IBM DB2 Everyplace

Application and Development Guide

Version 9 Release 1

SC18-9996-00

IBM DB2 Everyplace

Application and Development Guide

Version 9 Release 1

SC18-9996-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 453.

| **Eighth Edition (July 2006)**

| This edition applies to Version 9, Release 1 of IBM DB2 Everyplace and to all subsequent releases and modifications
| until otherwise indicated in new editions.

Contents

About this book

This book is designed to help database administrators, system programmers, application programmers, and system operators perform the following tasks:

- Design and write applications for DB2 Everyplace
- Diagnose and recover from DB2 Everyplace problems

Always check the DB2 Everyplace Library page for the most current version of this publication:

http://www.ibm.com/software/data/db2/everyplace/library.html

Conventions used in this book

This documentation uses the following highlighting conventions:

- **Boldface type** indicates commands or user interface controls such as names of fields, folder, icons, or menu choices.
- `Monospace type` indicates examples of text that you enter exactly as shown.
- *Italic type* indicates variables that you should replace with a value. It is also used to indicate book titles and to emphasize significant words.

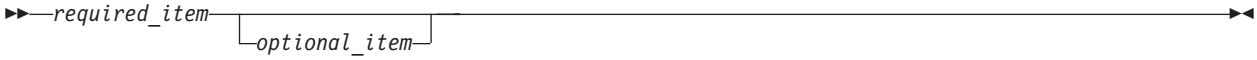
In this documentation, <DSYPATH> refers to the directory where DB2 Everyplace is installed. For instructions that are specific to Linux and UNIX systems, $DSYINSTDIR refers to the directory where the DB2® Everyplace® Sync Server instance is located for a given user ID.

How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol indicates that a syntax diagram is continued from the previous line.
  - The --->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path.



  If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

- If you can choose from two or more items, they appear vertically, in a stack.

If you *must* choose one of the items, one item of the stack appears on the main path.

If choosing one of the items is optional, the entire stack appears below the main path.

If one of the items is the default, it appears above the main path, and the remaining choices are shown below.

- An arrow returning to the left, above the main line, indicates an item that can be repeated.

If the repeat arrow contains a comma, you must separate repeated items with a comma.

A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown. Variables appear in all lowercase italic letters (for example, *column-name*). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).

Service updates and support information

To find service updates and support information, including software fix packs, Frequently Asked Questions (FAQs), technical notes, troubleshooting information, and downloads, refer to the following Web page:

http://www.ibm.com/software/data/db2/everyplace/support.html

Receiving information updates automatically

By registering with the IBM My Support service, you can automatically receive a weekly e-mail that notifies you when new DCF documents are released, when existing product documentation is updated, and when new product documentation is available. You can customize the service so that you receive information about only those IBM products that you specify.

To register with the My Support service:

1. Go to http://www.ibm.com/support/mysupport.
2. Enter your IBM ID and password, or create one by clicking **register now**.
3. When the My Support page is displayed, click **add products** to select those products that you want to receive information updates about. DB2 Everyplace is located under **Software** -> **Data and Information Management** -> **Databases**.
4. Click **Subscribe to email** to specify the types of updates that you would like to receive.
5. Click **Update** to save your profile.

The DB2 Everyplace information set

DB2 Everyplace technical information is available in the following formats:

PDF files

The PDF versions of the books are titled as follows:

- *DB2 Everyplace Application and Development Guide*
- *DB2 Everyplace Sync Server Administration Guide*
- *DB2 Everyplace Installation and User's Guide*

A description of each book in the DB2 Everyplace library is available from:

- The IBM® Publications Center at http://www.ibm.com/shop/publications/order
- The DB2 Everyplace Web site at http://www.ibm.com/software/data/db2/everyplace/library.html

IBM developerWorks

IBM developerWorks has technical articles about DB2 Everyplace and a forum for interacting with other DB2 Everyplace users. You can access the DB2 Everyplace section of IBM developerWorks at http://www.ibm.com/developerworks/db2/products/db2e/index.html .

Information center

The information center contains the entire DB2 Everyplace library in a convenient searchable format. More information is available on the DB2 Everyplace library page at http://www.ibm.com/software/data/db2/everyplace/library.html.

under the **Information Center** heading.

Online help

You can open HTML browser-based online help from the Mobile Devices Administration Center user interface.

Accessibility features

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully.

The Installer, Configuration Wizard, and Mobile Devices Administration Center are all accessible and include the following accessibility features:

- Operate all features by using the keyboard instead of the mouse.
- Customize the size and color of your fonts.
- Receive either visual or audio alert cues.
- Supports accessibility applications that use the Java™ Accessibility API.
- Comes with documentation that is provided in an accessible format.

Keyboard input
You can use keys or key combinations to perform operations that can also be done by using a mouse. You can access context-sensitive menus from the menu bar instead of right-clicking.

Accessible display
DB2 Everyplace has features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizing font properties.

Font settings
You can select the color, size, and font for the text in menus and dialog windows.

No dependence on color
You do not need to distinguish between colors in order to use any of the functions in this product.

Supports high contrast colors
The Mobile Devices Administration Center displays well when you use a high contrast color scheme.

No flashing or blinking content
No text or graphical user interface elements flash or blink during operation.

Alternative alert cues
You can specify whether you want to receive alerts through audio or visual cues.

Compatibility with assistive technologies
The Mobile Device Administration Center interface supports the Java Accessibility API, enabling use by screen readers and other assistive technologies used by people with disabilities.

Accessible documentation
Accessible documentation for DB2 Everyplace is available in the DB2 Everyplace information center.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other DB2 Everyplace documentation, use either of the following options:

- Use the online reader comment form, which is located at:
  http://www.ibm.com/software/data/rcf/
- Send your comments by e-mail to comments@us.ibm.com. Be sure to include the name of the book, the part number of the book, the version of DB2 Everyplace, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

IBM DB2 Everyplace Application and Development Guide version 9.1

Product overview

DB2 Everyplace is part of the IBM On Demand Business solution for synchronizing data between mobile devices and enterprise database servers.

By using DB2 Everyplace, mobile professionals (such as sales people, inspectors, auditors, field service technicians, doctors, realtors, and insurance claim adjusters) can have access to vital data that they need when they are away from the office. Organizations can deliver their enterprise data to mobile devices. By using the DB2 Everyplace mobile database, you can access and perform updates to databases that reside on mobile devices. By using the DB2 Everyplace Sync Server and the Sync Client, you can synchronize data from mobile devices to other data sources in your enterprise.

The following editions of DB2 Everyplace are available:

DB2 Everyplace Enterprise Edition
Provides database and synchronization software for mobile devices. This edition allows you to create a complete enterprise synchronization solution for your mobile devices. This edition includes:

- DB2 Everyplace Sync Server (including the Mobile Devices Administration Center and the XML Scripting tool)
- DB2 Everyplace mirror database
- DB2 Everyplace Sync Client
- DB2 Everyplace mobile database

DB2 Everyplace Database Edition
Provides the DB2 Everyplace mobile database. This edition is ideal if you only want a robust and small database for your mobile devices and do not need to synchronize data with an enterprise data source.

An example DB2 Everyplace scenario

DB2 Everyplace can increase the productivity and efficiency of a mobile workforce. In this example, an insurance adjuster uses a mobile device that runs a DB2 Everyplace application.

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 that runs a DB2 Everyplace application can considerably streamline this process. By 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. The adjusters can then 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 by using modem or wireless connection.

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 have instant access to their company's enterprise databases.

Components of the DB2 Everyplace solution

DB2 Everyplace Enterprise Edition is a robust solution for synchronizing enterprise data. You can configure the DB2 Everyplace environment in multiple ways depending on the needs of your network and your users.

DB2 Everyplace mobile database
The DB2 Everyplace mobile database engine runs on a mobile device and stores a local copy of data from a source system. Users can use the mobile device to access and modify this data. The DB2 Everyplace mobile database is included with DB2 Everyplace Database Edition, DB2 Everyplace Enterprise Edition, and the Mobility on Demand feature of DB2.

The DB2 Everyplace mobile database is a relational database that resides on your mobile device. You can interact with the database through DB2 Call Level Interface (CLI) functions, Java Database Connectivity (JDBC) methods, Open Database Connectivity (ODBC) methods, or ADO.NET methods.

DB2 Everyplace Sync Server
The DB2 Everyplace Sync Server is a servlet that synchronizes data and provides conflict resolution between embedded databases on mobile devices and a source database. When you install DB2 Everyplace, you install the DB2 Everyplace Sync Server servlet and a limited functionality embedded application server. You can also configure the DB2 Everyplace Sync Server to run inside a stand-alone application server such as WebSphere Application Server version 6.

You can administer the DB2 Everyplace Sync Server by using two tools:

Mobile Devices Administration Center
This graphical tool helps you manage and deliver synchronization services to groups of users with similar data synchronization needs.

The XML Scripting tool
The XML Scripting tool automates tasks otherwise performed using the Mobile Devices Administration Center. You can also use the XML Scripting tool to copy or move subscriptions, subscription sets, users, groups from one server to several other servers.

DB2 Everyplace mirror database
The DB2 Everyplace mirror database stores the data that you want to synchronize between your mobile devices and your enterprise databases. The DB2 Everyplace Sync Server uses the mirror database to perform conflict resolution between mobile devices and to minimize load on your enterprise database systems.

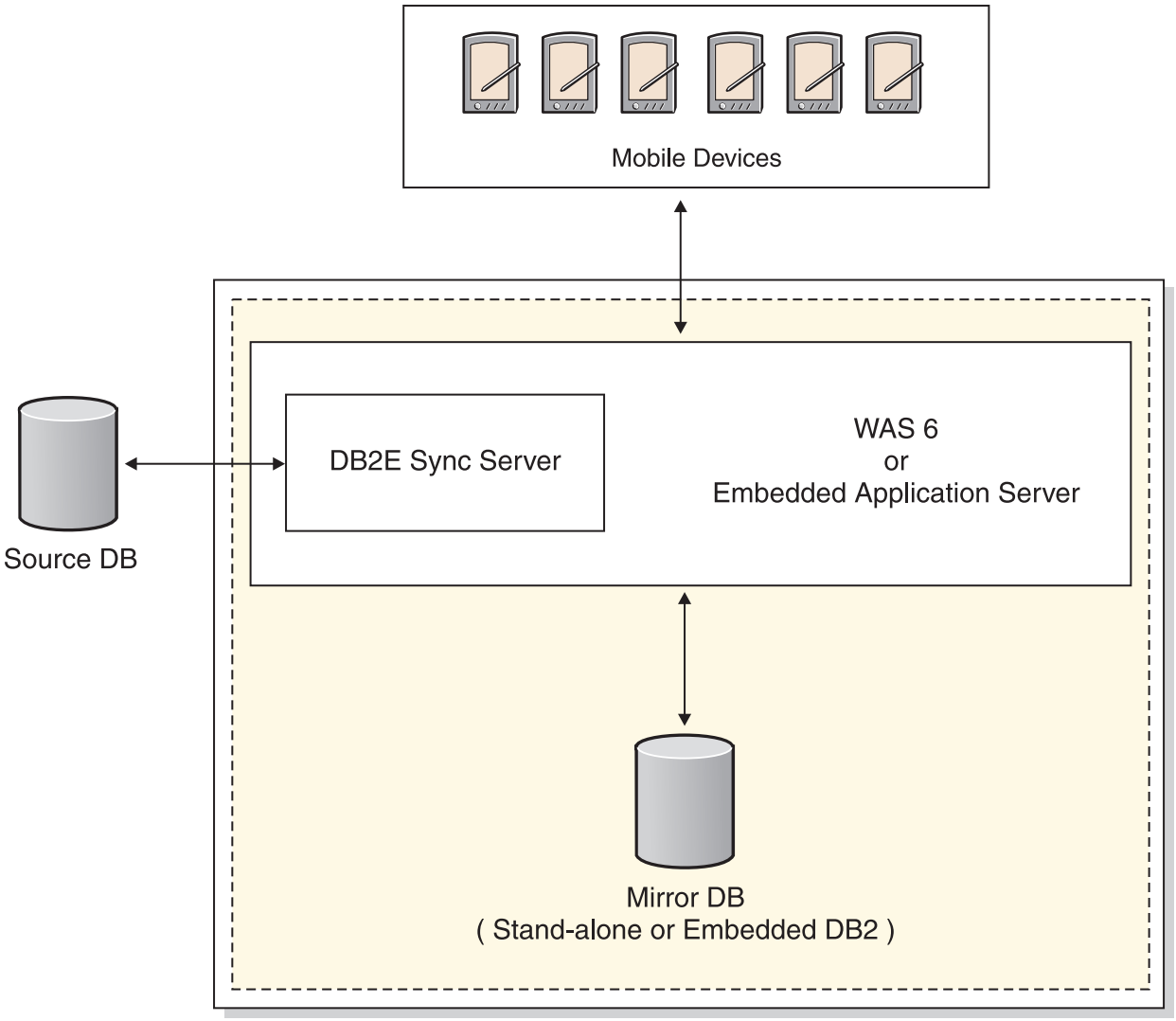
If you have a stand-alone copy of DB2 Version 9.1 on your system, create or catalog the mirror database on the local DB2 instance of DB2 Everyplace. If you do not have DB2 Version 9.1 on your system, DB2 Everyplace installs an embedded, restricted version of DB2 Version 9.1 to function as the mirror database.

DB2 Everyplace Sync Client
The DB2 Everyplace Sync Client is a component that mobile applications use to synchronize data between the DB2 Everyplace Sync Server. It handles bidirectional synchronization of enterprise relational data with the DB2 Everyplace mobile database. The DB2 Everyplace Sync Client also allows you to easily distribute and update mobile applications on your mobile devices as well as run stored procedures that reside on a DB2 database.

How data is exchanged between DB2 Everyplace and mobile devices

The DB2 Everyplace Sync Server defines relationships and access rights through DB2 Version 9.1 replication objects such as users, groups, subscriptions, and subscription sets. This information as well as a local copy of the source data are kept on the DB2 Everyplace mirror database. The DB2 Everyplace

Sync Server transfers data to mobile devices through the TCP/IP interface that is provided by the application server. Mobile devices can synchronize data through any channel that supports TCP/IP, such as a direct USB connection or an Internet connection.



DB2 Everyplace environments

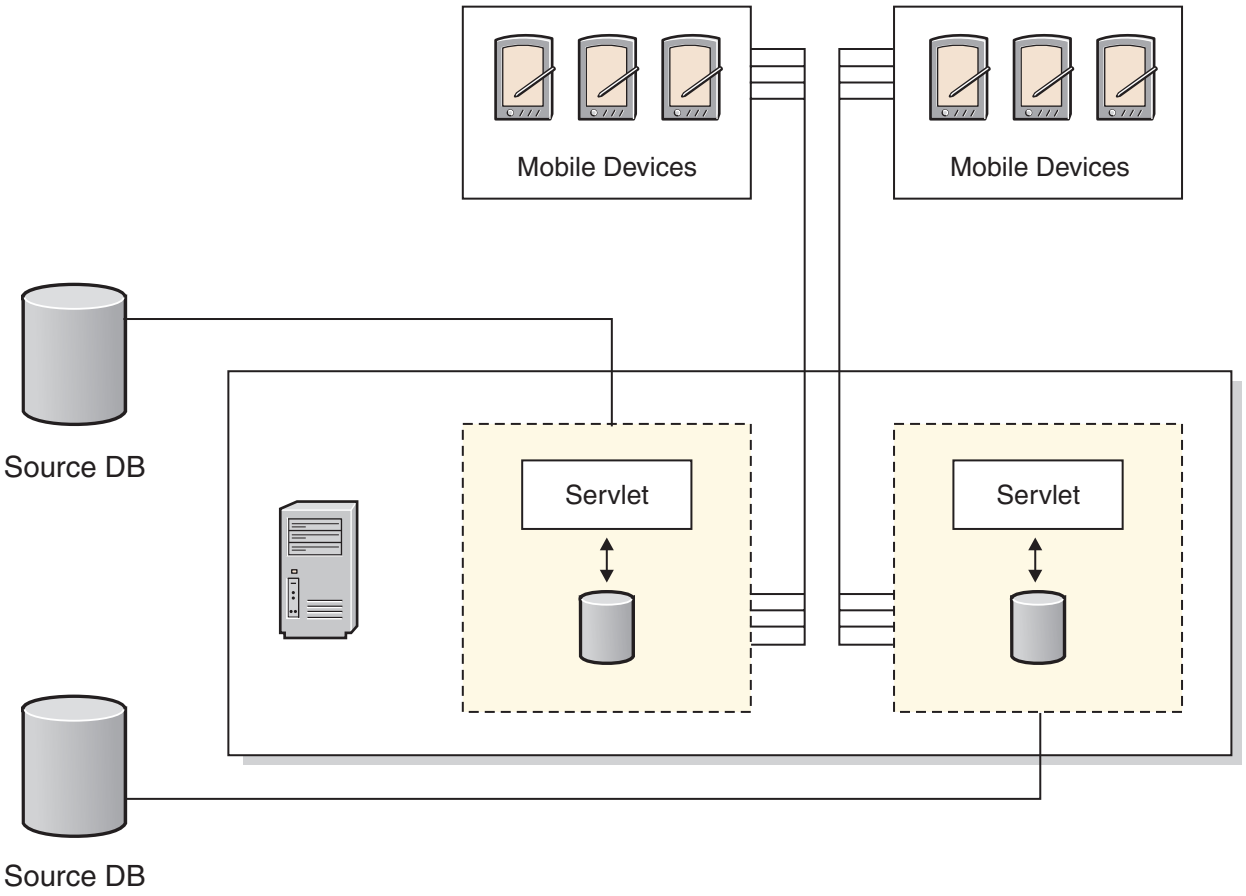
DB2 Everyplace Enterprise Edition is a robust solution for synchronizing enterprise data. You can configure the DB2 Everyplace environment in multiple ways depending on the needs of your network and your users.

Underlying concepts

Like DB2 Version 9.1, DB2 Everyplace operates using instances. A DB2 Everyplace instance consists of the following components:

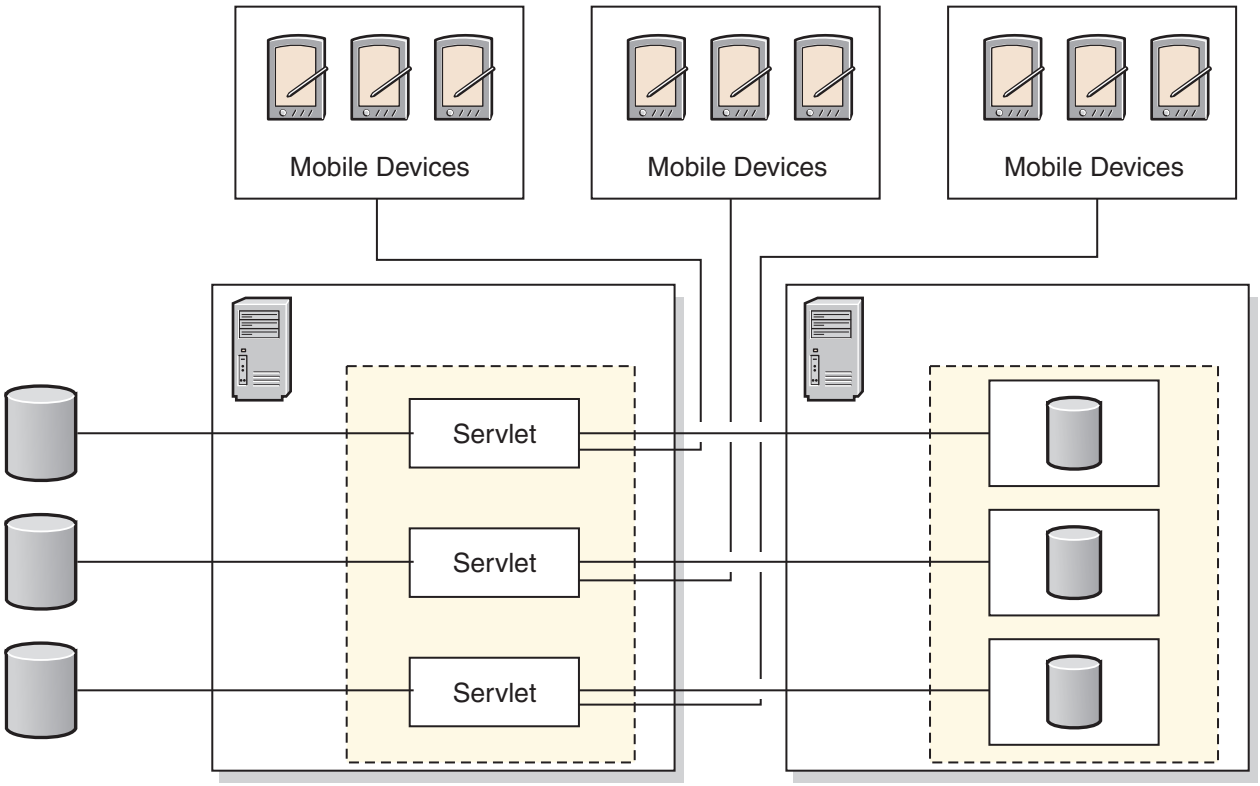
- The DB2 Everyplace Sync Server running in an embedded application server instance
- The DB2 Everyplace mirror database, which is managed by an instance of DB2 Version 9.1

The basic environment



A basic environment places the DB2 Everyplace instance on one physical machine.

The distributed environment



A distributed environment divides the DB2 Everyplace instance into two components:

Distributed server
The distributed server is a machine that runs an instance of the DB2 Everyplace Sync Server inside an embedded application server instance.

Distributed database
The distributed database is a machine that hosts the DB2 Everyplace mirror database on an instance of DB2 Version 9.1.

The clustered environment

A clustered environment allows you to scale the DB2 Everyplace Sync Server across several nodes in a WebSphere Application Server Network Deployment cluster. WAS ND empowers DB2 Everyplace with additional features such as dynamic load balancing, scalability, and fail over.

The remote administration environment

A remote administration environment allows you to administer a DB2 Everyplace Sync Server instance that is located on another physical system by using the Mobile Devices Administration Center. For example, you can install DB2 Everyplace on a Windows workstation, configure it for a remote administration environment, and use it to administer a DB2 Everyplace Sync Server on an AIX system.

The DB2 Everyplace sample applications

The sample applications provide examples of applications that use DB2 Everyplace.

The Visiting Nurse sample application demonstrates bidirectional synchronization between the mobile database and the Sync Server. The sample application has two parts: one part runs on the Sync Server and another part runs on the mobile device that has the DB2 Everyplace mobile database. The sample application on the mobile device demonstrates the database engine functionality in a standalone environment. When the Sync Server sample application and the DB2 Everyplace mobile database engine sample application are used together, they work as a complete application that invokes all components of DB2 Everyplace.

The DB2 Sync sample application demonstrates how to use the IBM Sync Client API to synchronize tables of the subscriptions that are defined in the Mobile Devices Administration Center.

Developing

DB2 Everyplace provides support for developing applications in several APIs and languages.

The following topics are discussed in this section.

Developing C/C++ applications using DB2 Everyplace

This topic provides information to help you develop applications using C or C++. It describes in detail how to use the CLI (Call Level interface) and the DB2 Everyplace Sync Client C API.

Developing DB2 Everyplace C/C++ applications

This topic presents information that will help you develop applications using the programming language C or C++.

Prerequisite:

Install DB2 Everyplace on the development workstation.

To develop DB2 Everyplace applications in C/C++ using the DB2 Everyplace CLI interface:

1. 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 mobile database.
2. Understand the DB2 CLI interface and determine what DB2 CLI functions to use in the application.
3. Write a C/C++ application program using the DB2 CLI functions supported in DB2 Everyplace.
4. Prepare, compile, and link the application code with the DB2 Everyplace header files and operating system library.
5. 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.

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 in this topic, return to "Developing DB2 Everyplace C/C++ applications."

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:

   <DSYPATH>\Clients\include\sqlcli.h
   <DSYPATH>\Clients\include\sqlcli1.h
   <DSYPATH>\Clients\include\sqlext.h
   <DSYPATH>\Clients\include\sqlsystm.h

   Note: <DSYPATH> denotes the root installation directory of DB2 Everyplace.
3. Include any header files specific to your application.
4. Include the appropriate DB2 Everyplace library in the project.
5. Optional: Define the macro UNICODE and _UNICODE in your project file to get UNICODE support.

See “DB2 Everyplace Unicode support” on page 353 for more information about 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 an integrated development environment. For additional information on compiling and linking a project, see the documentation included with your application development software.

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	<DSYPATH>\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 might 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	Emulator applications: <DSYPATH>\clients\Symbian7\database\wins\DB2e.lib Device applications: <DSYPATH>\clients\Symbian7\database\armi\DB2e.lib
Symbian OS 7s	Emulator applications: <DSYPATH>\clients\Symbian7s\database\wins\DB2e.lib Device applications: <DSYPATH>\clients\Symbian7s\database\armi\DB2e.lib
Windows® CE	ARM processor: • V3.00 <DSYPATH>\clients\wince\database\v3\armrel\DB2e.lib • V4.00 <DSYPATH>\clients\wince\database\v4\ARM4VRel\DB2e.lib MIPS processor: • V3.00 <DSYPATH>\clients\wince\database\v3\mipsrel\DB2e.lib • V4.00 <DSYPATH>\clients\wince\database\v4\MIPSIVRel\DB2e.lib Windows CE emulator: • V3.00 <DSYPATH>\clients\wince\database\v3\x86emrel\DB2e.lib (for Pocket PC emulator) <DSYPATH>\clients\wince\database\v4\x86rel\DB2e.lib (for Pocket PC 2002 emulator) • V4.00 <DSYPATH>\clients\wince\database\v4\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: • v4.00 <DSYPATH>\clients\wince\database\v4\XScaleRel\DB2e.lib
Windows	<DSYPATH>\clients\Win32\database\x86\DB2e.lib
Neutrino	libdb2e.so This file is located in the <DSYPATH>/clients/neutrino/database/*proc* directory.
Linux™	libdb2e.so This file is located in the $DSYINSTDIR/Clients/Linux/database/*proc* directory.

Related concepts

"The sample C/C++ applications" on page 13

Related tasks

"Testing a C/C++ application"
The testing procedure for a DB2 Everyplace C/C++ application depends on the type of mobile device. Use these instructions to properly test your application for your target platform.

Related reference

"C/C++ development tools" on page 13
Depending on the operating system of the mobile device, you will need a particular integrated development environment (IDE).

"C/C++ supported operating systems" on page 14
DB2 Everyplace supports C/C++ development on a variety of operating systems.

"DB2 CLI function summary" on page 163

"DB2 Everyplace Unicode support" on page 353

Testing a C/C++ application

The testing procedure for a DB2 Everyplace C/C++ application depends on the type of mobile device. Use these instructions to properly test your application for your target platform.

