLSMALLINT	*NameLength4*	Input	This field is ignored by DB2 Everyplace.

Note that the *TableName* arguments accept search patterns.

Usage:

Table information is returned in a result set where each table is represented by one row of the result set.

Sometimes, an application calls `SQLTables()` with null pointers *TableName* argument so that no attempt is made to restrict the result set returned. For some data sources that contain a large quantity of tables, this scenario maps to an extremely large result set and very long retrieval times.

The result set returned by `SQLTables()` contains the columns listed in Table 93 on page 265 in the order given. The rows are ordered by TABLE_NAME.

Calls to `SQLTables()` should be used sparingly, because in many cases they map to a complex and thus expensive query against the system catalog. The results should be saved rather than repeating calls.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside 128 characters (plus the null-terminator) for the output buffer, or alternatively, call `SQLGetInfo()` with the SQL_MAX_TABLE_NAME_LEN to determine the actual lengths of the TABLE_NAME column supported by the connected DBMS.

Table 93. Columns Returned By SQLTables

Column Name	Data type	Description
TABLE_CAT	VARCHAR(128)	This is always a zero-length string.
TABLE_SCHEM	VARCHAR(128)	This is always a zero-length string.
TABLE_NAME	VARCHAR(128)	The name of the table.
TABLE_TYPE	VARCHAR(128)	Identifies the type given by the name in the TABLE_NAME column. It always has the string value 'TABLE'.
REMARKS	VARCHAR(254)	Contains the descriptive information about the table.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 94. SQLTables SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to internal resources.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The valid of one of the name length arguments exceeded the maximum value supported for that data source. The maximum supported value can be obtained by calling the `SQLGetInfo()` function.

Restrictions:

None.

Related reference:

- "SQLGetInfo—Get general information" on page 240

Data conversion by DB2 CLI functions

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

For example, to bind a parameter marker that corresponds to an SQL data type of SQL_VARCHAR to an application's C buffer type of long integer, the appropriate `SQLBindParameter()` call would be:

```
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
                  SQL_VARCHAR, 0, 0, long_ptr, 0, NULL);
```

Table 95 shows the supported data conversions between C and SQL data types. The first column in Table 95 contains the SQL data type. The remaining columns represent the C data types. If the C data type column contains:

D The conversion is supported and is the default conversion for the SQL data type.

X DB2 Everyplace supports the conversion.

blank DB2 Everyplace does not support the conversion.

Limits on precision and scale and truncation and rounding rules for type conversions follow SQL syntax rules.

Table 95. Supported data conversions

SQL data type	Default conversion	Other supported conversions
BLOB	SQL C BINARY	SQL C CHAR
CHAR	SQL C CHAR	SQL C LONG SQL C SHORT SQL C TINYINT SQL C TYPE DATE SQL C TYPE TIME SQL C BINARY SQL C BIT SQL C TYPE TIMESTAMP
DATE	SQL C TYPE DATE	SQL C CHAR
DECIMAL	SQL C CHAR	SQL C LONG SQL C SHORT SQL C TINYINT SQL C BIT
INTEGER	SQL C LONG	SQL C CHAR SQL C SHORT SQL C TINYINT SQL C FLOAT SQL C DOUBLE SQL C BIT
SMALLINT	SQL C SHORT	SQL C CHAR SQL C LONG SQL C TINYINT SQL C FLOAT SQL C DOUBLE SQL C BIT

Table 95. Supported data conversions (continued)

SQL data type	Default conversion	Other supported conversions
TIME	SQL C TYPE TIME	SQL C CHAR
TIMESTAMP	SQL C TYPE TIMESTAMP	SQL C CHAR
VARCHAR	SQL C CHAR	SQL C LONG SQL C SHORT SQL C TINYINT SQL C TYPE DATE SQL C TYPE TIME SQL C BINARY SQL C BIT SQL C TYPE TIMESTAMP

Related reference:

- "Key to DB2 CLI function descriptions" on page 186
- "DB2 CLI function summary" on page 182

Supported JDBC methods

This chapter contains information about the JDBC methods that DB2 Everyplace supports. This chapter contains the following sections:

- "Overview of DB2 Everyplace JDBC support"
- "Interfaces in the java.sql package" on page 268
- "Interfaces in the javax.sql package" on page 285

Overview of DB2 Everyplace JDBC support

DB2 Everyplace supports a subset of methods defined in the Java Database Connectivity (JDBC) API specification offered in the Sun Java Developer's Kit. The information on JDBC methods that DB2 Everyplace supports is modified from Sun's Java Development Kit Version 1.4.1 documentation. DB2 Everyplace also supports extended Connection and Statement interfaces.

See "DB2eStatement class" on page 283 and "DB2eConnection class" on page 271 for more information.

The DB2 Everyplace JDBC driver is compatible with the JDBC Optional Package for CDC/Foundation Profile specified by JSR 169.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Blob interface" on page 268
- "Connection interface" on page 270
- "DB2eConnection class" on page 271
- "DatabaseMetaData interface" on page 272
- "Driver interface" on page 275
- "PreparedStatement interface" on page 275
- "ResultSet interface" on page 277
- "ResultSetMetaData interface" on page 281

Interfaces in the java.sql package

This chapter provides information about the JDBC methods in the java.sql package. The topics covered are:

Blob interface

The Blob interface represents (maps) an SQL BLOB in the Java™ programming language. An SQL BLOB is a built-in type that stores a binary large object as a column value in a row of a database table. A BLOB object is valid for the duration of the transaction in which it was created.

Methods in the interfaces ResultSet and PreparedStatement, such as getBlob and setBlob allow a programmer to access the SQL BLOB. The Blob interface provides methods for getting the length of an SQL BLOB (binary large object) value and for materializing a BLOB value on the client.

java.sql package

public interface **Blob**

Table 96 lists the methods in the Blob interface that are supported by DB2 Everyplace.

Table 96. Blob interface methods

Method return value type	Method
`InputStream`	**getBinaryStream()** Retrieves the BLOB designated by this BLOB instance as a stream.
`byte[]`	**getBytes**(long `pos`, int `length`) Returns as an array of bytes part or all of the BLOB value that this BLOB object designates.
`long`	**length()** Returns the number of bytes in the BLOB value designated by this BLOB object.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182

CallableStatement interface

The interface used to execute remote SQL stored procedures. The result parameter must be registered as an OUT parameter. The other parameters can be used for input, output or both. Parameters are referred to sequentially, by number. The first parameter is 1.

See the section called "The remote query and stored procedure adapter" in the *DB2 Everyplace Sync Server Administration Guide* for more details.

```
call <procedure-name> (?,?, ...)
```

IN parameter values are set using the set methods inherited from PreparedStatement. The type of all OUT parameters must be registered prior to executing the stored procedure; their values are retrieved after execution via the get methods provided here. The size of the output parameter is limited to 4K bytes.

A `CallableStatement` can return one `ResultSet`.

java.sql package

public interface **CallableStatement**

extends PreparedStatement

Table 97 lists the methods in the CallableStatement interface that are supported by DB2 Everyplace.

Table 97. CallableStatement interface methods

Method return value type	Method
`Blob`	**getBlob**(int `i`) JDBC 2.0 Gets the value of a JDBC BLOB parameter as a `Blob` object in the Java programming language.
`byte[]`	**getBytes**(int `parameterIndex`) Gets the value of a JDBC BINARY or VARBINARY parameter as an array of `byte` values in the Java programming language.
`Date`	**getDate**(int `parameterIndex`) Gets the value of a JDBC DATE parameter as a `java.sql.Date` object.
`int`	**getInt**(int `parameterIndex`) Gets the value of a JDBC INTEGER parameter as an `int` in the Java programming language.
`Object`	**getObject**(int `parameterIndex`) Gets the value of a parameter as an `object` in the Java programming language.
`short`	**getShort**(int `parameterIndex`) Gets the value of a JDBC SMALLINT parameter as a `short` in the Java programming language.
`String`	**getString**(int `parameterIndex`) Retrieves the value of a JDBC CHAR, VARCHAR, or LONGVARCHAR parameter as a `String` in the Java programming language.
`Time`	**getTime**(int `parameterIndex`) Gets the value of a JDBC TIME parameter as a `java.sql.Time` object.
`Timestamp`	**getTimestamp**(int `parameterIndex`) Gets the value of a JDBC TIMESTAMP parameter as a `java.sql.Timestamp` object.

Table 97. CallableStatement interface methods (continued)

Method return value type	Method
`void`	**registerOutParameter**(int `parameterIndex`, int `sqlType`) Registers the OUT parameter in ordinal position `parameterIndex` to the JDBC type `sqlType`.
`boolean`	**wasNull**() Indicates whether or not the last OUT parameter read had the value of SQL NULL.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182

Connection interface

The Connection interface establishes a connection (session) with a specific database. Within the context of a Connection, SQL statements are executed and results are returned.

A Connection's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on. This information is obtained with the getMetaData method.

java.sql package

public interface **Connection**

Table 98 lists the methods in the Connection interface that are supported by DB2 Everyplace.

Table 98. Connection interface methods

Method return value type	Method
`void`	**clearWarnings**() Clears all warnings reported for this Connection object.
`void`	**close**() Releases a Connection's database and JDBC resources immediately instead of waiting for them to be automatically released.
`void`	**commit**() Makes all changes made since the previous commit or rollback permanent and releases any database locks currently held by the Connection.
`Statement`	**createStatement**() Creates a `Statement` object for sending SQL statements to the database.
`Statement`	**createStatement**(int `resultSetType`, int `resultSetConcurrency`) JDBC 2.0. Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
`boolean`	**isClosed**() Tests to see if a Connection is closed.
`DatabaseMetaData`	**getMetaData**() Gets the metadata regarding this Connection's database.

Table 98. Connection interface methods (continued)

Method return value type	Method
`SQLWarning`	**getWarnings**() Returns the first warning reported by calls on this Connection.
`CallableStatement`	**prepareCall**(String `sql`) Creates a `CallableStatement` object for calling database stored procedures.
`PreparedStatement`	**prepareStatement** (String `sql`) Creates a `PreparedStatement` object for sending parameterized SQL statements to the database.
`PreparedStatement`	**prepareStatement**(String `sql`, int `resultSetType`, int `resultSetConcurrency`) JDBC 2.0. Creates a PreparedStatement object that will generate ResultSet objects with the given type and concurrency.
`void`	**rollback**() Drops all changes made since the previous commit or rollback and releases any database locks currently held by this Connection.
`void`	**setAutoCommit** (boolean `autoCommit`) Sets this connection's auto-commit mode.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182

DB2eConnection class

The DB2eConnection class gets and sets certain Connection attributes. To use the DB2eConnection class methods on a Connection object, the Connection object must first be cast to a DB2eConnection object. These methods are implemented by calls to the CLI/ODBC functions SQLGetConnectAttr and SQLSetConnectAttr with the appropriate arguments.

See Chapter 10, Supported DB2 CLI/ODBC functions for more information.

com.ibm.db2e.jdbc package

public class **DB2eConnection**

implements Connection

Table 99 lists the methods in the DB2eConnection class supported by DB2 Everyplace.

Table 99. DB2eConnection class methods

Method return type	Method
`void`	**enableFilenameFormat83** (`boolean enable`) Enables the database engine to create filenames in 8.3 format if enable is true, otherwise, enables filenames in long format. Filename format can only be changed if no catalog files exist in the path for this connection.
`boolean`	**isEnabledFilenameFormat83** () Does the database engine create filenames in 8.3 format? Or does it create filenames in long format?

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182
- "SQLGetConnectAttr—Get current setting of a connection attribute" on page 230
- "SQLSetConnectAttr—Set options related to a connection" on page 254

DatabaseMetaData interface

The DatabaseMetaData interface provides comprehensive information about the database as a whole.

Some of these methods take String arguments for catalog and schema names. These arguments are ignored by DB2 Everyplace.

Some of the methods here return lists of information in the form of ResultSet objects. You can use the normal ResultSet methods such as getString and getInt to retrieve the data from these ResultSets.

If a given form of metadata is not available, these methods throw an SQLException.

java.sql package

public interface **DatabaseMetaData**

Table 100 lists the fields in the DatabaseMetaData interface that are supported by DB2 Everyplace.

Table 100. DatabaseMetaData fields

Field type	Field
`static int`	**columnNoNulls** Indicates that the column might not allow NULL values.
`static int`	**columnNullable** Indicates that the column definitely allows NULL values.
`static int`	**columnNullableUnknown** Indicates that the nullability of columns is unknown.

Table 101 lists the methods in the DatabaseMetaData interface that are supported by DB2 Everyplace.

Table 101. DatabaseMetaData interface methods

Method return value type	Method
`ResultSet`	**getColumns**(String `catalog`, String `schemaPattern`, String `tableNamePattern`, String `columnNamePattern`) Gets a description of table columns available in the specified catalog.
`Connection`	**getConnection**() JDBC 2.0 Retrieves the connection that produced this metadata object.

Table 101. DatabaseMetaData interface methods (continued)

Method return value type	Method
`ResultSet`	**getCrossReference**(String `primaryCatalog`, String `primarySchema`, String `primaryTable`, String `foreignCatalog`, String `foreignSchema`, String `foreignTable`) Gets a description of the foreign key columns in the foreign key table that reference the primary key columns of the primary key table (describe how one table imports another's key.) This should normally return a single foreign key/primary key pair (most tables only import a foreign key from a table once.) They are ordered by FKTABLE_NAME and KEY_SEQ.
`String`	**getDatabaseProductName**() What is the name of this database product?
`String`	**getDatabaseProductVersion**() What is the version of this database product?
`int`	**getDriverMajorVersion**() What is this JDBC driver's major version number?
`int`	**getDriverMinorVersion**() What is this JDBC driver's minor version number?
`String`	**getDriverName**() What is the name of this JDBC driver?
`String`	**getDriverVersion**() What is the version of this JDBC driver?
`ResultSet`	**getExportedKeys**(String `catalog`, String `schema`, String `table`) Gets a description of the foreign key columns that reference a table's primary key columns (the foreign keys exported by a table).
`String`	**getIdentifierQuoteString**() What is the string used to quote SQL identifiers? This returns a space " " if identifier quoting is not supported.
`ResultSet`	**getImportedKeys**(String `catalog`, String `schema`, String `table`) Gets a description of the primary key columns that are referenced by a table's foreign key columns (the primary keys imported by a table).
`int`	**getMaxBinaryLiteralLength**() How many hex characters can you have in an inline binary literal?
`int`	**getMaxCharLiteralLength**() What is the maximum length for a character literal?
`int`	**getMaxColumnNameLength**() What is the limit of the column name length?
`int`	**getMaxColumnsInGroupBy**() What is the maximum number of columns in a GROUP BY clause?
`int`	**getMaxColumnsInIndex**() What is the maximum number of columns allowed in an index?
`int`	**getMaxColumnsInOrderBy**() What is the maximum number of columns in an ORDER BY clause?
`int`	**getMaxColumnsInSelect**() What is the maximum number of columns in a SELECT statement?
`int`	**getMaxConnections**() How many active connections can we have at a time to this database?
`int`	**getMaxIndexLength**() What is the maximum length of an index (in bytes)?
`int`	**getMaxRowSize**() What is the maximum length of a single row?

Table 101. DatabaseMetaData interface methods (continued)

Method return value type	Method
`int`	**getMaxStatementLength()** What is the maximum length of a SQL statement?
`int`	**getMaxStatements()** How many active statements can we have open at one time to this database?
`int`	**getMaxTableNameLength()** What is the maximum length of a table name?
`int`	**getMaxTablesInSelect()** What is the maximum number of tables in a SELECT statement?
`int`	**getMaxUserNameLength()** What is the maximum length of a user name?
`ResultSet`	**getPrimaryKeys**(String `catalog`, String `schema`, String `table`) Gets a description of a table's primary key columns.
`String`	**getSearchStringEscape()** Gets the string that can be used to escape wildcard characters.
`ResultSet`	**getTables**(String `catalog`, String `schemaPattern`, String `tableNamePattern`, String[] `types`) Gets a description of tables available in a catalog.
`ResultSet`	**getUDTs**(String `catalog`, String `schemaPattern`, String `typeNamePattern`, int[] `types`) JDBC 2.0 Gets a description of the user-defined types defined in a particular schema. DB2 Everyplace always returns an empty result set since it does not support UDTs.
`String`	**getURL()** What is the URL for this database?
`String`	**getUserName()** What is the user name as it is known to the database?
`boolean`	**supportsColumnAliasing()** Is column aliasing supported?
`boolean`	**supportsFullOuterJoins()** Are full nested outer joins supported?
`boolean`	**supportsMixedCaseIdentifiers()** Does the database treat mixed case unquoted SQL identifiers as case sensitive and as a result store them in mixed case?
`boolean`	**supportsMixedCaseQuotedIdentifiers()** Does the database treat mixed case quoted SQL identifiers as case sensitive and as a result store them in mixed case?
`boolean`	**supportsNonNullableColumns()** Can columns be defined as non-nullable?
`boolean`	**supportsOrderByUnrelated()** Can an ″ORDER BY″ clause use columns not in the SELECT statement?
`boolean`	**supportsOuterJoins()** Is some form of outer join supported?
`boolean`	**supportsPositionedDelete()** Is positioned DELETE supported?
`boolean`	**supportsPositionedUpdate()** Is positioned UPDATE supported?
`boolean`	**supportsResultSetType**(`int type`) JDBC 2.0 Does the database support the given result set type?
`boolean`	**supportsSchemasInTableDefinitions()** Can a schema name be used in a table definition statement?
`boolean`	**supportsTransactions()** Are transactions supported? If not, the isolation level is TRANSACTION_NONE.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182

Driver interface

The Driver interface is the Java SQL framework that allows for multiple database drivers.