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

1. Copy the DB2 Everyplace libraries to the emulator or device for your operating system. Without these files, a DB2 Everyplace application will not load. Table 2 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	<DSYPATH>\clients\palmos\database\DB2eCat.prc <DSYPATH>\clients\palmos\database\DB2eCLI.prc <DSYPATH>\clients\palmos\database\DB2eComp.prc <DSYPATH>\clients\palmos\database\DB2eRunTime.prc <DSYPATH>\clients\palmos\database\DB2eDMS.prc
Symbian OS	For emulator testing, copy the file <DSYPATH>\clients\symbian7\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 PC Suite connection software: <DSYPATH>\clients\symbian7\database\armi\DB2e.sis
Symbian OS 7s	For emulator testing, copy the file <DSYPATH>\clients\symbian7s\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 PC Suite connection software: <DSYPATH>\clients\symbian7s\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 <DSYPATH>\clients\wince\database\v3\armrel\DB2e.dll • V4.00 <DSYPATH>\clients\wince\database\v4\armv4rel\DB2e.dll • V5.00 <DSYPATH>\clients\wince\database\v4\armv4rel\DB2e.dll MIPS processor: • V3.00 <DSYPATH>\clients\wince\database\v3\mipsrel\DB2e.dll • V4.00 <DSYPATH>\clients\wince\database\v4\mipsivrel\DB2e.dll Windows CE emulator: • V3.00 For Pocket PC emulator: <DSYPATH>\clients\wince\database\v3\x86emrel\DB2e.dll For Pocket PC 2002 emulator: <DSYPATH>\clients\wince\database\v3\x86rel\DB2e.dll • V4.00 For STANDARDSDK emulator: <DSYPATH>\clients\wince\database\v4\emulatorrel\ DB2e.dll For Windows Mobile 2003 emulator: <DSYPATH>\clients\wince\database\v4\x86rel\ DB2e.dll • V5.00 For Windows Mobile 5.0 emulator: <DSYPATH>\clients\wince\database\v4\armv4rel\ DB2e.dll
Windows	Copy <DSYPATH>\clients\win32\database\x86\DB2e.dll to either the current directory of the application or the PATH environment variable of the system.
Neutrino	$DSYINSTDIR/database/<*platform*>/libdb2e.so where *platform* is one of the following options: **x86** x86 processor **strongarm** StrongARM processor
Linux	$DSYINSTDIR/database/<*platform*>/libdb2e.so where *platform* is one of the following options: **x86** x86 processor **strongarm** StrongARM processor **xscale** XScale processor

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 (this command only works in bash, with libdb2e.so in the current directory): export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<*dir*> where <*dir*>is the directory where libdb2e.so resides.

3. Load the files for the application you are testing.
4. Test the application.

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
This topic presents information that will help you develop applications using the programming language C or C++.

Related reference

"C/C++ development tools"
Depending on the operating system of the mobile device, you will need a particular integrated development environment (IDE).

"C/C++ supported operating systems" on page 14
DB2 Everyplace supports C/C++ development on a variety of operating systems.

C/C++ development tools

Depending on the operating system of the mobile device, you will need a particular integrated development environment (IDE).

Palm OS

You can use

- 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.palmsource.com/developers/.

Symbian OS

You can use the following tools to develop applications for Symbian OS:

- CodeWarrior for Symbian OS
- Borland C++ BuilderX Mobile or Mobile Studio
- Microsoft® Visual C++ 6

Recommendation: Obtain UIDs for your applications from Symbian Signed. Also refer to the UID FAQ at https://www.symbiansigned.com/app/page/uidfaq, and consider getting your application signed by Symbian.

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 C++ 4.0 to develop native C/C++ applications for .NET devices. For Windows Mobile 2003 for Pocket PC and .NET devices, use Microsoft Visual Studio .NET 2003 to develop managed applications for the Compact Framework.

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 must support ELF binaries enabled.

If you are developing the application on a system that has the same architecture as the target system, you can also use the GNU C/C++ tools.

Related concepts

"The sample C/C++ applications" on page 13

Related tasks

"Developing DB2 Everyplace C/C++ applications" on page 9
This topic presents information that will help you develop applications using the programming language C or C++.

C/C++ supported operating systems

DB2 Everyplace supports C/C++ development on a variety of operating systems.

The supported operating systems include:

- Palm OS
- Symbian OS
- Windows® CE® for Pocket PC
- Windows (Windows® 2000®, Windows® XP®, Windows® 2003)
- QNX Neutrino
- Linux and embedded Linux

Related concepts

"The sample C/C++ applications" on page 13

Related tasks

"Developing DB2 Everyplace C/C++ applications" on page 9
This topic presents information that will help you develop applications using the programming language C or C++.

Related reference

"C/C++ development tools" on page 13
Depending on the operating system of the mobile device, you will need a particular integrated development environment (IDE).

"DB2 CLI function summary" on page 163

CLI (call level interface)

This topic presents information that will help you develop applications using CLI, a programming interface that C and C++ applications can use to access DB2 Everyplace databases.

What is the CLI?

DB2 Everyplace Call Level Interface (CLI) is IBM's callable SQL interface to the DB2 Everyplace database. It is a C and C++ application programming interface for relational database access that uses function calls to pass dynamic SQL statements as function arguments.

DB2 Everyplace CLI is based on the DB2 Version 9.1 CLI, which is based on the Microsoft Open Database Connectivity (ODBC) specification and the International Standard for SQL/CLI. These specifications were chosen as the basis for the DB2 Everyplace CLI in an effort to follow industry standards and to provide a shorter learning curve for programmers already familiar with either of these database interfaces. DB2 Everyplace CLI includes support for many ODBC and ISO SQL/CLI functions, as well as DB2 Everyplace-specific features.

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

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

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

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

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

The DB2 Everyplace CLI library can be loaded as an ODBC driver by an ODBC driver manager. For ODBC application development, you must obtain an ODBC Software Development Kit. For the Windows platform, the ODBC SDK is available as part of the Microsoft Data Access Components (MDAC) SDK, available for download from http://www.microsoft.com/data/. For non-Windows platforms, the ODBC SDK is provided by other vendors. When developing ODBC applications that connect to DB2 Everyplace databases, use this book (for information about DB2 Everyplace-specific extensions and diagnostic information), in conjunction with the OBDC Programmer's Reference and SDK Guide available from Microsoft. Applications written directly to DB2 Everyplace CLI link directly to the DB2 Everyplace CLI library.

Connecting to the DB2 Everyplace mobile database

Applications typically create and access tables in a specific location, for example, the C:\TEMP directory. You can use the CLI call to specify a location when connecting to a DB2 Everyplace mobile database.

In the following example, *path* represents the path to the DB2 Everyplace mobile database.

```
rc = SQLConnect(hdbc, path, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

The path can include (but does not require) the database name. Thus, both of the following examples are correct, assuming a DB2 Everyplace mobile database exists in C:\TEMP.

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

Connecting to Sony Memory Stick extended memory under Palm OS requires a special path specification, as the following example shows.

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

Using DB2eCLP, you can connect to a specific location using the ″CONNECT TO″ command. For example, the following command connects to the DB2 Everyplace mobile database in C:\TEMP\ on a computer running Windows:

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

CAUTION:
For Windows and Windows CE platforms, it is unsafe to call DB2 Everyplace from within DllMain. This is especially important for version 8.2 because DB2 Everyplace added a background thread for performance. For example, an application that calls SQLConnect() within DllMain will experience a deadlock or other unexpected results. For more information about this issue, consult the Microsoft documentation.

Related concepts

"Overview of the DB2 Everyplace mobile database tables" on page 61
A DB2 Everyplace mobile database comprises several system catalog tables and a number of user-defined tables.

Related tasks

"Handling naming conflicts between tables" on page 62
This topic shows some examples of ways that you can handle file naming conflicts for user-defined tables.

Piecemeal retrieval of data through the CLI

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

There are two ways for you to retrieve a column value in pieces:

- SQLGetData()
- SQLGetSubstring()

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.

An example using SQLGetData():

```
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
         :
 }
```

While SQLGetData() works by breaking a BLOB down into consecutive segments and feeding them start-to-finish to the client application, SQLGetSubString() allows the application programmer additional flexibility because the data stream can start at any arbitrary location within the file, not just the

beginning. In addition, SQLGetSubString() allows the application to request number of bytes out of the BLOB (rather than reading all the way to the end of the BLOB). This requested portion of the BLOB is then split into segments and fed to the client.

For example:

```
/* get the LOB locator */
 sqlrc = SQLGetData(hstmt1, 1, SQL_C_BLOB_LOCATOR, (SQLPOINTER) &loc1, 0, &ind1);

 /* get the length of the BLOB */
 sqlrc = SQLGetLength(hstmt2, SQL_C_BLOB_LOCATOR, loc1, &len, NULL);

 /* retrieve the first 26 bytes from the LOB locator into the variable data1 */
 sqlrc = SQLGetSubString(hstmt2, SQL_C_BLOB_LOCATOR, loc1, 1, 26, SQL_C_BINARY,
                              data1, 52, &bufPos1, NULL);
```

Note: It is the application's responsibility to keep track of how many bytes have been read from the BLOB. The value of the *FromPosition* argument should be incremented by *BufferLength* with each subsequent call to SQLGetSubString().

Related reference

SQLGetSubString()

SQLGetData()

SQLSetStmtAttr()

SQLPrepare()

SQLBindParameter()

SQLExecute()

SQLFetch()

SQLGetLength()

Piecemeal insertion of data through the CLI

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

Prerequisites

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

Restrictions

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

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

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

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

```
/* dtlob.c */
/* ... */
SQLINTEGER      blobInd ;
/* ... */
blobInd = SQL_DATA_AT_EXEC;
sqlrc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
                         SQL_BLOB, BUFSIZ, 0, (SQLPOINTER)inputParam,
                         BUFSIZ, &blobInd);
```

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

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

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

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

Related reference

"SQLGetSubString function (CLI) - Retrieve portion of a string value" on page 232

"SQLGetData—Get data from a column" on page 217

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 DB2 Everyplace Sync Client C API.

Prerequisites:

Install DB2 Everyplace on the development workstation.

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

1. Define the synchronization application, including:
   - the data it will synchronize
   - the operations allowed
   - the users and the user groups
   - data security (for example, data encryption over the wire and local data encryption)
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.
3. Prepare, compile, and link the application code with the DB2 Everyplace Sync Client operating system libraries, isyncconf and isyncore.
4. Test the application:
   a. Install the DB2 Everyplace libraries on the emulator or device for your operating system.
   b. Test the application on an emulator, if applicable.
   c. Test the application on a device.

   Related concepts

   "The sample DB2 Everyplace Sync Client C/C++ application"
   This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

   Related reference

   "C/C++ development tools" on page 13
   Depending on the operating system of the mobile device, you will need a particular integrated development environment (IDE).

The sample DB2 Everyplace Sync Client C/C++ application

This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

You can download this example from the DB2 Everyplace Web site at http://www.ibm.com/software/data/db2/everyplace/. You can find more source code examples in <DSYPATH>\Clients\clientapisample\C_API, where <DSYPATH> is the directory where DB2 Everyplace is installed.

```
/*****************************************************************/
/**
 * 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

#include "isyncore.h"
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
    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 18
This topic provides an overview of how to develop DB2 Everyplace Sync Client applications using C/C++ based on the IBM DB2 Everyplace Sync Client C API.

Related reference

"C/C++ development tools" on page 13
Depending on the operating system of the mobile device, you will need a particular integrated development environment (IDE).

Developing Java applications using DB2 Everyplace

DB2 Everyplace provides a Java API that you can use to develop applications that use the DB2 Everyplace database.

Prerequisite: Install Java™ and JDBC on your workstation.

To develop DB2 Everyplace Java applications:

1. Import the java.sql package and any other necessary Java classes.
2. Connect to the database either using the DriverManager class or the DataSource interface. See the sample Java application for details. The JDBC URL syntax is jdbc:subprotocol:subname. The DB2 Everyplace subprotocol is db2e.

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

Related concepts

"Sample JDBC database engine applications" on page 26
This topic describes the DB2eAppl.java and the DB2eJavaCLP.java sample applications for the DB2 Everyplace database engine.

Related tasks

"Compiling and running sample Java applications on Palm OS targets" on page 373

"Compiling and running sample Java applications on non-Palm OS targets" on page 371

Related reference

"Overview of DB2 Everyplace JDBC support" on page 317

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.

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

To develop DB2 Everyplace Java applications:

1. Import the java.sql package and any other necessary Java classes.
2. Connect to the database either using the DriverManager class or the DataSource interface. See the sample Java application for details. The JDBC URL syntax is jdbc:subprotocol:subname. The DB2 Everyplace subprotocol is db2e.

Restriction: 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.

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

Related concepts

"Sample JDBC database engine applications" on page 26
This topic describes the DB2eAppl.java and the DB2eJavaCLP.java sample applications for the DB2 Everyplace database engine.

Related tasks

"Compiling and running sample Java applications on Palm OS targets" on page 373

"Compiling and running sample Java applications on non-Palm OS targets" on page 371

Related reference

"Overview of DB2 Everyplace JDBC support" on page 317

Overview of DB2 Everyplace Java 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 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 mobile devices.

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

- DB2Everyplace native synchronization providers
- DB2 Everyplace Java synchronization providers

You can find API documentation in the <DSYPATH>\doc*lang*\javadoc\SyncClientJavaAPI\ directory, where <DSYPATH> is the directory where DB2 Everyplace is installed and *lang* represents a language, for example, en_US. Information about how to create Java applications on the client device based on these providers is provided in the sample files.

Related concepts

"The sample Java native synchronization applications" on page 367

Related tasks

"Installing the JNI-based native synchronization provider" on page 31

DB2 Everyplace Java Sync Client for IBM Cloudscape Version 10

The DB2 Everyplace Java Sync Client for IBM Cloudscape™ Version 10 allows you to build applications that synchronize subscriptions to a IBM Cloudscape Version 10 database. The Java DB2 Everyplace Sync Client for IBM Cloudscape Version 10 is a set of libraries that work with the DB2 Everyplace Sync Server to simplify the synchronization of relational data between enterprise databases and a IBM Cloudscape client. The DB2 Everyplace Sync Server manages the movement of data to and from the device.

This topic includes the following information about the Java Sync Client for IBM Cloudscape Version 10:

- Required software to run the Java DB2 Everyplace Sync Client for IBM Cloudscape Version 10
- Features unsupported by IBM Cloudscape Version 10
- Java Sync Client for IBM Cloudscape Version 10 directory layout

- Setting the CLASSPATH environment variable

Required software to run the Java DB2 Everyplace Sync Client for IBM Cloudscape Version 10

In order to run the Java DB2 Everyplace Sync Client for IBM Cloudscape Version 10, you need the following software products:

- DB2 Everyplace version 8.2 or later
- JDK 1.3.1 or later

Note: JDK 1.3.x requires the JCE 1.2.2 package from Sun. Install the jars in the $JAVA_HOME/jre/lib/ext directory.

Supported Cloudscape drivers

- Embedded version 4
- Embedded version 5
- Embedded version 10

Features not supported by Java DB2 Everyplace Sync Client for IBM Cloudscape Version 10

- The CALL remote procedure
- Custom subscriptions on the server
- Network timeouts, which is set with the isync.timeout property to config.createSyncService
- Over-the-wire encryption, which is enabled with the **Encryption** combo box on the Identification tab of Create Subscription or Edit Subscription windows in the Mobile Devices Administration Center (You can still specify encryption settings, however, they will not be honored for IBM Cloudscape clients.)
- Per-table encryption, which is enabled with the **Encrypt** check box on the Define Replication Subscription window of the Mobile Devices Administration Center

Note: To encrypt the IBM Cloudscape Version 10 database, add the following option (in bold) to the JDBC url:

jdbc:cloudspace:mydb;create=true; **dataEncryption=true;bootPassword=Db2jeveryPlace**

Java DB2 Everyplace Sync Client for IBM Cloudscape Version 10 installation directory layout

The Java DB2 Everyplace Sync Client for IBM Cloudscape Version 10 files are in the following directories:

- <DSYPATH>\Clients\javaclient\ contains the IBM Cloudscape Version 10 ISync API jar.
- <DSYPATH>\Clients\clientapisample\Java_API contains the sample applications
- <DSYPATH>\doc*lang*\SyncClientJavaAPI contains the Javadoc.

Note: <DSYPATH> is the root installation directory of DB2 Everyplace.

Setting the CLASSPATH environment variable

To use the Java Sync Client for IBM Cloudscape Version 10, set your CLASSPATH environment variable to include the following files:

- The IBM Cloudscape Version 10 jar files from your IBM Cloudscape Version 10 installation.
- The IBM Cloudscape Version 10 ISync API jar file (db2jisync.jar).
- The sample applications (<DSYPATH>/Clients/clientapisample/Java_API).

  For example:

```
set CLASSPATH=<DSYPATH>\Clients\DB2j\db2jisync.jar
set CLASSPATH=%CLASSPATH%;%CS_INSTALL%\lib\cs.jar;%CS_INSTALL%\lib\cstools.jar
set CLASSPATH=%CLASSPATH%;<DSYPATH>\Clients\clientapisample\Java_API
```

Related tasks

"Compiling and running the DB2 Sync Console sample Java synchronization application" on page 366

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.

In DB2 Everyplace version 9.1, only one type of DB2 Everyplace native synchronization provider is supported:

- Java Native Interface (JNI) - based native synchronization provider

Character encoding in Java applications

Java strings are in Unicode. However, the application can specify the character encoding of target data by setting the isync.encoding property in the ISyncManager.getISyncService API. See the isync.encoding property in ISyncManager.getISyncService for more information about the supported encodings.

Related concepts

"Overview of DB2 Everyplace Java synchronization providers" on page 23
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 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 mobile devices.

Related tasks

"Installing the JNI-based native synchronization provider" on page 31

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. You can dynamically specify the data encoding format.

For Windows CE and Symbian OS operating systems, the DB2 Everyplace JDBC driver retrieves text from and inserts text to the database using 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 Windows 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.

You 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 Driver interface class in the java.sql package.

Related reference

"DB2 Everyplace Unicode support" on page 353

"DB2 Everyplace language enablers" on page 352

"DB2 Everyplace NLS support by operating system" on page 350

Sample JDBC database engine applications

This topic describes the DB2eAppl.java and the DB2eJavaCLP.java sample applications for the DB2 Everyplace database engine.

Sample 1: DB2eAppl.java

DB2eAppl.java demonstrates how to code a JDBC application for DB2 Everyplace. This application uses the standard input stream. To run this application, your Java environment must support java.lang.System.in.

To use the DB2eAppl.java application:

1. Import the java.sql package.
2. Connect to the database in the current directory, the directory that the DB2eAppl.java application will be run in.

   The sample application provides two ways to get a connection:
   - If the DriverManager class is successfully loaded (Step 2a in the code below) , the DB2 Everyplace JDBC driver com.ibm.db2e.jdbc.DB2eDriver is loaded, and the application uses the DriverManager class to obtain a connection.
   - If the DriverManager class is not found, the application uses the DataSource interface to obtain a connection. If the target environment includes JDBC Optional Package for CDC/Foundation Profile (specified by JSR 169), the application calls getConnection (Step 2b in the code below).
3. Create a Statement object.
4. Set up a very simple sample database that consists of an EMPLOYEE table with two records. This is done using the executeUpdate(String sql) method of the java.sql.Statement interface.
5. Select all records from the EMPLOYEE table, and retrieve the rows by calling the next() method of the java.sql.ResultSet interface.
6. Drop the EMPLOYEE table from the database.
7. Release 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 url = "jdbc:db2e:sample";
        try {
            Connection con; //Step 2
            try {
                Class.forName("java.sql.DriverManager");
                Class.forName("com.ibm.db2e.jdbc.DB2eDriver");
                con = DriverManager.getConnection(url); //Step 2a
            } catch (ClassNotFoundException e) {
                com.ibm.db2e.jdbc.DB2eDataSource ds =
          new com.ibm.db2e.jdbc.DB2eDataSource();
                ds.setUrl(url);
                con = ds.getConnection(); //Step 2b
            }
            Statement st = con.createStatement(); //Step 3
            //Create table: employee  //Step 4
            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 5
            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 6
            st.executeUpdate("Drop table employee");
            System.out.println("*** Deleted table: employee");
            //Release resources Step 7
            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 concepts

"The sample Java native synchronization applications" on page 367

Related tasks

"Developing DB2 Everyplace Java applications" on page 22
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.

"Creating a WebSphere Studio Device Developer project for DB2eAppl.java for Palm OS targets" on page 374

"Creating a WebSphere Studio Device Developer project and adding jar files to the build path for DB2eAppl.java for non-Palm OS targets" on page 372

"Compiling and running sample Java applications on Palm OS targets" on page 373

"Compiling and running sample Java applications on non-Palm OS targets" on page 371

"Running DB2eAppl.java on Windows" on page 375

"Running DB2eAppl.java on Windows CE" on page 376

"Running DB2eAppl.java on a Palm OS simulator" on page 378

"Running DB2eAppl.java on QNX Neutrino or embedded Linux" on page 379

"Running DB2eAppl.java on Symbian" on page 380

Developing with JDBC

This topic explains the basics of developing JDBC applications that interact with DB2 Everyplace.

JDBC interface supported operating systems

This topic presents the operating systems that support the JDBC interface.

The JDBC interface is supported on the following operating systems:

- Palm OS
- Symbian OS
- Windows CE® for Pocket PC
- Windows (Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, and Windows 2003)
- QNX Neutrino
- Linux and embedded Linux

Piecemeal retrieval and insertion of data through JDBC

The JDBC interface allows for piecemeal retrieval and insertion of data. The function InputStreams encapsulates the logic to chunk the data in pieces. The behavior of the BLOB infrastructure changes by allowing Blob objects to be invalidated when closing the associated statement object or by fetching the next row. This makes it necessary to keep the cursor at the same row while working with this object.

The following section contains examples of inserting and retrieving a BLOB:

Inserting a BLOB

```
// CREATE TABLE t1 (c1 INT PRIMARY KEY NOT NULL, c2 BLOB(5M));

            PreparedStatement pstmt

= conn.prepareStatement ("INSERT INTO t1 VALUES (?,?)");

            pstmt.setInt (1, 100);

            File fBlob = new File ( "image1.gif" );

            FileInputStream is = new FileInputStream ( fBlob );

      pstmt.setBinaryStream (2, is, (int) fBlob.length() );
```

```
            pstmt.execute ();

      ...
```

Retrieving a BLOB

```
JDBC Example:  Retrieving a BLOB

        // CREATE TABLE t1 (c1 INT PRIMARY KEY NOT NULL, c2 BLOB(5M));

            Statement stmt = conn.createStatement ();

            ResultSet rs= stmt.executeQuery("SELECT * FROM t1");

            while(rs.next()) {

            int val1 = rs.getInt(1);

            InputStream val2 = rs.getBinaryStream(2);

           ...

                }

            rs.close();

            ...
```

Setting JDBC statement attributes

To make JDBC applications more portable, DB2 Everyplace provides support for setting statement attributes (such as dirty bit control and enabling or disabling table reorganization) through connection properties.

The attributes become default values for newly-created java.sql.Statement, java.sql.PreparedStatement, and java.sql.CallableStatement instances. This approach is an alternative to the approach of casting a java.sql.Statement object into a com.ibm.db2e.jdbc.DB2eStatement object that can invoke methods to set these attributes.

Examples

Using java.sql.DriverManager

```
Properties pt = new Properties();
pt.setProperty("ENABLE_REORG", "false");
pt.setProperty("ENABLE_DELETE_PHYSICAL_REMOVE" , "true");
pt.setProperty("ENABLE_DIRTY_BIT_SET_BY_APPLICATION" , "false");
pt.setProperty("ENABLE_READ_INCLUDE_MARKED_DELETE" , "true");
con = DriverManager.getConnection(url, pt);
```

Using java.sql.Driver

```
Properties pt = new Properties();
pt.setProperty("ENABLE_REORG", "false");
pt.setProperty("ENABLE_DELETE_PHYSICAL_REMOVE" , "true");
pt.setProperty("ENABLE_DIRTY_BIT_SET_BY_APPLICATION" , "false");
pt.setProperty("ENABLE_READ_INCLUDE_MARKED_DELETE" , "true");
pt.setProperty("ENABLE_TABLE_CHECKSUM", "true");
con = Driver.getConnection(url, pt);
```

Using javax.sql.DataSource

```
com.ibm.db2e.jdbc.DB2eDataSource ds =
   new com.ibm.db2e.jdbc.DB2eDataSource();
ds.setUrl(url);
ds.setReorg(false);
ds.setDeletePhysicalRemove(true);
```

```
ds.setDirtyBitSetByApplication(false);
ds.setReadIncludeMarkedDelete(true);
ds.setEnabledTableChecksum(true);
con = ds.getConnection();
```

API reference information

java.sql.Driver interface

Method: Connection connect(String url, Properties info)

Description: Attempts to make a database connection to the given URL.

Table 3. Key/value pairs for info

Key	Value
ENABLE_REORG	True or false. Default is true.
ENABLE_DELETE_PHYSICAL_REMOVE	True or false. Default is false.
ENABLE_DIRTY_BIT_SET_BY_APPLICATION	True or false. Default is false.
ENABLE_READ_INCLUDE_MARKED_DELETE	True or false. Default is false.
ENABLE_TABLE_CHECKSUM	True or false. Default is false.

Method: Connection connect(String url, java.util.Hashtable info)

Description: DB2 Everyplace overloaded method for platforms that don't support java.util.Properties

Table 4. Key/value pairs for info

Key	Value
ENABLE_REORG	True or false. Default is true.
ENABLE_DELETE_PHYSICAL_REMOVE	True or false. Default is false.
ENABLE_DIRTY_BIT_SET_BY_APPLICATION	True or false. Default is false.
ENABLE_READ_INCLUDE_MARKED_DELETE	True or false. Default is false.
ENABLE_TABLE_CHECKSUM	True or false. Default is false.

javax.sql.DataSource interface

Table 5. DB2 Everyplace-specific properties for the DataSource interface

Property Name	Type	Description
reorg	boolean	Enable or disable table reorganization.
deletePhysicalRemove	boolean	Enable/disable physically removing records.
dirtyBitSetByApplication	boolean	Enable/disable application to set dirty bit.
readIncludeMarkedDelete	boolean	Enable/disable read logically deleted records.
isEnabledTableChecksum	boolean	Enable/disable checksums for database values.

Methods:

```
void setReorg(boolean enable)
void setDeletePhysicalRemove(boolean enable)
void setDirtyBitSetByApplication(boolean enable)
```

```
void setReadIncludeMarkedDelete(boolean enable)
void setEnabledTableChecksum(boolean enable)

boolean isReorg()
boolean isDeletePhysicalRemove()
boolean isDirtyBitSetByApplication()
boolean isEnabledTableChecksum()
```

Developing with JNI

This topic explains the basics of developing JNI applications that interact with DB2 Everyplace.

Installing the JNI-based native synchronization provider

The JNI-based synchronization provider requires the following files:

- isync4j.jar
- isyncore.dll
- isyncconf.dll
- imsadb2e.dll
- imsafile.dll
- imsaconfig.dll
- wbxmllib.dll
- isync4j.dll
- isyncxpt.dll
- isyncstat.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 9.1
- DB2 Everyplace Sync Client Libraries version 9.1
- JVM that supports the Java Native Interface

Important: When developing in the WebSphere Studio Device Developer environment, the build task tries to remove unused classes by default. Certain DB2 Everyplace Sync Client classes might be removed because they are not used in the application, even though they are required by the DB2 Everyplace Sync Client engine. If this occurs, a java.lang.AbstractMethodError might be thrown at runtime. To fix this problem, change ″-removeUnused″ to ″-noRemoveUnused″ in the .jxeLinkOptions file. You can also specify the following classes to be included in the .jxeLinkOptions file:

- -includeWholeClass ″com.ibm.mobileservices.isync.*″
- -includeWholeClass ″com.ibm.mobileservices.isync.db2e.jni.*″
- -includeWholeClass ″com.ibm.mobileservices.isync.event.*″
- -includeWholeClass ″com.ibm.mobileservices.isync.sql.*″

You can find sample applications at <DSYPATH>\clients\clientapisample\Java_API\.