When a Driver class is loaded, it should create an instance of itself and register it with the DriverManager. This means that a user can load and register the DB2 Everyplace JDBC driver by calling:

```
Class.forName("com.ibm.db2e.jdbc.DB2eDriver")
```

java.sql package

public interface **Driver**

Table 102 lists the methods in the Driver interface that are supported by DB2 Everyplace.

Table 102. Driver interface methods

Method return value type	Method
`boolean`	**acceptsURL**(String `url`)Returns true if the driver thinks that it can open a connection to the given URL.
`Connection`	**connect**(String `url`, Properties info) Attempts to make a database connection to the given URL. The java.util.Properties argument can be used to pass arbitrary string tag/value pairs as connection arguments. DB2 Everyplace supports the following driver-specific key and value pairs: • Key: LOGIN_TIMEOUT Value: number of seconds • Key: DB2e_ENCODING Value: character encoding
`int`	**getMajorVersion**() Gets the driver's major version number.
`int`	**getMinorVersion**() Gets the driver's minor version number.
`boolean`	**jdbcCompliant**() Reports whether this driver is a genuine JDBC COMPLIANT™ driver.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182

PreparedStatement interface

The PreparedStatement interface creates an object that represents a precompiled SQL statement.

A SQL statement is pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

java.sql package

public interface **PreparedStatement**

extends Statement

Table 103 lists the methods in the PreparedStatement interface that are supported by DB2 Everyplace.

Table 103. PreparedStatement interface methods

Method return value type	Method
`void`	**clearParameters**() Clears the current parameter values immediately.
`boolean`	**execute**() Executes any kind of SQL statement.
`ResultSet`	**executeQuery**() Executes the SQL query in this `PreparedStatement` object and returns the result set generated by the query.
`int`	**executeUpdate**() Executes the SQL INSERT, UPDATE or DELETE statement in this `PreparedStatement` object.
`void`	**setBigDecimal** (int `parameterIndex`, `BigDecimal x`) Sets the designated parameter to a java.lang.BigDecimal value. This method is not available in the DB2 Everyplace JDBC driver for Palm OS.
`void`	**setBoolean** (int `parameterIndex`, boolean `x`) Sets the designated parameter to a Java `boolean` value. The DB2 Everyplace JDBC driver converts this to an SQL SMALLINT value when it sends it to the database.
`void`	**setBlob**(int `i`, Blob `x`) JDBC 2.0 Sets a BLOB parameter.
`void`	**setBytes**(int `parameterIndex`, byte[ ]`x`) Sets the designated parameter to a Java array of bytes.
`void`	**setDate**(int `parameterIndex`, Date `x`) Sets the designated parameter to a java.sql.Date value.
`void`	**setDouble**(int `parameterIndex`, double `x`) Sets the designated parameter to a Java double value. The DB2 Everyplace JDBC driver converts this to an SQL DECIMAL value when it sends it to the database.
`void`	**setFloat**(int `parameterIndex`, float `x`) Sets the designated parameter to a Java float value. When a BigDecimal is converted to float, if the BigDecimal is too large to represent as a float, it will be converted to FLOAT.NEGATIVE_INFINITY or FLOAT.POSITIVE_INFINITY as appropriate.
`void`	**setInt** (int `parameterIndex`, int `x`) Sets the designated parameter to a Java `int` value.
`void`	**setLong**(int `parameterIndex`, long `x`) Sets the designated parameter to a Java long value.
`void`	**setNull** (int `parameterIndex`, int `sqlType`) Sets the designated parameter to SQL NULL.

Table 103. PreparedStatement interface methods (continued)

Method return value type	Method
`void`	**setObject**(int `parameterIndex`, Object `x`, int `targetSqlType`) Sets the value of the designated parameter with the given object. **DB2 Everyplace restrictions**: • `targetSqlType` must correspond with one of the data types DB2 Everyplace supports. • The basic and String conversions are supported. For example, if `targetSqlType` is `Types.INTEGER`, x should be either an `Integer` or a `String` object. • If `targetSqlType` is `Types.DECIMAL`, x can also be a `Double`, `Float`, or `Long` object. • If `targetSqlType` is `Types.SMALLINT`, x can also be a `Boolean` object. • On Palm OS, if `targetSqlType` is `Types.DECIMAL`, x should be a `String` object.
`void`	**setShort** (int `parameterIndex`, short `x`) Sets the designated parameter to a Java `short` value.
`void`	**setString** (int `parameterIndex`, String `x`) Sets the designated parameter to a Java `String` value.
`void`	**setTime** (int `parameterIndex`, Time `x`) Sets the designated parameter to a java.sql.Time value.
`void`	**setTimestamp** (int `parameterIndex`, Timestamp `x`) Sets the designated parameter to a java.sql.Timestamp value.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182

ResultSet interface

The ResultSet interface provides access to a table of data. A ResultSet object is usually generated by executing a Statement.

A ResultSet maintains a cursor pointing to its current row of data. Initially, the cursor is positioned before the first row. The `next()` method moves the cursor to the next row.

The getXXX methods retrieve column values for the current row. You can retrieve values using either the index number of the column or the name of the column. In general, using the column index is more efficient. Columns are numbered from one.

java.sql package

public interface **ResultSet**

Table 104 on page 278 lists the fields in the ResultSet interface that are supported by DB2 Everyplace.

Table 104. ResultSet interface fields

Field type	Field
`static int`	**CONCUR_READ_ONLY** The constant indicating the concurrency mode for a ResultSet object that may NOT be updated. **Note**: CONCUR_UPDATABLE is not supported by DB2 Everyplace. If CONCUR_UPDATABLE is specified for the concurrency mode for a ResultSet object when creating a Statement object, the DB2 Everyplace JDBC driver issues an SQLWarning on the Connection object that produced the Statement object, and uses CONCUR_READ_ONLY instead.
`static int`	**TYPE_FORWARD_ONLY** The constant indicating the type for a ResultSet object whose cursor can move only forward.
`static int`	**TYPE_SCROLL_INSENSITIVE** The constant indicating the type for a ResultSet object that is scrollable but generally not sensitive to changes made by others. **Note**: Use this type of ResultSet object sparingly, as it might affect performance. This type uses SQL_INSENSITIVE for the value of the CLI statement attribute SQL_ATTR_CURSOR_SENSITIVITY. Refer to the documentation for the CLI function SQLSetStmtAttr for details.
`static int`	**TYPE_SCROLL_SENSITIVE** The constant indicating the type for a ResultSet object that is scrollable and generally sensitive to changes made by others.**Note**: This type uses SQL_UNSPECIFIED for the value of the CLI statement attribute SQL_ATTR_CURSOR_SENSITIVITY. Refer to the documentation for the CLI function SQLSetStmtAttr for details.

Table 105 lists the methods in the ResultSet interface that are supported by DB2 Everyplace.

Table 105. ResultSet interface methods

Method return value type	Method
`boolean`	**absolute**(int `row`) JDBC 2.0. Moves the cursor to the given row number in the result set.
`void`	**afterLast**() JDBC 2.0. Moves the cursor to the end of the result set, just after the last row.
`void`	**beforeFirst**() JDBC 2.0. Moves the cursor to the front of the result set, just before the first row.
`void`	**clearWarnings**() Clears all warnings reported on this ResultSet object.
`void`	**close**() Releases this ResultSet object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.
`int`	**findColumn**(String `columnName`) Maps the given ResultSet column name to its ResultSet column index.
`boolean`	**first**() JDBC 2.0. Moves the cursor to the first row in the result set.
`BigDecimal`	**getBigDecimal**(int `columnIndex`) JDBC 2.0. Gets the value of a column in the current row as a java.math.BigDecimal object with full precision. This method is not supported by the DB2 Everyplace JDBC driver for Palm OS.

Table 105. ResultSet interface methods (continued)

Method return value type	Method
`BigDecimal`	**getBigDecimal**(int `columnIndex`, int `scale`) Gets the value of the designated column in the current row of this ResultSet object as a java.math.BigDecimal object in the Java programming language. This method is not supported by the DB2 Everyplace JDBC driver for Palm OS. **Deprecated.**
`BigDecimal`	**getBigDecimal**(String `columnName`) JDBC 2.0. Gets the value of a column in the current row as a java.math.BigDecimal object with full precision. This method is not supported by the DB2 Everyplace JDBC driver for Palm OS.
`BigDecimal`	**getBigDecimal**(String `columnName`, int `scale`) Gets the value of the designated column in the current row of this ResultSet object as a java.math.BigDecimal object in the Java programming language. This method is not supported by the DB2 Everyplace JDBC driver for Palm OS. **Deprecated.**
`Blob`	**getBlob**(int `columnIndex`) JDBC 2.0. Gets a BLOB value in the current row of this ResultSet object.
`Blob`	**getBlob**(String `columnName`) JDBC 2.0. Gets a BLOB value in the current row of this ResultSet object.
`boolean`	**getBoolean**(int `columnIndex`) Gets the value of a column in the current row as a Java boolean.
`boolean`	**getBoolean**(String `columnName`) Gets the value of a column in the current row as a Java boolean.
`byte`	**getByte**(int `columnIndex`) Gets the value of the designated column in the current row of this ResultSet object as a byte in the Java programming language.
`byte`	**getByte**(String `columnName`) Gets the value of the designated column in the current row of this ResultSet object as a byte in the Java programming language.
`byte[]`	**getBytes**(int `columnIndex`) Gets the value of the designated column in the current row of this ResultSet object as a byte array in the Java programming language.
`byte[]`	**getBytes**(String `columnName`) Gets the value of the designated column in the current row of this ResultSet object as a byte array in the Java programming language.
`int`	**getConcurrency**() JDBC 2.0. Returns the concurrency mode of the result set.
`Date`	**getDate**(int `columnIndex`) Gets the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
`Date`	**getDate**(int `columnIndex`, Calendar `cal`) Returns the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
`Date`	**getDate**(String `columnName`) Gets the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
`double`	**getDouble**(int `columnIndex`) Gets the value of a column in the current row as a Java double.
`double`	**getDouble**(String `columnName`) Gets the value of a column in the current row as a Java double.

Table 105. ResultSet interface methods (continued)

Method return value type	Method
`float`	**getFloat**(int `columnIndex`) Gets the value of a column in the current row as a Java float.
`float`	**getFloat**(String `columnName`) Gets the value of a column in the current row as a Java float.
`int`	**getInt**(int `columnIndex`) Gets the value of the designated column in the current row of this ResultSet object as an integer in the Java programming language.
`int`	**getInt**(String `columnName`) Gets the value of the designated column in the current row of this ResultSet object as an integer in the Java programming language.
`long`	**getLong**(int `columnIndex`) Gets the value of a column in the current row as a Java long.
`long`	**getLong**(String `columnName`) Gets the value of a column in the current row as a Java long.
`ResultSetMetaData`	**getMetaData**() Retrieves the number, types, and properties of this ResultSet object's columns.
`Object`	**getObject**(int `columnIndex`) Gets the value of a column in the current row as a Java object.
`Object`	**getObject**(String `columnName`) Gets the value of a column in the current row as a Java object.
`int`	**getRow**() JDBC 2.0. Retrieves the current row number.
`short`	**getShort**(int `columnIndex`)Gets the value of the designated column in the current row of this ResultSet object as a short in the Java programming language.
`short`	**getShort**(String `columnName`) Gets the value of the designated column in the current row of this ResultSet object as a short in the Java programming language.
`Statement`	**getStatement**() JDBC 2.0. Returns the Statement that produced this ResultSet object.
`String`	**getString**(int `columnIndex`) Gets the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
`String`	**getString**(String `columnName`) Gets the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
`Time`	**getTime**(int `columnIndex`) Gets the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.
`Time`	**getTime**(String `columnName`) Gets the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.
`Timestamp`	**getTimestamp**(String `columnName`) Gets the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.
`Timestamp`	**getTimestamp**(int `columnIndex`) Gets the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.
`int`	**getType**() JDBC 2.0. Returns the type of this result set.

Table 105. ResultSet interface methods (continued)

Method return value type	Method
`SQLWarning`	**getWarnings**() The first warning reported by calls on this ResultSet is returned.
`boolean`	**isAfterLast**() JDBC 2.0. Indicates whether the cursor is after the last row in the result set.
`boolean`	**isBeforeFirst**() JDBC 2.0. Indicates whether the cursor is before the first row in the result set.
`boolean`	**isFirst**() JDBC 2.0. Indicates whether the cursor is on the first row of the result set.
`boolean`	**isLast**() JDBC 2.0. Indicates whether the cursor is on the last row of the result set. This method is not supported for result sets with type TYPE_FORWARD_ONLY.
`boolean`	**last**() JDBC 2.0. Moves the cursor to the last row in the result set.
`boolean`	**next**() Moves the cursor down one row from its current position.
`boolean`	**previous**() JDBC 2.0. Moves the cursor to the previous row in the result set.
`boolean`	**relative**(int `rows`) JDBC 2.0. Moves the cursor a relative number of rows, either positive or negative.
`boolean`	**wasNull**() Reports whether the last column read had a value of SQL NULL.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182
- "SQLSetStmtAttr—Set options related to a statement" on page 257

ResultSetMetaData interface

The ResultSetMetaData interface creates an object that can be used to find out about the types and properties of the columns in a ResultSet.

java.sql package

public interface **ResultSetMetaData**

Table 106 lists the fields in the ResultSetMetaData interface that are supported by DB2 Everyplace.

Table 106. ResultSetMetaData interface fields

Field type	Field
`static int`	**columnNoNulls** The constant indicating that a column does not allow NULL values.
`static int`	**columnNullable** The constant indicating that a column allows NULL values.
`static int`	**columnNullableUnknown** The constant indicating that the nullability of a column's values is unknown.

Table 107 lists the methods in the ResultSetMetaData interface that are supported by DB2 Everyplace.

Table 107. ResultSetMetaData interface methods

Method return value type	Method
`String`	**getCatalogName**(int `column`) Gets a column's table's catalog name. DB2 Everyplace always returns "" (not applicable).
`int`	**getColumnCount**() Returns the number of columns in this `ResultSet` object.
`int`	**getColumnDisplaySize** (int `column`)Indicates the designated column's normal maximum width in characters.
`String`	**getColumnLabel**(int `column`) Gets the suggested column title for use in printouts and displays.
`String`	**getColumnName** (int `column`) Gets the designated column's name.
`int`	**getColumnType** (int `column`) Gets the designated column's SQL type.
`String`	**getColumnTypeName**(int `column`) Retrieves a column's database-specific type name.
`int`	**getPrecision** (int `column`) Gets the designated column's number of decimal digits.
`int`	**getScale** (int `column`) Gets the designated column's number of digits to the right of the decimal point.
`String`	**getSchemaName**(int `column`) Gets a column's table's schema name. DB2 Everyplace always returns "" (not applicable).
`int`	**isNullable** (int `column`) Indicates the nullability of values in the designated column.
`boolean`	**isWritable**(int `column`) Indicates whether it is possible for a write on the column to succeed.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182

Statement interface

The Statement interface creates an object that is used to execute a static SQL statement and obtain the results produced by it.

java.sql package

public interface **Statement**

Table 108 on page 283 lists the methods in the Statement interface that are supported by DB2 Everyplace.

Table 108. Statement interface methods

Method return value type	Method
`void`	**addBatch**(String `sql`) JDBC 2.0 Adds a SQL command to the current batch of commmands for the statement.
`void`	**clearBatch**() JDBC 2.0 Makes the set of commands in the current batch empty.
`void`	**close**() Releases this `Statement` object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closes.
`boolean`	**execute**(String `sql`) Executes an SQL statement that might return multiple results.
`int[]`	**executeBatch**() JDBC 2.0 Submits a batch of commands to the database for execution.
`ResultSet`	**executeQuery**(String `sql`) Executes an SQL statement that returns a single `ResultSet` object.
`int`	**executeUpdate**(String `sql`) Executes an SQL `INSERT`, `UPDATE`, or `DELETE` statement.
`Connection`	**getConnection** () JDBC 2.0. Returns the Connection object that produced this Statement object.
`boolean`	**getMoreResults**() Moves to a Statement's next result. DB2 Everyplace always returns false (there are no more results).
`ResultSet`	**getResultSet**() Returns the current result as a `ResultSet` object.
`int`	**getResultSetConcurrency**() JDBC 2.0. Retrieves the result set concurrency.
`int`	**getResultSetType**() JDBC 2.0. Determines the result set type.
`int`	**getUpdateCount**() Returns the current result as an update count; if the result is a ResultSet or there are no more results, -1 is returned.

Table 109 lists the fields in the Statement interface that are supported by DB2 Everyplace.

Table 109. Statement interface fields

Field type	Field
`static int`	**SUCCESS_NO_INFO** The constant indicating that a batch statement executed successfully, but that no count of the number of rows it affected is available.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182

DB2eStatement class

The DB2eStatement class gets and sets certain Statement attributes. To use the DB2eStatement class methods on a Statement object, the Statement object must first

be cast to a DB2eStatement object. These methods are implemented by calls to the CLI/ODBC functions SQLGetStmtAttr and SQLSetStmtAttr with the appropriate arguments.

See “DB2 CLI function summary” on page 182 for more information.

com.ibm.db2e.jdbc package

public class **DB2eStatement**

implements Statement

Table 110 lists the methods in the DB2eStatement class supported by DB2 Everyplace.

Table 110. DB2eStatement class methods

Method return type	Method
`void`	**enableDeletePhysicalRemove**(boolean `enable`) Enables or disables physically removing records, regardless of their dirty bit values, in a DELETE SQL statement.
`void`	**enableDirtyBitSetByApplication**(boolean `enable`) Enables the application mode if enable is true. Otherwise, enables the system mode.
`void`	**enableReadIncludeMarkedDelete** (boolean `enable`) Makes logically deleted records visible or invisible.
`void`	**enableReorg** (boolean `enable`) Enables or disables database reorganization by DB2 Everyplace or explicitly by the user with a REORG SQL statement.
`boolean`	**isEnabledDeletePhysicalRemove**() Will a delete SQL statement physically remove the records regardless of their dirty bit values? Or will the records only be marked as ″delete″?
`boolean`	**isEnabledDirtyBitSetByApplication**() Is the database system in the application mode? Or is it in the system mode?
`boolean`	**isEnabledReadIncludeMarkedDelete**() Are logically deleted records visible from SQL statements? Or are these records hidden from SQL?
`boolean`	**isEnabledReorg**() Can database reorganization be done by DB2 Everyplace or explicitly by the user with a REORG SQL statement? Or are REORG SQL statements restricted and is the automatic database reorganization of user-created tables disabled?

In these examples, `st` represents a Statement object, and `rs` represents a ResultSet object.

To physically remove some records from table T ignoring the status of the dirty bits:

```
DB2eStatement db2e_st = (DB2eStatement) st;
db2e_st.enableDeletePhysicalRemove(true);
st.executeUpdate("DELETE FROM T WHERE X<>0");
```

To read all records in table T with the dirty bit set, including those with dirty bit marked as DELETE:

```
DB2eStatement db2e_st = (DB2eStatement) st;
db2e_st.enableReadIncludeMarkedDelete(true);
rs = st.executeQuery("SELECT * FROM T WHERE $dirty<>0");
```

To clean the dirty bit of a record in table T:

```
DB2eStatement db2e_st = (DB2eStatement) st;
db2e_st.enableDirtyBitSetByApplication(true);
st.executeUpdate("UPDATE T SET $dirty=0 WHERE $dirty>0");
```

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182
- "SQLGetStmtAttr—Get current setting of a statement attribute" on page 243
- "SQLSetStmtAttr—Set options related to a statement" on page 257

Interfaces in the javax.sql package

This chapter provides information about the JDBC methods in the javax.sql package. The topics covered are:

- "DataSource interface"

DataSource interface

A factory for connections to the physical data source that this DataSource object represents. A replacement for the DriverManager facility, a DataSource object is the preferred means of getting a connection.

An instance of a DataSource object can be used in a stand alone program to create Connection objects. In the following example, an instance of DB2eDataSource is used to create a Connection to a DB2 Everyplace database with url "jdbc:db2e:myDataSource":

```
com.ibm.db2e.jdbc.DB2eDataSource ds = new com.ibm.db2e.jdbc.DB2eDataSource();
    ds.setUrl("jdbc:db2e:myDataSource");
    Connection con = ds.getConnection();
```

javax.sql package

public interface **DataSource**

Table 111 and Table 112 on page 286 list the properties for the DataSource interface that are supported by DB2 Everyplace. The properties can be accessed using "getter" and "setter" methods. (DataSource properties follow the convention specified for properties of JavaBeans™ components in the JavaBeans 1.01 Specification.)

Table 111. Standard DataSource properties supported by DB2 Everyplace

Property Name	Type	Description
description	String	description of this data source
password	String	database password
user	String	user's account name

Table 112 on page 286 lists the supported properties for the DataSource interface that are specific to DB2 Everyplace.

Table 112. DB2 Everyplace-specific properties for the DataSource interface

Property Name	Type	Description
encoding	String	character encoding
URL	String	data source

Table 113 lists the methods in the DataSource interface that are supported by DB2 Everyplace.

Table 113. DataSource interface methods

Method return value type	Method
`Connection`	**getConnection**() Attempts to establish a connection with the data source that this DataSource object represents.
`Connection`	**getConnection** (java.lang.String `username`, java.lang.String `password`) Attempts to establish a connection with the data source that this DataSource object represents.
`int`	**getLoginTimeout**() Gets the maximum time in seconds that this data source can wait while attempting to connect to a database.
`java.io.PrintWriter`	**getLogWriter**() Retrieves the log writer for this DataSource object.
`void`	**setLoginTimeout**(int `seconds`) Sets the maximum time in seconds that this data source will wait while attempting to connect to a database.
`void`	**setLogWriter**(java.io.PrintWriter `out`) Sets the log writer for this DataSource object to the given java.io.PrintWriter object.

Related tasks:

- "Developing DB2 Everyplace Java applications" on page 17

Related reference:

- "Overview of DB2 Everyplace JDBC support" on page 267
- "SQLState messages reported by JDBC" on page 182

Supported .NET classes

This chapter contains information about the .NET classes that DB2 Everyplace supports. This chapter contains the following sections:

DB2eCommandBuilder Members

Table 114. Public Static (Shared) Methods

Method	Description
DeriveParameters	Retrieves parameter information from the stored procedure specified in the *DB2eCommand* and populates the Parameters collection of the specified *DB2eCommand* object.

Table 115. Public Instance Constructors

Constructor	Description
DB2eCommandBuilder()	Overloaded. Initialize a new instance of the *DB2eCommandBuilder* class.
DB2eCommandBuilder(DB2eDataAdapter)	Overloaded. Initialize a new instance of the *DB2eCommandBuilder* class with the associated *DB2eDataAdapter* object.

Table 116. Public Instance Properties

Property	Description
DataAdapter	Gets or sets an *DB2eDataAdapter* object for which this *DB2eCommandBuilder* object will generate SQL statements.

Table 117. Public Instance Methods

Method	Description
GetDeleteCommand	Gets the automatically generated *DB2eCommand* object required to perform deletions at the database.
GetInsertCommand	Gets the automatically generated *DB2eCommand* object required to perform insertions at the database.
GetUpdateCommand	Gets the automatically generated *DB2eCommand* object required to perform updates at the database.
RefreshSchema	Refreshes the database schema information used to generate INSERT, UPDATE, or DELETE statements.

Table 118. Protected Instance Methods

Method	Description
Dispose	Overloaded.

DB2eCommand Members

Table 119. Public Instance Constructors

Constructor	Description
DB2eCommand()	Overloaded. Initialize a new instance of the *DB2eCommand* class.
DB2eCommand(string)	Overloaded. Initialize a new instance of the *DB2eCommand* class with the text of the query.
DB2eCommand(string, DB2eConnection)	Overloaded. Initialize a new instance of the *DB2eCommand* class with the text of the query and an *DB2eConnection* object.
DB2eCommand(string, DB2eConnection, DB2eTransaction)	Overloaded. Initialize a new instance of the *DB2eCommand* class with the text of the query, an *DB2eConnection* object, and the *DB2eTransaction* object.

Table 120. Public Instance Properties

Property	Description
CommandText	Gets or sets the SQL statement or stored procedure to execute against the database.
CommandType	Gets or sets a value indicating how the *CommandText* property is interpreted.
Connection	Gets or sets the *DB2eConnection* used by this instance of the *DB2eCommand*.
DesignTimeVisible	Gets or sets a value indicating whether the command object should be visible in a customized interface control.
Parameters	Gets the *DB2eParameterCollection*.
Transaction	Gets or sets the *DB2eTransaction* within which the *DB2eCommand* executes.
UpdatedRowSource	Gets or sets a value that specifies how the *Update* method should apply command results to the *DataRow*.

Table 121. Public Instance Methods

Method	Description
CreateParameter	Creates a new instance of a *DB2eParameter* object.
Dispose	Overloaded. Clean up.

Table 121. Public Instance Methods (continued)

Method	Description
EnableDeletePhysicalRemove	Enables or disables physically removing records.
EnableDirtyBitSetByApplication	Enables the application mode if enable is true. Otherwise, enables the system mode.
EnableReadIncludeMarkedDelete	Makes logically deleted records visible or invisible.
EnableReorg	Enables or disables database reorganization by DB2 Everyplace or explicitly by the user with a REORG SQL statement.
ExecuteNonQuery	Executes an SQL statement against the *Connection* and returns the number of rows affected.
ExecuteReader	Overloaded. Sends the *CommandText* to the Connection and builds an *DB2eDataReader*.
ExecuteScalar	Executes the query, and returns the first column of the first row in the resultset returned by the query. Extra columns or rows are ignored.
IsEnabledDeletePhysicalRemove	Check if physical remove is enabled or not. If enabled, returns true; otherwise false.
IsEnabledDirtyBitSetByApplication	Check if the database system is in application mode or system mode. If enabled, returns true; otherwise false.
IsEnabledReadIncludeMarkedDelete	Check if logically deleted records are visible or not to application. Returns true if logically deleted records are visible to application; otherwise returns false.
IsEnabledReorg	Checks if database reorganization is enabled. Returns true if it is enabled; otherwise returns false.
Prepare	Creates a prepared (or compiled) version of the command at the database.

DB2eConnection Members

Table 122. Public Static (Shared) Methods

Method	Description
ReleaseObjectPool	Indicates that the DB2e environment handle can be released when the last underlying connection is released.

Table 123. Public Instance Constructors

Constructor	Description
DB2eConnection()	Overloaded. Initialize a new instance of the *DB2eConnection* class.
DB2eConnection(string)	Overloaded. Initialize a newInitialize a new instance of the *DB2eConnection* class with a specified connection string.

Table 124. Public Instance Properties

Property	Description
ConnectionString	Gets or sets the string used to open a database.
ConnectionTimeout	Gets or sets the time to wait while trying to establish a connection before terminating the attempt and generating an error.
Database	Gets the name of the current database or the database to be used after a connection is opened.
ServerVersion	Gets a string containing the version of the server to which the client is connected.
State	Gets the current state of the connection.

Table 125. Public Instance Methods

Method	Description
BeginTransaction	Overloaded. Begins a transaction at the database.
ChangeDatabase	Changes the current database associated with an open *DB2eConnection*.
Close	Closes the connection to the database. This is the preferred method of closing any open connection.
CreateCommand	Creates and returns an *DB2eCommand* object associated with the *DB2eConnection*.
Open	Opens a connection to a data source with the property settings specified by the *ConnectionString*.

Table 126. Public Instance Events

Event	Description
InfoMessage	Occurs when the DB2 Everyplace sends a warning or an informational message.
StateChange	Occurs when the state of the connection changes.

DB2eDataAdapter Members

Table 127. Public Instance Constructors

Constructor	Description
DB2eDataAdapter()	Overloaded. Initialize a new instance of the *DB2eDataAdapter* class.
DB2eDataAdapter(DB2eCommand)	Overloaded. Initialize a new instance of the *DB2eDataAdapter* class with the specified SQL SELECT statement.
DB2eDataAdapter(string, DB2eConnection)	Overloaded. Initialize a new instance of the *DB2eDataAdapter* class with the specified SQL SELECT statement and an *DB2eConnection* object.
DB2eDataAdapter(string, string)	Overloaded. Initialize a new instance of the *DB2eDataAdapter* class with the specified SQL SELECT statement and a connection string.

Table 128. Public Instance Properties

Property	Description
AcceptChangesDuringFill (inherited from **DataAdapter**)	Gets or sets a value indicating whether *AcceptChanges* is called on a *DataRow* after it is added to the *DataTable*.
ContinueUpdateOnError (inherited from **DataAdapter**)	Gets or sets a value that specifies whether to generate an exception, or the row in error when an error is encountered during a row update.
DeleteCommand	Gets or sets an SQL statement or stored procedure used to delete records in the database.
InsertCommand	Gets or sets an SQL statement or stored procedure used to insert new records into the data source.
MissingMappingAction (inherited from **DataAdapter**)	Determines the action to take when incoming data does not have a matching table or column.
MissingSchemaAction (inherited from **DataAdapter**)	Determines the action to take when existing *DataSet* schema does not match incoming data.
SelectCommand	Gets or sets an SQL statement or stored procedure used to select records in the database.
TableMappings (inherited from **DataAdapter**)	Gets a collection that provides the master mapping between a source table and a *DataTable* .
UpdateCommand	Gets or sets an SQL statement or stored procedure used to update records in the database.

Table 129. Public Instance Methods

Method	Description
Clone	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Fill (inherited from **DbDataAdapter**)	Adds or refreshes rows to a *DataSet* or *DataTable* to match those in the data source.
FillSchema (inherited from **DbDataAdapter**)	Adds a *DataTable* to a *DataSet* and configures the schema to match that in the data source.
GetFillParameters (inherited from **DbDataAdapter**)	Gets the parameters set by the user when executing an SQL SELECT statement.
Update (inherited from **DbDataAdapter**)	Invokes the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the DataSet.

Table 130. Public Instance Events

Event	Description
FillError (inherited from **DbDataAdapter**)	Returned when an error occurs during a fill operation.
RowUpdated	Occurs during an update operation after a command is executed against the database.
RowUpdating	Occurs during *Update* before a command is executed against the database.

DB2eDataReader Members

Table 131. Public Instance Properties

Property	Description
Depth	Gets a value indicating the depth of nesting for the current row.
FieldCount	Gets the number of columns in the current row.
IsClosed	Indicates whether the *DB2eDataReader* is closed.
Item	Overloaded. Gets the value of the specified column in its native format given the column ordinal.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Table 132. Public Instance Methods

Method	Description
Close	Closes the *DB2eDataReader* object.
GetByte	Gets the value of the specified column as a byte.
GetBytes	Reads a stream of bytes from the specified column offset into the buffer as an array, starting at the given buffer offset.
GetDataTypeName	Gets the name of the source data type.
GetDate	Gets the value of the specified column as a *DateTime* object.
GetDateTime	Gets the value of the specified column as a *DateTime* object.
GetDecimal	Gets the value of the specified column as a *Decimal* object.
GetDouble	Gets the value of the specified column as a double-precision floating point number.
GetFieldType	Gets the *Type* that is the data type of the object.
GetFloat	Gets the value of the specified column as a single-precision floating-point number.
GetInt16	Gets the value of the specified column as a 16-bit signed integer.
GetInt32	Gets the value of the specified column as a 32-bit signed integer.
GetInt64	Gets the value of the specified column as a 64-bit signed integer.
GetName	Gets the name of the specified column.
GetOrdinal	Gets the column ordinal, given the name of the column.
GetSchemaTable	Returns a *DataTable* that describes the column metadata of the *DB2eDataReader*.
GetString	Gets the value of the specified column as a string.
GetTime	Gets the value of the specified column as a *TimeSpan* object.
GetValue	Gets the value of the column at the specified ordinal in its native format.
GetValues	Gets all the attribute columns in the current row.

Table 132. Public Instance Methods (continued)

Method	Description
IsDBNull	Gets a value indicating whether the column contains non-existent or missing values.
NextResult	Advances the *DB2eDataReader* to the next result, when reading the results of batch SQL statements. DB2 Everyplace does not currently support batch SQL statements.
Read	Advances the *DB2eDataReader* to the next record.

DB2eError Members

Table 133. Public Instance Properties

Property	Description
Message	Gets a short description of the error.
NativeError	Gets the error information from DB2 Everyplace.
SQLState	Gets the five-character error code that follows the ANSI SQL standard for the database.

DB2eException Members

Table 134. Public Instance Properties

Property	Description
Errors	Gets a collection of one or more *DB2eError* objects that give detailed information about exceptions generated by the DB2 Everyplace .NET Data Provider.
Message	Gets the textual description describing the error.

DB2eParameter Members

Table 135. Public Instance Constructors

Constructor	Description
DB2eParameter()	Overloaded. Initialize a new instance of the *DB2eParameter* class.
DB2eParameter(string, object)	Overloaded. Initialize a new instance of the *DB2eParameter* class with the parameter name and the value of the parameter.
DB2eParameter(string, DB2eType)	Overloaded. Initialize a new instance of the *DB2eParameter* class with the parameter name and data type.
DB2eParameter(string, DB2eType, int)	Overloaded. Initialize a new instance of the *DB2eParameter* class with the parameter name, data type, and width.
DB2eParameter(string, DB2eType, int, string)	Overloaded. Initialize a new instance of the *DB2eParameter* class with the parameter name, data type, width, and source column name.

Table 135. Public Instance Constructors (continued)

Constructor	Description
DB2eParameter(string, DB2eType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object)	Overloaded. Initialize a new instance of the *DB2eParameter* class with the parameter name, data type, width, parameter direction, nullable boolean, numeric precision, scale, source column name, source version and value of the parameter.

Table 136. Public Instance Properties

Property	Description
DB2eType	Gets or sets the *DB2eType* of the parameter.
DbType	
Direction	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
IsNullable	Gets or sets a value indicating whether the parameter accepts null values.
ParameterName	Gets or sets the name of the *DB2eParameter*.
Precision	Gets or sets the maximum number of digits used to represent the *Value* property.
Scale	Gets or sets the number of decimal places to which *Value* is resolved.
Size	Gets or sets the maximum size, in bytes, of the data within the column.
SourceColumn	Gets or sets the name of the source column mapped to the *DataSet* and used for loading or returning the *Value*.
SourceVersion	Gets or sets the *DataRowVersion* to use when loading *Value*.
Value	Gets or sets the value of the parameter.

Table 137. Public Instance Methods

Method	Description
ToString	Gets a string containing the *ParameterName*.

DB2eTransaction Members

Table 138. Public Instance Properties

Property	Description
Connection	Specifies the *DB2eConnection* object associated with the transaction.
IsolationLevel	Specifies the *IsolationLevel* for this transaction.

Table 139. Public Instance Methods

Method	Description
Commit	Commits the database transaction.
Rollback	Rolls back a transaction from a pending state.

DB2eType Enumeration

Specifies the data type of a field, property, or *DB2eParameter*.

[Visual Basic]

Public Enum DB2eType

[C#]

public enum DB2eType

The following table shows mappings between *DB2eType* data types, DB2 Everyplace data types (shown in parentheses), and .NET Framework types.