Read the following topics for more information about installing the JNI-based synchronization provider on each of the supported operating systems:

- "Installing the JNI-based synchronization provider on Windows" on page 33
- "Installing the JNI-based synchronization provider on Symbian OS devices" on page 33
- "Installing the JNI-based synchronization provider on Windows CE"

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 mobile devices is supported on J9 JVM.

Complete the following steps to install the JNI-based synchronization provider on Windows CE:

1. Compile the ISyncSample program on your workstation.
   a. Type the following command to compile ISyncSample.java with isync4j.jar in the classpath:
      ```
      javac -target 1.1 -classpath isync4j.jar ISyncSample.java
      ```
   b. Edit db2sync_db2e.properties to specify the server URL, username, and password.
2. Run the ISyncSample program.
   a. Verify that the J9 run-time environment is installed on the mobile device. In addition, verify that the DB2 Everyplace and DB2 Everyplace Sync Client libraries are installed.
   b. Copy the ISyncSample.class and db2sync_db2e.properties files to the mobile device.
   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>
      ```

      where <*property file*> is a file containing variables that your program reads.

      For example:

      ```
      j9.exe -bp:\wsdd\classes.zip -cp:\wsdd;\Windows\isync4j.jar ISyncSample
      db2sync_db2e.properties
      ```

      Important: If you get an UnsupportedEncodingException when running the sample with J9, you might also need to include charconv.zip in the classpath from \ive\runtimes\common\ive\lib.

      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"
      "db2sync_db2e.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 that is specified in the property file.

Related tasks

"Installing the JNI-based native synchronization provider" on page 31

Installing the JNI-based synchronization provider on Symbian OS devices

JNI-based implementations have been tested on Symbian PersonalJava JVM.

To install the JNI-based synchronization provider on Symbian OS devices:

1. Edit and compile the ISyncSample program on your workstation.
   a. Edit ISyncSample.java to take db2sync_db2e.properties as a parameter.
   b. Compile ISyncSample.java with isync4j.jar in your classpath by typing the following command: `javac -target 1.1 -classpath isync4j.jar ISyncSample.java`
   c. Edit db2sync_db2e.properties to specify the server URL, user, and password.
2. Run the ISyncSample program.
   a. Make sure the DB2 Everyplace mobile database and DB2 Everyplace Sync Client libraries are installed on the device.
   b. Copy the ISyncSample.class and db2sync_db2e.properties files to the a directory on the device. Alternatively, you can create a ISyncSample.sis with the Symbian aifbuilder tool. Specify Java as the application language, input an application name, UID, and Command Line Text ISyncSample. In the .pkg package file, include the .app, .aif, .class and .properties files and have it install into C:\System\Apps\ISyncSample. (Refer to the Symbian AIF Builder documentation for more information.)
   c. Select the ISyncSample.class file. If you created and installed ISyncSample.sis, start the application by selecting it in the applications menu.
   d. Use the Redirect program to transfer the output from the Java program and then either display this output on your console or write this output to a file.

Installing the JNI-based synchronization provider on Windows

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

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

      <DSYPATH>\Clients\Win32\database\x86

      <DSYPATH>\Clients\Win32\sync

      where <DSYPATH> is the root installation directory of DB2 Everyplace
   b. Change the CLASSPATH variable to include the isync4j.jar file:

      <DSYPATH>\Clients\Win32\Sync\isync4j.jar

      Note: If you get an UnsupportedEncodingException when running the sample with J9, you might also need to include charconv.zip in the classpath from \ive\runtimes\common\ive\lib.
   c. Compile the sample files that are included in the <DSYPATH>\Clients\clientapisample\Java_API directory, where <DSYPATH> is the directory where DB2 Everyplace is installed. For example:

      ```
      javac -target 1.1 ISyncSample.java
      ```
2. Edit the db2sync_db2e.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 db2sync_db2e.properties`

Related tasks

"Installing the JNI-based native synchronization provider" on page 31

Developing DB2 Everyplace applications with the .NET framework

This topic describes the DB2 Everyplace interfaces and providers you can use to develop .NET applications. It also presents some sample code.

Overview of .NET support for building applications on the DB2 Everyplace mobile 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 mobile 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 Windows:** <DSYPATH>\Clients\Win32\database\nmp\IBM.Data.DB2.DB2e.dll, where <DSYPATH> is the root installation directory for DB2 Everyplace
- **For WinCE:** <DSYPATH>\Clients\WinCE\database\nmp\IBM.Data.DB2.DB2e.CF.dll, where <DSYPATH> is the root installation directory for DB2 Everyplace

The API specifications are located in the <DSYPATH>\Clients\Win32\database\nmp\doc\readme.html directory, where <DSYPATH> is the root installation directory for DB2 Everyplace.

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

"Sample DB2 Everyplace .NET Data Provider application code for WinCE and Windows" on page 41

"Simple example application using the ISync.NET API" on page 40

"APIs for developing DB2 Everyplace Sync Server applications" on page 38
The DB2 Everyplace Sync Client provides two APIs, ISyncComponent and ISync.NET, that you can use to build managed applications for the DB2 Everyplace Sync Server.

Related tasks

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

"Using ISyncComponent" on page 40

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

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

Component	Minimum requirement
Microsoft.NET Framework	Microsoft.NET Framework 1.1 Must be installed prior to installing the DB2 Everyplace .NET Data Provider for application development.
Microsoft Visual Studio	Microsoft Visual Studio .NET 2003 or 2005 for developing mobile applications
Microsoft.NET Compact Framework	Microsoft .NET Compact Framework 1.0 or 2.0 for mobile development Must be installed on the device prior to installing the DB2 Everyplace .NET Data Provider for mobile application development.

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

Component	Minimum requirement
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. The DB2 Everyplace native DLL files and the DB2 Everyplace .NET Data Provider must be at the same version level in order to function properly.

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. 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.

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.

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	Description
DB2eConnection	Establishes a connection to a DB2 Everyplace data source and can begin a *Transaction*.

Table 7. DB2 Everyplace .NET Data Provider, core objects (continued)

Object	Description
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.

The DB2 Everyplace .NET Data Provider also contains the classes listed in the following table.

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

Object	Description
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.

1. 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;
   ```

2. 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
   ```

3. 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.

C# example

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

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

Table 12. DB2 Everyplace .NET Provider Connection string keywords

Keyword	Description
@ DATABASE	Database location. For example: C:\data1\
ENCODING	Specifies the database encoding. For example, to connect to a UTF-8 based database, encoding = UTF-8
IO WRITETHROUGH	A boolean value that specifies whether changes to the database are pushed to storage media without delay or handed to the operating system.
LOCK TIMEOUT	A positive integer value that represents the number of seconds to wait before rolling back a transaction when a lock cannot be obtained. The default value is 20.

Table 12. DB2 Everyplace .NET Provider Connection string keywords (continued)

Keyword	Description
PWD	Password
SHARED DB ACCESS	Gets or sets a boolean value that indicates whether the database allows connections to share access. The default value is false.
UID	User ID

Related concepts

"APIs for developing DB2 Everyplace Sync Server applications"
The DB2 Everyplace Sync Client provides two APIs, ISyncComponent and ISync.NET, that you can use to build managed applications for the DB2 Everyplace Sync Server.

"Simple example application using the ISync.NET API" on page 40

"Overview of .NET support for building applications on the DB2 Everyplace mobile database" on page 34

"Sample DB2 Everyplace .NET Data Provider application code for WinCE and Windows" on page 41

Related tasks

"Using the ISync.NET API" on page 39

"Using ISyncComponent" on page 40

APIs for developing DB2 Everyplace Sync Server applications

The DB2 Everyplace Sync Client provides two APIs, ISyncComponent and ISync.NET, that you can use to build managed applications for the DB2 Everyplace Sync Server.

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

Table 13. ISync.NET managed provider location and namespaces. In the table below, <DSYPATH> is the root installation directory for DB2 Everyplace

Available providers	Namespaces	Supported operating systems	Location
Non-Unicode for .NET Framework	IBM.Data.Sync IBM.Data.Sync.DB2e	Windows	<DSYPATH>\clients\win32\sync\nmp\IBM.Data.Sync.DB2e.dll
Unicode for .NET Framework	IBM.Data.Sync IBM.Data.Sync.DB2e	Windows Unicode	<DSYPATH>\clients\win32\sync\nmp\Unicode\IBM.Data.Sync.DB2e.dll
.NET Compact Framework	IBM.Data.Sync IBM.Data.Sync.DB2e.CF	Windows CE	<DSYPATH>\clients\wince\sync\nmp\IBM.Data.Sync.DB2e.CF.dll

ISync.NET sample applications

DB2 Everyplace provides two sample applications, DB2 Sync Console and ISyncSample, that demonstrate the API's functionality. See "DB2 Everyplace sample applications" on page 357 for the location of these samples.

ISync.NET API specification

Specifications for the ISync.NET API are located in the <DSYPATH>\doc*lang*\javadoc\ISyncNetAPI\ directory, where <DSYPATH> is the root installation directory for DB2 Everyplace.

Related concepts

"Sample DB2 Everyplace .NET Data Provider application code for WinCE and Windows" on page 41

"Overview of .NET support for building applications on the DB2 Everyplace mobile database" on page 34

"Simple example application using the ISync.NET API" on page 40

Related tasks

"Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 34

"Using ISyncComponent" on page 40

"Using the ISync.NET API"

Using the ISync.NET API

Software Requirements

- DB2 Everyplace Version 8.1.4 or later
- Microsoft .NET Standard Framework 1.0 (included with Visual Studio 2002) - needed for developing applications on Windows
- 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 DB2 Everyplace Sync Client libraries. Both the provider and the DB2 Everyplace 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.

Strings in .NET are in Unicode. However, the ISync .NET synchronization provider converts strings into bytes according to the format specified by the application. The application-specified format overrides the operating system's default character encoding. See the isync.encoding property in ISyncProvider.CreateSyncService for encodings that you can set.

You can find the API specification for ISync.NET in <DSYPATH>\doc\<*lang*>\javadoc\ISyncNetAPI\ ISync.NET.chm, where <DSYPATH> is the directory where DB2 Everyplace is installed.

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. Here 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 the <DSYPATH>\Clients\clientapisample\NMP directory, where <DSYPATH> is the directory where DB2 Everyplace is installed.
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 a command line, type: **csc /t:exe /r:IBM.Data.Sync.DB2e.dll DB2SyncConsole.cs**.

Related concepts

"APIs for developing DB2 Everyplace Sync Server applications" on page 38
The DB2 Everyplace Sync Client provides two APIs, ISyncComponent and ISync.NET, that you can use to build managed applications for the DB2 Everyplace Sync Server.

"Simple example application using the ISync.NET API" on page 40

"Overview of .NET support for building applications on the DB2 Everyplace mobile database" on page 34

"Sample DB2 Everyplace .NET Data Provider application code for WinCE and Windows" on page 41

Related tasks

"Using ISyncComponent"

"Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 34

Using ISyncComponent

When developing a Visual Studio Windows Application, add the DB2 Everyplace component **IBM.Data.Sync.DB2e.dll** to your **Toolbox**.

Note: The native DB2 Everyplace Sync Client libraries must already be in the user path for this process to complete successfully.

ISyncComponent 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. 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

"APIs for developing DB2 Everyplace Sync Server applications" on page 38
The DB2 Everyplace Sync Client provides two APIs, ISyncComponent and ISync.NET, that you can use to build managed applications for the DB2 Everyplace Sync Server.

"Simple example application using the ISync.NET API"

"Overview of .NET support for building applications on the DB2 Everyplace mobile database" on page 34

"Sample DB2 Everyplace .NET Data Provider application code for WinCE and Windows" on page 41

Related tasks

"Using the ISync.NET API" on page 39

"Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 34

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();
```

You can find complete code examples at <DSYPATH>\Samples\clientapisample\NMP\, where <DSYPATH> is the root installation directory for DB2 Everyplace.

Related concepts

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

"Overview of .NET support for building applications on the DB2 Everyplace mobile database" on page 34

"APIs for developing DB2 Everyplace Sync Server applications" on page 38
The DB2 Everyplace Sync Client provides two APIs, ISyncComponent and ISync.NET, that you can use to build managed applications for the DB2 Everyplace Sync Server.

Related tasks

"Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 34

"Using ISyncComponent" on page 40

"Using the ISync.NET API" on page 39

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

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

- DB2eSample1.cs
- DB2eSample2.cs

Both files are located in the<DSYPATH>\Clients\Win32\database\nmp\samples directory or the <DSYPATH>\Clients\wince\database\nmp\samples directory, where <DSYPATH> is the root installation directory of DB2 Everyplace.

Here 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

"Overview of .NET support for building applications on the DB2 Everyplace mobile database" on page 34

"Simple example application using the ISync.NET API" on page 40

"APIs for developing DB2 Everyplace Sync Server applications" on page 38
The DB2 Everyplace Sync Client provides two APIs, ISyncComponent and ISync.NET, that you can use to build managed applications for the DB2 Everyplace Sync Server.

Related tasks

"Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 34

"Using ISyncComponent" on page 40

"Using the ISync.NET API" on page 39

Character encoding in .NET applications

On Windows operating systems, a DB2 Everyplace database can be either in UTF-8 encoding or local code page encoding. A DB2 Everyplace .NET Provider application uses the connection string to request a connection to an UTF-8 based database. The keyword is `encoding`, and the value is `UTF-8`. For example, a connection to a UTF-8 based database at c:\db\ would be `database=C:\db1\;encoding=UTF-8`.

Related concepts

"Overview of .NET support for building applications on the DB2 Everyplace mobile database" on page 34

"Simple example application using the ISync.NET API" on page 40

"APIs for developing DB2 Everyplace Sync Server applications" on page 38
The DB2 Everyplace Sync Client provides two APIs, ISyncComponent and ISync.NET, that you can use to build managed applications for the DB2 Everyplace Sync Server.

"Sample DB2 Everyplace .NET Data Provider application code for WinCE and Windows" on page 41

Related tasks

"Overview of developing ADO.NET applications using the DB2 Everyplace .NET Data Provider" on page 34

"Using ISyncComponent" on page 40

"Using the ISync.NET API" on page 39

Platform-specific SQL and stored procedures

This topic presents information that will help you develop applications using SQL and stored procedures. It also describes how to use the remote query and stored procedure adapter.

Overview of parameter markers

For SQL statements that need to be issued 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 issued, 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 issued 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"
DB2 Everyplace provides a rich set of standard interfaces including CLI/ODBC, JDBC, and ADO.NET to access data efficiently. The example code snippets in this topic show the use of prepared statement with parameter markers for each data access API.

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 example code snippets in this topic 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 14. Example table schema

Column name	DB2 Everyplace data type	Nullable
c1	INTEGER	false
c2	SMALLINT	true

Table 14. Example table schema (continued)

Column name	DB2 Everyplace data type	Nullable
c3	CHAR(20)	true
c4	VARCHAR(20)	true
c5	DECIMAL(8,2)	true
c6	DATE	true
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 44
For SQL statements that need to be issued 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.

DB2 Everyplace supported parameter markers

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

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.

Table 15. 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

The remote query and stored procedure adapter

DB2 Everyplace includes a remote query and stored procedure adapter. This adapter enables DB2 Everyplace application to use the DB2 Everyplace Sync Server architecture to call a stored procedure located at a remote data source.

The results of the stored procedure are returned directly to the application on the device. The stored procedure call allows a DB2 Everyplace application to directly access data in a remote server without synchronizing.

The remote query and stored procedure adapter enables some unique capabilities of the DB2 Everyplace database engine such as the ability to remotely call a DB2 Version 9.1 stored procedure. This topic details the requirements and techniques for using the remote query and stored procedure adapter in a DB2 Everyplace application.

| As of version 9.1, DB2 Everyplace now supports IPv6. You can use the remote stored procedure by
| entering a valid IPv6 or IPv4 URL. For example, the following formats are valid:

```
| http://[::1]:9081/db2e/com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample
| http://[::1]:9081/db2e/agent?DB=mysample
|
| http://127.0.0.1:9081/db2e/com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample
| http://127.0.0.1:9081/db2e/agent?DB=mysample
```

Supported data types for stored procedures

DB2 Everyplace supports calling stored procedures on a remote DB2 server through the CLI or JDBC interface. The client application uses the CALL statement to run the remote stored procedure. The CALL statement names the procedure to be called and specifies its parameters. The following types are supported: INTEGER, SMALLINT, DECIMAL, CHAR, VARCHAR, DATE, TIME, TIMESTAMP and BLOB.

Related concepts

"The remote query and stored procedure adapter" on page 49
DB2 Everyplace includes a remote query and stored procedure adapter. This adapter enables DB2 Everyplace application to use the DB2 Everyplace Sync Server architecture to call a stored procedure located at a remote data source.

"Restrictions for result sets" on page 55

Related tasks

"Creating the Custom subscription for the sample application" on page 54

"Using the remote query and stored procedure adapter"
The remote query and stored procedure adapter supports the Windows 32-bit (non-Unicode), Windows CE, Symbian, and Palm OS client platforms. The remote query and stored procedure adapter require stored procedures to be registered to DB2.

Related reference

"Sample application code" on page 380
This topic contains the code in Java and C for the sample application. The code requires a connection string for the SQLConnect() function to connect to the remote data source.

"Creating a stored procedure using the sample application" on page 51

"Testing the remote query and stored procedure adapter" on page 54

Using the remote query and stored procedure adapter

The remote query and stored procedure adapter supports the Windows 32-bit (non-Unicode), Windows CE, Symbian, and Palm OS client platforms. The remote query and stored procedure adapter require stored procedures to be registered to DB2.

Restrictions

Multiple connections
DB2 Everyplace supports multiple connections to databases with some limitations. The remote connection uses the local database (the last connection before the remote connection) to store its temporary files. If no local connection exists, then the current directory is used.

Statement handle
Allocate only one statement handle for the remote connection.

On Palm OS
The application stack size might need to be increased.

On Windows 32-bit operating systems
At run time, the IBM DB2 Everyplace Sync Client DLL files must be included in the local directory or system path.

In a DB2 Version 9.1 stored procedure
When a binary large object (BLOB) is used as an input or output parameter, the first four bytes of the BLOB data are reserved to indicate length.

Stored procedures
DB2 Everyplace supports DB2 Version 9.1 stored procedures on only the Windows and UNIX platforms.

Message size

Do not use remote stored procedure calls to transfer large amounts of data. Instead, use DB2 Everyplace synchronization. Each message size must be less than 32 KB.

The following example shows how to create a stored procedure, a subscription to the stored procedure, and a DB2 Everyplace application to use the stored procedure. This sample application allows a mobile user to check an account balance and transfer money between a savings and a checking account by using a DB2 Everyplace remote stored procedure call.

Related reference

"Sample application code" on page 380
This topic contains the code in Java and C for the sample application. The code requires a connection string for the SQLConnect() function to connect to the remote data source.

Creating a stored procedure using the sample application:

This example uses a stored procedure named `MYPROC()`. This procedure takes five parameters: Account Name, Option, Transfer Amount, Saving Balance®, Checking Balance. The following list identifies the purpose of each of the parameters:

1. Account Name: Input parameter to identify the account.
2. Optional: Input parameter to determine what to do. There are three options:
   - Check balance
   - Transfer from saving to checking
   - Transfer from checking to saving
3. Transfer Amount: Input parameter of the amount to transfer between checking and saving
4. Saving Balance: Output parameter returning the balance of saving account
5. Checking Balance: Output parameter returning the balance of checking account

The following code builds the stored procedure:

```
SQL_API_RC SQL_API_FN
myProc(char * szName, int * nCmd, int * nAmount, int * nSaving, int * nChecking)
{
 SQLHENV  henv;
 SQLHDBC  hdbc;
 SQLHSTMT hstmt;
 SQLRETURN rc;
 int   nRetSize;

 SQLCHAR  str1[]="select saving, checking from db2e.myaccount where name = ?";
 SQLCHAR  str2[]="update db2e.myaccount set saving=saving - ?,
          checking=checking + ? where name=?";
 SQLCHAR  str3[]="update db2e.myaccount set saving=saving + ?,
          checking=checking - ? where name=?";

 //*************************************************************
 //* Prepare connection and statement
 //*************************************************************
 rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
 //checkerror
 rc = SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc);
 //checkerror
 rc = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF, SQL_NTS);
 //checkerror
     rc = SQLConnect(hdbc, NULL, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
 //checkerror
 rc = SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &hstmt);
 //checkerror

 //*************************************************************
 //* Update account
```

```
//*************************************************************
if ( *nCmd == 2 || *nCmd == 3 ){
 if ( *nCmd == 2 ){ //Transfer from saving to checking
  rc = SQLPrepare(hstmt, str2, SQL_NTS); //checkerror
 }
 if ( *nCmd == 3 ){ //Transfer from checking to saving
  rc = SQLPrepare(hstmt, str3, SQL_NTS); //checkerror
 }
 rc = SQLBindParameter(hstmt,
   1,
   SQL_PARAM_INPUT,
   SQL_C_LONG,
   SQL_INTEGER,
   0,
   0,
   (SQLPOINTER)nAmount,
   0,
   NULL ); //checkerror
 rc = SQLBindParameter(hstmt,
   2,
   SQL_PARAM_INPUT,
   SQL_C_LONG,
   SQL_INTEGER,
   0,
   0,
   (SQLPOINTER)nAmount,
   0,
   NULL ); //checkerror

 rc = SQLBindParameter(hstmt,
   3,
   SQL_PARAM_INPUT,
   SQL_C_CHAR,
   SQL_CHAR,
   0,
   0,
   (SQLPOINTER)szName,
   0,
   NULL ); //checkerror
 rc = SQLExecute(hstmt); //checkerror
}

//*************************************************************
//* Retrieve account balance
//*************************************************************
rc = SQLPrepare(hstmt, str1, SQL_NTS); //checkerror
rc = SQLBindParameter(hstmt,
  1,
  SQL_PARAM_INPUT,
  SQL_C_CHAR,
  SQL_CHAR,
  0,
  0,
  (SQLPOINTER)szName,
  0,
  NULL );//checkerror
rc = SQLExecute(hstmt);//checkerror
if ( rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO )
{
 while ( (rc = SQLFetch(hstmt) ) == SQL_SUCCESS ){
  rc = SQLGetData( hstmt,
   (SQLSMALLINT)1,
   SQL_C_LONG,
   nSaving,
   sizeof(int) ,
   &nRetSize ) ; //checkerror
  rc = SQLGetData( hstmt,
```

```
    (SQLSMALLINT)2,
    SQL_C_LONG,
    nChecking,
    sizeof(int) ,
    &nRetSize ) ; //checkerror
  }
 }
 //*************************************************************
 //* Clean up
 //*************************************************************
     rc = SQLEndTran( SQL_HANDLE_DBC, hdbc, SQL_COMMIT );
 SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
 SQLDisconnect(hdbc);
 SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
 SQLFreeHandle(SQL_HANDLE_ENV, henv);
 return (0);
}
```

On the Windows platform, after building the stored procedure into a dynamic link library (mydll.dll), copy it to the \SQLLIB\function directory. Next, register the stored procedure.

1. Open a DB2 command window.
2. Connect to the MYSAMPLE database using the following command:

   ```
   DB2 CONNECT TO MYSAMPLE
   ```

3. Register the stored procedure using a script named regscript.scr to configure options. The following code is used for this script:

   ```
   CREATE PROCEDURE db2e.MYPROC (IN szName CHAR(16),
                                 IN nCmd INTEGER,
                                 IN nAmount INTEGER,
                                 OUT nSaving INTEGER,
                                 OUT nChecking INTEGER )
   DYNAMIC RESULT SETS 1
   LANGUAGE C
   PARAMETER STYLE GENERAL
   NO DBINFO
   FENCED
   MODIFIES SQL DATA
   PROGRAM TYPE SUB
   EXTERNAL NAME 'mydll!myProc'@
   ```

   To run the script, enter the following command: `db2 -td@ -vf regscript.scr`

The stored procedure db2e.MYPROC is now configured. Next, create a subscription using the Mobile Devices Administration Center.

Related concepts

"The remote query and stored procedure adapter" on page 49
DB2 Everyplace includes a remote query and stored procedure adapter. This adapter enables DB2 Everyplace application to use the DB2 Everyplace Sync Server architecture to call a stored procedure located at a remote data source.

Related tasks

"Creating the Custom subscription for the sample application" on page 54

"Using the remote query and stored procedure adapter" on page 50
The remote query and stored procedure adapter supports the Windows 32-bit (non-Unicode), Windows CE, Symbian, and Palm OS client platforms. The remote query and stored procedure adapter require stored procedures to be registered to DB2.

Related reference

"Sample application code" on page 380
This topic contains the code in Java and C for the sample application. The code requires a connection string for the SQLConnect() function to connect to the remote data source.

"Testing the remote query and stored procedure adapter"

Creating the Custom subscription for the sample application:

To create a custom subscription for the sample application:

Perform the following steps in the Mobile Devices Administration Center.

1. Open the Mobile Devices Administration Center.
2. Right-click on the **Subscriptions** folder in the left pane of the Mobile Devices Administration Center window and select **Create → Custom Subscription**. This opens the Create Custom Subscription window.
3. Type `subex` in the **Name** field.
4. Click the **Launch Customizer** button. This opens the Source database window.
   a. In the **User ID** field, type the DB2 user ID that has access privileges to the target database.
   b. Type the password for the user ID in the **Password** and **Verify password** fields.
   c. In the **Other** field, type:
      ```
      dbname=mysample;procname=db2e.MYPROC
      ```
5. Click **OK** to close the Source database window. Click **OK** to close the Create Custom Subscription window.

After you create the Custom subscription, create a user, group, and subscription set.

Related concepts

"The remote query and stored procedure adapter" on page 49
DB2 Everyplace includes a remote query and stored procedure adapter. This adapter enables DB2 Everyplace application to use the DB2 Everyplace Sync Server architecture to call a stored procedure located at a remote data source.

Related tasks

"Using the remote query and stored procedure adapter" on page 50
The remote query and stored procedure adapter supports the Windows 32-bit (non-Unicode), Windows CE, Symbian, and Palm OS client platforms. The remote query and stored procedure adapter require stored procedures to be registered to DB2.

Related reference

"Sample application code" on page 380
This topic contains the code in Java and C for the sample application. The code requires a connection string for the SQLConnect() function to connect to the remote data source.

"Creating a stored procedure using the sample application" on page 51

"Testing the remote query and stored procedure adapter"

Testing the remote query and stored procedure adapter:

This sample uses a DB2 Everyplace Windows console application to test the remote query and stored procedure adapter. The sample application is called myclient.exe. It uses the following three parameters:

1. Account Name: Identify the account to access.
2. Option: Identify the action to perform. The options are:
   1: Check balance.
   2: Transfer from savings to checking.
   3: Transfer from checking to savings.
3. Amount: Amount to transfer between checking and saving.

For example, to transfer $1000 from savings to checking on the Michael account, enter the following command: `myclient.exe Michael 2 1000`.

Assuming Michael has $5000 in each account before the transfer, the following response is returned:

```
Saving = 4000
Checking = 6000
```

Related concepts

"The remote query and stored procedure adapter" on page 49
DB2 Everyplace includes a remote query and stored procedure adapter. This adapter enables DB2 Everyplace application to use the DB2 Everyplace Sync Server architecture to call a stored procedure located at a remote data source.

Related tasks

"Creating the Custom subscription for the sample application" on page 54

"Using the remote query and stored procedure adapter" on page 50
The remote query and stored procedure adapter supports the Windows 32-bit (non-Unicode), Windows CE, Symbian, and Palm OS client platforms. The remote query and stored procedure adapter require stored procedures to be registered to DB2.

Related reference

"Sample application code" on page 380
This topic contains the code in Java and C for the sample application. The code requires a connection string for the SQLConnect() function to connect to the remote data source.

"Creating a stored procedure using the sample application" on page 51

Restrictions for result sets:

Result sets are a useful way to retrieve data from a stored procedure. If a client application runs a stored procedure that generates a result set, it can then use the regular CLI functions or JDBC methods such as SQLFetch() and SQLGetData() to retrieve the data. DB2 Everyplace does not support multiple result sets.

Related concepts

"The remote query and stored procedure adapter" on page 49
DB2 Everyplace includes a remote query and stored procedure adapter. This adapter enables DB2 Everyplace application to use the DB2 Everyplace Sync Server architecture to call a stored procedure located at a remote data source.

"Supported data types for stored procedures" on page 50

Related tasks

"Creating the Custom subscription for the sample application" on page 54

"Using the remote query and stored procedure adapter" on page 50
The remote query and stored procedure adapter supports the Windows 32-bit (non-Unicode), Windows CE, Symbian, and Palm OS client platforms. The remote query and stored procedure adapter require stored procedures to be registered to DB2.

Related reference

"Sample application code" on page 380
This topic contains the code in Java and C for the sample application. The code requires a connection string for the SQLConnect() function to connect to the remote data source.

"Creating a stored procedure using the sample application" on page 51

"Testing the remote query and stored procedure adapter" on page 54

Developing VisualBasic applications

This topic presents information that will help you develop applications using VisualBasic.

Developing DB2 Everyplace Visual Basic applications

To develop a DB2 Everyplace application in Visual Basic, 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.

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.

The basic steps to developing 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"
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 Windows operating systems.

Related reference

"Visual Basic Interface supported operating systems"
This topic presents the operating systems that support the Visual Basic Interface.

"DB2 CLI function summary" on page 163

Visual Basic Interface supported operating systems

This topic presents the operating systems that support the Visual Basic Interface.

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

- Windows CE® for Pocket PC
- Windows Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, and Windows 2003)

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 Windows 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 Windows 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 "Visual Basic example: db2evb.bas" on page 57 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 (Windows CE), the files are located in <DSYPATH>\clients**wince**\database\visualbasic. For 32-bit Windows operating systems, the files are located in <DSYPATH>\clients**win32**\database\visualbasic.

Note: <DSYPATH> is the root installation directory of DB2 Everyplace.

The sample Visual Basic application includes the following files:

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.vbp (For WinCE, DB2eSample.ebp)
Application project file.

DB2eSample.vbw
Application project file.

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.

Visual Basic example: db2evb.bas

The major steps used in the sample application (db2evb.bas) are:

- Connect to the DB2 Everyplace mobile database.
  - Step 1: Allocate an environment handle.
  - Step 2: Allocate a DB2 Everyplace mobile database handle.
  - Step 3: Connect to the DB2 Everyplace mobile 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.

Note: Make sure that the application closes the connection to the DB2 Everyplace mobile 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 statement 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 56
To develop a DB2 Everyplace application in Visual Basic, 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.

Compiling and testing the sample Visual Basic program

The testing procedure for a DB2 Everyplace Visual Basic application depends on the type of mobile device. Use these instructions to properly test your application for your target platform.

The Visual Basic project directory, which contains the sample application, is located under the directory where you installed DB2 Everyplace. For Pocket PC (Windows CE), the files are located in \db2everyplace\clients**wince**\database\visualbasic. For Windows 32-bit operating systems, the files are located in \db2everyplace\clients**win32**\database\visualbasic.

1. Open the Visual Basic project file DB2eSample.vbp (for Windows CE, DB2eSample.ebp).).
2. Build the sample program.
   - For Windows: Select **File** → **DB2eSample.exe**. DB2eSample.exe will be built.
   - For Windows CE: Select **File** → **DB2eSample.vb**. DB2eSample.vb will be built.
3. Copy the following files:
   - For Windows: Copy DB2e.dll (for your Windows operating system) into your current project directory or the path of DB2e.dll in the environment variable PATH.
   - For Windows CE: 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 Windows or DB2Sample.vb for WinCE.

Related concepts

"Overview of the sample Visual Basic application" on page 56
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 Windows operating systems.

Related reference

"Visual Basic Interface supported operating systems" on page 56
This topic presents the operating systems that support the Visual Basic Interface.

"DB2 CLI function summary" on page 163

Advanced Development with DB2 Everyplace

This topic explains the development of applications that use the advanced features of DB2 Everyplace.

Overview of the DB2 Everyplace mobile database tables

A DB2 Everyplace mobile database comprises several system catalog tables and a number of user-defined tables.

Each table is stored in two files: one for the data itself, and one for indexes. All indexes are kept in the same index file. Unlike DB2 Version 9.1, DB2 Everyplace mobile databases do not have names and cannot be cataloged or uncataloged. Therefore, the database name is ignored.

A DB2 Everyplace mobile database is a set of files that can be copied or moved to another location. A DB2 Everyplace mobile database must contain the following system catalog tables:

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

System catalog tables contain metadata about user-defined tables. For example, if you remove files for a user-defined table without deleting a corresponding entry in the catalog tables, you will cause an inconsistency.

To access catalog tables in a query, you must use delimited identifiers. For example, the following query returns 1 if the table T exists:

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

Related reference

"DB2 Everyplace System Catalog base tables" on page 66
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.

Setting the checksum attribute to detect file changes

DB2 Everyplace supports a connection attribute called SQL_TABLE_CHECKSUM that allows an application to detect if vendor software has altered the contents of a database or if the contents of the database have been corrupted.

When the SQL_TABLE_CHECKSUM_ATTR attribute is set to ON, DB2 Everyplace will store files with checksums enabled. This connection property is used with the SQLSetConnectAttr() and SQLGetConnectAttr() functions. To enable this feature, follow this step:

Before you create a database, call SQLSetConnectAttr() and set the SQL_ATTR_TABLE_CHECKSUM to SQL_TABLE_CHECKSUM_ON, like in the following CLI example:
```
rc = SQLSetConnectAttr(hdbc,
SQL_ATTR_TABLE_CHECKSUM, (SQLPOINTER) SQL_TABLE_CHECKSUM_ON, 0);
```

You cannot change this attribute in existing databases. After you connect to a database, applications can use the SQLGetConnectAttr() function to find out if the checksum property is enabled.

SQLGetConnectAttr

SQLSetConnectAttr

Handling naming conflicts between tables

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 DB2eCLP:
```
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

"Connection serialization"
A DB2 Everyplace data source accepts connections from one process at a time. When more than one process tries to connect to the same data source at the same time, the requests are put into a queue through a mechanism called *connection serialization*.

"Overview of the DB2 Everyplace mobile database tables" on page 61
A DB2 Everyplace mobile database comprises several system catalog tables and a number of user-defined tables.

Related tasks

"Connecting to the DB2 Everyplace mobile database" on page 15
Applications typically create and access tables in a specific location, for example, the C:\TEMP directory. You can use the CLI call to specify a location when connecting to a DB2 Everyplace mobile database.

Connecting to the DB2 Everyplace mobile database

Applications typically create and access tables in a specific location, for example, the C:\TEMP directory. You can use the CLI call to specify a location when connecting to a DB2 Everyplace mobile database.

In the following example, *path* represents the path to the DB2 Everyplace mobile database.

```
rc = SQLConnect(hdbc, path, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

The path can include (but does not require) the database name. Thus, both of the following examples are correct, assuming a DB2 Everyplace mobile database exists in C:\TEMP.

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

Connecting to Sony Memory Stick extended memory under Palm OS requires a special path specification, as the following example shows.

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

Using DB2eCLP, you can connect to a specific location using the ″CONNECT TO″ command. For example, the following command connects to the DB2 Everyplace mobile database in C:\TEMP\ on a computer running Windows:

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

CAUTION:
For Windows and Windows CE platforms, it is unsafe to call DB2 Everyplace from within DllMain. This is especially important for version 8.2 because DB2 Everyplace added a background thread for performance. For example, an application that calls SQLConnect() within DllMain will experience a deadlock or other unexpected results. For more information about this issue, consult the Microsoft documentation.

Related concepts

"Overview of the DB2 Everyplace mobile database tables" on page 61
A DB2 Everyplace mobile database comprises several system catalog tables and a number of user-defined tables.

Related tasks

"Handling naming conflicts between tables" on page 62
This topic shows some examples of ways that you can handle file naming conflicts for user-defined tables.

Connection serialization

A DB2 Everyplace data source accepts connections from one process at a time. When more than one process tries to connect to the same data source at the same time, the requests are put into a queue through a mechanism called *connection serialization*.

Connection serialization requires developers to decide how long an application should wait to get a connection. This interval, called the *timeout period*, can be set using the SQL_ATTR_LOGIN_TIMEOUT attribute of the SQLSetConnectAttr() function. The following CLI and JDBC examples set the connection timeout period to 10. If the application cannot connect to the database within 10 seconds, it returns an error code.

The timeout period

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:

- The default timeout period is 0 seconds.
- The default LOCK_TIMEOUT period is 20 seconds.
- @ @ A multi-thread application can connect to a database using one thread and disconnect from the database using a different thread. This does not apply to Symbian OS and Palm OS.
- Connection serialization might not work with a database on a network drive.
- In a JDBC program, the timeout value is ignored and set to zero if it is passed in a property to the DriverManager.getConnection() method.
- DB2 Everyplace allows concurrent database access *within the same process* (or address space). For example, the connect method in the java.sql.Driver interface supports ENABLE_SHARED_DATABASE_ACCESS, a Boolean property that you can set to true to enable concurrent access. DB2 Everyplace supports similar properties and methods for other languages, such as SHARED_DB_ACCESS for ADO.NET applications, and SQL_DB_SHARED for CLI applications.

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. In DB2 Everyplace, 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 fetches rows from a table T
- Handle #2 deletes rows from the same table

Each handle might 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 can 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 topic, 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 DB2eCLP. 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 rolling back an entire transaction.

Upon a partial transaction rollback such as a ROLLBACK TO SAVEPOINT statement, an open cursor remains open if the UPON ROLLBACK RETAIN CURSORS clause was specified for the given savepoint. Otherwise, the cursor is closed.

Object dependency

Preparing an SQL statement via a statement handle H can 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 might be different or an error might be returned.

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 16. System catalog base tables

Description	Catalog base table
tables	"DB2eSYSTABLES"
columns	"DB2eSYSCOLUMNS"
referential constraints	"DB2eSYSRELS" on page 67
users	"DB2eSYSUSERS" on page 67

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 17. DB2eSYSTABLES system catalog base table

Column name	Data type	Nullable	Description
TNAME	VARCHAR (129)		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 (32767)	Yes	Check constraint (internal use only)
IDXINFO	BLOB (4096)	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 18. DB2eSYSCOLUMNS system catalog base table

Column name	Data type	Nullable	Description
CNAME	VARCHAR (129)		Column name
TNAME	VARCHAR (129)		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)
KEYSEQ	INTEGER (4)		Ordinal position of the column in the primary key
SCALE	INTEGER (4)		Scale for decimal column
@ DEF	VARCHAR (32767)	Yes	Default value (internal use)

DB2eSYSRELS

This system catalog base table contains a row for each referential constraint.

Table 19. 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 (129)		Parent table name
PKCOLUMN_NAME	VARCHAR (129)		Parent table primary key column
FKTABLE_NAME	VARCHAR (129)		Child table name
FKCOLUMN_NAME	VARCHAR (129)		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 20. DB2eSYSUSERS system catalog base table

Column name	Data type	Nullable	Description
USERNAME	VARCHAR (129)		Part of primary key and is case sensitive. The name of the user associated with this row.
DATABASENAME	VARCHAR (129)		For future use. Empty string is stored. Part of primary key.
TABLENAME	VARCHAR (129)		For future use. Empty string is stored. Part of primary key.
ENCMETHOD	VARCHAR (129)		For future use. Empty string is stored. Part of primary key.
PRIVILEGES	VARCHAR (129)	Yes	Defines user privileges. Currently, only the value 'E', indicating encryption, is allowed.

Table 20. DB2eSYSUSERS system catalog base table (continued)

Column name	Data type	Nullable	Description
ENCKEYDATA	BLOB (280)	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 (280)	Yes	Verifies that the record is authentic and the user was added by an authenticated user.
GRANTOR	VARCHAR (129)	Yes	The user name that registered the user name in column 1.
INTERNALDATA	BLOB (255)	Yes	(Internal future use)

Tuning database applications

Topics in this section describe techniques for improving the performance of database applications.

A DB2 Everyplace data source accepts connections from one process at a time. When more than one process tries to connect to the same data source at the same time, the requests are put into a queue. However, DB2 Everyplace allows multiple database connections *within the same process* (or address space). For example, the connect method in the java.sql.Driver interface supports ENABLE_SHARED_DATABASE_ACCESS, a Boolean property that you can set to true to enable concurrent access. DB2 Everyplace supports similar properties and methods for other languages, such as SHARED_DB_ACCESS for ADO.NET applications, and SQL_DB_SHARED for CLI applications.

Before developing applications that use multiple connections, you should understand the following concepts.

Concurrency issues

Concurrency refers to the sharing of resources by multiple interactive users or application programs at the same time. DB2 Everyplace supports concurrent transactions, enabling an application to establish several distinct connections to the same database.

When developing such an application, take care to prevent undesirable effects, such as:

- **Lost updates**. Two applications, A and B, might both read the same row from the database and both calculate new values for one of its columns based on the data these applications read. If A updates the row with its new value and B then also updates the row, the update performed by A is lost.
- **Access to uncommitted data**. Application A might update a value in the database, and application B might read that value before it was committed. Then, if the value of A is not later committed, but backed out, the calculations performed by B are based on uncommitted (and presumably invalid) data.
- **Non-repeatable reads**. Some applications involve the following sequence of events: application A reads a row from the database, then goes on to process other SQL requests. Meanwhile, application B either modifies or deletes the row and commits the change. Later, if application A attempts to read the original row again, it receives the modified row or discovers that the original row has been deleted.
- **Phantom reads**. The phantom read phenomenon occurs when:
  1. Your application executes a query.
  2. Another application inserts or updates data that satisfies your application's query criteria.
  3. Your application repeats the query from step 1 (within the same unit of work), but the result set is different because it includes additional ″phantom″ rows inserted or updated by the other application.

You can prevent such behavior in a DB2 Everyplace application by managing locks and isolation levels. If your application does not require multiple database connections, you can avoid concurrency issues altogether by disabling shared access. For example, the connect method in the java.sql.Driver interface supports ENABLE_SHARED_DATABASE_ACCESS, a Boolean property that you can set to false to disable concurrent access. DB2 Everyplace supports similar properties and methods for other languages, such as SHARED_DB_ACCESS for ADO.NET applications, and SQL_DB_SHARED for CLI applications. Consult the reference for more information.

Table locking

A *lock* associates a database manager resource with an application to control how other applications can access the same resource. DB2 Everyplace supports *table locking*. That is, you either lock the entire table or you don't lock it at all. You cannot lock specific rows in a table.

DB2 Everyplace supports two types of table locks:

- Exclusive locks, used on DDL and DML statements.
- Shared locks, used on SELECT statements.

The following table shows how these lock types can be combined when multiple users or transactions access a table.

Table 21. Lock compatibility

	SHARED	**EXCLUSIVE**
SHARED	Compatible	Incompatible
EXCLUSIVE	Incompatible	Incompatible

An application can lock a table by calling the SQL statement LOCK TABLE. For example the following code obtains an exclusive lock on the table EMP.

```
LOCK TABLE EMP IN EXCLUSIVE MODE
```

Table locking is appropriate for read-only transactions and single-user access. When two or more transactions are updating the same table, table locking can lead to deadlock. For example, consider this scenario:

1. Two transactions, A and B, obtain a shared lock on table T.
2. Later, both transactions need to write to table T, which requires an exclusive lock.
3. Neither transaction can obtain an exclusive lock, because the other transaction has a shared lock, and shared locks and exclusive locks are incompatible.
4. Each transaction waits for the other to release the shared lock, resulting in a deadlock.

DB2 Everyplace provides a timeout mechanism that applications can use to resolve deadlocks. If an application cannot obtain a lock within a specified amount of time, the database engine rolls back the transaction and returns SQLSTATE 40001. The default lock timeout is 20 seconds.

Guidelines for locking

This topic presents the guidelines you should consider when tuning locking for concurrency and data integrity.

- DB2 Everyplace locks entire tables.

  You either lock an entire table or you don't lock it at all. You cannot lock specific rows in a table.
- Create small units of work with frequent COMMIT statements to promote concurrent access of data by many users.

  Include COMMIT statements when the data you have changed is consistent. When a COMMIT is issued, all locks are released except those related to open cursors (in DB2 Everyplace, cursors are held across a COMMIT). After a COMMIT, all remaining locks are SHARED locks. All locks are released upon ROLLBACK.
- Specify an appropriate isolation level.

  Shared locks are acquired by serializable, repeatable-read and read-committed isolation levels, even in read-only applications. To release these locks, close cursors that are not in use.

The database manager ensures that your application does not retrieve uncommitted data (rows that have been updated by other applications but are not yet committed) unless you are using the uncommitted read isolation level.

- Use the LOCK TABLE statement appropriately.

  Only the table specified in the LOCK TABLE statement is locked. Parent and dependent tables of the specified table are not locked. You must determine whether locking other tables is necessary to achieve the desired result in terms of concurrency and performance. The lock is not released until the unit of work is committed or rolled back.

  LOCK TABLE IN SHARE MODE

  You want to access data that is consistent in time; that is, data current for a table at a specific point in time. If the table experiences frequent activity, the only way to ensure that the entire table remains stable is to lock it. For example, your application wants to take a snapshot of a table. However, during the time your application needs to process some rows of a table, other applications are updating rows you have not yet processed. This is allowed with repeatable read, but this action is not what you want.

  As an alternative, your application can issue the LOCK TABLE IN SHARE MODE statement: no rows can be changed, regardless of whether you have retrieved them or not. You can then retrieve as many rows as you need, knowing that the rows you have retrieved have not been changed just before you retrieved them.

  With LOCK TABLE IN SHARE MODE, other users can retrieve data from the table, but they cannot update, delete, or insert rows into the table.

  LOCK TABLE IN EXCLUSIVE MODE

  With LOCK TABLE IN EXCLUSIVE MODE, all other users are locked out; no other applications can access the table unless they are uncommitted read applications.

- Close cursors to release the locks that they hold.

  In DB2 Everyplace, cursors are held across commits by default, and are closed implicitly upon executing the next statement. If an application no longer needs the cursor at commit time, it should close the cursor explicitly before committing the transaction to release its shared locks. Also, the isolation level of a connection can only be set if there are no open cursors and auto commit is on; otherwise SQLSTATE HY011 is returned.

Isolation levels

An *isolation level* specifies how much one transaction is isolated from other transactions in a multiple-connection environment. DB2 Everyplace supports the following ANSI SQL isolation levels.

Note: The levels are listed with their DB2 Version 9.1 equivalents in decreasing order of performance impact, but in increasing order of care required when accessing and updating data (for example, the potential for deadlock situations varies with the isolation level). Details about each level follow the table.

Table 22. Isolation levels

ANSI SQL Isolation Level	DB2 Version 9.1 equivalent
SERIALIZABLE	Repeatable read (RR)
REPEATABLE READ	Read stability (RS)
READ COMMITTED (default)	Cursor stability (CS)
READ UNCOMMITTED	Uncommitted read (UR)

SERIALIZABLE (DB2 Version 9.1: Repeatable Read)

Locks the table within a unit of work. An application can retrieve and operate on rows in the table as many times as needed. However, the entire table is locked, not just the rows that are retrieved. Until the unit of work completes, no other application can update, delete, or insert a row that would affect the table.

SERIALIZABLE applications cannot see uncommitted changes made by other applications. Therefore, a SELECT statement issued repeatedly within the unit of work gives the same result each time. Lost updates, access to uncommitted data, and phantom rows are not possible.

REPEATABLE READ (DB2 Version 9.1: Read Stability)

Because DB2 Everyplace locks entire tables (not specific rows), REPEATABLE READ behaves exactly like SERIALIZABLE.

READ COMMITTED (DB2 Version 9.1: Cursor Stability)

The entire table is locked. Shared locks are released when the associated cursors are closed (isolation levels higher than READ COMMITTED hold shared locks until the end of a transaction). Exclusive locks are held until the end of the transaction.

No other application can perform any DML operation on a table while an open cursor is accessing it. READ COMMITTED applications cannot see uncommitted changes of other applications.

Both nonrepeatable reads and phantom reads are possible. READ COMMITTED is the default isolation level, allowing maximum concurrency while seeing only committed rows from other applications.

READ UNCOMMITTED (DB2 Version 9.1: Uncommitted Read)

An application can access *some* uncommitted changes of other transactions: tables and indexes that are being created or dropped by other transactions are not available while the transaction is processing. Any other changes can be read before they are committed or rolled back.

At this level, the application does not lock other applications out of the table it is reading.

The following table summarizes isolation levels in terms of their undesirable effects.

Table 23. Summary of isolation levels

Isolation Level	Access to uncommitted data	Nonrepeatable reads	Phantom read phenomenon
SERIALIZABLE	Not possible	Not possible	Not possible
REPEATABLE READ	Not possible	Possible	Possible
READ COMMITTED	Not possible	Possible	Possible
READ UNCOMMITTED	Possible	Possible	Possible

The following table can help you choose an initial isolation level for your applications. Consider this table a starting point, and refer to the previous discussions of the various levels for factors that might make another isolation level more appropriate.

Table 24. Guidelines for choosing an isolation level

Application Type	High data stability required	High data stability not required
Read-write transactions	REPEATABLE READ	READ COMMITTED
Read-only transactions	SERIALIZABLE or REPEATABLE READ	READ UNCOMMITTED

Other points to consider:

- INSERT, UPDATE, and DELETE statements always behave the same regardless of the isolation level. Only the behavior of SELECT statements varies.
- You can set the isolation level only at the beginning of a transaction, thus it remains in effect for the duration of the unit of work.

Connection serialization

A DB2 Everyplace data source accepts connections from one process at a time. When more than one process tries to connect to the same data source at the same time, the requests are put into a queue through a mechanism called *connection serialization*.

Connection serialization requires developers to decide how long an application should wait to get a connection. This interval, called the *timeout period*, can be set using the SQL_ATTR_LOGIN_TIMEOUT attribute of the SQLSetConnectAttr() function. The following CLI and JDBC examples set the connection timeout period to 10. If the application cannot connect to the database within 10 seconds, it returns an error code.

The timeout period

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:

- The default timeout period is 0 seconds.
- The default LOCK_TIMEOUT period is 20 seconds.
- @ @ A multi-thread application can connect to a database using one thread and disconnect from the database using a different thread. This does not apply to Symbian OS and Palm OS.
- Connection serialization might not work with a database on a network drive.
- In a JDBC program, the timeout value is ignored and set to zero if it is passed in a property to the DriverManager.getConnection() method.
- DB2 Everyplace allows concurrent database access *within the same process* (or address space). For example, the connect method in the java.sql.Driver interface supports ENABLE_SHARED_DATABASE_ACCESS, a Boolean property that you can set to true to enable concurrent access. DB2 Everyplace supports similar properties and methods for other languages, such as SHARED_DB_ACCESS for ADO.NET applications, and SQL_DB_SHARED for CLI applications.

Security in DB2 Everyplace

Topics in this section describe techniques that you can use to make applications more secure.

Encrypting local data

Encryption in DB2 Everyplace is designed for securing data on a mobile or embedded device. This topic provides a quick overview of the significance of local data encryption and a series of pertinent tasks to help get you started. It also describes how encryption is enabled for each platform and lists the libraries that are needed in addition to those required by the DB2 Everyplace mobile database.

Libraries needed:

For Windows:

- plug-in library: CryptoPlugin.dll (provided by DB2 Everyplace)
- encryption library: Crypt32.dll (128-bit cipher strength encryption package, comes with Internet Explorer 5.5 or above). Go to http://www.microsoft.com/windows/ie/downloads/default.mspx to download Internet Explorer.

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/windowsmobile/downloads/highencryption.mspx to download the encryption pack. This pack is part of Pocket PC 2003, but you must install it on Pocket PC 2002. If the CryptoPlugin.dll library is present, but the encryption pack is not installed, applications cannot connect to any database (for example DB2eCLP cannot start). If an application requires encryption, install the Microsoft High Encryption Pack for Pocket PC. If encryption is not needed, delete the CryptoPlugin.dll from the Windows directory on the Pocket PC device.

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)

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 information about using data encryption, see the following topics.

Establishing a connection to the DB2 Everyplace mobile database

Any interaction with the DB2 Everyplace mobile 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.

This task is part of the main task of encrypting local data. After completing these steps, return to "Encrypting local data" on page 75.

The following is an example of establishing the connection to a database. It uses the CLI function:

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

where *"C:\temp\"* is the directory of the DB2 Everyplace mobile database that the application is connected 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
A DB2 Everyplace data source accepts connections from one process at a time. When more than one process tries to connect to the same data source at the same time, the requests are put into a queue through a mechanism called *connection serialization*.

Granting a user encryption privileges

Before creating the first encrypted table, the application must grant a user encryption privileges.

This task is part of the main task of encrypting local data. After completing these steps, return to "Encrypting local data" on page 75.

For example, the application can issue the following SQL statement:

```
rc = SQLExecDirect(..., "GRANT ENCRYPT ON DATABASE 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 mobile database to access encrypted data. This is because a different database 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 78
This topic contains 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.

Related tasks

"Managing encryption privileges"
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.

"Creating an encrypted table"
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.

Creating an encrypted table

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.

This task is part of the main task of encrypting local data. After completing these steps, return to "Encrypting local data" on page 75.