Table 140. Data type mappings

Member	Desciption
SmallInt	Exact numeric value with precision 5 and scale 0 (signed: –32,768 <= n <= 32,767, unsigned: 0 <= n <= 65,535) (SMALLINT). This maps to Int16.
Integer	Exact numeric value with precision 10 and scale 0 (signed: –2[31] <= n <= 2[31] – 1, unsigned:0 <= n <= 2[32] – 1) (INTEGER). This maps to Int32.
Char	A fixed-length character string (CHAR). This maps to String.
VarChar	A variable-length character string (VARCHAR). This maps to String.
Decimal	Signed, exact, numeric value with a precision of at least p and scale s, where 1 <= p <= 31 and s <= p. (DECIMAL). This maps to Decimal.
Date	Date data in the format yyyy-mm-dd (DATE). This maps to DateTime.
Time	Time data in the format hh:mm:ss (TIME). This maps to TimeSpan.
Timestamp	Timestamp data in the format yyyy-mm-dd-hh.mm.ss.zzzzzz (TIMESTAMP). This maps to DateTime.
Blob	A stream of binary data (BLOB). This maps to an Array of type Byte.

Requirements:

NameSpace: IBM.Data.DB2.DB2e Namespace

Assembly: IBM.Data.DB2.DB2e.dll

IBM Sync Client C-API

This chapter provides information about the IBM Sync Client C-API. The topics covered are:

Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2

This section summarizes the major changes made to the IBM Sync Client C-API in Version 8.1:

- Three handles are now available: service, configuration, and engine. (If you do not want to perform synchronizations, you do not need to open the sync engine handle.)
- The preferences in the IBM Sunc Client C-API Version 8.1 are not persistent, and some preferences, which actually are essential information have been removed. For example, the host name, port, user name, and password in the old `isyncSetPref` function are now the required parameters in the `iscOpenService` function for opening a service handle.
- The synchronization mode is now implicit to the application, and the synchronization mode parameter is no longer required when invoking a synchronization.
- The interface to the synchronization listener is now event based. Event structures that contain event information are now passed to the application.
- The synchronization status of a subscription set (from its last synchronization) is persistent and can be inquired afterward.
- The default listener has been removed. When a default action for an event is needed, the application simply returns the `ISCRTNCB_Default` code.
- DB2e Everyplace now supports data encryption to protect the table which contain sensitive data. When developing a sync client application to synchronize encrypted tables, you can implement (in the listener) the query from the sync engine for the DB2 Everyplace user name and password.
- Rejected records (including records with conflicts or illegal operations) are now passed to the application through the listener.
- The log file (LOGDB-ISYNC) is now managed by the application. That is, the Version 8.1 synchronization engine no longer generates the log file (LOGDB-ISYN) in a natively language as in version 7.2.1. Instead, for service purpose, the synchronization engine will generate a trace file (TRACE-ISYN), which is in English only
- The IBM Sync Client engine stores all files (including the configuration, trace file, data, and preferences (if applicable) in one directory:
  - On Windows CE® operating systems: \ (root directory)
  - On EPOC operating systems: `C:\Systems\Data\ISync\`
  - On Palm operating systems: the main memory
  - On other operating systems: the current directory
- The functionality of the IBM Sync Client API Version 7.2.1 is still supported through an API wrapper (the `isynce` library), which will handle the backward compatibility of the API. The API wrapper also generates the log file (`LOGDB-ISYN`) in native languages in the same directory as in Version 7.2.1, that is:
  - On Windows CE® operating systems: `\Progran Files\ISync\`

- On EPOC operating systems: `C:\Systems\Apps\ISync\`
- On Palm operating systems: the main memory
- On other operating systems: the current directory

In addition, the `ISYNCOPTION_SkipConfig` and `ISYNCOPTION_UseAppSignature` options will not work with the `isyncGo` and `isyncSetSyncMoe` functions.

Note: The API wrapper (`isynce`) library does need to be installed if choose to use IBM Sync Client API Version 8.1.

Table 141 lists the major differences between the functions in the IBM Sync Client C-API Version 8.1 and the IBM Sync Client Version 7.2.

Table 141. The IBM Sync Client C-API Version 8.1 and Version 7.2 comparison

Version 8.1	Version 7.2	Remarks
`iscGetVersion`	`isyncGetVersion`	No handles needed in `iscGetVersion`.
`iscServiceOpen` `iscConfigOpeniscEngineOpen`	`isyncOpen`	Need to open three handles. Host, port, user name, and password are specified in `iscServiceOpen` and are not persistent.
`iscServiceClose` `iscConfigClose iscEngineClose`	`isyncClose`	Need to close three handles.
`iscEngineSetListener`	`isyncSetListener`	The listener prototype and interface have changed.
(None)	`isyncDefaultListener`	No more external default listener. For default event handling, returns the ISCRTNCB_Default code.
`iscEngineSetPref` `iscEngineGetPref`	`isyncSetPref` `isyncGetPref`	Only two preferences (trace and timeout) are required. These preferences are not persistent.
`iscEngineSync` `iscEngineSyncConfig`	`isyncGo`	Sync mode is no longer required. Can update the configuration only with `iscEngineSyncConfig`.

`iscConfigEnableSubsSet`
`iscConfigDisableSubsSet`
`iscConfigResetSubsSet`
	`isyncSetSyncMode`	No more general sync mode setting.
		
		Synchronization of a subscription set can be skipped (disabled) by `iscConfigDisableSubsSet`.

`iscConfigOpenCursor`
`iscConfigCloseCursor`
`iscConfigGetNextSubsSet`
`iscConfigSubsSetIsEnabled`
`iscConfigSubsSetIsReset`
	`isyncGetFirstApp`
	`isyncGetNextApp`
		Opens a cursor before iterating subscription sets.
		
		A subscription-set ID is needed to query a subscription set.

`iscEngineGetInfo`
`iscConfigPurge`
`iscConfigGetSubsSetStatus`
		New C-APIs in Version 8.1.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "IBM Sync Client C-API function summary"
- "IBM Sync Client C-API data types" on page 297
- "Key to IBM Sync Client C-API function descriptions" on page 299

IBM Sync Client C-API function summary

Table 142 lists the IBM Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

Table 142. IBM Sync Client C-API function list

Function name	Purpose
iscGetVersion	Gets the version number of the Sync Client C-API.

Table 143. IBM Service API function list

Function name	Purpose
iscServiceOpen	Opens a new service.
iscServiceOpenEx	Opens a new service using properties.
iscServiceClose	Closes a service.

Table 144. IBM Configuration API function list

Function name	Purpose
iscConfigOpen	Opens a connection to the config store.
iscConfigClose	Closes a connection to the config store.
iscConfigPurge	Reinitializes the configuration.
iscConfigOpenCursor	Gets (handle of) a cursor for iterating subscription sets.
iscConfigCloseCursor	Disposes an opened cursor.
iscConfigGetNextSubsSet	Gets the description of the next subscription set (if any).
iscConfigEnableSubsSet	Enables a subscription set for synchronization.
iscConfigDisableSubsSet	Disables synchronization on a subscription set.
iscConfigResetSubsSet	Changes a subscription set back to the reset mode.
iscConfigSubsSetIsEnabled	Queries if a subscription set is enabled for synchronization.
iscConfigSubsSetIsReset	Queries if a subscription set is reset.
iscConfigGetSubsSetStatus	Queries the sync status of the previous synchronization.

Table 145. IBM Sync Engine API function list

Function name	Purpose
iscEngineOpen	Opens a handle to the synchronization engine.
iscEngineClose	Closes an opened handle to the synchronization engine.
iscEngineGetInfo	Gets general information about the synchronization engine.
iscEngineSetListener	Informs the synchronization about the user-defined listener function to use.
iscEngineListenerPF	Data type for the user-defined listener function.
iscEngineSetPref	Sets a preference.
iscEngineGetPref	Retrieves a preference value.
iscEngineSync	Launches a synchronization session.
iscEngineSyncConfig	Synchronizes the provided config with the server.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types"
- "Key to IBM Sync Client C-API function descriptions" on page 299

IBM Sync Client C-API data types

Table 146 lists the new data types defined by the IBM Sync Client C-API. When calling the C-API functions, make sure that the argument type complies with the prototype of the functions.

Table 146. Data types for IBM Sync Client C-API

Data type	Description
isy_VOID	Void type
isy_INT	Integer
isy_UINT	Unsigned integer
isy_INT16	Two-byte integer
isy_UINT16	Two-byte unsigned integer
isy_INT32	Four-byte integer
isy_UINT32	Four-byte unsigned integer
isy_ULONG	Unsigned long integer
isy_BYTE	One-byte type
isy_WORD	One-word type
isy_DWORD	Two-word type
isy_TCHAR	Character type
isy_BOOL	Boolean type

Table 146. Data types for IBM Sync Client C-API (continued)

Data type	Description
HISCSERV	Data type of the service handle
HISCCONF	Data type of the config handle
HISCENG	Data type of the synchronization engine handle
HISCCSR	Data type of an iterating cursor for subscription sets

ISCEVT	Data type of a listener event:
	`typedef struct {`
	`  isy_INT32  code;`
	`  isy_UINT32  type;`
	`  isy_INT32  retry;`
	`  ISCSTATE  state;`
	`  ISCLISTENARG  *info;`
	`} ISCEVT;`

ISCSTATE	Data type of the event state:
	`typedef struct {`
	`  isy_TCHAR  currSubsSet[ISCLEN_SubsSetName];`
	`  isy_TCHAR  currSubs[ISCLEN_SubsName];`
	`  isy_UINT32  subsType;`
	`  isy_INT32  syncProg;`
	`} ISCSTATE;`

ISCLISTENARG	Data type of information for the sync listener and consists of a list of string arguments (argc, argv):
	`typedef struct {`
	`  isy_INT32  argc;`
	`  isy_TCHAR  **argv;`
	`} ISCLISTENARG;`

ISCLISTENCOLUMN	Data type of information for the sync listener and consists of a table column that contains the column position, primary key sequence, column type, data size, and the actual column data:
	`typedef struct {`
	`  isy_INT32  pos;`
	`  isy_INT32  pkseq;`
	`  isy_INT32  type;`
	`  isy_INT32  size;`
	`  isy_BYTE  *data;`
	`} ISCLISTENCOLUMN;`
	
	Various column type constants for the column type are defined in a DB2 Everyplace header file, `sqlcli.h`. The column data is represented as a null-terminated text string. This is true except for the blob column type where the actual column data (the data field) is represented as a plain byte string and is NOT null-terminated. In addition, its size (# of bytes) is given in the size field.

ISCLISTENCONFLICT	Data type of information for the sync listener and consists of a table record that contains the table name, operation, the number of columns, and an array of column information (`ISCLISTENCOLUMN`):
	`typedef struct {`
	`  isy_TCHAR  table[ISCLEN_Table];`
	`  isy_INT32  op;`
	`  isy_INT32  colc;`
	`  ISCLISTENCOLUMN *colv;`
	`} ISCLISTENCONFLICT;`
	
	The *op* field indicates the rejected operation, which is one of the following operation constants(with actual values in the parenthesis):
	• ISCCONST_OpDelete (1)
	• ISCCONST_OpInsert (2)
	• ISCCONST_OpUpdate (3)

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API function summary" on page 296
- "Key to IBM Sync Client C-API function descriptions"

IBM Sync Client C-API function descriptions

This chapter describes the functions in the IBM Sync Client C-API.

Key to IBM Sync Client C-API function descriptions

Descriptions for each Sync Client C-API function contain the following sections:

Purpose
Gives a brief overview of what the function does.

Syntax
Contains the generic C prototype. The generic prototype is used for all environments, including Windows.

Function arguments
Lists the arguments of each function along each argument's data type, description, and type of use (input or output).

Usage Provides information about how to use the function and describes any special considerations.

Return codes
Lists all the possible function return codes.

Restrictions
Indicates any differences or limitations when applying each Sync Client C-API function.

References
Lists related Sync Client C-API functions.

Note: There is no Diagnostics section in the IBM Sync Client C-API.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296

iscGetVersion()

Purpose: `iscGetVersion()` gets the version number of the Sync Client C-API.

Syntax:

```
isy_UINT32 iscGetVersion();
```

Function arguments:

None.

Usage:

`iscGetVersion()` is used to retrieve the version number of the Sync Client C-API. The version number returns as a 32-bit unsigned integer in the format of *0xmmnnrrxx*, where *mm*, *nn*, and *rr* are the hexadecimal representation of major, minor, and modification version numbers, respectively. *xx* are reserved values.

Example::

```
isy_UINT32  version;
       int verMajor, verMinor, verModi;
       version = iscGetVersion();
       verMajor = (int) (version >> 24);
       verMinor = (int) ((version >> 16) & 0x000000FF);
       verModi  = (int) ((version >> 8)  & 0x000000FF);
```

Return codes:

The Sync Client C-API version number.

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "iscEngineGetInfo()" on page 317
- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "Key to IBM Sync Client C-API function descriptions" on page 299
- "IBM Sync Client C-API function summary" on page 296

iscServiceOpen()

Purpose: `iscServiceOpen()` opens a new service handle.

Syntax:

```
isy_INT32 iscServiceOpen(
    isy_CONST isy_TCHAR* host,
    isy_CONST isy_TCHAR* port,
    isy_CONST isy_TCHAR* username,
    isy_CONST isy_TCHAR* password,
    isy_CONST isy_VOID*  reserved,
    HISCSERV*            phServ);
```

Function arguments:

Table 147 lists the valid arguments used with the `iscServiceOpen()` function.

Table 147. iscServiceOpen() arguments

Data type	Argument	Use	Description
isy_CONST isy_TCHAR*	*host*	input	Host name or the IP
isy_CONST isy_TCHAR*	*port*	input	Port number
isy_CONST isy_TCHAR*	*username*	input	User name for the requested service
isy_CONST isy_TCHAR*	*password*	input	Password for the requested service
isy_CONST isy_TCHAR*	*reserved*	input	(Reserved)
HISCSERV*	*phServ*	output	Handle to a service

Usage:

`iscServiceOpen()` is used to request a new handle for a specific service that is identified by the host name and port number. The user name and password are specified when requesting a service. Upon success, a service handle (HISCSERV) returns through *phServ. Otherwise, *phServ is NULL, and the error code returns.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_OutOfMemory : Out of memory
- ISCRTN_ResourceBusy : Resource locked (for example, by another application)
- ISCRTN_NotPermitted : Resource not accessible (for example, not readable)
- ISCRTN_NotFound : Resource not found (for example, path not found)
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- “The sample Sync Client C/C++ application” on page 111

Related tasks:

- “Developing DB2 Everyplace Sync Client applications using C/C++” on page 15

Related reference:

- “Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2” on page 294
- “IBM Sync Client C-API data types” on page 297
- “IBM Sync Client C-API function summary” on page 296

- "iscServiceClose()" on page 303
- "iscServiceOpenEx()"

iscServiceOpenEx()

Purpose: `iscServiceOpenEx()` opens a new service handle based on a property array.

Syntax:

```
isy_INT32 iscServiceOpenEx(
    isy_CONST isy_TCHAR* URL,
    ISCPROPERTY*          property,
    isy_INT32            propertyNum,
    HISCSERV*            phServ);
```

Function arguments:

Table 148 lists the valid argument used with the `iscServiceOpenEx()` function.

Table 148. iscServiceOpenEx() argument

Data type	Argument	Use	Description
isy_CONST isy_TCHAR	*URL*	input	Server information as a URL sting

ISCPROPERTY	property	input	Array of properties of the ISCPROPERTY type:
			`typedef struct {`
			`  isy_TCHAR *key;  //property ID string`
			`  isy_TCHAR *value; //property value string`
			`} ISCPROPERTY;`
			
			There are three properties available:
			• *isync.user* — Sync Client user name
			• *isync.password* — Sync Client password
			• *isync.encoding* — Character encoding of the target data
			
			The user name and password properties are mandatory.

isy_INT32	propertyNum	input	Number of properties.
HISCSERV	phServ	output	Handle to a service.

Usage:

`iscServiceOpenEx()` is used to request a new handle for a specific service from a server with settings represented as a property array. The server is identified by a Uniform Resource Locator (URL) string, which might contain the protocol, the host name (or IP), and the port number. If the Sync Server is configured for Secure Socket Layer (SSL), the protocol part for the URL should be "https://", otherwise, it is "http://". The port number can be omitted, and the default ports for SSL and non-SSL are port 443 and port 80, respectively. All of the settings (including the user name and password) are specified in the property array. Upon success, a service handle (HISCSERV) is returned through *phServ*; otherwise, *phServ* is NULL, and the error code is returned. Upon completion, the service handle is closed with `iscServiceClose()`.

Example:

```
int rc = 0;
    HISCSERV hSyncServ;
    ISCPROPERTY properties[3] = {{"isync.user", "myUserName"},
                    {"isync.password", "myPassword"},
                    {"isync.encoding", "ISO8859_1"}}
    rc = iscServiceOpenEx("http://localhost.mycom.com:80", properties, 3, &hSyncServ);
```

Return codes:
- ISCRTN_Succeeded : OK
- ISCRTN_OutOfMemory : Out of memory
- ISCRTN_ResourceInUse : Resource locked (for example, by another application)
- ISCRTN_NotPermitted : Resource not accessible (for example, not readable)
- ISCRTN_NotFound : Resource not found (for example, path not found)
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related reference:
- "iscConfigClose()" on page 305

iscServiceClose()

Purpose: `iscServiceClose()` closes an opened service handle.

Syntax:
```
isy_INT32 iscServiceClose(
    HISCSERV  hServ);
```

Function arguments:

Table 149 lists the valid argument used with the `iscServiceClose()` function.

Table 149. iscServiceClose() argument

Data type	Argument	Use	Description
HISCSERV	*hServ*	input	Service handle

Usage:

Use `iscServiceClose()` to free the storage of a previously opened service handle.

Return codes:
- ISCRTN_Succeeded : OK
- ISCRTN_Failed : Otherwise

Restrictions:

Multiple calls to `iscServiceClose()` can cause errors and should be avoided.

Related concepts:
- "The sample Sync Client C/C++ application" on page 111

Related tasks:
- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:
- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294

- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscServiceOpen()" on page 300

iscConfigOpen()

Purpose: `iscConfigOpen()` opens a connection to the config store.

Syntax:

```
isy_INT32 iscConfigOpen(
    HISCSERV   hServ,
    isy_TCHAR *path,
    HISCCONF  *phConf);
```

Function arguments:

Table 150 lists the valid arguments used with the `iscConfigOpen()` function.

Table 150. iscConfigOpen() arguments

Data type	Argument	Use	Description
HISCSERV	*hServ*	input	Service handle
isy_TCHAR*	*path*	input	Path of the working directory
HISCCONF*	*phConf*	output	Config connection

Usage:

`iscConfigOpen()` opens a connection to the config store as specified in the given path for a specific service. Upon success, a config connection (HISCCONF) returns through *phServ. Otherwise, *phServ is NULL, and the error code returns. If this is a new service (either a new host or a new port), a new empty config is created for that service.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_OutOfMemory : Out of memory
- ISCRTN_ResourceBusy : Resource locked (for example, by another application)
- ISCRTN_NotPermitted : Resource not accessible (for example, not readable)
- ISCRTN_NotFound : Resource not found (for example, path not found)
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigClose()"

iscConfigClose()

Purpose: `iscConfigClose()` closes an opened config store connection.

Syntax:

```
isy_INT32 iscConfigClose(
    HISCCONF    hConf);
```

Function arguments:

Table 151 lists the valid argument used with the `iscConfigClose()` function.

Table 151. iscConfigClose() argument

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	config connection

Usage:

`iscConfigClose()` closes a previously opened config store connection.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigOpen()" on page 304

iscConfigPurge()

Purpose: `iscConfigPurge()` empties all of the subscription information from the config store.

Syntax:

```
isy_INT32 iscConfigPurge(
    HISCCONF    hConf);
```

Function arguments:

Table 152 lists the valid argument used with the `iscConfigPurge()` function.

Table 152. iscConfigPurge() argument

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection

Usage:

`iscConfigPurge()` removes all the user subscription information in the config store. During the next synchronization, the engine fetches the configuration again from the server and performs a total refresh on all the subscription sets.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigResetSubsSet()" on page 312

iscConfigOpenCursor()

Purpose: `iscConfigOpenCursor()` gets a cursor in order to repeatedly process all the subscription sets.

Syntax:

```
isy_INT32 iscConfigOpenCursor(
    HISCCONF    hConf,
    HISCCSR    *phCursor);
```

Function arguments:

Table 153 on page 307 lists the valid arguments used with the `iscConfigOpenCursor()` function.

Table 153. iscConfigOpenCursor() arguments

Data type	Argument	Use	Description
HSYNCCONF	*hConf*	input	Config connection
HISCCSR*	*phCursor*	output	Returned cursor for iterating subscription sets

Usage:

When an iteration process over all subscription sets is needed, use `iscConfigOpenCursor()` to get an appropriate cursor. Then, use `iscConfigGetNextSubsSet()` to get each subscription set and its corresponding description.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_Failed : Otherwise

Restrictions:

When `iscConfigOpenCursor()` is called, all previously opened cursors are invalidated and should be closed. Any attempt to process subscription sets with closed cursors generates the ISCRTN_Failed return code. That is, an iteration of the subscription sets cannot be nested within another iteration. Similarly, opened cursors are invalidated when the configuration is synchronized (using either `iscEngineSync()` or `iscEngineSyncConfig()`).

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigCloseCursor()"
- "iscConfigGetNextSubsSet()" on page 308

iscConfigCloseCursor()

Purpose: `iscConfigCloseCursor()` disposes an opened cursor.

Syntax:

```
isy_INT32 iscConfigCloseCursor(
    HISCCONF    hConf,
    HISCCSR     hCursor);
```

Function arguments:

Table 154 lists the valid arguments used with the `iscConfigCloseCursor()` function.

Table 154. iscConfigCloseCursor() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection
HISCCSR	*hCursor*	input	Cursor for iterating subscription sets

Usage:

When a cursor is opened with `iscConfigOpenCursor()` but the cursor is not needed, close this cursor with `iscConfigCloseCursor()`. Otherwise, the open cursor might cause memory leaks or other configuration consistency problems. Do not attempt to use the closed handle after the cursor closes since this can cause unexpected errors.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigOpenCursor()" on page 306
- "iscConfigGetNextSubsSet()"

iscConfigGetNextSubsSet()

Purpose: `iscConfigGetNextSubsSet()` gets the description (if any) of and moves the cursor to the next subscription set.

Syntax:

```
isy_INT32 iscConfigGetNextSubsSet(
    HISCCONF    hConf,
    HISCCSR     hCursor,
    isy_TCHAR*  id,
    isy_TCHAR*  name);
```

Function arguments:

Table 155 on page 309 lists the valid arguments used with the `iscConfigGetNextSubsSet()` function.

Table 155. iscConfigGetNextSubsSet() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection
HISCCSR	*hCursor*	input	Cursor for iterating subscription sets
isy_TCHAR*	*id*	output	ID of the subscription set
isy_TCHAR*	*name*	output	Name of the subscription set

Usage:

`iscConfigGetNextSubsSet()` gets the subscription-set ID from the server, retrieves the subscription-set name (if any), and moves the cursor to the next subscription set.

Example:

```
isy_TCHAR  id[ISCLEN_SubsSetID];
isy_TCHAR  name[ISCLEN_SubsSetName];
isy_INT32  isReset, isEnabled;
HISCCSR    hCursor;
isy_INT32  rc;

// start iteration of all subscription sets
rc = iscConfigOpenCursor(hConf, &hCursor);
while (rc == ISCRTN_Succeeded) {
    rc = iscConfigGetNextSubsSet(hConf, hCursor, id, name);
    if (rc == ISCRTN_Succeeded) {
        isReset = iscConfigSubsSetIsReset(hConf, id);
        isEnabled = iscConfigSubsSetIsEnabled(hConf, id);
        // processing the subscription set
        ...
        // get next subscription
    } // end of processing
} // end of iteration
iscConfigCloseCursor(hConf, hCursor);
```

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_Empty : No more subscription sets
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- “The sample Sync Client C/C++ application” on page 111

Related tasks:

- “Developing DB2 Everyplace Sync Client applications using C/C++” on page 15

Related reference:

- “Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2” on page 294
- “IBM Sync Client C-API data types” on page 297
- “IBM Sync Client C-API function summary” on page 296
- “iscConfigSubsSetIsReset()” on page 314
- “iscConfigSubsSetIsEnabled()” on page 313

iscConfigEnableSubsSet()

Purpose: `iscConfigEnableSubsSet()` enables a subscription set in the config for synchronization.

Syntax:

```
isy_INT32 iscConfigEnableSubsSet(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments:

Table 156 lists the valid arguments used with the `iscConfigEnableSubsSet()` function.

Table 156. iscConfigEnableSubsSet() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection
isy_TCHAR*	*id*	input	Subscription-set ID

Usage:

All subscription sets are initially enabled for synchronization. The `iscConfigEnableSubsSet()` and `iscConfigDisableSubsSet()` functions enable and disable the synchronization capability of a subscription set, which is specified by the given ID.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_NotFound : The subscription set is not found.
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigDisableSubsSet()" on page 311
- "iscConfigSubsSetIsEnabled()" on page 313

iscConfigDisableSubsSet()

Purpose: `iscConfigDisableSubsSet()` disables the synchronization on a subscription set.

Syntax:

```
isy_INT32 iscConfigDisableSubsSet(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments:

Table 157 lists the valid arguments used with the `iscConfigDisableSubsSet()` function.

Table 157. iscConfigDisableSubsSet() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection
isy_TCHAR*	*id*	input	Subscription-set ID

Usage:

All subscription sets are initially enabled for synchronization. The `iscConfigEnableSubsSet()` and `iscConfigDisableSubsSet()` functions enable and disable the synchronization capability of a subscription set, which is specified by the given ID.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_NotFound : The subscription set is not found.
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigEnableSubsSet()" on page 310
- "iscConfigSubsSetIsEnabled()" on page 313

iscConfigResetSubsSet()

Purpose: `iscConfigResetSubsSet()` resets a subscription set in the config back to the reset mode.

Syntax:

```
isy_INT32 iscConfigResetSubsSet(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments:

Table 158 lists the valid arguments used with the `iscConfigResetSubsSet()` function.

Table 158. iscConfigResetSubsSet() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection
isy_TCHAR*	*id*	input	Subscription-set ID

Usage:

If a subscription set is in reset mode when synchronized, the sync engine drops the client data for that subscription set. The sync engine simply fetches (or re-fetches) the server data; this process is called a refresh. After a subscription set is synchronized, this subscription set is *no longer* in reset mode. Use `iscConfigResetSubsSet()` to change the specified subscription set back to reset mode.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_NotFound : The subscription set is not found.
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigSubsSetIsReset()" on page 314

iscConfigSubsSetIsEnabled()

Purpose: `iscConfigSubsSetIsEnabled()` queries if a subscription set is enabled for synchronization.

Syntax:

```
isy_INT32 iscConfigSubsSetIsEnabled(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments:

Table 159 lists the valid arguments used with the `iscConfigSubsSetIsEnabled()` function.

Table 159. iscConfigSubsSetIsEnabled() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection
isy_TCHAR*	*id*	input	Subscription-set ID

Usage:

`iscConfigSubsSetIsEnabled()` is used to perform a query if a subscription set, which is specified by the given ID, is enabled for synchronization. All subscription sets are initially enabled for synchronization.

Return codes:

- ISCRTN_True : The subscription set is enabled for synchronization.
- ISCRTN_False : The subscription set is not enabled for synchronization.
- ISCRTN_NotFound : The subscription set is not found.
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigSubsSetIsReset()" on page 314

iscConfigSubsSetIsReset()

Purpose: `iscConfigSubsSetIsReset()` performs a query if a subscription set is in reset mode.

Syntax:

```
isy_INT32 iscConfigSubsSetIsReset(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments:

Table 160 lists the valid arguments used with the `iscConfigSubsSetIsReset()` function.

Table 160. iscConfigSubsSetIsReset() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection
isy_TCHAR*	*id*	input	Subscription-set ID

Usage:

All subscription sets are initially set to reset mode. However, if a subscription set is synchronized, the subscription-set mode changes. Use `iscConfigResetSubsSet()` to change a subscription set, which is specified by the given ID, back to reset mode.

Return codes:

- ISCRTN_True : The subscription set is in reset mode.
- ISCRTN_False : The subscription set is not in reset mode.
- ISCRTN_NotFound : The subscription set is not found.
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigSubsSetIsEnabled()" on page 313

iscConfigGetSubsSetStatus()

Purpose: `iscConfigGetSubsSetStatus()` gets the synchronization status of a subscription set.

Syntax:

```
isy_INT32 iscConfigGetSubsSetStatus(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments:

Table 161 lists the valid arguments used with the `iscConfigGetSubsSetStatus()` function.

Table 161. iscConfigGetSubsSetStatus() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection
isy_TCHAR*	*id*	input	Subscription-set ID

Usage:

Use `iscConfigGetSubsSetStatus()` to query the sync status of a subscription set (with the provided ID) during its last synchronization.

Return codes:

- ISCRTN_Succeeded : The synchronization of the subscription set succeeded.
- ISCRTN_Ready : The subscription set is enabled. The synchronization process started but has not yet synced the subscription set.
- ISCRTN_Canceled : The synchronization of the subscription set is canceled.
- ISCRTN_Failed : The synchronization of the subscription set failed.
- ISCRTN_NotFound : The subscription set is not found.

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscEngineSync()" on page 329
- "iscConfigSubsSetIsEnabled()" on page 313

iscEngineOpen()

Purpose: `iscEngineOpen()` opens a handle to the synchronization engine.

Syntax:

```
isy_INT32 iscEngineOpen(
    HISCCONF     hConf,
    HISCENG     *phEngine);
```

Function arguments:

Table 162 lists the valid arguments used with the `iscEngineOpen()` function.

Table 162. iscEngineOpen() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config handle
HISCENG*	*phEngine*	output	Handle to the synchronization engine

Usage:

Use `iscEngineOpen()` to open a handle to the sync engine (HISCENG) when synchronizing the specified configuration. The handle returns through *phEngine upon successful completion of the synchronization. If the synchronization does not complete successfully, the *phEngine value is NULL, and an error code returns.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_OutOfMemory : Out of memory
- ISCRTN_ResourceBusy : Resource locked (for example, by another application)
- ISCRTN_NotPermitted : Resource not accessible (for example, resource is not readable)
- ISCRTN_NotFound : Resource not found (for example, the path is not found)
- ISCRTN_Failed : Otherwise

Restrictions:

Avoid multiple calls to `iscEngineOpen()` since multiple calls open multiple handles to the synchronization engine and might cause consistency problems.

Related concepts:

- “The sample Sync Client C/C++ application” on page 111

Related tasks:

- “Developing DB2 Everyplace Sync Client applications using C/C++” on page 15

Related reference:

- “Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2” on page 294
- “IBM Sync Client C-API data types” on page 297
- “IBM Sync Client C-API function summary” on page 296
- “iscEngineClose()” on page 317

iscEngineClose()

Purpose: `iscEngineClose()` closes an opened handle to the synchronization engine.

Syntax:

```
isy_INT32 iscEngineClose(
   HISCENG    hEngine);
```

Function arguments:

Table 163 lists the valid argument used with the `iscEngineClose()` function.

Table 163. iscEngineClose() argument

Data type	Argument	Use	Description
HISCENG	*hEngine*	input	Handle to the synchronization engine

Usage:

Use `iscEngineClose()` to close an opened handle to the synchronization engine.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_Failed : Otherwise

Restrictions:

Multiple calls to `iscEngineClose()` can cause errors and should be avoided.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscEngineOpen()" on page 316

iscEngineGetInfo()

Purpose: `iscEngineGetInfo()` gets general information about the synchronization engine.

Syntax:

```
isy_INT32 iscEngineGetInfo(
    HISCENG       hEngine,
    isy_TCHAR    *info,
    isy_INT32     infoLen);
```

Function arguments:

Table 164 lists the valid arguments used with the `iscEngineGetInfo()` function.

Table 164. iscEngineGetInfo() arguments

Data type	Argument	Use	Description
HISCENG	*hEngine*	input	Handle to the synchronization engine
isy_TCHAR*	*info*	output	Pointer to the buffer that stores the return information
isy_INT32	*infoLen*	input	Size of the provided buffer

Usage:

`iscEngineGetInfo()` provides synchronization engine information for service purposes. The content and format of the information might change in the future. Therefore, your applications should simply display or log this information. Do not use this information as input for application program processing.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_ValTruncated : The actual length of the information is longer than the infoLen.
- ISCRTN_Failed : Otherwise

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscGetVersion()" on page 300

iscEngineSetListener()

Purpose: `iscEngineSetListener()` registers the user-defined listener function with the synchronization engine. During a synchronization session, the listener function is called when a synchronization event (such as a starting synchronization) or an error occurs.

Syntax:

```
isy_INT32 iscEngineSetListener(
    HISCENG             hEngine,
    iscEngineListenerPF syncListener,
    isy_UINT32          syncListenerData);
```

Function arguments:

Table 165 lists the valid arguments used with the `iscEngineSetListener()` function.

Table 165. iscEngineSetListener() arguments

Data type	Argument	Use	Description
HISCENG	*hEngine*	input	Handle to the synchronization engine
iscEngineListenerPF	*syncListener*	input	Address of the user-defined listener function
isy_UINT32	*syncListenerData*	input	Data that the application wants to forward to the user-define listener function

Usage:

By registering a user-defined listener function, the application has a view into the synchronization process. The application is notified when events or errors occur during synchronization. The application can customize methods to present these events or errors to the users.

Example:

```
// Function syncListener is defined with the following prototype:
isy_INT32 mySyncListener(
    isy_UINT32 listenerData,
    ISCEVT*  event,
    isy_VOID*  pExtraInfo);
...
// Handle to the synchronization engine is passed to the listener function
iscEngineSetListener(hEngine, mySyncListener, (isy_UINT32) hEngine);
```

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_Failed : Otherwise

Restrictions:

The user-defined listener function should follow the protocol of the synchronization engine, or the synchronization engine might not work correctly.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscEngineSync()" on page 329

iscEngineListenerPF

Purpose: `iscEngineListenerPF` defines the prototype that the user-defined listener function registered in `iscEngineSetListener()` should comply with.

Syntax:

```
typedef isy_INT32 (*iscEngineListenerPF)(
    isy_UINT32  listenerData,
    ISCEVT*     event,
    isy_VOID*   pExtraInfo);
```

Function arguments:

Table 166 lists the valid arguments used with the `iscEngineSetListenerPF` function type.

Table 166. iscEngineListenerPF arguments

Data type	Argument	Use	Description
isy_UINT32	*listenerData*	input	Data set in the *syncListenerData* argument by *iscEngineSetListener()* is forwarded back to the listener function
ISCEVT*	*event*	input	Event object
isy_VOID*	*pExtraInfo*	input	Reserved

Usage:

To use a user-define listener function for monitoring the progress of synchronization you must first make the function comply with the `iscEngineSetListenerPF` function type. Next, register the listener function using the `iscEngineSetListener()` function. Then, the user-defined listener function will be notified when synchronization events occur. The `event` argument is a structure, which contains various information about that event.

Table 167 on page 321 lists all the fields in the event structure and the purpose of each field.