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)
```

For 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.

Related concepts

"Encryption using the DB2eCLP" on page 78
This topic contains 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.

Related tasks

"Managing encryption privileges"
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.

"Granting a user encryption privileges" on page 76
Before creating the first encrypted table, the application must grant a user encryption privileges.

Managing encryption privileges

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.

This task is part of the main task of encrypting local data. After you have complete these steps, return to "Encrypting local data" on page 75.

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"
This topic contains 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.

Related tasks

"Granting a user encryption privileges" on page 76
Before creating the first encrypted table, the application must grant a user encryption privileges.

"Creating an encrypted table" on page 77
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.

Encryption using the DB2eCLP

This topic contains 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;

-- because 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 mobile 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 the database 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 the database 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 the database 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

"Encrypting local data" on page 75
Encryption in DB2 Everyplace is designed for securing data on a mobile or embedded device. This topic provides a quick overview of the significance of local data encryption and a series of pertinent tasks to help get you started. It also describes how encryption is enabled for each platform and lists the libraries that are needed in addition to those required by the DB2 Everyplace mobile database.

"Granting a user encryption privileges" on page 76
Before creating the first encrypted table, the application must grant a user encryption privileges.

"Creating an encrypted table" on page 77
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.

"Managing encryption privileges" on page 77
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.

Encrypted DB2 Everyplace Sync Server passwords

This topics introduces how different types of passwords are encrypted and where they are saved.

DB2 Everyplace Sync Server passwords appear in properties files and XML scripts

Two properties files contain passwords used by the DB2 Everyplace Sync Server:

- DSYIdflt.properties contains the password for the control database, DSYCTLDB.
- DSYLDAP.properties contains the password for the SOAP router HTTP connection.

DB2 Everyplace Sync Server passwords can also appear in XML scripts, and they can be specified using the Mobile Devices Administration Center.

To prevent accidental or unauthorized access to these resources, passwords can be encrypted.

DSYEncrypt utility encrypts passwords

DB2 Everyplace provides a command-line utility that encrypts passwords. Given a password, the utility returns an encrypted version of that password. The utility, named dsyencrypt.bat, is installed by default in the<DSYPATH>\Server\bin directory, where <DSYPATH> is the directory where DB2 Everyplace is installed.

Here's an example of how to use the tool to encrypt the password `db2admin`.

1. From the command line, enter `dsyencrypt db2admin`. You will see a message similar to the following one:

```
Encrypted form of your input text is: nw4SCU6x1ok=
If this is an encrypted password you want to place in a properties file,
then the value you should place in the properties file is: {DSY}nw4SCU6x1ok=
```

2. Use the generated value (prefixed by `{DSY}` in a properties file instead of the plain text password. For example, in DSYIdflt.properties, instead of using this:

   `SSDB2.Password=db2admin` use this:

   `SSDB2.Password={DSY}nw4SCU6x1ok=`

   Similarly, in DSYLdap.properties, instead of using this:

   `WEBSERVICE_HTTP_PASSWORD=db2admin`

   use this:

   `WEBSERVICE_HTTP_PASSWORD={DSY}nw4SCU6x1ok=`

DB2 Everyplace Sync Server passwords in XML scripts are encrypted in the database

When creating XML scripts manually, you can specify passwords for the DB2 Everyplace Sync Server in plain text, for example, `<Password>db2admin<Password>`. Such plain text passwords are automatically encrypted when stored in the control database.

Note: DB2 Everyplace Sync Client passwords stored in DSYCTLDB are *not* encrypted. DB2 Everyplace Sync Client passwords are stored in plain text.

In XML scripts generated by the XML Scripting Tool, master and mirror database passwords are encrypted automatically, with output similar to the following examples:

```
<AddJdbcMaster>
  <Databasej>dbc:db2:VNURSE</Database>
  <Driver>COM.ibm.db2.jdbc.app.DB2Driver<Driver>
  <Userid>db2admin<Userid>
  <Password Encryption="DSY">QmlUOzeUngArzGq1xptlhA==</Password>
<AddJdbcMaster>
<AddJdbcMirror>
  <Databasej>dbc:db2:M_VN2</Database>
  <Driver>COM.ibm.db2.jdbc.app.DB2Driver<Driver>
  <Userid>db2admin</Userid>
  <Password Encryption="DSY">bbRtum49DRuMMRxwD5eSlAA==</Password>
  <SyncWindow>5000</SyncWindow>
<AddJdbcMirror>
```

Mobile Devices Administration Center encrypts passwords for master and mirror databases

When you use the Mobile Devices Administration Center to specify passwords for master and mirror databases, they are saved in encrypted form. If you forget your passwords, you can't retrieve them by looking into these tables.

Note: Only new records and updated records are encrypted. Existing data in DSYCTLDB, specifically DSY.REPL_MIRROR.PASSWORD, DSY.REPL_MASTER.PASSWORD, DSY.JDBC_MIRROR.PASSWORD, and DSY.JDBC_MIRROR.PASSWORD, is not encrypted when migrated.

DB2 Everyplace support and troubleshooting

This topic presents tools, utilities, and techniques that you can use to identify and solve DB2 Everyplace problems.

Diagnostic data for the DB2 Everyplace mobile database

DB2 Everyplace provides the following methods to log the activity of your applications. This data can help you troubleshoot problems throughout the development and testing process.

Trace files
Tracing allows you to view detailed information about each transaction between your application and the DB2 Everyplace database engine. To enable tracing, install the development libraries on the mobile device.

Log files
When an application encounters a severe system error (SQLState 58005), DB2 Everyplace logs the error in a log file.

Dump files
When an application encounters a severe system error (SQLState 58005), DB2 Everyplace captures the system state in a dump file.

The following table lists the names of each type of diagnostic file. DB2 Everyplace creates these files in the same directory as the mobile database.

Table 25. Tracing and diagnostics file names

Type of	Long file name	Short file name (8.3 format)
Trace file	DSY_DB2eTRACE	_trc.DBs
Log file	db2ediag.log	_diag.DBs
Dump files	DB2e_c*xxxxx*	_c*xxxxx*.DBs

where *xxxxx* is the process identifier (PID) number of the affected process.

Diagnostic data for the DB2 Everyplace Sync Server

Use the following files to troubleshoot problems with the DB2 Everyplace Sync Server or the Mobile Devices Administration Center.

Table 26. Log and trace files for the DB2 Everyplace Sync Server and Mobile Devices Administration Center

File type	Path
DB2 Everyplace Sync Server log file	<DSYPATH>\Server\logs\IBMDB2eServer\ syncadapterinit.log
DB2 Everyplace Sync Server trace file	<DSYPATH>\Server\logs\IBMDB2eServer\dsy*nnnn*.trace
Mobile Devices Administration Center trace file	<DSYPATH>\Server\logs\dsyadmin*nnnn*.trace

Important: If you are using Windows, view the log and trace files in Wordpad. Non-English characters in the log files might not display correctly if you view them from the command prompt.

To change the directory to which DB2 Everyplace stores its log files, edit the DSYGdflt.properties file, which is located in the <DSYPATH>\Server\properties\com\ibm\mobileservices directory. Set the `Trace.Path` property to the directory to which you want the log files to be stored. Use "\\" to denote subdirectories.

Example: To store the log files in the C:\log\DB2e directory, enter the following value for the `Trace.Path` property:`c:\\log\\DB2e`

Enabling tracing for the DB2 Everyplace Sync Client

Each time that you synchronize a mobile device, the DB2 Everyplace Sync Client generates a trace file called trace-isyn that stores information about the synchronization process. Follow these steps to enable tracing in your CLI-based application.

1. Call the iscServiceOpenEx() function to create a new service handle. When you call the function, set the value of the *isync.trace* argument to "detailed".
2. Call the iscConfingOpen() function to connect to the configuration store. When you call the function, set the value of the *path* argument to the path to which you want to store the trace-isyn file.

Important: For debugging purposes, view the file by using a text editor on a workstation. If you are using Windows, open the trace file in Wordpad. Windows Notepad might not display international characters properly.

Important: When you report a synchronization problem, include a detailed-level trace file.

Verifying database integrity with the data integrity check tool

The data integrity check tool reports whether tables and indexes are corrupted.

To run this tool, execute the DBCHECK command in DB2eCLP. The command syntax is:

`DBCHECK` *`outputfile`*

The *outputfile* parameter specifies a text file in the database directory where the tool will write the results.

Note: This tool is supported on Linux and Windows 32-bit operating systems only.

Handling DB2 Everyplace synchronization problems

When synchronization is interrupted, the DB2 Everyplace Sync Server writes messages to the log in the administration control database.

The following topics explain how to handle synchronization problems:

Synchronization conflict resolution

At times, changes that a client submits to the DB2 Everyplace Sync Server conflict with changes that other clients or applications previously made or are simultaneously making to the source tables. The DB2 Everyplace Sync Server tracks the version of each record in each table in a replication subscription. Each client is similarly tracked to maintain a version of each record for each client's last synchronization with each table. This information allows the DB2 Everyplace Sync Server to determine whether a client is attempting to update a row based on an obsolete version of the data for that row. If a client attempts to update a row based on an obsolete version of the data for that row, the update is rejected.

Note: Restoring a DB2 Everyplace target (device) database from an earlier backup image will result in unpredictable behavior, including data inconsistencies that might affect users in addition to the user of the offending device.

Conflict resolution happens when data is staged to the mirror tables on the mid-tier system, as shown in Figure 1. This occurs in the replication cycle following the client's synchronization session. As a result, conflicts from a client's updates will not be detected until after response messages are returned to the client during that synchronization. Rejections of client changes will be communicated back to the client in the first synchronization session following the replication in which the conflict was discovered. If a client change is based on an obsolete record, a correct version of that record will be returned in the original synchronization request.

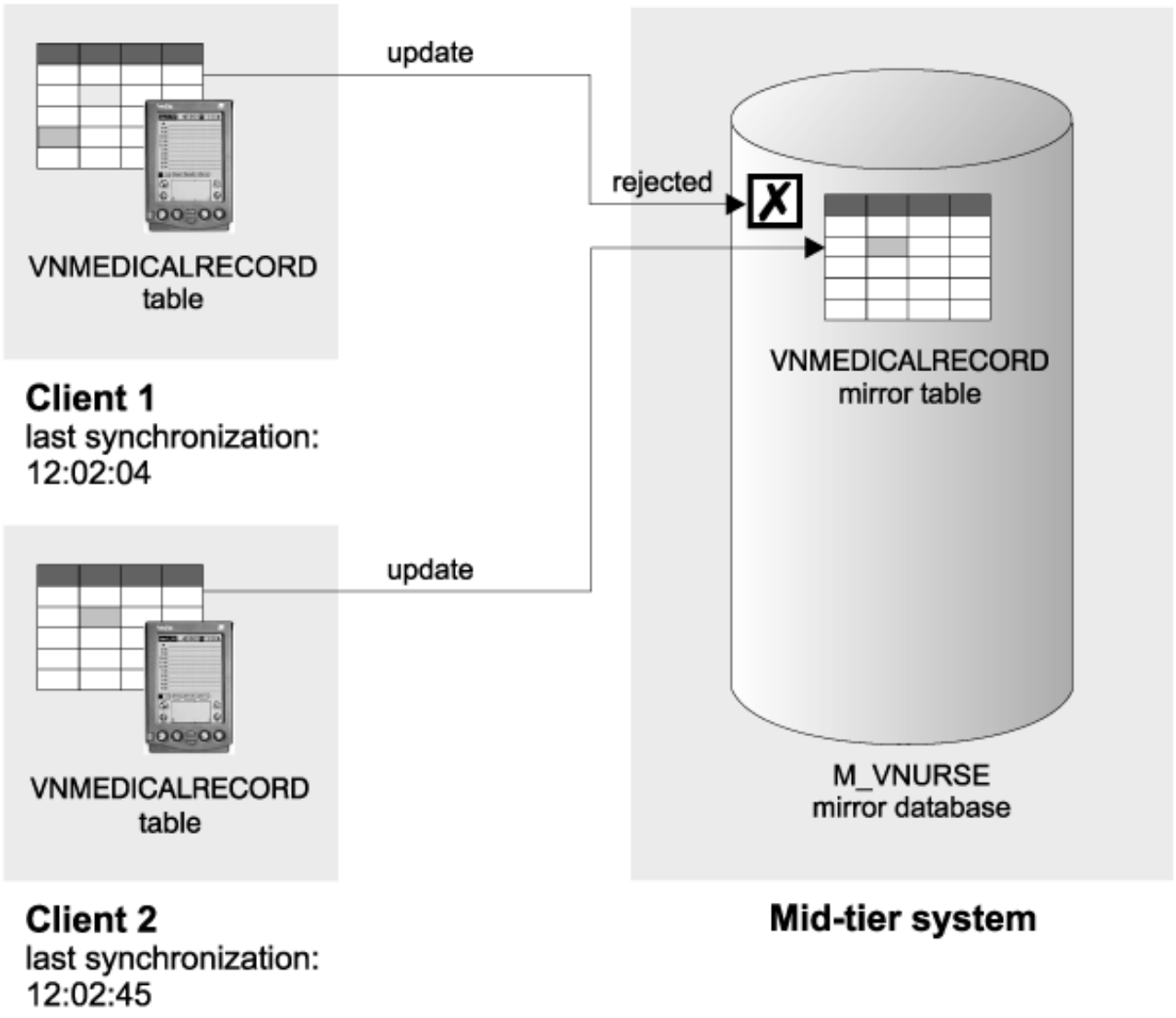


Figure 1. How the DB2 Everyplace Sync Server handles conflicts

The client whose update was rejected receives both the rejected record and a correct version of that record. The rejected record is recorded in the log on the client or provided to the application by the client API. The correct version of that record replaces the original (rejected) record on the client's DB2 Everyplace mobile database.

When DataPropagator™ applies the changed data from the mid-tier to the source database, additional types of conflicts can occur. See the DB2 Version 9.1 documentation for more information about how these conflicts are managed and resolved.

Related concepts

"Synchronization conflict resolution" on page 88

"DB2 Everyplace environments" on page 5
DB2 Everyplace Enterprise Edition is a robust solution for synchronizing enterprise data. You can configure the DB2 Everyplace environment in multiple ways depending on the needs of your network and your users.

The order of synchronization and reception of error messages

Subscriptions are synchronized in the order that you added them when you created the subscription set in the Mobile Devices Administration Center. Similarly, for each subscription, the tables are synchronized in the order that you added them when you created or altered the subscription. It's important to understand the order of synchronization so that you can interpret the logs and resolve synchronization problems.

You can change the order of the subscriptions and subscription sets by editing them in Mobile Devices Administration Center. The subscriptions listed in the Change Subscription Set or Change Subscription window are in the order that you added them, with the earliest at the top. You can alter the order of synchronization in the Change Subscription Set, Create Subscription Set, Change Group, Create Group, and Define Replication Subscriptions windows.

If a record was rejected by the DB2 Everyplace Sync Server for JDBC and DataPropagator subscriptions, the client receives error messages about the rejection during the synchronization that follows a replication of that mirror database.

Related concepts

"Handling DB2 Everyplace synchronization problems" on page 88
When synchronization is interrupted, the DB2 Everyplace Sync Server writes messages to the log in the administration control database.

Related tasks

"Viewing the error log to diagnose problems"

"Purging error log entries automatically" on page 92

"Defining the tracing level" on page 91

"Providing error-handling logic for user-exits" on page 93

"Viewing the log on the mobile device" on page 92

Viewing the error log to diagnose problems

When you encounter synchronization problems, you can troubleshoot by using the Mobile Devices Administration Center to view the error log.

To determine the actions that you should take for a particular message, see "Error messages" on page 112, which explains each message code and suggests actions that you can take to resolve the problem.

Additional logs are created in the form of trace files. You can use a text editor to view the trace files. The location of the trace files are defined by the `Trace.Path` value which is found in the **DSYGdflt.properties** file. The value in `Trace.Path` is identified by the `.trace` suffix. By default, this value is set to store the trace files in the <DSYPATH>\Server\logs directory, where <DSYPATH> is the directory where DB2 Everyplace is installed. You can change the default value by editing the `Trace.Path` value.

For Websphere Everyplace Access customers, the location of the trace files (for the DB2 Everyplace Sync Server Servlet) are found in the WAS node for the DB2 Everyplace Sync Sever Servlet. By default the location of the trace files are set to the <DSYPATH>\Server\logs directory, where <DSYPATH> is the directory where DB2 Everyplace is installed. This value can be changed, so it's advisable to check for the current location of the trace files. The trace file location for errors in the Mobile Devices Administration

Center, replication involved inside by the DB2 Everyplace Sync Server servlet, and the XML Scripting tool are located the DB2 Everyplace **DSYGdflt.properties** file.

To view the log to diagnose problems:

1. Start the Mobile Devices Administration Center.
2. In the object tree, select the Logs folder to open the log.

   When you open the **Logs** folder, the contents pane displays the following information:

   Timestamp
   This field displays the time the message was written to the log.

   Code This field displays the number of the message.

   Description
   This field displays the text of the message. Message text is truncated to 255 characters.

   User name
   User for which this log entry is associated, if applicable.

   Subscription
   Subscription for which this log entry is associated, if applicable.

   Database
   Database for which this log entry is associated, if applicable.

   Host:Port
   Identification of DB2 Everyplace Sync Server reporting this log entry.

Defining the tracing level

By default, the DB2 Everyplace Sync Server only logs error messages in the trace files. However, for diagnostic purposes, you might want to turn on tracing to include more detailed information. Use DSYTrace to turn on or off tracing.

The DSYTrace command changes the Trace.Level in file DSYGdflt.properties in the \<DSYPATH>\Server\properties\com\ibm\mobileservices\ directory, where <DSYPATH> is the directory where DB2 Everyplace is installed. For additional tracing options, type `DSYTrace` with no options at the command line or open the DSYGdflt.properties file in a text editor and modify the `Trace.`*`var`* parameters where *var* is a specific parameter variable.

1. To save all trace messages in a `.trace` file, turn tracing on.
   a. Open a command prompt.
   b. Change to the <DSYPATH>/Server/bin directory, where <DSYPATH> is the directory where DB2 Everyplace is installed.
   c. At the command prompt, type: `DSYTrace 1 -console`
2. To save only error messages in a`.trace` file, turn tracing off.
   a. Open a command prompt.
   b. Change to the <DSYPATH>/Server/bin directory, where <DSYPATH> is the directory where DB2 Everyplace is installed.
   c. At the command prompt, type: `DSYTrace 0 -console`

Related concepts

"Handling DB2 Everyplace synchronization problems" on page 88
When synchronization is interrupted, the DB2 Everyplace Sync Server writes messages to the log in the administration control database.

"The order of synchronization and reception of error messages" on page 90

Related tasks

"Viewing the error log to diagnose problems" on page 90

"Purging error log entries automatically"
"Providing error-handling logic for user-exits" on page 93
"Viewing the log on the mobile device"

Viewing the log on the mobile device

If you are using DB2 Sync on a Palm device or emulator, a synchronization log is kept (LOGDB-ISYN.pdb). To view the log in the DB2 Sync application , click the **Log** button. If you want to save its contents for debugging purposes, use any utility that can view the contents of a .pdb file to open this file.

The contents of the synchronization log are overwritten each time a new synchronization starts.

In addition to the synchronization log that the DB2 Sync provides, the synchronization engine generates a trace file named trace-isyn.

Related concepts

"Handling DB2 Everyplace synchronization problems" on page 88
When synchronization is interrupted, the DB2 Everyplace Sync Server writes messages to the log in the administration control database.
"The order of synchronization and reception of error messages" on page 90

Related tasks

"Viewing the error log to diagnose problems" on page 90
"Purging error log entries automatically"
"Defining the tracing level" on page 91
"Providing error-handling logic for user-exits" on page 93

Purging error log entries automatically

You can specify how many days the entries in the error log in the Mobile Devices Administration Center should be kept. The log entries that are older than the specified days are automatically purged. This feature helps to keep the size of the error log small.

To purge error log entries automatically:

1. Use the dsysetproperty tool to set the DSYGdflt Log.KeepDays property.
2. You can also set a maximum number of log entries which will be stored before purging will take place. This is dictated by the DSYGdflt Log.PruneToSize property, which defaults to 10000 entries.

Related concepts

"Handling DB2 Everyplace synchronization problems" on page 88
When synchronization is interrupted, the DB2 Everyplace Sync Server writes messages to the log in the administration control database.
"The order of synchronization and reception of error messages" on page 90

Related tasks

"Viewing the error log to diagnose problems" on page 90
"Defining the tracing level" on page 91
"Providing error-handling logic for user-exits" on page 93
"Viewing the log on the mobile device"

Providing error-handling logic for user-exits

Frequently, error messages from the Mobile Devices Administration Center logs require that you take action to resolve the problem indicated in the message text. To simplify day-to-day synchronization management, you might choose to add your own logic to automatically perform these actions when a particular error message is issued.

For example, suppose that you want to include a user exit that pages you when message DSYD000E is issued. You could write a program called pager.exe that dials your pager, and includes the pager number as a parameter. The line in the DSYUserExits file might look something like this:

```
DSYD000E=pager.exe number=9980674
```

You can also include the following parameters and variables with the command:

ID Use this parameter to write the number of the message. For the parameter value, specify the variable *DSYID*.

MSG Use this parameter to write the message text. For the parameter value, specify DSYMSG to write the actual message text, or DSYMSG_ to write the message text, but convert all blank spaces in the text to underscore characters.

The following example dials the same pager, but submits both the message number and its text as parameters so they appear in the pager window:

```
DSYD000E=pager.exe number=9980674 id=dsyid msg=dsymsg
```

The DB2 Everyplace Sync Server does not check the validity of your entries in the DSYUserExits properties file, nor does it validate that the action associated with the message has been completed.

Use the DSYUserExitsTest.bat file tool to test a user exit routine. Use the following format to test:

```
DSYUserExitsTest.bat dsy_message_id
```

where *dsy_message_id* is the message number you want to simulate. If you do not provide a message ID, a list of available message IDs is displayed.

Example command with message id DSYS001I:

```
DSYUserExitsTest.bat dsys001i
```

The DSYS001I message is generated and the user exit defined for this message number is started.

Example command with an invalid message number:

```
DSYUserExitsTest.bat zzz
```

The output from the DSYUserExitsTest tool is:

```
DSYUserExitsTest
    DSY message id 'ZZZ' not found. Valid DSY message id's are:
DSYA000E, DSYA001E, DSYD000E, DSYD002E, DSYD006E, DSYD007E, ...
```

To define automatic processing of errors:

1. Open the DSYUserExits.properties file for editing. This file associates a message number with a routine or program that runs when the message number is written to the log. This properties file supports many different parameters.

   ```
   # Formats:
   # {DSY message id}={class to execute} {environment parameters}
   # {DSY message id}={command to execute} {environment parameters}
   #   where
   #     {DSY message id}:
   ```