Table 167. iscEngineListenerPF event fields

Field	Description
type	The event type can be one of the following values (actual values in the parentheses): **ISCEVTTYPE_Info (1)** Information regarding the synchronization progress. **ISCEVTTYPE_Conflict (2)** Conflicting or rejected operations in the synchronization process. **ISCEVTTYPE_Query (3)** Some information is needed in order for the synchronization to continue. The application must provide some required information (based on the event code) for the sync engine to continue. **ISCEVTTYPE_Retry (4)** An exception occurs, and a retry or cancel instruction needs to continue synchronizing. **ISCEVTTYPE_Error (5)** An error occurred, and the sync engine cannot continue synchronizing the current subscription set. **ISCEVTTYPE_Fatal (6)** A fatal error occurred, and the sync engine cannot continue synchronizing subscription sets.
state	The event state, which contains the following sub-fields: **currSubsSet** The subscription-set name, if not empty. **currSubs** The subscription name, if not empty. **subsType** Subscription type, if not 0, arranged as: • 100–999 : Reserved • 1000–9999 : Registered subscription type • 10000+ : Custom subscription type The pre-defined originators are (actual values in the parentheses): – ISCSUBSTYPE_Config (100) : Configuration – ISCSUBSTYPE_File (101) : File subscription – ISCSUBSTYPE_DB2e (102) : DB2 Everyplace table subscription **syncProg** The synchronization progress expressed as a percentage.
retry	The number of retries on the same event, if not 0.
info	Optional event-specific information (if not NULL), which is an array of string arguments for non-conflict events. For conflict events, the data type is ISCLISTENCONFLICT.

The **event.info** field contains some optional event-specific information. The event code is used to identify and interpret this information.

Table 168 on page 322 lists all the event codes by category of event type.

Table 168. iscEngineListenerPF event codes. Event type: ISCEVTTYPE_Info

Event code	Event info (argc)	Description
ISCEVT_InfGeneral (1000)	NULL	General information (for debugging).
ISCEVT_InfSyncStarted (1001)	NULL	Synchronization started.
ISCEVT_InfPrepMsg (1002)	NULL	Preparing message.
ISCEVT_InfSendMsg (1003)	NULL	Sending message.
ISCEVT_InfWaitMsg (1004)	NULL	Awaiting server reply.
ISCEVT_InfApplyMsg (1005)	NULL	Applying server message.
ISCEVT_InfCancelingSync (1006)	NULL	Canceling synchronization.
ISCEVT_InfSubsSetStarted (1007)	NULL	Synchronization of a subscription set started.
ISCEVT_InfSyncingSubs (1008)	NULL	Synchronization of a subscription has started.
ISCEVT_InfSubsSetFailed (1009)	NULL	Synchronization of a subscription set failed.
ISCEVT_InfSubsSetCanceled (1010)	NULL	Synchronization of a subscription set has been canceled.
ISCEVT_InfSubsSetSucceeded (1011)	NULL	Synchronization of a subscription set completed successfully.
ISCEVT_InfSyncSucceeded (1012)	NULL	Synchronization succeeded.
SCEVT_InfSyncFailed (1013)	NULL	Synchronization (on some subscription sets) failed.
ISCEVT_InfSyncCanceled (1014)	NULL	Synchronization canceled (by the user).
ISCEVT_InfSyncProg (1015)	NULL	Synchronization progress expressed as a percentage.
ISCEVT_InfNoNewChange (1016)	NULL	No new server change; skip pull and confirm phases.
ISCEVT_InfLoginFailed (1017)	NULL	Specified login information does pass the authentication process.

Table 169. iscEngineListenerPF event codes. Event type: ISCEVTTYPE_Conflict

Event code	Event info (argc)	Description
ISCEVT_CftReject (2000)	ISCLISTENCONFLICT	Data conflicts found in the synchronization.The actual conflicting data is represented as a `ISCLISTENCONFLICT` structure, and its reference pointer is given back to the application through *event.info*.

Table 170. iscEngineListenerPF event codes. Event type: ISCEVTTYPE_Retry

Event code	Event info (argc)	Description
ISCEVT_TryNetConn (4601)	NULL	Try again to connect to the server.
ISCEVT_TrySendRequest (4602)	NULL	Try again to send the request.
ISCEVT_TryRecvReply (4603)	NULL	Try again to receive the reply.
ISCEVT_TryRecvTimeout (4604)	NULL	Wait longer for the receiving reply.
SCEVT_TryRecvAck (4605)	NULL	Try again to receive an acknowledgement.

Table 171. iscEngineListenerPF event codes. Event type: ISCEVTTYPE_Query

Event code	Event info (argc)	Description
ISCEVT_QueCancel (5000)	NULL	Inquiry if the user cancels and returns (actual values in the parentheses): • ISCRTNCB_ReplyYes (3): If the user cancels • ISCRTNCB_ReplyNo (2) : If the user chooses to continue • ISCRTNCB_Default (0) : The default (that is, ISCRTNCB_ReplyNo)
ISCEVT_QueCancelUponError (5001)	NULL	Inquiry if the user cancels and returns (actual values in the parentheses): • ISCRTNCB_ReplyYes (3): If the user cancels • ISCRTNCB_ReplyNo (2) : If the user chooses to continue • ISCRTNCB_Default (0) : The default (that is, ISCRTNCB_ReplyNo)
ISCEVT_QueLogin (5002)	ISCLISTENARG(3) info->argv[0] info->argv[1] info->argv[2]	Login information requested by an adapter. The listener must provide the requested information in the event info and should return *ISCRTNCB_Done* with the actual value (1). Target name of data source Blank buffer for holding the user name Blank buffer for holding the password

Table 171. iscEngineListenerPF event codes (continued). Event type: ISCEVTTYPE_Query

Event code	Event info (argc)	Description
ISCEVT_QueSubsTarget (5003)	ISCLISTENARG(1) info->argv[0]	Database information requested by an adapter. The listener may provide the requested information in the event info and return ISCRTNCB_Done or return ISCRTNCB_Default to use the default target directory. Directory for subscription.

Table 172. iscEngineListenerPF event codes. Event type: ISCEVTTYPE_Error

Event code	Event info (argc)	Description
ISCEVT_ErrOpenAdapter (300)	NULL	Failed to open adapter *<adapter name>*.
ISCEVT_ErrLoadAdapter (301)	NULL	Failed to load adapter *<adapter name>*.
ISCEVT_ErrCloseAdapter (302)	NULL	Failed to close adapter *<adapter name>*.
ISCEVT_ErrAuthenticateKey (306)	NULL	Authentication failed (invalid encryption key); synchronization aborted.
ISYNCEVT_ErrClientCryptoFailed (307)	NULL	Client encryption or decryption failed; synchronization aborted.
ISCEVT_ErrEncryptNotAvail (308)	NULL	Encryption not available.
ISCEVT_ErrEncryptLibOpen (309)	NULL	Failed to open encryption library.
ISCEVT_ErrSubsNotFound (311)	NULL	Subscription not found by the server.
ISCEVT_ErrSubsNotAvail (312)	NULL	Subscription blocked by the server.
ISCEVT_ErrSubsDefAltered (316)	NULL	Subscription definition altered since the last time the sync engine synchronizes the configuration.
ISCEVT_ErrAllocResource (400)	NULL	Failed to allocate adapter resources.
ISCEVT_ErrConnectData (401)	NULL	Failed to connect to the target data.
ISCEVT_ErrDisconnectData (402)	NULL	Failed to disconnect from the target data.
ISCEVT_ErrNoData (403)	NULL	No data found.
ISCEVT_ErrMessageFormat (412)	NULL	Unexpected message format.
ISCEVT_ErrNotFound (413)	ISCLISTENARG(2) info->argv[0] info->argv[1]	Requested data not found. Target name of data source Data name
ISCEVT_ErrEndOfData (414)	NULL	Unexpected end-of-data.
ISCEVT_ErrDataTooLong (415)	ISCLISTENARG(3) info->argv[0] info->argv[1] info->argv[2]	Data is too long and is truncated. Target name of data source Data name Data element name (if not empty)
ISCEVT_ErrSyncDisabled (417)	NULL	Server reported that the user is not enabled.
ISCEVT_ErrServerException (418)	NULL	Server reported unknown exceptions.
ISCEVT_ErrReadOnly (420)	ISCLISTENARG(2) info->argv[0] info->argv[1]	Attempted to update read-only data. Target name of data source Data name
ISCEVT_ErrOperation (421)	NULL	Illegal operation on the data.
ISCEVT_ErrUnauthorized (423)	NULL	Not authorized to access the target data.
ISCEVT_ErrNotAvailable (424)	ISCLISTENARG(2) info->argv[0] info->argv[1]	Requested data not available. Target name of data source Data name
ISCEVT_ErrNotSupported (425)	ISCLISTENARG(3) info->argv[0] info->argv[1] info->argv[2]	Requested data is not supported. Target name of data source Data name Data element name (if not empty)

Table 172. iscEngineListenerPF event codes (continued). Event type: ISCEVTTYPE_Error

Event code	Event info (argc)	Description
ISCEVT_ErrSubsTargetDir (426)	NULL	The target database (path) provided in the ISCEVT_QueSubsTarget event is invalid, for example, an absolute path.
ISCEVT_ErrNetConn (601)	NULL	Failed to connect to the server.
ISCEVT_ErrSendRequest (602)	NULL	Failed to send request.
ISCEVT_ErrRecvReply (603)	NULL	Failed to receive reply.
ISCEVT_ErrRecvTimeout (604)	NULL	Timeout occurred while receiving the reply.
ISCEVT_ErrRecvAck (605)	NULL	Failed to receive an acknowledgement.
ISCRTN_ErrCloseNetLib (608)	NULL	Failed to close the network library
ISCEVT_ErrOutOfMemory (610)	NULL	Out of memory.
ISCEVT_ErrInternal (698)	ISCLISTENARG(1) info->argv[0]	Other internal errors occurred. Error state (as a string).

Table 173. iscEngineListenerPF event codes. Event type: ISCEVTTYPE_Fatal

Event code	Event info (argc)	Description
ISCEVT_FatSyncCfgAbort (303)	NULL	Config sync failed; synchronization aborted.
ISCEVT_FatAuthenticateFailed (304)	NULL	Authentication failed; synchronization aborted.
ISCEVT_FatIncompVersion (310)	NULL	Incompatible sync client version.
ISCEVT_FatInvalidSession (313)	NULL	Invalid session ID.
ISCEVT_FatSyncGroup (314)	NULL	User does not belong to any sync group.
ISCEVT_FatRegisterDevice (315)	NULL	Failed to register the device for the user.
ISCEVT_FatNetOpenConn (600)	NULL	Failed to open a connection to the server.
ISCEVT_FatOpenNetLib (606)	NULL	Failed to load the Network library.
ISCEVT_FatResolveHost (609)	NULL	Failed to resolve the host name.
ISCEVT_FatServerForbidden (611)	NULL	Forbidden to sync to the server.
ISCEVT_FatServerNotFound (612)	NULL	Server not found
ISCEVT_FatServer (613)	NULL	Server error.
ISCEVT_FatServerNotAvail (614)	NULL	Server not responding.
ISCEVT_FatProtocolNotSupported (615)	NULL	Protocol specified in the URL is not supported.
ISCEVT_FatNetUnknown (699)	NULL	Unknown network error.

Example:

```
isy_INT32 mySyncListener(
    isy_UINT32 listenerData,
    ISCEVT*  event,
    isy_VOID*  pExtraInfo)
{
    char *statusMsg = appEventCodeToMessage(event);
    int   timesRetried;

    switch (event->type) {
      case ISCEVTTYPE_Info:
        appStatusBar(statusMsg);
        // appStatusBar can be any routine which shows the statusMsg (e.g., in a
        // status bar)
        return ISCRTNCB_Done;

      case ISCEVTTYPE_Retry:
        timesRetried = event->retry;
        if (timesRetried >= 3) // Try no more than 3 times
            return ISCRTNCB_ReplyNo;
        else
            return appRetryCancelBox(statusMsg, 10); // 10 sec timeout
        // appRetryCancelBox can be any routine which shows a window with two
        // buttons: Cancel and Retry.  It returns
        //   ISCRTNCB_ReplyYes, if user clicks Retry
        //   ISCRTNCB_ReplyNo, if user clicks Cancel
```

```
        // If the user doesn't make choice, it returns ISCRTNCB_Default.
        break;

      // all other event types, don't care
      default:
        return ISCRTNCB_Default;
    } // switch (event->type)
} // mySyncListener
```

Return codes:

- ISCRTNCB_ReplyYes : The user replies `Yes` to the query.
- ISCRTNCB_ReplyNo* : The user replies `No` to the query.
- ISCRTNCB_Default : No reply; take the default action.

If the event type is ISCEVTTYPE_Retry, the listener function returns one of the following codes:

If the event type is ISCEVTTYPE_Query, the meaning of the return code depends on the value of event code. In other words, the listener checks the event code and returns the appropriate value. But if the user does not reply to the query, the application returns the following code:

- ISCRTNCB_Default : No reply; take the default action.

For event types other than ISCEVTTYPE_Retry and ISCEVTTYPE_Query, the sync engine ignores the return code. The listener simply returns ISCRTNCB_Done.

Note: For those events not of interest, the listener function simply returns ISCRTNCB_Default and allows the sync engine to take the default action.

Note: An asterisk (*) above indicates the default action for various event types.

Restrictions:

The user-defined listener function should follow the protocol of the synchronization engine. Otherwise, the synchronization engine might not work correctly.

Related concepts:

- “The sample Sync Client C/C++ application” on page 111

Related tasks:

- “Developing DB2 Everyplace Sync Client applications using C/C++” on page 15

Related reference:

- “Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2” on page 294
- “IBM Sync Client C-API data types” on page 297
- “IBM Sync Client C-API function summary” on page 296
- “iscEngineSync()” on page 329
- “iscEngineSetListener()” on page 318

iscEngineSetPref()

Purpose: `iscEngineSetPref()` sets the preferences of the synchronization engine.

Syntax:

```
isy_INT32 iscEngineSetPref(
    HISCENG             hEngine,
    isy_CONST isy_INT32  prefID,
    isy_CONST isy_TCHAR *prefVal);
```

Function arguments:

Table 174 lists the valid arguments used with the `iscEngineSetPref()` function.

Table 174. iscEngineSetPref() arguments

Data type	Argument	Use	Description
HISCENG	*hEngine*	input	Handle to the synchronization engine
isy_CONST isy_INT32	*prefID*	input	Preference ID, which is one of the following values: • ISCPREF_Timeout: Timeout length for receiving messages • ISCPREF_Trace: Detailed trace.
isy_CONST isy_TCHAR*	*prefVal*	input	New preference value to set. There are some pre-defined preference constants. For the ISCPREF_Trace preference: • ISCCONST_TraceON: Turn on detailed debugging trace • ISCCONST_TraceOFF: Turn off detailed debugging trace For the ISCPREF_Timeout preference: • ISCCONST_TimeoutNever: Never timeout while waiting for the server reply . • ISCCONST_TimeoutMinimum: Minimum timeout length

Usage:

Use `iscEngineSetPref()` to set the preferences of the synchronization engine. These preferences are not persistent, and they must be reset each time a new handle to the synchronization engine opens.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_UnknownID: Unknown
- ISCRTN_ValTooLong: The length of the given prefVal is too long.
- ISCRTN_Failed: Other errors

Restrictions:

The provided preference values should be within the specified preference limits:

- ISCPREF_Trace : 1
- ISCPREF_Timeout : 11

The iscEngineSetPref() and iscEngineGetPref() functions are deprecated. Please use iscServiceOpenEx with respective properties for the trace and timeout settings.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscEngineGetPref()"

iscEngineGetPref()

Purpose: `iscEngineGetPref()` retrieves the current preference setting.

Syntax:

```
isy_INT32 iscEngineGetPref(
    HISCENG              hEngine,
    isy_CONST isy_INT32  prefID,
    isy_TCHAR           *prefVal,
    isy_CONST isy_INT32  prefLen);
```

Function arguments:

Table 175 lists the valid arguments used with the `iscEngineGetPref()` function.

Table 175. iscEngineGetPref() arguments

Data type	Argument	Use	Description
HISCENG	*hEngine*	input	Handle to the synchronization engine
isy_CONST isy_INT32	*prefID*	input	Preference ID, which is one of the following values: • ISCPREF_Timeout: Timeout length for receiving messages • ISCPREF_Trace: Detailed trace.

Table 175. iscEngineGetPref() arguments (continued)

Data type	Argument	Use	Description
isy_TCHAR*	*prefVal*	output	Pointer to the buffer for storing the returned preference value. There are some pre-defined preference constants. For the ISCPREF_Trace preference: • ISCCONST_TraceON: Turn on detailed debugging trace • ISCCONST_TraceOFF: Turn off detailed debugging trace For the ISCPREF_Timeout preference: • ISCCONST_TimeoutNever: Never timeout while waiting for the server reply . • ISCCONST_TimeoutMinimum: Minimum timeout length
isy_CONST isy_INT32	*prefLen*	input	The size of the provided buffer (prefVal)

Usage:

Use `iscEngineGetPref()` to get the preference setting (which is either a default value or the value set by `iscEngineSetPref()`) of a synchronization engine.

Return codes:

- ISCRTN_Succeeded : OK
- ISCRTN_UnknownID : Unknown prefID provided
- ISCRTN_ValTruncated : The actual length of the preference value is longer than the prefLen.
- SCRTN_Failed : Other errors

Restrictions:

The provided buffer should be large enough to store the values of the various preferences:

- ISCPREF_Trace : 1
- ISCPREF_Timeout : 11
- ISCPREF_CodePage: 15

The iscEngineSetPref() and iscEngineGetPref() functions are deprecated. Please use iscServiceOpenEx with respective properties for the trace and timeout settings.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:

- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297

- "IBM Sync Client C-API function summary" on page 296
- "iscEngineSetPref()" on page 326

iscEngineSync()

Purpose: `iscEngineSync()` launches a synchronization session.

Syntax:

```
isy_INT32 iscEngineSync(
    HISCENG       hEngine);
```

Function arguments:

Table 176 lists the valid argument used with the `iscEngineSync()` function.