```
#       a DSY* message id that you want to define a user exit for (such as
#       DSYD000E)
#     {class to execute}:
#       the name the Java .class to execute. This class must implement the
#       com.ibm.mobileservices.DSYUserExitsInterface
#     {command to execute}:
#       the name the command to execute (such as pager.exe)
#     {environment parameters}:
#       a series of parameters to pass in to the class or command to execute
#
#   Optional command tags:
#     <DSYID>            = the message id
#     <DSYIDMSG>         = the message id message text
#     <DSYIDMSG_>        = the message id message text, but all blanks are
#                          converted to underscores
#     <DSYMSG>           = the message text
#     <DSYMSG_>          = the message text, but all blanks are converted to
#                          underscores.
#     <SERVER_IPADDRESS> = the server ip address (such as 9.112.19.143)
#     <SERVER_NAME>      = the server name (such as mpauser.stl.ibm.com)
#
# Refer to the messagestopic for available DSY message ids.
#
# Example 1:
#   If you wanted to have user exists for DSYD000E, you would add a line
#   similar to the following:
#
# DSYD000E=pager.exe number=5551234 id=<DSYID> msg=<DSYMSG_>
#
# When a DSYD000E message was issued, the pager.exe command would be executed
# with two environment parameters would be set: number=5551234, id=DSYD000E
# and msg=DSY message text, substituting an underscore (_) for blanks.
#
# Example 2:
#   If you wanted to have a class executed when a DSYD020E was encountered,
#   you would add a line similar to the following:
#
# DSYD020E=com.ibm.mobileservices.DSYUserExitsSample.class
#
# When a DSYD020E was issued, the com.ibm.mobileservices.DSYUserExitsSample
# class would be executed.
```

The action to be performed must be a reference to a valid routine or program.

2. At the end of the file, pair a message number with an action to be performed when that message is written to the log. The action can be a command (such as an executable or batch file) or a Java class. Use the following format:

   message_number=action parameter=value

   where:

   message_number
   : The number of the message. See "Error messages" on page 112 for message numbers and their associated text.

   action
   : The file referencing the command or Java class that is called. The Java class must implement the com.ibm.mobileservices.DSYUserExitsInterface.

   parameter=value
   : A series of parameter sets (such as `lastname=Doe firstname=John`), with each set separated by a space.

Related concepts

"Handling DB2 Everyplace synchronization problems" on page 88
When synchronization is interrupted, the DB2 Everyplace Sync Server writes messages to the log in the administration control database.

“The order of synchronization and reception of error messages” on page 90

Related tasks

“Viewing the error log to diagnose problems” on page 90
“Purging error log entries automatically” on page 92
“Defining the tracing level” on page 91
“Viewing the log on the mobile device” on page 92

Reference for DB2 Everyplace

This section presents reference information.

Data type mappings between DB2 Everyplace and data sources

This topic shows the default mirror and client data types to which various backend data source data types are mapped.

Important:

1. Because of the inherent differences between non-DB2 data types and DB2 data types, the creation of certain subscriptions and the replication or synchronization of certain values might not be possible.
2. If a data type is not listed in the data type mapping tables, it is not supported.
3. You cannot include a table in a subscription if it contains a column of an unsupported data type.

Supported database default values

The following tables list the default values that a column of a table in a data source might have. A source table having columns with any of the default values listed can be synchronized by the DB2 Everyplace Sync Server. A table having columns with default values not listed in the table for the data source cannot be synchronized by the DB2 Everyplace Sync Server.

Note: If you are using the IBM Toolbox for Java driver to connect to DB2 on AS/400, your table cannot have any columns with non-null default values.

Table 27. Supported DB2 default values

DB2 data type	DB2 default value
BIGINT	constant, NULL
CHAR(n)	constant, NULL
DATE	current date, NULL
DECIMAL(p,s)	constant, NULL
DOUBLE	constant, NULL
FLOAT	constant, NULL
GRAPHIC(n)	NULL
INTEGER	constant, NULL
LONG VARCHAR	constant, NULL
LONG VARCHAR FOR BIT DATA	NULL
LONG VARGRAPHIC	NULL
REAL	constant, NULL
SMALLINT	constant, NULL
TIME	current time, NULL
TIMESTAMP	current timestamp, NULL
VARCHAR(n)	constant, NULL
VARCHAR(n) FOR BIT DATA	NULL

Table 27. Supported DB2 default values (continued)

VARGRAPHIC(n)	NULL

Table 28. Supported Informix default values

Informix data type	Informix default value
CHAR	NULL
CHARACTER VARYING(m,r)	NULL
DATE	NULL
DATETIME HOUR TO SECOND	NULL
DATETIME HOUR TO FRACTION	NULL
DATETIME YEAR TO DAY	NULL
DATETIME YEAR TO SECOND	NULL
DATETIME YEAR TO FRACTION	NULL
DATETIME YEAR TO FRACTION(5)	NULL
DECIMAL(p,s)	NULL
DOUBLE PRECISION	NULL
FLOAT(n)	NULL
INT8	NULL
INTEGER	NULL
INTERVAL largest_qualifier(p) TO smallest_qualifier(s)	NULL
LVARCHAR	NULL
MONEY(p,s)	NULL
NCHAR(n)	NULL
NUMERIC(p,s)	NULL
NVARCHAR(m)	NULL
REAL	NULL
SMALLFLOAT	NULL
SMALLINT	NULL
VARCHAR(m)	NULL

Table 29. Supported Oracle default values

Oracle data type	Oracle default value
CHAR(n)	constant, NULL
DATE	SYSDATE, NULL
NUMBER(p,s)	constant, NULL
RAW(n)	NULL
TIMESTAMP	NULL
VARCHAR2(n)	constant, NULL

Table 30. Supported Microsoft SQL Server default values

Microsoft SQL Server data type	Microsoft SQL Server default value
BIGINT	constant, NULL

Table 30. Supported Microsoft SQL Server default values (continued)

BIT	constant, NULL
CHAR	constant, NULL
DATETIME	NULL
DECIMAL	constant, NULL
FLOAT	constant, NULL
INTEGER	constant, NULL
MONEY	constant, NULL
NCHAR	constant, NULL
NUMERIC	constant, NULL
NVARCHAR	constant, NULL
REAL	constant, NULL
SMALLDATETIME	NULL
SMALLINT	constant, NULL
SMALLMONEY	constant, NULL
TINYINT	constant, NULL
VARCHAR	constant, NULL

DB2™ family data type mappings

Table 31 lists the data type mapping that is performed when the source data type is a DB2 Version 9.1 or DB2 Universal Database (UDB) Version 8.2 data type.

Table 31. Data type mapping from DB2 Universal Database source data types

DB2 Version 9.1 and DB2 UDB Version 8.2 source data type	DB2 Version 9.1 mirror data type	DB2 Everyplace device data type	IBM Cloudscape Version 10 device data type
BIGINT	BIGINT	VARCHAR	BIGINT
BLOB(n [K|M|G])	unsupported	unsupported	unsupported
CHAR(n)	CHARACTER	CHARACTER	CHARACTER
CHAR(n) FOR BIT DATA	unsupported	unsupported	unsupported
CLOB(n [K|M|G])	unsupported	unsupported	unsupported
DATALINK	unsupported	unsupported	unsupported
DATE	DATE	DATE	DATE
DBCLOB(n [K|M|G])	unsupported	unsupported	unsupported
DECIMAL(p,s)	DECIMAL	DECIMAL	DECIMAL
DOUBLE	FLOAT	VARCHAR	DOUBLE PRECISION
DOUBLE PRECISION	FLOAT	VARCHAR	DOUBLE PRECISION
FLOAT	FLOAT	VARCHAR	DOUBLE PRECISION
GRAPHIC(n)	GRAPHIC	CHARACTER	unsupported
INTEGER	INTEGER	INTEGER	INTEGER
LONG VARCHAR	LONG VARCHAR	VARCHAR	LONG VARCHAR
LONG VARCHAR FOR BIT DATA	LONG VARCHAR FOR BIT DATA	BLOB	LONG BIT VARYING

Table 31. Data type mapping from DB2 Universal Database source data types (continued)

DB2 Version 9.1 and DB2 UDB Version 8.2 source data type	DB2 Version 9.1 mirror data type	DB2 Everyplace device data type	IBM Cloudscape Version 10 device data type
LONG VARGRAPHIC	LONG VARGRAPHIC	VARCHAR	unsupported
REAL	REAL	VARCHAR	REAL
SMALLINT	SMALLINT	SMALLINT	SMALLINT
TIME	TIME	TIME	TIME
TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP
VARCHAR(n)	VARCHAR	VARCHAR	VARCHAR
VARCHAR(n) FOR BIT DATA	VARCHAR() FOR BIT DATA	BLOB	BIT VARYING
VARGRAPHIC(n)	VARGRAPHIC	VARCHAR	unsupported
XML	unsupported	unsupported	unsupported

Related reference

"Data type mappings between DB2 Everyplace and data sources" on page 97
This topic shows the default mirror and client data types to which various backend data source data types are mapped.

"Informix data type mappings"

"Oracle data type mappings" on page 101

Informix data type mappings

Table 32 lists the data type mapping that is performed when the source data type is an Informix® data type.

Table 32. Data type mapping from Informix source data types

Informix source data type	DB2 Version 9.1 mirror data type	DB2 Everyplace device data type	IBM Cloudscape Version 10 device data type
BLOB	unsupported	unsupported	unsupported
BOOLEAN	unsupported	unsupported	unsupported
BYTE	unsupported	unsupported	unsupported
CHAR(n)	CHARACTER	CHARACTER	CHARACTER
CHARACTER VARYING(m,r)	VARCHAR	VARCHAR	VARCHAR
CLOB	unsupported	unsupported	unsupported
DATE	DATE	DATE	DATE
DATETIME HOUR TO SECOND	TIME	TIME	TIME
DATETIME HOUR TO FRACTION	TIMESTAMP	TIMESTAMP	TIMESTAMP
DATETIME YEAR TO DAY	DATE	DATE	DATE
DATETIME YEAR TO SECOND	TIMESTAMP	TIMESTAMP	TIMESTAMP
DATETIME YEAR TO FRACTION	TIMESTAMP	TIMESTAMP	TIMESTAMP

Table 32. Data type mapping from Informix source data types (continued)

Informix source data type	DB2 Version 9.1 mirror data type	DB2 Everyplace device data type	IBM Cloudscape Version 10 device data type
DATETIME YEAR TO FRACTION(5)	TIMESTAMP	TIMESTAMP	TIMESTAMP
DECIMAL(p,s)	DECIMAL	DECIMAL	DECIMAL
DOUBLE PRECISION	DECIMAL	DECIMAL	DECIMAL
FLOAT(n)	FLOAT	VARCHAR	FLOAT
INT8	BIGINT	VARCHAR	BIGINT
INTEGER	INTEGER	INTEGER	INTEGER
INTERVAL largest_qualifier(p) TO smallest_qualifier(s)	CHARACTER	CHARACTER	CHARACTER
LVARCHAR	VARCHAR	VARCHAR	VARCHAR
MONEY(p,s)	DECIMAL	DECIMAL	DECIMAL
NCHAR(n)	CHARACTER	CHARACTER	CHARACTER
NUMERIC(p,s)	NUMERIC	DECIMAL	NUMERIC
NVARCHAR(m)	VARCHAR	VARCHAR	VARCHAR
REAL	REAL	VARCHAR	REAL
SERIAL(n)	unsupported	unsupported	unsupported
SERIAL8	unsupported	unsupported	unsupported
SMALLFLOAT	REAL	VARCHAR	REAL
SMALLINT	SMALLINT	SMALLINT	SMALLINT
TEXT	unsupported	unsupported	unsupported
VARCHAR(m)	VARCHAR	VARCHAR	VARCHAR

Related reference

"Data type mappings between DB2 Everyplace and data sources" on page 97
This topic shows the default mirror and client data types to which various backend data source data types are mapped.

"DB2™ family data type mappings" on page 99

"Oracle data type mappings"

Oracle data type mappings

Table 33 lists the data type mapping that is performed when the source data type is an Oracle data type.

Table 33. Data type mapping from Oracle source data types

Oracle source data type	DB2 mirror data type	DB2 Everyplace device data type	IBM Cloudscape Version 10 device data type
BFILE	unsupported	unsupported	unsupported
BLOB	unsupported	unsupported	unsupported
CHAR(n)	CHARACTER	CHARACTER	CHARACTER
CLOB	unsupported	unsupported	unsupported
DATE	TIMESTAMP	TIMESTAMP	unsupported
FLOAT	unsupported	unsupported	unsupported

Table 33. Data type mapping from Oracle source data types (continued)

Oracle source data type	DB2 mirror data type	DB2 Everyplace device data type	IBM Cloudscape Version 10 device data type
LONG	unsupported	unsupported	unsupported
LONG RAW	unsupported	unsupported	unsupported
NCHAR(n)	unsupported	unsupported	unsupported
NCLOB	unsupported	unsupported	unsupported
NUMBER(p,s)	DECIMAL	DECIMAL	DECIMAL
NVARCHAR2(n)	unsupported	unsupported	unsupported
RAW(n)	VARCHAR() BIT FOR DATA	BLOB	BIT VARYING
REAL	unsupported	unsupported	unsupported
ROWID	unsupported	unsupported	unsupported
TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP
UROWID	unsupported	unsupported	unsupported
VARCHAR2(n)	VARCHAR	VARCHAR	VARCHAR

Related reference

"Data type mappings between DB2 Everyplace and data sources" on page 97
This topic shows the default mirror and client data types to which various backend data source data types are mapped.

"DB2™ family data type mappings" on page 99

"Informix data type mappings" on page 100

Microsoft SQL Server data type mappings

Table 34 lists the data type mapping that is performed when the source data type is Microsoft SQL Server. In DB2 Everyplace version 8.1.4 and below, the mapping for Microsoft SQL Server BIT data type is inconsistent between JDBC and upload subscriptions. In JDBC subscriptions, the Microsoft SQL Server data type BIT is mapped to the DB2 Everyplace data type SMALLINT. In upload subscriptions, the Microsoft SQL Server data type BIT is mapped to the DB2 Everyplace data type VARCHAR(1). In DB2 Everyplace version 8.2, the BIT is mapped to SMALLINT in both cases. If you desire the old, inconsistent behavior, run the following script and restart the DB2 Everyplace Sync Server: `dsysetproperty "DatatypeMappings Generic Target:*" -7="12 VARCHAR"`

Table 34. Data type mapping from Microsoft SQL Server

Microsoft SQL Server source type	DB2 Version 9.1 mirror data type	DB2 Everyplace device data type	IBM Cloudscape Version 10 device data
BIGINT	BIGINT	VARCHAR	BIGINT
BINARY	unsupported	unsupported	unsupported
BIT	SMALLINT	SMALLINT	SMALLINT
CHAR	CHARACTER	CHARACTER	CHARACTER
CURSOR	unsupported	unsupported	unsupported
DATETIME	TIMESTAMP	TIMESTAMP	TIMESTAMP
DECIMAL	DECIMAL	DECIMAL	DECIMAL
FLOAT	FLOAT	VARCHAR	FLOAT
IMAGE	unsupported	unsupported	unsupported
INT	INTEGER	INTEGER	INTEGER

Table 34. Data type mapping from Microsoft SQL Server (continued)

Microsoft SQL Server source type	DB2 Version 9.1 mirror data type	DB2 Everyplace device data type	IBM Cloudscape Version 10 device data
MONEY	DECIMAL	DECIMAL	DECIMAL
NCHAR	GRAPHIC	CHARACTER	CHARACTER
NTEXT	unsupported	unsupported	unsupported
NUMERIC	DECIMAL	DECIMAL	DECIMAL
NVARCHAR	VARGRAPHIC	VARCHAR	VARCHAR
REAL	REAL	VARCHAR	REAL
SMALLDATETIME	TIMESTAMP	TIMESTAMP	TIMESTAMP
SMALLINT	SMALLINT	SMALLINT	SMALLINT
SMALLMONEY	DECIMAL	DECIMAL	DECIMAL
TEXT	unsupported	unsupported	unsupported
TIMESTAMP	unsupported	unsupported	unsupported
TINYINT	SMALLINT	SMALLINT	SMALLINT
UNIQUEIDENTIFIER	unsupported	unsupported	unsupported
VARBINARY	unsupported	unsupported	unsupported
VARCHAR	VARCHAR	VARCHAR	VARCHAR

Related reference

"Data type mappings between DB2 Everyplace and data sources" on page 97
This topic shows the default mirror and client data types to which various backend data source data types are mapped.

"DB2™ family data type mappings" on page 99

"Informix data type mappings" on page 100

"Oracle data type mappings" on page 101

Data type mapping restrictions

The following restrictions exist when data type mapping is performed:

- If DB2 mirror types GRAPHIC, VARGRAPHIC are used, the DB2 mirror database has to be created in DBCS.
- Mobile device applications must take care that the type of the data entered into a column of a device table is compatible with the types of the columns of the corresponding mirror and source tables and the length of the data does not exceed the length of the corresponding mirror and source columns. Source applications must ensure the same with data entered into a column of a source table.
- Informix columns of the type DECIMAL, NVARCHAR, and VARCHAR must not be defined using the following syntax: DECIMAL(p), NVARCHAR(m,r), and VARCHAR(m,r).
- The following data types are not supported as primary keys by DB2 Everyplace Sync Server:
  - **DB2 Version 9.1 Type:** LONG VARCHAR, LONG VARCHAR FOR BIT DATA, LONG VARGRAPHIC, and VARCHAR() FOR BIT DATA
  - **Informix Type:** DATETIME and INTERVAL
  - **Oracle Type:** RAW
  - **SQL Server Type:** MONEY, REAL, and SMALLMONEY

Because of the inherent differences between non-DB2 data types and DB2 data types, the creation of certain subscriptions and the replication or synchronization of certain values might not be possible. If a data type is not listed in the data type mapping tables, it is not supported.

Data source restrictions for DataPropagator subscriptions

Restrictions

The following restrictions apply to both Windows and UNIX® platforms:

@ • Before you can apply changes, you must first start the Capture program to capture the changes on the
@ source database.
- Do not create a DataPropagator subscription on tables that have referential-integrity constraints or triggers. Otherwise, replication would fail and would be unrecoverable.
- A mirror database must only replicate with a single source database.
- For z/OS® source systems, you must issue a bind Capture job to start the Capture program on the source system. For more information about how to bind the Capture program, see the DB2 Version 9.1 documentation.

@ • For iSeries:

@ – You must start the Capture program on the source database before DB2 Everyplace Sync Server can
@ be started. Use the STRDPRCAP command to start the Capture program.

@ – Do not use the XML scripting tool to create control tables for the source database on iSeries because
@ the tool does not support that function. In other words, do not set the **CreateDPropRControlTables**
@ attribute of tag <AddReplMaster> to TRUE. You must create control tables manually by using the
@ CRTDPRTBL command.

@ – You must journal source tables manually. Use the STRJRNPF command to journal source tables.

@ **Note:** If you use the CREATE SCHEMA statement to create a library that contains the source tables,
@ journaling will perform automatically.

@ For more information on the commands, see the iSeries Information Center and SQL Replication Guide
@ and Reference.

- The mirror database must be located on the same server as the replication engine, which is a replication-enabled DB2 Everyplace Sync Server or a command-line process running the dsyreplicate script. The DataPropagator Capture program is not able to capture any remote databases.
- By default, DataPropagator will create the mirror tables in their own, non-system managed, table spaces if the subscription is created with Replication Center. The XML Scripting tool does not create the tables in their own non-system managed table spaces. If you want the DB2 Everyplace control tables associated with the mirror tables to be created in the same table spaces, you will need to either override the table space creation to specify that the table space is system-managed, or increase the table space container size to accomodate these control tables (a general recommendation is to increase the container to 5 times the Data Propagator-generated size). If you do not specify a table space in the XML to generate the DB2 Everyplace Subscription, the default table space USERSPACE1 is used for the DB2 Everyplace control tables.
- If `Replicate = "FALSE"` is not specified as an attribute in the AddDProprSubscription tag, then the mirror database must be local to the machine on which the XML Scripting Tool is executing, and the source capture process must be running.
- By default, the CommitCount value for a DataPropagator table subscription is 0, which forces all replicating changes to be performed within a single transaction. If an error occurs during replication, all changes are rolled back, and when recovery is attempted, changes are not replayed. However, using this feature increases the amount of transaction space required, especially when an application generates many changes. Depending on the client environment and application requirements, you might need to set CommitCount to a positive value or increase the transaction log space to support a CommitCount value of 0. You can set the CommitCount value in the XML script that creates or alters the DataPropagator table subscription, or modify it using the DB2 Version 9.1 Replication Center.
- The DB2 Everyplace Sync Server does not synchronize times or timestamps with an hour of 24 correctly due to differences in how time is represented in Java and a data source. A time of ″24:00:00″, for example, is converted to ″00:00:00″ and causes the data saved in the mobile database to differ from the data saved in the source database. You should avoid using such an hour in your applications.

- If you are adding a table to a subscription, its source and target schema names, column names, and table names cannot be keywords, reserved words, or special registers in SQL or DB2 Version 9.1.
- A mirror database can replicate only with one source database. DB2 Everyplace does not allow a mirror database to be associated with multiple source databases.
- In rare cases, this can result in an inability to create a table due to non-uniqueness of the table name or column name. In other rare cases it can result in a conversion of a character to multiple characters, as in the German “sharp s” (ß), which will be converted to “SS”.

DB2 Everyplace limits

Table 35 describes the limits for the DB2 Everyplace database and SQL support. If you adhere to the most restrictive case, your programs might be easier to port to other platforms. Some mobile devices might have further restrictions on these limits due to physical memory and system limitations. Refer to the documentation that came with your mobile device for more information about these limitations. Each constraint applies to all clients unless otherwise specified.

Table 35. DB2 Everyplace database and SQL limits

Description	Limit
Maximum combined length for INT, SMALLINT, CHAR, DECIMAL, DATE, TIME, and TIMESTAMP columns in a single record	32767 bytes
Maximum length of a BLOB column	2 Gigabytes -1 byte
Maximum length of a CHAR column	32767 bytes
Maximum length of a SQL statement	64 kilobytes
Maximum length of a VARCHAR column	32767 bytes
Maximum length of a check constraint	32767 bytes
Maximum length of a column name (Cloudscape 10.0 client)	30
Maximum length of a column name (Other clients)	128
Maximum length of a default value	32767 bytes
Maximum length of a row in a table	64 kilobytes
Maximum length of a table name	128
Maximum length of an identifier	128
Maximum length of an index name (Cloudscape 10.0)	18
Maximum length of an index name (Other clients)	128
Maximum length of each column in a single index	1024 bytes
Maximum number of columns in a foreign key	8
Maximum number of columns in an index	8
Maximum number of columns in a primary key	8
Maximum number of columns in a table	256
Maximum number of indices in a table	15
Maximum number of LOB locators	256
Maximum number of rows in a table	Limited by table size
Maximum number of statement handles per connection	256
Maximum number of tables in a data store	65535
Maximum size of a decimal	31 digits
Maximum size of a literal	32672 bytes

Table 35. DB2 Everyplace database and SQL limits (continued)

Description	Limit
Maximum size of a table (on a 32 bit system)	2 Gigabytes
Maximum year for a date value	9999
Minimum year for a date value	0001

DB2 Everyplace has additional limits about the size of fields that it synchronizes with the DB2 Everyplace Sync Server. If the synchronization size limit is not shown in the table below, it is the same size as the limit that is shown in Table 35 on page 105. Each constraint applies to all clients unless otherwise specified.

Table 36. DB2 Everyplace synchronization limits

Description	Limit
Maximum length of a column name	30
Maximum length of a table name (Cloudscape 10.0 and 10.1 clients)	126

Special limitations for the length of table names

Restriction: The actual length of table names might be less than 128 characters due to expansion after the name is converted to UTF-8.

Starting in DB2 Everyplace version 9.1, the maximum identifier length for table names, column names, and user names has been increased from 18 to 128 for newly created databases. If an older database exists, the maximum identifier length remains at 18.

Restriction: Previous versions of DB2 Everyplace do not support table names that are as long as DB2 Everyplace version 9.1. In order to create tables that have long names, you must create the database in DB2 Everyplace version 9.1. Otherwise, DB2 Everyplace limits the length of table names to the maximum length of the version with which the table was created.

The following platforms cannot support the 128 character limit:

Palm OS
Table names on Palm OS mobile devices cannot be longer than 26 characters. This is because Palm OS limits filenames to 31 characters, and DB2 Everyplace appends the 5 characters ″DSY_i″ for the index of each table.

QNX Neutrino version 6.2 and earlier
Table names cannot be longer than 43 characters. This is because QNX Neutrino limits filenames to 48 characters, and DB2 Everyplace appends the 5 characters ″DSY_i″ for the index of each table.

QNX Neutrino version 6.2.1 and newer
To avoid the 43 character table size limit:

1. Log in as the root user.
2. Change to the root directory.
3. Create an empty, read-only file named .longfilenames.
4. Reboot.

DB2 Everyplace reserved words

The following DB2 Everyplace reserved words cannot be used as identifiers unless they are specified as delimited identifiers. This restriction also applies to items that are added to subscriptions. Identifiers cannot be keywords, reserved words, or special registers that are used by:

- SQL
- DB2
- the data source

Example:

The following statement causes a SQL error:

```
CREATE TABLE tab1 (select int)
```

The following statement does not cause a SQL error:

```
CREATE TABLE tab1 ("select" int)
```

Table 37. DB2 Everyplace reserved words

ADD	ENCRYPTION	OR
ALL	ESCAPE	ORDER
ALTER	EXCLUSIVE	PRIMARY
ALWAYS	EXPLAIN	QUERYNO
AND	FETCH	READ
AS	FOR	REFERENCES
ASC	FOREIGN	RELEASE
BEGIN	FROM	REORG
BIT	GENERATED	RETAIN
BLOB	GRANT	REVOKE
BY	GROUP	ROLLBACK
CALL	IDENTITY	SAVEPIONT
CHAR	IN	SELECT
CHARACTER	INDEX	SET
CHECK	INSERT	SHARE
COLUMN	INT	SMALLINT
COMMIT	INTEGER	START
CONCAT	INTO	TABLE
CREATE	IS	TIME
CURRENT	KEY	TIMESTAMP
CURSORS	LIKE	TO
DATA	LIMIT	TRANSACTION
DATABASE	LOCK	TYPE
DATE	LOCKS	UNIQUE
DECIMAL	MODE	UPDATE
DEFAULT	NEW	USING
DELETE	NOT	VALUES
DESC	NULLSYM	VARCHAR
DISTINCT	OF	WHERE
DROP	ON	WITH
ENCRYPT	ONLY	WORK

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 as follows:

Table 38. IBM SQL reserved words that are not currently used by DB2 Everyplace

ACQUIRE	DISCONNECT	LONG	RESULT
AFTER	DO	LOOP	RETURN
ALIAS	DOUBLE	MAX	RETURNS
ALLOCATE	DSSIZE	MICROSECOND	RIGHT
ALLOW	DYNAMIC	MICROSECONDS	ROW
ANY	EDITPROC	MIN	ROWS
ASUTIME	ELSE	MINUTE	RRN
AUDIT	ELSEIF	MINUTES	RUN
AUTHORIZATION	END	MODIFIES	SCHEDULE
AUX	END-EXEC	MONTH	SCHEMA
AUXILIARY	ERASE	MONTHS	SCRATCHPAD
AVG	EXCEPT	NAME	SECOND
BEFORE	EXCEPTION	NAMED	SECONDS
BETWEEN	EXECUTE	NHEADER	SECQTY
BINARY	EXISTS	NO	SECURITY
BUFFERPOOL	EXIT	NODEN	SIMPLE
CALLED	EXTERNAL	AME	SOME
CAPTURE	FENCED	NODENUMBER	SOURCE
CASCADED	FIELDPROC	NULLS	SPECIFIC
CASE	FILE	NUMPARTS	SQL
CAST	FINAL	OBID	STANDARD
CCSID	FREE	OPEN	STATIC
CLOSE	FULL	OPTIMIZATION	STATISTICS
CLUSTER	FUNCTION	OPTIMIZE	STAY
COLLECTION	GENERAL	OPTION	STOGROUP
COLLID	GO	OUT	STORES
COMMENT	GOTO	OUTER	STORPOOL
CONDITION	GRAPHIC	PACKAGE	STYLE
CONNECT	HANDLER	PAGE	SUBPAGES
CONNECTION	HAVING	PAGES	SUBSTRING
CONSTRAINT	HOUR	PARAMETER	SUM
CONTAINS	HOURS	PART	SYNONYM
CONTINUE	IDENTIFIED	PARTITION	TABLESPACE
COUNT	IF	PATH	THEN
COUNT_BIG	IMMEDIATE	PCTFREE	TRIGGER
CROSS	INDICATOR	PCTINDEX	TRIM
CURRENT_DATE	INNER	PIECESIZE	UNDO
CURRENT_LC_PATH	INOUT	PLAN	UNION
CURRENT_PATH	INSENSITIVE	POSITION	UNTIL
CURRENT_SERVER	INTEGRITY	PRECISION	USAGE
CURRENT_TIME	INTERSECT	PREPARE	USER
CURRENT_TIMESTAMP	ISOBID	PRIQTY	USING
CURRENT_TIMEZONE	ISOLATION	PRIVATE	VALIDPROC
CURRENT_USER	JAVA	PRIVILEGES	VARIABLE
DAY	JOIN	PROCEDURE	VARIANT
DAYS	LABEL	PROGRAM	VCAT
DBA	LANGUAGE	PSID	VIEW
DBINFO	LC_CTYPE	PUBLIC	VOLUMES
DBSPACE	LEAVE	READS	WHEN
DB2GENERAL	LEFT	RECOVERY	WHILE
DB2SQL	LINKTYPE LOCAL	RENAME	WLM
DECLARE	LOCALE	REPEAT	WRITE
DESCRIPTOR	LOCATOR	RESET	YEAR
DETERMINISTIC	LOCATORS	RESOURCE	YEARS
DISALLOW	LOCKSIZE	RESTRICT	

Table 39. ISO/ANSI SQL92 reserved words that are not used by IBM SQL

ABSOLUTE	CONVERT	FOUND
ACTION	CORRESPONDING	FULL
ARE	DEALLOCATE	GET
ASSERTION	DEC	GLOBAL
AT	DEFERRABLE	IDENTITY
BIT_LENGTH	DEFERRED	INITIALLY
BOTH	DESCRIBE	INPUT
CATALOG	DIAGNOSTICS	INTERVAL
CHAR_LENGTH	DOMAIN	LAST
CHARACTER_LENGTH	EXEC	LEADING
COALESCE	EXTRACT	LEVEL
COLLATE	FALSE	LOWER
COLLATION	FIRST	MATCH
CONSTRAINTS	FLOAT	MODULE

Table 40. ISO/ANSI SQL92 reserved words that are not used by IBM SQL, continued

NAMES	PRESERVE	TIMEZONE_HOUR
NATIONAL	PRIOR	TIMEZONE_MINUTE
NATURAL	REAL	TRAILING
NCHAR	RELATIVE	TRANSLATION
NEXT	SCROLL	TRUE
NULLIF	SESSIONSESSION_USER	UNKNOWN
NUMERIC	SIZE	UPPER
OCTET_LENGTH	SPACESQLCODE	VALUE
OUTPUT	SQLERROR	VARYING
OVERLAPS	SQLSTATE	WHENEVER
PAD	SYSTEM_USER	ZONE
PARTIAL	TEMPORARY	

Overview of the DB2 Everyplace mobile database tables

A DB2 Everyplace mobile database comprises several system catalog tables and a number of user-defined tables.

Each table is stored in two files: one for the data itself, and one for indexes. All indexes are kept in the same index file. Unlike DB2 Version 9.1, DB2 Everyplace mobile databases do not have names and cannot be cataloged or uncataloged. Therefore, the database name is ignored.

A DB2 Everyplace mobile database is a set of files that can be copied or moved to another location. A DB2 Everyplace mobile database must contain the following system catalog tables:

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

System catalog tables contain metadata about user-defined tables. For example, if you remove files for a user-defined table without deleting a corresponding entry in the catalog tables, you will cause an inconsistency.

To access catalog tables in a query, you must use delimited identifiers. For example, the following query returns 1 if the table T exists:

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

Related reference

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 41. System catalog base tables

Description	Catalog base table
tables	"DB2eSYSTABLES" on page 66
columns	"DB2eSYSCOLUMNS" on page 66
referential constraints	"DB2eSYSRELS" on page 67
users	"DB2eSYSUSERS" on page 67

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 42. DB2eSYSTABLES system catalog base table

Column name	Data type	Nullable	Description
TNAME	VARCHAR (129)		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 (32767)	Yes	Check constraint (internal use only)
IDXINFO	BLOB (4096)	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 43. DB2eSYSCOLUMNS system catalog base table

Column name	Data type	Nullable	Description
CNAME	VARCHAR (129)		Column name
TNAME	VARCHAR (129)		Table name
TYPE	INTEGER (4)		Data type

Table 43. DB2eSYSCOLUMNS system catalog base table (continued)

Column name	Data type	Nullable	Description
ATTR	INTEGER (4)		(Internal use only)
LENGTH	INTEGER (4)		Length of the column
POS	INTEGER (4)		Column number
FLAGS	INTEGER (4)		(Internal use only)
KEYSEQ	INTEGER (4)		Ordinal position of the column in the primary key
SCALE	INTEGER (4)		Scale for decimal column
@ DEF	VARCHAR (32767)	Yes	Default value (internal use)

DB2eSYSRELS

This system catalog base table contains a row for each referential constraint.

Table 44. 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 (129)		Parent table name
PKCOLUMN_NAME	VARCHAR (129)		Parent table primary key column
FKTABLE_NAME	VARCHAR (129)		Child table name
FKCOLUMN_NAME	VARCHAR (129)		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 45. DB2eSYSUSERS system catalog base table

Column name	Data type	Nullable	Description
USERNAME	VARCHAR (129)		Part of primary key and is case sensitive. The name of the user associated with this row.
DATABASENAME	VARCHAR (129)		For future use. Empty string is stored. Part of primary key.
TABLENAME	VARCHAR (129)		For future use. Empty string is stored. Part of primary key.
ENCMETHOD	VARCHAR (129)		For future use. Empty string is stored. Part of primary key.
PRIVILEGES	VARCHAR (129)	Yes	Defines user privileges. Currently, only the value 'E', indicating encryption, is allowed.
ENCKEYDATA	BLOB (280)	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.

Table 45. DB2eSYSUSERS system catalog base table (continued)

Column name	Data type	Nullable	Description
VALIDATE	BLOB (280)	Yes	Verifies that the record is authentic and the user was added by an authenticated user.
GRANTOR	VARCHAR (129)	Yes	The user name that registered the user name in column 1.
INTERNALDATA	BLOB (255)	Yes	(Internal future use)

DB2 Everyplace error messages

This topic explains error messages that are returned by DB2 Everyplace and its subsystems.

Error messages

This topic lists the error messages and their meanings, and it provides suggested actions to correct the problem identified in the message. Use this information with "Handling DB2 Everyplace synchronization problems" on page 88 to identify and correct problems.

DSYA002E: User *user name* **attempt to register device** *device id* **failed as the device is already registered to user** *registered user name***. Action canceled.**

Explanation: The DB2 Everyplace control database could not assign the device to the specified user as the device is currently assigned to the specified registered user.

User Response: If the device should be registered to the specified user, use the DB2 Everyplace Mobile Devices Administration Center to delete the device associated with the registered user and try again.

DSYC2000C: Requested data changes rejected.

Explanation: Some of the data changes that the DB2 Everyplace Sync Client requested have been rejected due to data conflicts or no permission.

User Response: Check with the DB2 Everyplace Sync Server administrator to ensure that the user has permission to perform the desired operation. because the requested changes have been rejected, the user has to re-apply the changes, or have the DB2 Everyplace Sync Server administrator reset the user and try again. If the problem persists, contact IBM software support.

DSYC300E: Failed to open adapter: *adapter name***.**

Explanation: The DB2 Everyplace Sync Client could not find the adapter library for synchronizing a subscription.

User Response: Check that the library 'adapter name' is present on the device. If the library name is unknown, contact the DB2 Everyplace Sync Server administrator, and then have the administrator reset the user and try again. If the problem persists, contact IBM software support.

DSYC301E: Failed to load adapter: *adapter name***.**

Explanation: The DB2 Everyplace Sync Client could not load the adapter library for synchronizing a subscription.

User Response: Check that the library for 'adapter name' is present on the device. If the library does exist in the device, it might be because the operating system has reached its limits of the opened shared libraries. Close unused applications, or restart the DB2 Everyplace Sync Client application, and/or do a soft reset on the device, and then try again. If the library name is unknown, contact the DB2 Everyplace Sync Server administrator, and then have the administrator reset the user and try again. If the problem persists, contact IBM software support.

DSYC302E: Failed to close adapter: *adapter name***.**

Explanation: The DB2 Everyplace Sync Client could not close the adapter library after synchronization.

User Response: The library for 'adapter name' have been used and locked by other DB2 Everyplace Sync Client applications. Check if there are any other pending DB2 Everyplace Sync Client applications, and/or do a soft reset on the device, and then try again. If the problem persists, contact IBM software support.

DSYC303E: Configuration synchronization failed, synchronization aborted.

Explanation: The DB2 Everyplace Sync Client could not properly synchronize the configuration information.

User Response: Contact the DB2 Everyplace Sync Server Administrator to check the health of the DB2 Everyplace Sync Server. Restart the DB2 Everyplace Sync Server, and/or do a soft reset on the user, and then try again. If problem persists, contact IBM software support.

DSYC304E: Authentication failed - synchronization aborted.

Explanation: The provided username/password does not pass the authentication on the DB2 Everyplace Sync Server.

User Response: Make sure that the password is entered correctly, and try again. If problem persists, contact the DB2 Everyplace Sync Server administrator.

DSYC306E: Unrecognizable client message format

Explanation: The DB2 Everyplace Sync Server does not recognize the format of the client message.

User Response: Make sure that encryption is supported on the DB2 Everyplace Sync Client platform, and/or do a soft reset on the device, and then try again. If problem persists, contact IBM software support.

DSYC307E: Client encryption/decryption failed.

Explanation: The DB2 Everyplace Sync Client could not successfully encrypt the outgoing message or decrypt the received message.

User Response: Make sure that the client's operating system supports encryption and the chosen encryption level, do a soft reset on the device and then try again. If problem persists, contact IBM software support.

DSYC308E: Encryption not available.

Explanation: Encryption is not supported for Palm OS 3.2 and earlier. The encryption library not be installed properly or the encryption path is not set correctly.

User Response: Make sure that the client's operating system supports encryption and the chosen encryption level, the encryption library is properly installed, and the encryption path is set correctly. Please refer to the Installation of User's Guide for the instruction of installation. If problem persists, contact the DB2 Everyplace Sync Server administrator.

DSYC309E: Failed to open the encryption library.

Explanation: The encryption library not be installed properly or the encryption path is not set correctly.

User Response: Make sure that the client's operating system supports encryption and the chosen encryption level, the encryption library is properly installed, and the encryption path is set correctly. Please refer to the Installation of User's Guide for the instruction of installation. If problem persists, contact the DB2 Everyplace Sync Server administrator.

DSYC310E: Incompatible DB2 Everyplace Sync Client version.

Explanation: The version for the DB2 Everyplace Sync Client is not compatible with the DB2 Everyplace Sync Server.

User Response: Check with the DB2 Everyplace Sync Server administrator to make sure a compatible DB2 Everyplace Sync Client version is installed on the device. If the problem persists, contact IBM software support.

DSYC311E: Subscription not found.

Explanation: The DB2 Everyplace Sync Server cannot find the subscription that the client is requesting. It be because the configuration has changed because the client starts synchronization.

User Response: Try to synchronize again, and the configuration will be updated. If the problem still occurs, contact the DB2 Everyplace Sync Server administrator regarding the problematic subscription, and/or reset the user, then synchronize again. If the problem persists, contact IBM software support.

DSYC312E: Subscription blocked by the Server.

Explanation: The subscription has been blocked by the DB2 Everyplace Sync Server. It be because the DB2 Everyplace Sync Server administrator is performing maintenance on the control database or modifying the subscription, or the DB2 Everyplace Sync Server is replicating the subscription, or the last replication failed, but has not yet been recovered.

User Response: Please wait for a while and try to synchronize again. If the problem persists, contact the DB2 Everyplace Sync Server administrator to check the replication status of the subscription.

DSYC313E: Attempt to synchronize from a backed-up client

Explanation: The server has detected and rejected the attempt to synchronize from a backed-up client.

User Response: Please make sure the current client image (including both the configuration and synchronized data) is good and wanted. If yes, select the ″Allow synchronization from backup″ check box; otherwise, have the DB2 Everyplace Sync Server administrator reset the user. Then, try to synchronize again. If the problem persists, contact the DB2 Everyplace Sync Server administrator.

DSYC314E: User not assigned to any group with synchronization privilege

Explanation: The user has not been added to any group with synchronization privilege.

User Response: Check that the username is entered correctly and contact the DB2 Everyplace Sync Server administrator to verify that the user is in a group with synchronization enabled. Try to synchronize again. If the problem persists, contact IBM software support.

DSYC315E: Failed to register the device.

Explanation: The DB2 Everyplace Sync Server cannot register the device with the user. There be already a device registered with the user, while the user is configured to allow to synchronize from one device.

User Response: Check that the username is entered correctly and contact the DB2 Everyplace Sync Server administrator to check the user association of the device, reset the user, and then try to synchronize again. If the problem persists, contact IBM software support.

DSYC316E: Subscription definition altered.

Explanation: The definition of a subscription has changed because the client starts synchronization.

User Response: Try to synchronize again, and the subscription definition will be updated. If the problem still occurs, contact the DB2 Everyplace Sync Server administrator to reset the user, and synchronize again. If the problem persists, contact IBM software support.

DSYC317E: Attempt to create client image without a valid device ID

Explanation: The DB2 Everyplace Sync Client engine cannot create a client image without a valid deviceID.

User Response: Creating a client image should be performed by the administrator. You can only create a client image from a client which has successfully synchronized before and has a valid device ID. Please first synchronize with the ″CreateImage″ option off, then try to create the client image again. If the problem persists, contact IBM software support.

DSYC400E: Failed to allocate adapter resources.

Explanation: The DB2 Everyplace Sync Client could not allocate adapter-specific resources (for example, database connections, or statement handles for DB2 Everyplace) for synchronizing the subscribed data.

User Response: Check that the required adapter-specific resources are not occupied and try again. If the problem persists, consult the DB2 Everyplace Application Development Guide for the limitation on those resources or contact IBM software support.

DSYC401E: Failed to connect to the target data.

Explanation: The DB2 Everyplace Sync Client detected problems when connecting to or accessing the target data.

User Response: Check the access rights and existence of the resource or make sure the resource is not being used by another application. If synchronizing encrypted tables, make sure the provided DB2 Everyplace username and password have been granted the encryption privilege. Have the DB2 Everyplace Sync Server administrator reset the user and try again. If the problem persists, contact IBM software support.

DSYC402E: Failed to disconnect from the target data.

Explanation: The DB2 Everyplace Sync Client detected problems disconnecting or releasing the target data.

User Response: Check the access rights and existence of the resource or make sure the resource is not being used by another application. Have the DB2 Everyplace Sync Server administrator reset the user and try again. If the problem persists, contact IBM software support.

DSYC403E: No data subscribed in subscription

Explanation: The DB2 Everyplace Sync Client received unexpected empty or missing subscription information (for example, no subscribed table information) from the server.

User Response: There be missing information in the subscription. Contact the DB2 Everyplace Sync Server administrator.

DSYC412E: Unexpected message format.

Explanation: The DB2 Everyplace Sync Client does not recognize some message information from the server during synchronization of a subscription.

User Response: Have the DB2 Everyplace Sync Server administrator reset the user and try again. If the problem persists, contact IBM software support.

DSYC413E: Requested target data not found.

Explanation: The DB2 Everyplace Sync Client cannot find the target data (for example, table not found) for synchronization.

User Response: The target data does not exist, which might have been removed by other applications. To restore the data of the subscription, reset the containing subscription set, and synchronize again. If the problem persists, contact IBM software support.

DSYC414E: Unexpected end of data.

Explanation: The DB2 Everyplace Sync Client has encountered unexpected missing or end-of-data during synchronization of a subscription.

User Response: Contact IBM software support.

DSYC415E: Data too long for corresponding field.

Explanation: The DB2 Everyplace Sync Client received data that was too large (for example, over-sized file). Data have been truncated.

User Response: The data from the DB2 Everyplace Sync Server be invalid. Check with the DB2 Everyplace Sync Server administrator.

DSYC417E: Server reported subscription not enabled for synchronization.

Explanation: The user is not enabled to synchronize the subscription.

User Response: Make sure that the username is entered correctly, and contact the DB2 Everyplace Sync Server administrator to make sure the user is enabled to synchronize the subscription, and try to synchronize again. If the problem persists, contact IBM software support.

DSYC418E: Server reported exceptions.

Explanation: The DB2 Everyplace Sync Server has encountered problems (or exceptions) synchronizing the subscription.

User Response: Contact the DB2 Everyplace Sync Server administrator to check the health of the DB2 Everyplace Sync Server or the status of the source data.

DSYC420E: Attempt to upload changes to read-only data.

Explanation: The client attempts to upload changes to the target data which is configured as read-only in the subscription definition.

User Response: Contact the DB2 Everyplace Sync Server administrator to check the permission to the target data in the subscription definition. If the subscribed data is configured read-only, the user shouldn't change the target data. To refresh the target data, reset the subscription set, and synchronize again. If the problem persists, contact IBM software support.

DSYC421E: Attempt to upload impermissible operations.

Explanation: The client attempts to upload changes to the target data, which are not permissible in the subscription definition.

User Response: Contact the DB2 Everyplace Sync Server administrator to check the permission to the target data in the subscription definition. The user shouldn't perform impermissible operations to the target data. To refresh the target data, reset the subscription set, and synchronize again. If the problem persists, contact IBM software support.

DSYC423E: Not authorized to access the target data.

Explanation: The user is not authorized to access the requested data.

User Response: Check that the username and password (to access the target data) are entered correctly, and that the user is authorized to access the requested data. Have the DB2 Everyplace Sync Server administrator reset the user and try to synchronize again. If the problem persists, contact the DB2 Everyplace Sync Server administrator.

DSYC424E: Requested data not available.

Explanation: The requested data is not available. The target data is being used by another application.

User Response: Check that the DB2 Everyplace Sync Client has permission to modify the target data, and that it is not being used by another application. Have the DB2 Everyplace Sync Server administrator reset the user and try again. If the problem persists, contact IBM software support.

DSYC425E: Requested data type is not supported.

Explanation: The subscription adapter does not support the requested data type. For example, you can only synchronize .prc and .pdb files to a Palm device and there are some columns types which are not supported by DB2 Everyplace.

User Response: Have the DB2 Everyplace Sync Server administrator verify that all the subscribed data are supported (for example, file type and column types), and try to synchronize again. If the problem persists, contact IBM software support.

DSYC426E: Invalid subscription target path.

Explanation: The user-provided subscription target path is either not a valid path name, or the path is not relative to the device target path.

User Response: Make sure the specified subscription target path is a valid path name and relative to the device target directory.

DSYC600E: Failed to open connection.

Explanation: The DB2 Everyplace Sync Client has a problem opening a connection with the DB2 Everyplace Sync Server because either you have a wrong server URL (Uniform Resource Locator), or the server is not up.

User Response: Check to ensure: 1)Server URL is correctly entered 2)The DB2 Everyplace Sync Server is currently operational. Try to synchronize again. If the problem persists, contact IBM software support.

DSYC601E: Failed to establish connection.

Explanation: The DB2 Everyplace Sync Client has a problem opening connection with the DB2 Everyplace Sync Server. This could be due to: 1)Wrong server URL (Uniform Resource Locator) 2)Server is not up 3)Network is busy 4)Network connection is not operational.

User Response: Check to ensure: 1)Server URL is correctly entered 2)The DB2 Everyplace Sync Client has access to the DB2 Everyplace Sync Server through either a serial or USB cable, a modem, or a network connection. 3)The DB2 Everyplace Sync Server is currently operational. 4)If using Windows RAS, or some PPP connection software (for example, Mocha PPP for the serial port, Softick PPP for the USB port), make sure the PPP connection between the mobile device and the desktop computer is established. 5)Try to synchronize again. If the problem persists, contact IBM software support.

DSYC602E: Failed to send request.

Explanation: The DB2 Everyplace Sync Client has successfully connected to the DB2 Everyplace Sync Server, but has a problem sending requests to the server. The causes might be: 1)Server is not up 2)Network connection is lost.

User Response: Check to ensure: 1)The DB2 Everyplace Sync Client has access to the DB2 Everyplace Sync Server through either a serial or USB cable, a modem, or a network connection. 2)The DB2 Everyplace Sync Server is currently operational. 3)If using Windows RAS, or some PPP connection software (for example, Mocha PPP for the serial port, Softick PPP for the USB port), make sure the PPP connection between the mobile device and the desktop computer is established. 4)Try to synchronize again. If the problem persists, contact IBM software support.

DSYC603E: Failed to receive reply.

Explanation: The DB2 Everyplace Sync Client has successfully connected to the DB2 Everyplace Sync Server, but has a problem receiving messages from the server, or the received message is corrupted or in an unexpected format. The causes could be: 1)Server is not up 2)Network connection is lost.

User Response: Check to ensure: 1)The DB2 Everyplace Sync Server is currently operational. 2)If using Windows RAS, or some PPP connection software (for example, Mocha PPP for the serial

port, Softick PPP for the USB port), make sure the PPP connection between the mobile device and the desktop computer is established. 3)Try to synchronize again. If the problem persists, contact IBM software support.

DSYC604E: Timeout while receiving reply.

Explanation: The DB2 Everyplace Sync Client has successfully connected sent a request to the DB2 Everyplace Sync Server, but timed out while receiving the server message. Either the server needs more time preparing the acquired information, the network is busy, the server is not up, or the network connection is lost.

User Response: Check to ensure: 1)The DB2 Everyplace Sync Server is currently operational. 2)If using Windows RAS, or some PPP connection software (for example, Mocha PPP for the serial port, Softick PPP for the USB port), make sure the PPP connection between the mobile device and the desktop computer is established. 3)Length the network timeout. 4)Try to synchronize again. If the problem persists, contact IBM software support.

DSYC605E: Failed to receive acknowledgment.

Explanation: The DB2 Everyplace Sync Client has successfully sent a request and received a reply from the DB2 Everyplace Sync Server, but does has not received an acknowledgement from the server. Either the server needs more time preparing the inquired information, the network is busy, the server is not up, or the network connection is lost.

User Response: Check to ensure: 1)The DB2 Everyplace Sync Server is currently operational. 2)If using Windows RAS, or some PPP connection software (for example, Mocha PPP for the serial port, Softick PPP for the USB port), make sure the PPP connection between the mobile device and the desktop computer is established. 3)Try to synchronize again. If the problem persists, contact IBM software support.

DSYC606E: Failed to open the Network library.

Explanation: The DB2 Everyplace Sync Client has a problem opening the Network library.

User Response: Check to ensure: 1) The Network library is present. 2) If using the PalmOS emulator, make sure the check box under Settings->Properties is set (checked). 3) Try a soft reset on the device, and synchronize again. If the problem persists, contact IBM software support.

DSYC608E: Failed to close the Network library.

Explanation: The DB2 Everyplace Sync Client has a problem closing the Network library.

User Response: The Network library be in a corrupted state. Do a soft reset on the device, and then synchronize again. If the problem persists, contact IBM software support.

DSYC609E: Failed to resolve hostname.

Explanation: The DB2 Everyplace Sync Client cannot resolve the IP for the provided hostname in the server URL (Uniform Resource Locator).

User Response: Make sure the server hostname is correctly specified.

DSYC610E: Out of memory.

Explanation: The DB2 Everyplace Sync Client could not allocate sufficient memory to complete synchronization.

User Response: The device could be low on available memory or the dynamic heap memory. Close out or delete some unused applications, and try again. If the problem persists, consult the DB2 Everyplace documentation for system requirements or contact IBM software support.

DSYC611E: Forbidden to synchronize to the server.

Explanation: The user is not allowed to synchronize to the server.

User Response: Contact the DB2 Everyplace Sync Server administrator to check if the user is authorized to synchronize to the DB2 Everyplace Sync Server.

DSYC612E: Server not found.

Explanation: The server cannot be found because either a wrong server URL (Uniform Resource Locator) is being used, the server is not up, or the DB2 Everyplace Sync Server is not installed properly.

User Response: Check to ensure: 1)Server URL (Uniform Resource Locator) is correctly entered; 2)The DB2 Everyplace Sync Server is currently operational; 3)The DB2 Everyplace Sync Server is installed properly. Try to synchronize again. If the problem persists, contact the DB2 Everyplace Sync Server administrator.

DSYC613E: Internal DB2 Everyplace Sync Server error.

Explanation: An internal error has occurred to the DB2 Everyplace Sync Server.

User Response: Contact the DB2 Everyplace Sync Server administrator to check the health of the DB2 Everyplace Sync Server. Restart the DB2 Everyplace Sync Server and try to synchronize again. If the problem persists, contact IBM software support.

DSYC614E: Server not responding.

Explanation: No servers available to service the client requests. All servers are either busy or blocked.

User Response: Contact the DB2 Everyplace Sync Server administrator to check the health and the availability of the DB2 Everyplace Sync Server. Try to synchronize again when the server is less busy. If the problem persists, contact IBM software support.

DSYC615E: Transport protocol not supported

Explanation: The transport protocol specified in the server URL (Uniform Resource Locator) is not supported.

User Response: Make sure the protocol specified in the server URL is supported. Currently, the only supported protocol is HTTP (HyperText Transfer Protocol). Use HTTPS if both DB2 Everyplace Sync Server and DB2 Everyplace Sync Client have been configured for SSL (Secure Socket Layer).

DSYC616E: Server busy

Explanation: No servers available to service the client requests. All servers are busy.

User Response: Contact the DB2 Everyplace Sync Server administrator to check the availability of the DB2 Everyplace Sync Server. Try to synchronize again when the server is less busy. If the problem persists, contact IBM software support.

DSYC698E: Internal error or inconsistent state.

Explanation: The subscription adapter encountered errors or an inconsistent state the target data manager during synchronization.

User Response: Check the DB2 Everyplace Sync Client configuration log, and identify the subscription type (config, table, or file). If it is a table subscription, consult the DB2 Everyplace Application Development Guide with the provided SQL state in the error message. Contact the DB2 Everyplace Sync Server administrator regarding the error state. If the problem persists, contact IBM software support.

DSYC699E: Unknown network error.

Explanation: The DB2 Everyplace Sync Client has encountered an unknown error while communicating with the DB2 Everyplace Sync Server.

User Response: Contact IBM technical support.

DSYD006E: The MDSS session monitor encountered an exception from the SQL statement: *SQL statement*.

Explanation: The DB2 Everyplace Sync Server records information about the state of each synchronization session in session monitoring tables in the administration control database, DSYCTLDB. The DB2 Everyplace Sync Server issues an SQL statement to add an entry for each new session so that session state information is persistent. The SQL statement failed because the session monitor tables could not be accessed.

User Response: Ensure that the DSYCTLDB database is functional and the storage is not exhausted. If no problems are found, contact IBM Software Support.

DSYD007E: The MDSS connection pool encountered the exception: *exception details*.

Explanation: The DB2 Everyplace Sync Server creates a pool of database connections for each database accessed. In this case, the DB2 Everyplace Sync Server attemped to use a connection from this pool for the named database failed. A possible explanation for this error is because the named database is not functional.

User Response: Ensure that the named database is functional. If no problems are found, contact IBM Software Support.

DSYD010E: The DB2 adapter failed to generate a DataPropagator password file for the Apply qualifier *apply qualifier*.

Explanation: Each time you start the DB2 Everyplace Sync Server, a password file is generated for each Apply qualifier. The DB2 Data Propagator uses the user ID and password in this file to access the source database. Synchronization fails for all subscriptions whose Apply qualifier does not have a corresponding password file. File creation fails because of inadequate storage in the directory where the DB2 Everyplace Sync Server is running.

User Response: First, ensure that adequate storage is available in the directory where the DB2 Everyplace Sync Server is running. Then, stop and restart the DB2 Everyplace Sync Server to attempt to generate the password file. If problems persist, contact IBM Software Support.

DSYD011E: The DB2 adapter was unable to create the tables required for synchronizing table *schema name.table name* **in database** *database name*.

Explanation: The DB2 Everyplace Sync Server creates staging tables for each mirror table involved in synchronization. These tables are created when you create a subscription against the database that corresponds to the mirror referenced in the message. Inadequate storage or a non-functional database causes the creation of these tables to fail.

User Response: Ensure that the database referenced in the message is functional and that adequate storage is available. Open the Mobile Devices Administration Center and remove and recreate the subscriptions for the referenced database to attempt to create the staging tables again. If you continue to receive this message, contact IBM Software Support.

DSYD012E: The DB2 adapter was unable to drop the tables required for synchronizing table *schema name.table name* **in database** *database name*.

Explanation: To manage synchronization, the DB2 Everyplace Sync Server creates multiple tables in association with a given mirror database. When you delete subscription sets associated with the mirror in the named database, the DB2 Everyplace Sync Server drops the tables associated with the deleted subscription set. This drop operation might fail if the database is not functional.

User Response: Ensure that the database named in the message is functional.

DSYD014E: The DB2 adapter was unable to access the synchronization mirror table *schema name.table name* **in database** *database name*.

Explanation: An error occurred while attempting to access the mirror table.

User Response: Ensure that you are connected to the mirror database named in the message and that the database has adequate storage capacity. If problems persist, contact IBM Software Support.

DSYD015E: The DB2 adapter was unable to access the synchronization tables peripheral to mirror table *schema name.table name* **in database** *database name***.**

Explanation: To manage synchronization, the DB2 Everyplace Sync Server creates multiple tables in association with a given mirror table. An error occurred when the DB2 Everyplace Sync Server attempted to access these tables for the mirror table referenced in the message.

User Response: Ensure that the database named in the message is functional. If problems persist, contact IBM Software Support.

DSYD018E: The DB2 adapter was unable to access the ASN.IBMSNAP_APPLYTRAIL table in database *database name***.**

Explanation: An error occurred while attempting to access the DataPropagator apply trail table.

User Response: Ensure that the database named in the message is functional, and that the DB2 Data Propagator subscription was properly set up. In addition, see the DB2 Replication Guide and Reference for more information about why the Apply Trail table could not be accessed. If problems persist, contact IBM Software Support.

DSYD019E: The DB2 adapter was unable to access the ASN.IBMSNAP_UOW table in database *database name***.**

Explanation: An error occurred while attempting to access the DataPropagator unit-of-work table.

User Response: Ensure that the database named in the message is functional, and that the Data Propagator subscription was properly set up. Additionally, see the DB2 Replication Guide and Reference for more information on why the unit-of-work table could not be accessed. If problems persist, contact IBM Software Support.

DSYD022E: The DB2 adapter was unable to access the change data table *schema.table name* **in database** *database name***.**

Explanation: An error occurred while attempting to access the Data Propagator change data table.

User Response: Ensure that the database named in the message is functional, and that the subscription is properly set up in the Mobile Devices Administration Center. If problems persist, contact IBM Software Support.

DSYD027E: The DataPropagator Apply process failed for database *database name***, Apply qualifier** *apply qualifier***.**

Explanation: Synchronization of this database is not possible until the DataPropagator Apply error is addressed.

User Response: See the DB2 Replication Guide and Reference for information about why the Apply program failed. If problems persist, contact IBM Software Support.

DSYD028I: A synchronization element for database *database name***, table** *schema name.table name***, primary key** *primary key value***, from device** *device id* **was rejected due to** *reject code***.**

Explanation: The synchronization element for the relational database row specified was not accepted for synchronization for the reason given.

User Response: No action required.

DSYD029W: The DB2 Everyplace Sync Server detected a syntax error in the user WHERE clause of the mirror table *full table name* **in database** *database name***.** *Message from parser WHERE clause*