Table 176. iscEngineSync() argument

Data type	Argument	Use	Description
HISCENG	*hEngine*	input	Handle to the synchronization engine

Usage:

Use `iscEngineSync()` to launch a synchronization session that synchronizes the configuration that is specified in `iscEngineOpen()`. A subscription set is in reset mode if that subscription set has never been synchronized. When the sync engine performs a synchronization on that subscription set, the sync client fetches the data from the Sync Server; this process is called a refresh. After the refresh completes, the sync engine synchronizes the changed data when the subscription set is synchronized again; this process is called a synchronize. The sync engine always synchronizes the configuration first. If the configuration synchronization fails, the sync engine does not continue processing the subsequent subscription sets, and the synchronization session stops. If the sync engine fails on one subscription set (but not on the configuration), the sync engine continues processing the remaining subscription sets, if any.

Return codes:

- ISCRTN_Succeeded : The synchronization ended successfully.
- ISCRTN_Failed : The synchronization failed.
- ISCRTN_Canceled : The synchronization was canceled by the users.

The return code of `iscEngineSync()` is the aggregate (following the precedence listed below) of the sync status for all the subscription sets it has synchronized:

ISCRTN_Canceled > ISCRTN_Failed > ISCRTN_Succeeded

Restrictions:

None.

Related concepts:

- "The sample Sync Client C/C++ application" on page 111

Related tasks:

- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:
- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscConfigPurge()" on page 305
- "iscEngineSyncConfig()"

iscEngineSyncConfig()

Purpose: `iscEngineSyncConfig()` launches a synchronization session that synchronizing only the configuration.

Syntax:

```
isy_INT32 iscEngineSyncConfig(
    HISCENG        hEngine);
```

Function arguments:

Table 177 lists the valid argument used with the `iscEngineSyncConfig()` function.

Table 177. iscEngineSyncConfig() argument

Data type	Argument	Use	Description
HISCENG	*hEngine*	input	Handle to the synchronization engine.

Usage:

When the configuration changes on the server, `iscEngineSyncConfig()` updates the configuration without re-synchronizing all of the subscription sets.

Return codes:
- ISCRTN_Succeeded : The synchronization ended successfully.
- ISCRTN_Failed : The synchronization failed.
- ISCRTN_Canceled : The synchronization was canceled by the users.

Restrictions:

None.

Related concepts:
- "The sample Sync Client C/C++ application" on page 111

Related tasks:
- "Developing DB2 Everyplace Sync Client applications using C/C++" on page 15

Related reference:
- "Comparisons between IBM Sync Client C-API Version 8.1 and Version 7.2" on page 294
- "IBM Sync Client C-API data types" on page 297
- "IBM Sync Client C-API function summary" on page 296
- "iscEngineSync()" on page 329

- “iscConfigPurge()” on page 305

Chapter 19. DB2 Everyplace System Catalog base tables

The database manager creates and maintains a set of system catalog base tables. This appendix contains a description of each system catalog base table, including column names and data types. All of the system catalog base tables are created by the database manager. The system catalog base tables cannot be explicitly created or dropped. The system catalog base tables are updated during normal operation in response to SQL data definition statements, environment routines, and certain utilities. Data in the system catalog base tables is available through normal SQL query facilities. The system catalog base tables cannot be modified using normal SQL data manipulation commands. In order to access the system catalog tables, you need to use a delimited identifier.

Table 178. System catalog base tables

Description	Catalog base table
tables	333
columns	333
referential constraints	334
users	334

DB2eSYSTABLES:

This system catalog base table contains one row for each table that is created. All of the catalog tables have entries in the DB2eSYSTABLES catalog.

Table 179. DB2eSYSTABLES system catalog base table

Column name	Data type	Nullable	Description
TNAME	VARCHAR (19)		Table name
NUMCOLS	INTEGER (4)		Number of columns
FLAGS	INTEGER (4)		(Internal use only)
NUMKEY	INTEGER (4)		Number of columns in the primary key
CHK	BLOB (512)	Yes	Check constraint (internal use only)
IDXINFO	BLOB (700)	Yes	Index (internal use only)
NUMREFS	INTEGER (4)	Yes	Primary and foreign key (internal use only)
F_ID	INTEGER (4)	Yes	(Internal use only)
PD	BLOB (4096)	Yes	(Internal use only)

DB2eSYSCOLUMNS:

This system catalog base table contains one row for each column that is defined for a table.

Table 180. DB2eSYSCOLUMNS system catalog base table

Column name	Data type	Nullable	Description
CNAME	VARCHAR (19)		Column name
TNAME	VARCHAR (19)		Table name
TYPE	INTEGER (4)		Data type
ATTR	INTEGER (4)		(Internal use only)
LENGTH	INTEGER (4)		Length of the column
POS	INTEGER (4)		Column number
FLAGS	INTEGER (4)		(Internal use only)

Table 180. DB2eSYSCOLUMNS system catalog base table (continued)

Column name	Data type	Nullable	Description
KEYSEQ	INTEGER (4)		Ordinal position of the column in the primary key
SCALE	INTEGER (4)		Scale for decimal column
DEF	VARCHAR (128)	Yes	Default value (internal use)

DB2eSYSRELS:

This system catalog base table contains a row for each referential constraint.

Table 181. DB2eSYSRELS system catalog base table

Column name	Data type	Nullable	Description
RMD_ID	INTEGER (4)		Primary and foreign key (internal use only)
PKTABLE_NAME	VARCHAR (19)		Parent table name
PKCOLUMN_NAME	VARCHAR (19)		Parent table primary key column
FKTABLE_NAME	VARCHAR (19)		Child table name
FKCOLUMN_NAME	VARCHAR (19)		Child table foreign key column name
ORDINAL_POSITION	INTEGER (4)		Position of the column in the foreign key reference

DB2eSYSUSERS:

The DB2eSYSUSERS table is created automatically when the first encrypted table is created or when the first GRANT statement is executed. This table is tightly bound to the database and encrypted data; it cannot be moved to another DB2 Everyplace database that contains different encrypted data.

This system catalog base table contains one row for each registered user name that is defined for a database.

Table 182. DB2eSYSCOLUMNS system catalog base table

Column name	Data type	Nullable	Description
USERNAME	VARCHAR (19)		Part of primary key and is case sensitive. The name of the user associated with this row.
DATABASENAME	VARCHAR (19)		For future use. Empty string is stored. Part of primary key.
TABLENAME	VARCHAR (19)		For future use. Empty string is stored. Part of primary key.
ENCMETHOD	VARCHAR (198)		For future use. Empty string is stored. Part of primary key.
PRIVILEGES	CHAR (19)	Yes	Defines user privileges. Currently, only the value 'E', indicating encryption, is allowed.
ENCKEYDATA	BLOB (64)	Yes	Used to regenerate encryption key.
ATTIME	TIMESTAMP (26)	Yes	Time when the user was added or the record was most recently modified, whichever is most recent.
VALIDATE	BLOB (64)	Yes	Verifies that the record is authentic and the user was added by an authenticated user.
GRANTOR	VARCHAR (19)	Yes	The user name that registered the user name in column 1.
INTERNALDATA	BLOB (255)	Yes	(Internal future use)

Chapter 20. DB2 Everyplace limits

The following table describes certain DB2 Everyplace and SQL limits. Adhering to the most restrictive case can help the programmer design application programs that are easily portable. Many of these limits might be further restricted due to physical memory and system limitations imposed by the devices.

Table 183. DB2 Everyplace database and SQL limits

Description	Limit
Maximum table size (on a 32 bit system)	2 Gigabytes
Maximum length of a database	75 bytes
Maximum number of tables in a data store	65535
Maximum number of indices on a table	15
Maximum number of foreign keys in a table	8
Maximum number of columns in an index	8
Maximum number of columns in a primary key	8
Maximum length of SQL statement	64 kilobytes
Maximum number of connections to a data store path	1
Maximum number of rows in a table	Limited by table size
Maximum number of columns in a table	256
Maximum length for a CHAR column	32767 bytes
Maximum length for VARCHAR or BLOB column	32767 bytes
Maximum cumulative length for a row's 32767 fixed-length columns	32767 bytes
Maximum number of statement handles per connection	20
Maximum length of check constraints	512 bytes
Maximum size of decimals	31 digits
Maximum length of each column in a single index	1024 bytes

Related reference:

• "Overview of DB2 Everyplace SQL statement support" on page 129

Chapter 21. DB2 Everyplace reserved words

The following DB2 Everyplace reserved words cannot be used as identifiers unless they are specified as delimited identifiers. For example:

The following statement will cause an SQL error:

```
CREATE TABLE tab1 (select int)
```

Uses double quotation marks and will not cause a SQL error:

```
CREATE TABLE tab1 ("select" int)
```

DB2 Everyplace reserved words:

```
ALL, AND, AS, ASC,
BEGIN, BLOB, BY,
DATABASE
CALL, CHAR, CHAR, CHECK, COMMIT, CONCAT, CREATE, CURRENT,
DATE, DECIMAL, DEFAULT, DELETE, DESC, DISTINCT, DROP,
ENCRYPT
FETCH, FOR, FOREIGN, FROM,
GRANT, GROUP,
IN, INDEX, INSERT, INT, INTEGER, INTO, IS,
KEY,
LIKE, LIMIT,
NEW, NOT, NULL,
OF, ON, ONLY, OR, ORDER,
PRIMARY,
READ, REFERENCES, REORG,REVOKE,ROLLBACK,
SELECT, SET, SMALLINT,
TABLE, TIME, TIMESTAMP, TO,TRANSACTION
UPDATE, UPSERT, USING
VALUES, VARCHAR,
WHERE, WITH
```

For future compatibility, do not use the following IBM SQL and ISO/ANSI SQL92 reserved words as identifiers. The IBM SQL reserved words that are not currently used by DB2 Everyplace are:

```
ACQUIRE ADD AFTER ALIAS ALLOCATE
ALLOW ALTER ANY ASUTIME AUDIT AUTHORIZATION
AUX AUXILIARY AVG BEFORE BETWEEN BINARY
BUFFERPOOL CALLED CAPTURE CASCADED CASE
CAST CCSID CHARACTER CLOSE CLUSTER COLLECTION
COLLID COLUMN COMMENT CONDITION CONNECT
CONNECTION CONSTRAINT CONTAINS CONTINUE
COUNT COUNT_BIG CROSS CURRENT_DATE CURRENT_LC_PATH
CURRENT_PATH CURRENT_SERVER CURRENT_TIME
CURRENT_TIMESTAMP CURRENT_TIMEZONE CURRENT_USER
CURSOR DATA DATABASE DAY DAYS DBA DBINFO
DBSPACE DB2GENERAL DB2SQL DECLARE DESCRIPTOR
DETERMINISTIC DISALLOW DISCONNECT DO DOUBLE
DSSIZE DYNAMIC EDITPROC ELSE ELSEIF END
END-EXEC ERASE ESCAPE EXCEPT EXCEPTION
EXCLUSIVE EXECUTE EXISTS EXIT EXPLAIN
EXTERNAL FENCED  FIELDPROC FILE FINAL FREE
FULL FUNCTION GENERAL GENERATED GO GOTO
GRANT GRAPHIC HANDLER HAVING HOUR HOURS
IDENTIFIED IF IMMEDIATE INDICATOR INNER
INOUT INSENSITIVE INTEGRITY INTERSECT
ISOBID ISOLATION JAVA JOIN LABEL LANGUAGE
LC_CTYPE LEAVE LEFT LINKTYPE LOCAL LOCALE
```

```
LOCATOR LOCATORS LOCK LOCKSIZE LONG LOOP
MAX MICROSECOND MICROSECONDS MIN MINUTE
MINUTES MODE MODIFIES MONTH MONTHS NAME
NAMED NHEADER NO NODENAME NODENUMBER NULLS
NUMPARTS OBID OPEN OPTIMIZATION OPTIMIZE
OPTION OUT OUTER PACKAGE PAGE PAGES
PARAMETER PART PARTITION PATH PCTFREE
PCTINDEX PIECESIZE PLAN POSITION PRECISION
PREPARE PRIQTY PRIVATE PRIVILEGES PROCEDURE
PROGRAM PSID PUBLIC QUERYNO READS RECOVERY
RELEASE RENAME REPEAT RESET RESOURCE RESTRICT
RESULT RETURN RETURNS REVOKE RIGHT ROW ROWS
RRN RUN SCHEDULE SCHEMA SCRATCHPAD SECOND
SECONDS SECQTY SECURITY SHARE SIMPLE SOME
SOURCE SPECIFIC SQL STANDARD STATIC STATISTICS
STAY STOGROUP STORES STORPOOL STYLE SUBPAGES
SUBSTRING SUM SYNONYM TABLESPACE THEN TO
TRANSACTION TRIGGER TRIM TYPE UNDO UNION
UNIQUE UNTIL USAGE USER USING VALIDPROC
VARIABLE VARIANT VCAT VIEW VOLUMES WHEN
WHILE WLM WORK WRITE YEAR YEARS
```

The ISO/ANS SQL92 reserved words that are not used by the IBM SQL are as follows.

```
ABSOLUTE ACTION ARE ASSERTION AT BIT_LENGTH
BOTH CATALOG CHAR_LENGTH CHARACTER_LENGTH
COALESCE COLLATE COLLATION CONSTRAINTS CONVERT
CORRESPONDING DEALLOCATE DEC DEFERRABLE DEFERRED
DESCRIBE DIAGNOSTICS DOMAIN EXEC EXTRACT FALSE
FIRST FLOAT FOUND FULL GET GLOBAL IDENTITY
INITIALLY INPUT INTERVAL LAST LEADING LEVEL
LOWER MATCH MODULE NAMES NATIONAL NATURAL
NCHAR NEXT NULLIF NUMERIC OCTET_LENGTH OUTPUT
OVERLAPS PAD PARTIAL PRESERVE PRIOR REAL
RELATIVE SCROLL SECTION SESSION SESSION_USER
SIZE SPACE SQLCODE SQLERROR SQLSTATE SYSTEM_USER
TEMPORARY TIMEZONE_HOUR TIMEZONE_MINUTE
TRAILING TRANSLATION TRUE UNKNOWN UPPER
VALUE VARYING WHENEVER ZONE
```

Related reference:

- "Overview of DB2 Everyplace SQL statement support" on page 129

Chapter 22. National language support (NLS)

This chapter contains information about the national language support (NLS) provided by DB2 Everyplace, including information about countries, languages, and code pages (code sets) supported, and how to configure and use DB2 Everyplace NLS features with your devices and applications. DB2 Everyplace supports single-byte, double-byte, and multi-byte character sets, and UNICODE. Both UNICODE and non-UNICODE (ANSI) are supported on all Win32 operating systems. Read the following topics for an explanation of how code pages and UNICODE are supported.

DB2 Everyplace NLS support by operating system

Table 184 lists which operating systems and corresponding languages have NLS support.

Table 184. NLS support

Language	Win32	WinCE	Linux	Palm OS	Symbian OS	Neutrino
English	Codepage/ UNICODE	UNICODE	Codepage/ UTF-8	Codepage	UNICODE	UTF-8
French	Codepage/ UNICODE	UNICODE	Codepage/ UTF-8	Codepage	UNICODE	UTF-8
German	Codepage/ UNICODE	UNICODE	Codepage/ UTF-8	Codepage	UNICODE	UTF-8
Italian	Codepage/ UNICODE	UNICODE	Codepage/ UTF-8	Codepage	UNICODE	UTF-8
Spanish	Codepage/ UNICODE	UNICODE	Codepage/ UTF-8	Codepage	UNICODE	UTF-8
Simplified Chinese	Codepage/ UNICODE	UNICODE	Codepage/ UTF-8	Codepage • Install enabler	N/A	UTF-8
Traditional Chinese	Codepage/ UNICODE	UNICODE • Install enabler for Pocket PC	Codepage/ UTF-8	Codepage • Install enabler Acer S60 has a built-in Traditional Chinese Palm OS.	N/A	UTF-8
Korean	Codepage/ UNICODE	UNICODE • Install enabler	Codepage/ UTF-8	Codepage • Install enabler	N/A	UTF-8

Table 184. NLS support (continued)

Language	Win32	WinCE	Linux	Palm OS	Symbian OS	Neutrino
Japanese	Codepage/ UNICODE	UNICODE	Codepage/ UTF-8	Codepage	N/A	UTF-8
Hebrew	N/A	N/A	N/A	Codepage • Install enabler	N/A	N/A
Czech	Codepage/ UNICODE	UNICODE • Install enabler	N/A	Codepage • Install enabler	UNICODE	N/A
Arabic	N/A	N/A	N/A	Codepage • Install enabler	N/A	N/A

For Palm OS, QNX Neutrino, Linux, Windows NT and Windows 2000 operating systems without UNICODE support, locale information is used to determine the correct code page. There is no internal string conversion inside DB2 Everyplace. Each passed string is stored as is. Querying applications must use the same code page settings that were used at storage time. This is similar to how DB2 Universal Database provides NLS. DB2 Everyplace does not provide code page conversion functions. DB2 Everyplace databases created on a system using a specific code page can be deployed only on systems using the same code page. Tables created with a specific code page are usable on all devices that support that code page, except when a specific language enabler is required. Applications accessing a database are responsible for interpreting the character data correctly.

DB2 Everyplace detects the currently used encoding format by examining the currently set or available locale.

On Palm OS, the presence of language enablers is also used to determine the code page.

Related reference:

- "Character encoding in Java applications"
- "DB2 Everyplace language enablers" on page 341
- "DB2 Everyplace UNICODE support" on page 342

Character encoding in Java applications

Java programs use UNICODE text internally; however, the character data in a DB2 Everyplace table could be in a format other than UNICODE, depending on the operating system and language in which the table was created. For Windows CE and Symbian OS operating systems, the DB2 Everyplace JDBC driver retrieves and inserts text as UTF-8 format. For other supported operating systems, the system's default character encoding is used. The default is usually determined by the "file.encoding" attribute of the Java system property.

For example, on the Win32 operating system, a user might choose to use a UNICODE or non-UNICODE version of the CLI interface; on the same machine, therefore, one database could have UTF-8 format encoding and one local codepage encoding. To enable a JDBC application to access the data from both databases,

DB2 Everyplace provides a way for users to dynamically indicate which data encoding format an application should use.

The DB2 Everyplace JDBC driver converts Java strings into bytes according to the format specified by the application. The application-specified format overrides the operating system's default character encoding.

Users can dynamically specify the application's data encoding format through the JDBC interface. To do this:

1. Create a java.util.Properties object.
   - Key: DB2e_ENCODING
   - Value: character encoding.

   Use the value UTF-8 to specify DB2 Everyplace using UTF-8 coding, or use any character encoding supported by the JVM.
2. Use one of the following two methods to pass the java.util.properties object:
   - To establish a connection to a given database URL:

     Use the static method Connection getConnection(String url, Properties info) in the DriverManager class in the java.sql package.
   - To make a database connection to the given URL:

     Use the Connection connect(String url, Properties info) method in the Interface Driver class in the java.sql package.

Related reference:

- "DB2 Everyplace NLS support by operating system" on page 339
- "DB2 Everyplace language enablers"
- "DB2 Everyplace UNICODE support" on page 342

DB2 Everyplace language enablers

To ensure that your mobile device can display all the characters of the language that you are using, you can install language enablers on your mobile device. The following table lists the enablers that you can use with DB2 Everyplace.

Table 185. Language enablers for mobile devices

Language	Enabler and operating system
Arabic	Sakhr Arabic Palm 2.0
Simplified Chinese	CWP v1.0 for Palm
Traditional Chinese	• CJKOS 3.21 for Palm OS color devices (The sort records in the CJK option can cause unexpected results.) • Gismosoft Chinese Small_Knife 2.0 for Pocket PC only • Acer S60 has a built-in Traditional Chinese Palm OS
Czech	• RedGrep GNU-czech0.71 for Palm OS • Sunnysoft InterWrite5.5P Pro for Windows CE
Hebrew	Penticon Technologies Ltd. Hebrew Support+3.20c for Palm OS

Table 185. Language enablers for mobile devices (continued)

Language	Enabler and operating system
Korean	• HANME 2.0 for Palm OS • HANTIP 2.01for Palm OS CessHan for Casio E-115 1.0 on Windows CE

Related reference:

- "Character encoding in Java applications" on page 340
- "DB2 Everyplace NLS support by operating system" on page 339
- "DB2 Everyplace UNICODE support"

DB2 Everyplace UNICODE support

On operating systems that support UNICODE (Windows CE, Symbian OS, Windows NT and Windows 2000), DB2 Everyplace takes UNICODE strings only as Input/Output strings. These UNICODE strings are saved as UTF–8 format inside the DB2 Everyplace engine. A UNICODE character might require one to three bytes of storage space after the UTF–8 conversion. A character string stored in a database server such as IBM DB2 Universal Database might require more space when the string is downloaded and stored in DB2 Everyplace on Windows CE devices.

CLI UNICODE interface notes:

- The DB2 Everyplace CLI UNICODE functions have a character ″W″ appended at the end. By defining the macro UNICODE (which is the system default on Windows CE), the regular CLI functions map to the corresponding UNICODE functions automatically. To write portable code, define the macro ″UNICODE″, and let the system do the conversions.
- When UNICODE support is enabled, the data types SQL_C_CHAR, SQL_C_TCHAR, and SQL_C_WCHAR have the same meaning.
- Many CLI functions have a string (or buffer) length as an input/output parameter.
  - For functions with *Argument Type* as SQLCHAR* (or SQLWCHAR* for the W function), the length is the number of characters. For example:

```
SQLRETURN   SQLExecDirect    (SQLHSTMT          hstmt,
                              SQLCHAR     FAR   *szSqlStr,
                              SQLINTEGER        cbSqlStr);
```

    UNICODE string `L"ABCD"` is four characters.
  - For functions with *Argument Type* as SQLPOINTER, the length is the number of bytes. For example:

```
SQLRETURN  SQLGetData       (SQLHSTMT          hstmt,
                             SQLUSMALLINT      icol,
                             SQLSMALLINT       fCType,
                             SQLPOINTER        rgbValue,
                             SQLINTEGER        cbValueMax,
                             SQLINTEGER  FAR   *pcbValue);
```

    The length for the input parameter `cbValueMax` and output parameter `*pcbValue` are in bytes. UNICODE string `L"ABCD"` is eight bytes.
- The UNICODE functions can also take SQL_NTS to indicate a NULL-terminated string.

Tips for writing portable code:

- Use SQLTCHAR instead of SQLCHAR or SQLWCHAR.
- Use the `_tcs`*XXXX* functions instead of `str`*XXXX* (ANSI) or `wcs`*XXXX* (UNICODE). For example, `use _tcslen()` instead of `wcslen()` or `strlen()`.
- Use `_TEXT()` (or `TEXT()`) to wrap literal strings. For example, `_TEXT("ABCD")` can be interpreted as either an ANSI or UNICODE string depending on the macro definition.
- Use `sizeof(ArrayName)/sizeof(TCHAR)` to find out the size of a character array.

For an example, see the Windows CE SampleCLP sample code included with DB2 Everyplace.

Related reference:

- "Character encoding in Java applications" on page 340
- "DB2 Everyplace language enablers" on page 341
- "DB2 Everyplace NLS support by operating system" on page 339

Chapter 23. The DB2 Everyplace information set

The DB2 Everyplace library consists of the online help in HTML and books in PDF and HTML format. This section describes the information that is provided, and how you can access it.

All product information is also available online at www.ibm.com/software/data/db2/everyplace/library.html

DB2 Everyplace PDF and HTML files

The DB2 Everyplace books and Release Notes are viewable in HTML and PDF formats directly from the CD-ROM. DB2 Everyplace information is translated into different languages; however, not all the information is translated into every language. Whenever information is not available in a specific language, the English information is provided.

When you install DB2 Everyplace on your workstation, the documentation is stored in `\DB2everyplace\docs`. The following table lists the books stored in the **docs** directory.

Table 186. Books available for DB2 Everyplace

Book title	Description	PDF file name	HTML directory
DB2 Everyplace Installation and User's Guide (SC18–7184–00)	• Installing DB2 Everyplace components to a workstation. • Installing the DB2 Everyplace database and sample applications to a mobile or embedded device. • Configuring and maintaining a mobile or embedded device. • Using the DB2 Everyplace sample applications.	dsyiug.pdf	dsyiug
DB2 Everyplace Application Development Guide (SC18–7185–00)	• Building DB2 Everyplace applications on the available platforms. • The DB2 Everyplace sample applications and source code. • Supported SQL statements, SQLStates, DB2 CLI/ODBC, JDBC methods, IBM Sync Client C-API, IBM Java Sync APIs, and National Language Support. • Accessing a DB2 Everyplace database • Using local data encryption.	dsyadg.pdf	dsyadg

Table 186. Books available for DB2 Everyplace (continued)

Book title	Description	PDF file name	HTML directory
DB2 Everyplace Sync Server Administration Guide (SC18–7186–00)	• Configuring and maintaining the Sync Server. • Connecting the Sync Server to data sources. • Configuring communications between the Sync Server and mobile and embedded devices. • Configuring and maintaining local and remote databases. • Managing users and data.	dsysag.pdf	dsysag

DB2 Everyplace online documentation

Online help is available with the DB2 Everyplace Sync Server Mobile Devices Administration Center and the DB2 Everyplace Mobile Application Builder.

Part 5. Appendixes

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 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 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 which has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information 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 illustrates 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.

This product includes software developed by 3Com and its contributors.:

Copyright (c) 1998 3Com/Palm Computing Division. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by 3Com and its contributors.
4. Neither 3Com nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE 3COM AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL 3COM OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Trademarks

The following terms, which may be denoted by an asterisk(*), are trademarks of International Business Machines Corporation in the United States, other countries, or both.

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2
z/OS

The following terms are trademarks or registered trademarks of other companies:

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Java or all Java-based trademarks and logos, and Solaris are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, other countries, or both.

UNIX is a registered trademark in the United States, other countries or both and is licensed exclusively through X/Open Company Limited.

Other company, product, or service names, which may be denoted by a double asterisk(**) may be trademarks or service marks of others.

Glossary

A

Apply qualifier. A character string that identifies subscription definitions that are unique to each instance of the DataPropagator Apply program.

authentication. The process of validating a user's ID and password against entries in the administration control database to ensure that the user is authorized to use the Sync Server to synchronize data.

authorization. In computer security, the right granted to a user to communicate with or make use of a computer system.

B

binary large object (BLOB). A sequence of bytes, where the size of the sequence ranges from 0 to 2 gigabytes. This byte sequence does not have an associated code page and character set. Image, audio, and video objects are stored in BLOBs.

bind. In SQL, the process by which the output from the SQL precompiler is converted to a usable structure called an access plan. During this process, access paths to the data are selected and some authorization checking is performed.

BLOB. See *binary large object*.

C

client. A program or user that communicates with and accesses a database server. You define clients using the Administrator.

conflict detection. The process of detecting an out-of-date row in a target table that was updated by a user application. When a conflict is detected, the transaction that caused the conflict is rejected.

Control Center. A graphical interface that shows database objects (such as databases and tables) and their relationship to each other. From the Control Center, you can perform the tasks provided by the DBA Utility, Visual Explain, and Performance Monitor tools.

D

data filter. See *filter*.

data synchronization. See *mobile data synchronization*.

database management system (DBMS). A computer program that manages data by providing the services of centralized control, data independence, and complex physical structures for efficient access, integrity, recovery, concurrency control, privacy, and security.

database server. A functional unit that provides database services for databases.

DB2 Control Center. See *Control Center*.

DB2 DataPropagator. A replication product that provides an automated method of replicating data from sources to targets. During mobile data synchronization, the mirror and remote databases serve as both source and target. DataPropagator replicates clients' changes from the mirror to the remote database, and also replicates changes from the remote database to the mirror database.

DBCS. See *double-byte character set*.

DHCP. See *Dynamic Host Configuration Protocol*.

DPROP. See *DB2 DataPropagator*.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes.

Dynamic Host Configuration Protocol (DHCP). An Internet protocol for automating the configuration of computers that use TCP/IP.

E

enterprise database. See *source database*.

enterprise server. See *source server*.

F

filter. A device or program that separates data, signals, or material in accordance with specified criteria.

G

group. A collection of clients that have similar mobile data synchronization needs. You define synchronization characteristics for each group, such as which applications the users in the group need to access to perform their jobs and what subsets of enterprise data they need to access.

H

handheld device. Any computing device that can be held in the hand. Handheld devices include palm-sized PCs and personal digital assistants (PDAs).

I

IBM Sync. The name for the icon representing the client component of the DB2 Everyplace Sync Server software.

J

join. A relational operation that allows for retrieval of data from two or more tables based on matching column values.

K

key. A column or an ordered collection of columns that are identified in the description of a table, index, or referential constraint.

L

large object (LOB). A sequence of bytes, where the length can be up to 2 gigabytes. It can be any of three types: BLOB (binary), CLOB (single-byte character or mixed), or DBCLOB (double-byte character).

LOB. See *large object*.

local database. A database that is physically located on the computer in use. Contrast with *remote database*.

log. A Administrator object containing synchronization error messages and their descriptions.

M

master database. See *source database*.

MDAC. See *Administrator*.

mid-tier system. The machine where the DB2 Everyplace Sync Server is installed. In a two-tier synchronization configuration, the mid-tier and source systems refer to the same machine.

mirror database. A database that the Sync Server uses internally to store the data that is required for synchronization and replication.

mobile. Pertaining to computing that is performed on a portable computer or a handheld device by a user who is frequently moving among various locations and using different types of network connections (for example, dial-up, LAN, or wireless).

mobile data synchronization. A two–step process whereby mobile users, or *clients*, submit changes that they made to local copies of source data and receive any changes that were made to source data (in a remote database) since the last time they synchronized.

Administrator (MDAC). A graphical interface that allows you to create, edit, and view synchronization objects and their relationships to each other. The Administrator also allows you to view synchronization status of individual clients and error messages.

O

object.

1. Anything that can be created or manipulated with SQL—for example, tables, views, indexes, or packages.
2. In object-oriented design or programming, an abstraction consisting of data and operations associated with that data.

ODBC. See *Open Database Connectivity*.

Open Database Connectivity (ODBC). An API that allows access to database management systems using callable SQL, which does not require the use of an SQL preprocessor. The ODBC architecture allows users to add modules, called database drivers, that link the application to their choice of database management systems at run time. Applications do not need to be linked directly to the modules of all the supported database management systems.

P

PDA. See *personal digital assistant*.

persistent. Pertaining to data that is maintained across session boundaries, usually in nonvolatile storage such as a database system or a directory.

personal digital assistant (PDA). A handheld device that is used for personal organization tasks (such as managing a calendar and note-taking) and includes telephone, fax, and networking features.

pervasive computing (PVC). The use of a computing infrastructure that includes specialized appliances, known as information appliances, from which users can access a broad range of network-based services (including services that are typically offered through the Internet). These information appliances include televisions, automobiles, telephones, refrigerators, and microwave ovens. Pervasive computing provides convenient access to relevant information and the ability to take action on that information.

primary key. A unique key that is part of the definition of a table. A primary key is the default parent key of a referential constraint definition. With

the DB2 Everyplace Sync Server Version 7, each replication source must have one and only one primary key.

privilege. The right to access a specific database object in a specific way. These rights are controlled by users with SYSADM (system administrator) authority or DBADM (database administrator) authority or by creators of objects. Privileges include rights such as creating, deleting, and selecting data from tables.

PVC. See *pervasive computing*.

Q

QBE. See *Query-by-Example*.

query. A request for information from the database based on specific conditions; for example, a request for a list of all customers in a customer table whose balance is greater than $1000.

Query-by-Example. An application that allows a user to dynamically view and modify the data stored in a DB2 Everyplace table.

R

RAS. See *Remote Access Service*.

refresh. A process in which all of the data of interest in a user table is copied to the target table, replacing existing data.

remote database. A database that is physically located on a computer other than the one in use. Contrast with local database. The remote computing device can be stationary and nonportable, or it can be portable.

Remote Access Service (RAS). A Windows program that manages connections between two systems.

replication. The process of taking changes that are stored in the database log or journal at a source server and applying them to a target server.

replication source. A database table that is defined as a source for replication. After you define a database table as a replication source, the table can accept copy requests.

S

SQL. See *Structured Query Language*.

source database. A database residing on a source server containing data to be copied to a target system.

source server. The database location of the replication source.

source table. A table that contains the data that is to be copied to a target table. The source table must be a replication source table. Contrast with *target table*.

subscription. A specification for how the information in a source database is to be replicated to a target database. A subscription allows you to define which subsets of data and files can be copied from the source database. You can create two types of subscriptions: file subscriptions for files stored at the source server and table subscriptions for tables in the source database.

subscription set. A Administrator object containing replication subscriptions. To provide group members with access to the data and files defined in replication subscriptions, you create a subscription set and assign subscriptions to it, then assign the subscription set to a group. The subscription set object replaces the application object.

synchronization. *See* mobile data synchronization.

synchronization object. A manageable item within the Administrator that contains information about aspects of the synchronization process in your organization. There are five types of synchronization objects: group, client, subscription set, subscription, and log.

synchronization session. A transaction in which mobile users, or *clients*, submit changes that they made to local copies of source data and receive any changes that were made to source data (residing on the remote server) since the last time they synchronized.

Structured Query Language (SQL). A programming language that is used to define and manipulate data in a relational database.

T

target database. A DB2 Everyplace database residing on a mobile device to which data from a source database is copied.

target table. A table to which data from a source table is copied. Mirror tables on the mid-tier server are targets, and DB2 Everyplace tables on the mobile device are targets.

tap. To use a stylus to interact with a handheld device.

temporary table. A table created during the processing of an SQL statement to hold intermediate results.

V

view. A logical table that consists of data that is generated by a query.

W

wireless LAN. In wireless uses, a mobile user can connect to a local area network (LAN) through a radio connection. Wireless technologies for LAN connection include speed spectrum, microwave, and infrared light.

Index

Special characters

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Contacting IBM

For information or to order any of the DB2 Everyplace products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

http://www.ibm.com/software/data/db2/everyplace/
The DB2 Everyplace World Wide Web pages provide current DB2 Everyplace information about news, product descriptions, education schedules, and more.

http://www.ibm.com/software/data/db2/everyplace/library.html
The DB2 Everyplace Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 Everyplace technical information.

Note: This information may be in English only.

http://www.ibm.com/software/data/
The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more.

http://www.ibm.com/software/data/db2/library/
The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information.

Note: This information may be in English only.

http://www.elink.ibmlink.ibm.com/pbl/pbl/
The International Publications ordering Web site provides information on how to order books.

http://www.ibm.com/education/certify/
The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products, including DB2.

ftp.software.ibm.com
Log on as anonymous. In the directory `/ps/products/db2`, you can find demos, fixes, information, and tools relating to DB2 and many other products.

comp.databases.ibm-db2, bit.listserv.db2-l
These Internet newsgroups are available for users to discuss their experiences with DB2 products.

On Compuserve: GO IBMDB2

Enter this command to access the IBM DB2 Family forums. All DB2 products are supported through these forums.

For information on how to contact IBM outside of the United States, refer to Appendix A of the *IBM Software Support Handbook*. To access this document, go to the following Web page: http://www.ibm.com/support/, and then select the IBM Software Support Handbook link near the bottom of the page.

Note: In some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

Program Number: 5724–D04

Printed in USA

SC18-7185-01