Explanation: The DB2 Everyplace Sync Server parses the WHERE clause to obtain the information to handle WHERE clauses that involve multiple tables. If the WHERE clause refers to only one table, this warning can be safely ignored.

User Response: Correct the WHERE clause in the Mobile Devices Administration Center. If problems persist, contact IBM Software Support.

DSYD030I: Usage: com.ibm.mobileservices.adapter.rdb.Replicate mirror_database_name

Explanation: com.ibm.mobileservices.adapter.rdb.Replicate was not executed with one command-line argument.

User Response: Run com.ibm.mobileservices.adapter.rdb.Replicate with a mirror database name as the command-line argument.

DSYD031I: Replication of mirror database *mirror database name* **succeeded.**

Explanation: Replication for the mirror database was successful.

User Response: No action is required.

DSYD032E: Look-up table *table name* **is not found in the subscription** *subscription name***, but is mentioned in the mirror filter of** *domain table name***.**

Explanation: The filter makes reference to a table that does not exist in Subscription.

User Response: Correct the mirror filter in the Mobile Device Administration Center to make sure the look-up table specified in the filter is part of the subscription definition and try the operation again.

DSYD033E: The DB2 adapter received a request for an invalid subscription ID *subscription id* **for device type** *device type***, device ID** *device id***, and user** *user name***.**

Explanation: The DB2 adapter received a request for an invalid subscription from the specified user and device. The specified subscription ID no longer exists.

User Response: Reset the users device and try again.

DSYD035E: Replication of *mirror database* **is blocked:** *reason***.**

Explanation: The DB2 Everyplace Sync Server cannot ensure that it is safe to perform the replication because it cannot commnicate with some synchronization-enabled servers.

User Response: Check the log folder of the Mobile Device Administration Center. Look for DSYD038E to find out which server is unreachable. If the unreachable server is down, mark it as offline. If the server is up, stop it and restart it.

DSYD037E: No table subscription is defined for *mirror database***.**

Explanation: No table subscription is defined for the specified mirror database.

User Response: Create a table subscription using the specified mirror database name before requesting replication.

DSYD038E: The last three attempts to communicate with server *server* **has resulted in communication failure.**

Explanation: The server is either down, unreachable, or there are unknown communication problems.

User Response: Check that the server is up and reachable on the network.

DSYD039E: The data filter *data filter name* **for column** *column name* **in table** *table name* **is undefined in group** *group name* **or user** *user name* **.**

Explanation: A required data filter is undefined for an unsubscribed column.

User Response: Ensure that the parameters of vertical filtering are spelled correctly and defined by the group or user. Make corrections in the Subscription, Group, and/or User folders in the Mobile Device Administration Center.

DSYD040E: The migration of DB2 Everyplace is not complete.

Explanation: The migration phase of the configuration of DB2 Everyplace failed.

User Response: Run the DSYconfig utility to complete the migration of DB2 Everyplace. This utility migrates all the DB2 Everyplace Sync Server internal control tables to the current version.

DSYD041E: The mirror filter of *table name* **in** *subscription name* **is invalid. Error:** *error***. Filter:** *filter*

Explanation: The filter has one of the following problems: 1. The filter is syntactically incorrect. 2. A value for a parameter in the filter is incorrect. 3. The DB2 Everyplace Sync Server cannot identify the tables referenced in the filter.

User Response: Check the filter with the following steps: 1. Check the filter syntax with the DB2 SQL Reference. If the source database is DB2 and all the target table and column names are identical to their corresponding names in the source, you can run the filter on the source to validate it. 2. Check the default values of the parameters at the group and user levels. 3. If neither of the above checking shows any error, modify the filter by adding a pound sign (#) in front of each table referenced in the filter.

DSYD042E: The following mirror databases have not completed replication: *failed databases*

Explanation: The product cannot be upgraded until all mirror databases have completed replication.

User Response: Replicate the specified mirror databases before starting installation.

DSYD043E: The following DB2 Everyplace Sync Servers are running: *running servers*

Explanation: The product cannot be upgraded until all DB2 Everyplace Sync Servers have been shut down.

User Response: Shut down all DB2 Everyplace Sync Servers before starting installation.

DSYD044E: The following mirror databases have not completed the internal staging operation: *failed databases*

Explanation: The product cannot be upgraded until all mirror databases have completed the internal staging operation.

User Response: Block all subscriptions. Start the DB2 Everyplace Sync Server. Wait until there is no activity. Shut down the DB2 Everyplace Sync Server. Start installation again.

DSYD045E: The following tables must be migrated to a tablespace with a large enough page size: *failed tables*

Explanation: The product cannot be upgraded until the specified tables have been migrated to a tablespace with a large enough page size.

User Response: Please search for DSYD045E in the Installation Notes.

DSYD046W: Before installing DB2 Everyplace version 8.2, all users that have data changes on the device should perform synchronization. Otherwise, if the DB2 Everyplace Sync Client program has not been upgraded to version 8.2, these data changes on the device might be lost in the first synchronization session after this installation.

Explanation: Some users might be forced to perform a refresh by the DB2 Everyplace Sync Server in the first synchronization session after this installation. Only DB2 Everyplace Sync Client version 8.2 or above can perform a refresh without potential data loss.

User Response: If users have important data on the device that is not synchronized, we recommend that you do not install. Ask the users to synchronize their data and then start the installation again.

DSYD047E: Migration failed because *reason***.**

Explanation: Migration failed because of the specified reason.

User Response: Correct the problem. Then, run dsymigration.

DSYD048E: Mirror database name is not specified.

Explanation: A mirror database name is required to perform the operation.

User Response: Provide the mirror database name as input to the requested operation.

DSYD049I: The replication process is preparing to replicate the tables in the database *database name***.**

Explanation: The replication process is preparing to replicate the tables in the specified database.

User Response: No action is required.

DSYD050I: The replication process is replicating the tables in the database *database name***.**

Explanation: The replication process is replicating the tables in the specified database.

User Response: No action is required.

DSYD051I: The replication process is replicating data from the database *database name* **to the database** *database name***.**

Explanation: The replication process is replicating data from the first database specified to the second database specified.

User Response: No action is required.

DSYD052I: The replication process is replicating data from *table name* **in** *database name* **to** *table name* **in** *database name***.**

Explanation: The replication process is replicating data from the first table specified to the second table specified.

User Response: No action is required.

DSYD053I: The replication process is replicating the row with primary key *primary key value* **from** *table name* **in** *database name* **to** *table name* **in** *database name***.**

Explanation: The replication process is replicating the specified row from the first table specified to the second table specified.

User Response: No action is required.

DSYD054I: The replication process is finished replicating *x* **rows from** *table name* **in** *database name* **to** *table name* **in** *database name***.**

Explanation: The replication process is finished replicating the specified number of rows from the first table specified to the second table specified.

User Response: No action is required.

DSYD055I: The replication process is pruning the control tables associated with *table name* **in** *database name***.**

Explanation: The replication process is pruning the control tables associated with the specified table in the specified database.

User Response: No action is required.

DSYD056I: The replication process is finished pruning *x* **rows from the control tables associated with** *table name* **in** *database name***.**

Explanation: The replication process is finished pruning the specified number of rows from the control tables associated with the specified table in the specified database.

User Response: No action is required.

DSYD057I: The replication process is finished replicating the tables in the database *database name***.**

Explanation: The replication process is finished replicating the tables in the specified database.

User Response: No action is required.

DSYD058I: The replication process is finalizing the replication of the tables in the database *database name***.**

Explanation: The replication process is finalizing the replication of the tables in the specified database.

User Response: No action is required.

DSYD059I: The replication process is recovering the previous replication of the database *database name* **(state =** *state***).**

Explanation: The previous replication of the specified database failed. The current replication is performing recovery starting from the specified state.

User Response: No action is required.

DSYF000E: MDSS encountered an exception *exception details***.**

Explanation: The MDSS Servlet encountered an exception.

User Response: Gather trace and log files, and contact IBM Software Support for the specific adapter.

DSYG001E: An unexpected exception occurred: *exception***.**

Explanation: DB2 Everyplace Sync Server encountered the specified unexpected exception.

User Response: Report the error to your administrator.

DSYG003E: Unable to connect to database *database name***.**

Explanation: A connection to the specified database could not be established.

User Response: Verify that the database name, driver, user ID and password is valid and that it is operational.

DSYG004I: Connection to database *database name* **was successful.**

Explanation: A connection to the specified database was successfully established.

User Response: No action required.

DSYG005E: A valid license for this product installation was not found.

Explanation: A valid license must be installed for this product.

User Response: Please contact IBM for information on purchasing this product.

DSYG006I: The Evaluation License for this product installation expires in *remaining days* **days.**

Explanation: This product installation has been licensed for evaluation purposes only, and will cease to operate in the number of days reported.

User Response: Please contact IBM for information on purchasing this product to avoid interruption in service.

DSYG007I: Unable to open log file, exception: *exception***.**

Explanation: The DB2 Everyplace Sync Server was unable to open the log file because the specified exception was encountered.

User Response: Verify that the DSYGdflt.properties file, Trace.Path variable specifies a valid path and that the current user has the necessary create/read/write permissions.

DSYG008I: Unable to write to log file *log file name***, exception:** *exception***.**

Explanation: The DB2 Everyplace Sync Server was unable to write to the specified log file because the specified exception was encountered.

User Response: Verify that the DSYGdflt.properties file, Trace.Path variable specifies a valid path and that the current user has the necessary create/read/write permissions.

DSYG010E: The DSYCTLDB control database is incompatible with this DB2 Everyplace Sync Server installation: *level*

Explanation: This DB2 Everyplace Sync Server requires that the installation be at the same level as the DSYCTLDB control database.

User Response: If you are upgrading or installing a new DB2 Everyplace Sync Server, the DSYCTLDB control database needs to be installed or upgraded to the same level using the install package. If the DSYCTLDB database has been upgraded, then this DB2 Everyplace Sync Server installation needs to be upgraded to the same level. If neither of these actions applies, or is unsuccessful in resolving the problem, please contact IBM Software Support.

DSYG011E: Unable to read file *filename***.**

Explanation: The specified file could not be read because the file does not exist or the permissions prohibit this action.

User Response: Verify that the specified file exists and that it is accessible.

DSYG013E: Could not load the CustomLogicPolicy subclass *classname***.**

Explanation: A class designated by the DB2 Everyplace Administrator to be an implementation of a CustomLogicPolicy could not be loaded.

User Response: Ensure that the specified CustomLogicPolicy subclass is accessible by all SyncServer instances, by any standalone dsyreplicate process, and by the Mobile Devices Administration Center and XML Scripting Tool. If this is unsuccessful in resolving the problem, please contact IBM Software Support.

DSYG014I: Blocking mirror database *database name***.**

Explanation: Requesting the servers to stop servicing synchronization and replication requests for this mirror database.

User Response: No action required.

DSYG015I: Updating control database.

Explanation: Applying the requested changes in the control database.

User Response: No action required.

DSYG016I: Unblocking mirror database *database name***.**

Explanation: Requesting the servers to continue servicing synchronization and replication requests for this mirror database.

User Response: No action required.

DSYG017E: Cannot edit subscription because there was an error loading the subscription details: *error message*

Explanation: Subscription was not fully initialized earlier and editing at this time is not possible because subscription details are not known.

User Response: Make sure that source and if required mirror databases are accessible and their contents are valid. If there is connection problem, check to make sure URL is correct and drivers are on CLASSPATH. The subscription relies on information in the source and the mirror to fully define itself. When source and/or mirror database is down, actions that can be applied to the subscription are limited.

DSYG018I: Replicating mirror database *database name***.**

Explanation: Replicating the mirror database to bring it in sync with the source.

User Response: No action required.

DSYG019I: There are no changes to apply to control database for subscription *subscription name***.**

Explanation: No changes will be made in the control database for this operation.

User Response: No action required.

DSYG020I: Creating triggers.

Explanation: This is a progress message.

User Response: No action required.

DSYG021I: Creating replication control tables.

Explanation: This is a progress message.

User Response: No action required.

DSYG022I: Setting up join filtering control structures.

Explanation: This is a progress message.

User Response: No action required.

DSYG023I: Requesting replication.

Explanation: This is a progress message.

User Response: No action required.

DSYG024I: Removing join filtering control structures.

Explanation: This is a progress message.

User Response: No action required.

DSYG025I: Removing replication control tables.

Explanation: This is a progress message.

User Response: No action required.

DSYG026I: Removing triggers.

Explanation: This is a progress message.

User Response: No action required.

DSYG027E: Connection to the database *database name* **failed therefore it is not possible to fully initialize the subscription** *subscription name***.**

Explanation: To be able to retrieve subscription details, access to the specified database is required but the database is either down or there is another connection problem.

User Response: Make sure the specified database is up and running and configured properly. Make sure JDBC Url is correct and JDBC drivers are on Java CLASSPATH.

DSYG028E: Table *table name* **is not found in database** *database name***.**

Explanation: Subscription uses a table that does not exist in the database or the database was not accesible at the time of this error and list of tables could not be retrieved.

User Response: Make sure the specified table exists in the database and verify that the specified database is up and running and configured properly for JDBC access.

DSYG029E: Could not load the tables of subscription *subscription name***. Root cause:** *exception message*

Explanation: Details of the subscription's tables were not retrieved due to an error. Actual cause is likely to be a source or a mirror database being down, or a bad database Url or a missing JDBC driver on CLASSPATH.

User Response: Verify that the source and mirror databases are up and running and configured properly for JDBC access. Make sure that the tables which the subscription references still exist.

DSYG030E: DB2 Everyplace Sync Server object *name of missing object* **is not defined.**

Explanation: An attempt was made to use an undefined object such as a database, subscription, subscription set, group, user or another type that is not yet defined in DB2 Everyplace Sync Server. This object needs to be defined using the Administration console or the XML Scripting tool before it can used.

User Response: Verify that the object referenced here is actually defined in DB2 Everyplace Sync Server configuration. Cause can be a typo, or misconfiguration.

DSYG031E: DB2 Everyplace Sync Server database *database name* **is not defined.**

Explanation: An attempt was made to use an undefined mirror or master database. This database needs to be defined using the Administration console or the XML Scripting tool before it can be used.

User Response: Verify that the database referenced here is actually defined in DB2 Everyplace Sync Server configuration. Cause can be a typo, or misconfiguration.

DSYG032E: The DB2 Everyplace Sync Server Context Root is not defined.

Explanation: The DB2 Everyplace Sync Server installation is incomplete as the Context Root is not defined.

User Response: Define a valid context root using command-line configuration tool.

DSYG036E: The executable code retrieved from *code URL* **has a build date of** *jar build date***, which is inconsistent with the control database build date** *db build date***.**

Explanation: The DB2 Everyplace Sync Server upgrade installation and configuration has not been completed.

User Response: Ensure that the DB2 Everyplace Sync Server upgrade installation and configuration has been completed for all components. Refer to the DB2e installation documentation for instructions on completing the install configuration.

DSYG037E: Unable to verify the consistency of build dates between DB2e executable code and the control database due to the following error: *exception***.**

Explanation: The DB2 Everyplace Sync Server executable code build date or control database build date could not be accessed.

User Response: Ensure that the DB2 Everyplace Sync Server upgrade installation and configuration has been completed for all components. Refer to the DB2e installation documentation for instructions on completing the install configuration.

DSYJ000E: The database adapter detected a non-insertion operation in a Put subscription: *operation***.**

Explanation: In a Put subscription, only insertion is allowed. Deletion and update are rejected.

User Response: No action is required.

DSYJ001E: A database error occurred. *the error message from database*

Explanation: A database error occurred. The error message is obtained from the database.

User Response: Refer to the documentation of your database to diagnose the problem. If the problem persists, contact your database administrator.

DSYJ002E: Invalid JDBC driver name *driver name***.**

Explanation: The MDSS is unable to determine which database the specified JDBC driver name is using.

User Response: Verify that the JDBC driver name is correct. If it is correct, add the JDBC driver to the file com\\ibm\\mobileservices\\DSYJdbcDriverList.properties.

DSYM000E: Unable to start the DB2 Everyplace Mobile Devices Administration Center because the required control database 'DSYCTLDB' does not exist or is invalid.

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not be started because the required control database 'DSYCTLDB' does not exist or is invalid.

User Response: Verify the existence of the control database 'DSYCTLDB.' Rerun the script to create the control database (such as dsyctldb.bat).

DSYM001E: Unable to locate ID for subscription set *subscription set name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate the specified subscription set.

User Response: Correct the error and try again.

DSYM006E: Invalid or duplicate subscription set name *subscription set name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription set because the name is invalid or another subscription set with the same name exists.

User Response: Specify a unique, valid subscription set name and try again.

DSYM007E: Update subscription set *subscription set name* **name failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription set because the name is invalid or it already exists.

User Response: Specify a unique, valid subscription set name and try again.

DSYM008E: Update subscription set *subscription set name* **description failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription set description because the description is invalid.

User Response: Specify a valid subscription set description and try again.

DSYM009E: Update subscription set *subscription set name* **signature failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription set signature because the signature is invalid.

User Response: Specify a valid subscription set signature and try again.

DSYM010E: Update subscription set *subscription set name* **subscriptions failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription set subscriptions. Possible reason: the subscription is invalid or does not exist.

User Response: Verify that the subscription is valid and try again.

DSYM011E: Update subscription set *subscription set name* **groups failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription set groups. Possible reason: the group is invalid or does not exist.

User Response: Verify that the group is valid and try again.

DSYM012E: Unable to locate ID for user *user name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate the specified user.

User Response: Correct the error and try again.

DSYM013E: Invalid or duplicate user name *user name*. **Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified user because the name is invalid or another user with the same name exists.

User Response: Specify a unique, valid user name and try again.

DSYM014E: Update user *user name* **name failed. Action canceled.**

Explanation: The DB2 Everywhere Mobile Devices Administration Center could not update the specified user because the name is invalid or it already exists.

User Response: Specify a unique, valid user name and try again.

DSYM015E: Update user *user name* **description failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified user description because the description is invalid.

User Response: Specify a valid user description and try again.

DSYM016E: Unable to associate group with user *user name*. **Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not associate a group with the specified user. Possible reason: the group is invalid or does not exist.

User Response: Verify that the group is valid and try again.

DSYM017E: Update user *user name* **password failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified user password because the password is invalid.

User Response: Specify a valid user password and try again.

DSYM018E: Update user *user name* **enable state failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified user enable state because the enable state is invalid.

User Response: Specify a valid user enable state and try again.

DSYM019E: Update user *user name* **data filters failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified user data filters. Possible reasons: the group level data filter is invalid or does not exist or the user data filter value is invalid.

User Response: Correct the error and try again.

DSYM020E: Unable to remove data filter *data filter name* **from the user. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not remove the specified data filter from the user. Possible reasons: the data filter is invalid or is no longer defined at the group level.

User Response: Correct the error and try again.

DSYM021E: Unable to remove data filter *data filter name* **from the group. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not remove the specified data filter from the group. Possible reason: the data filter is invalid or no longer exists.

User Response: Correct the error and try again.

DSYM023E: Unable to locate the ID for group *group name*. **Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate the specified group.

User Response: Correct the error and try again.

DSYM024E: Invalid or duplicate group name *group name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified group because the name is invalid or another group with the same name exists.

User Response: Specify a unique, valid group name and try again.

DSYM025E: Update group *group name* **name failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified group because the name is invalid or it already exists.

User Response: Specify a unique, valid group name and try again.

DSYM026E: Update group *group name* **description failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified group description because the description is invalid.

User Response: Specify a valid group description and try again.

DSYM028E: Update group *group name* **enable state failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified group enable state because the enable state is invalid.

User Response: Specify a valid group enable state and try again.

DSYM029E: Unable to associate user with group *group name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not associate a user with the specified group. Possible reason: the user is invalid or does not exist.

User Response: Verify that the user is valid and try again.

DSYM030E: Unable to associate subscription set with group *group name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not associate a subscription set with the specified group. Possible reason: the subscription set is invalid or does not exist.

User Response: Verify that the subscription set is valid and try again.

DSYM031E: Update group *group name* **data filters failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified group data filters. Possible reason: the group data filter value is invalid.

User Response: Correct the error and try again.

DSYM032E: Unable to locate ID for subscription *subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate the specified subscription.

User Response: Correct the error and try again.

DSYM040E: Update file subscription *file subscription name* **timestamp failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified file subscription timestamp because the timestamp is invalid.

User Response: Correct the error and try again.

DSYM042E: Update file subscription *file subscription name* **source failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified file subscription source because the source is invalid.

User Response: Specify a valid file subscription source and try again.

DSYM043E: Update file subscription *file subscription name* **target failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified file subscription target because the target is invalid.

User Response: Specify a valid file subscription target and try again.

DSYM045E: Invalid source database for table subscription *table subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected an invalid source database for the specified table subscription.

User Response: Verify that the source database is correct and in the {SYSTEM}.{INSTANCE}.{DATABASE} format.

DSYM046E: Invalid or duplicate table subscription name *table subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription because the name is invalid or another table subscription with the same name exists.

User Response: Specify a unique, valid table subscription name and try again.

DSYM047E: Invalid or missing DataPropagator subscription. Action canceled.

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate the associated DataPropagator subscription.

User Response: Correct the error and try again.

DSYM048E: Unable to create source replication properties for table subscription *table subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not create the source replication properties for the specified table subscription.

User Response: Correct the error and try again.

DSYM049E: Unable to create mirror replication properties for table subscription *table subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not create the specified table subscription mirror replication properties.

User Response: Correct the error and try again.

DSYM052E: Unable to locate source replication properties id for table subscription *table subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate the specified table subscription source replication properties.

User Response: Correct the error and try again.

DSYM053E: Unable to locate mirror replication properties id for table subscription *table subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate the specified table subscription mirror replication properties.

User Response: Correct the error and try again.

DSYM056E: Update table subscription *table subscription name* **source system failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription source system because the source system is invalid.

User Response: Specify a valid table subscription source system and try again.

DSYM057E: Update table subscription *table subscription name* **source instance failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription source instance because the source instance is invalid.

User Response: Specify a valid table subscription source instance and try again.

DSYM058E: Update table subscription *table subscription name* **source database failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription source database because the source database is invalid.

User Response: Specify a valid table subscription source database and try again.

DSYM059E: Update table subscription *table subscription name* **source user ID failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription source user ID because the source user ID is invalid.

User Response: Specify a valid table subscription source user ID and try again.

DSYM060E: Update table subscription *table subscription name* **source password failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription source password because the source password is invalid.

User Response: Specify a valid table subscription source password and try again.

DSYM061E: Update table subscription *table subscription name* **mirror database failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription mirror database because the mirror database is invalid.

User Response: Specify a valid table subscription mirror database and try again.

DSYM062E: Update table subscription *table subscription name* **mirror user ID failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription mirror user ID because the mirror user ID is invalid.

User Response: Specify a valid table subscription mirror user ID and try again.

DSYM063E: Update table subscription *table subscription name* **mirror password failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription mirror password because the mirror password is invalid.

User Response: Specify a valid table subscription mirror password and try again.

DSYM064E: Update table subscription *table subscription name* **mirror sync window failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription mirror sync window because the mirror sync window is invalid.

User Response: Specify a valid table subscription mirror sync window and try again.

DSYM066E: Update table subscription *table subscription name* **target database failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription target database because the target database is invalid.

User Response: Specify a valid table subscription target database and try again.

DSYM067E: Update table subscription *table subscription name* **Apply qualifier failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription Apply qualifier because the Apply qualifier is invalid.

User Response: Specify a valid table subscription Apply qualifier and try again.

DSYM070E: Update table subscription *table subscription name* **subtables failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription subtables because a subtable is invalid.

User Response: Specify a valid table subscription subtable and try again.

DSYM072E: Unable to create DataPropagator table manager control tables for table subscription *table subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not complete the changes to the specified subscription due to an exception.

User Response: Correct the error and try again.

DSYM073I: Unable to update DataPropagator table manager control tables for table subscription *table subscription name***.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not complete the changes to the specified subscription due to an exception.

User Response: No action required.

DSYM074E: The replication subscription using source table *source table name* **and target table** *target table name* **does not contain a target column with a primary key. Action canceled.**

Explanation: The specified replication subscription table is invalid because it does not contain a target column defined as a primary key.

User Response: Use the advanced subscription definition, dialog, target columns tab to select one or more target columns to be a primary key.

DSYM075E: The DB2 Everyplace Mobile Devices Administration Center encountered an unexpected exception: *exception***. Attempt to correct the error specified in the exception. If unable to do so, close the DB2 Everyplace Mobile Devices Administration Center and try again.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center encountered the specified exception. The exception can include additional information that can be used to identify the cause of the error and corrective steps.

User Response: Attempt to correct the error specified in the exception. If unable to do so, close the DB2 Everyplace Mobile Devices Administration Center and try again. If the error persists, contact the database administrator.

DSYM077E: User *user name* **password and verify password are not the same. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified user password because the password and verify password are not the same.

User Response: Specify a valid, matching password and verify password and try again.

DSYM079E: Table subscription *table subscription name* **source database password and verify password are not the same. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription source database password because the source database password and source database verify password are not the same.

User Response: Specify a valid, matching source database password and verify password and try again.

DSYM080E: Table subscription *table subscription name* **mirror database password and verify password are not the same. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription mirror database password because the mirror database password and mirror database verify password are not the same.

User Response: Specify a valid, matching mirror database password and verify password and try again.

DSYM082E: Duplicate target table names for table subscription *table subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected duplicate target table names for the specified table subscription. The target table names within a single table subscription must be unique.

User Response: Specify unique target table names and try again.

DSYM083E: Duplicate target table names for table subscriptions assigned to subscription set *subscription set name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected duplicate target table names for two or more table subscriptions assigned to the specified subscription set. The target table names of table subscriptions assigned to a subscription set must be unique.

User Response: Ensure that the table subscriptions assigned to the specified subscription set use unique target table names and try again.

DSYM084E: Duplicate target table names for table subscriptions assigned to subscription sets of the group *group name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected duplicate target table names for two or more table subscriptions in subscription sets assigned to the specified group. The target table names of table subscriptions assigned to subscription sets of a group must be unique.

User Response: Ensure that the table subscriptions assigned to the specified group use unique target table names and try again.

DSYM085E: Assignment of subscription set *subscription set name* **to one or more selected groups would result in duplicate table subscription target table names. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the assignment of the specified subscription set to the group(s) would result in duplicate target table names. The target table names of table subscriptions used by a group must be unique.

User Response: Ensure that the groups using the table subscriptions assigned in the specified subscription set use unique target table names and try again.

DSYM086E: Assignment of the the table subscription *table subscription name* **to one or more selected subscription sets would result in duplicate table subscription target table names. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the assignment of the specified subscription to the subscription set(s) would result in duplicate target table names. The target table names of table subscriptions used by a subscription set must be unique.

User Response: Ensure that the subscription sets using the specified table subscription use unique target table names and try again.

DSYM087E: Assignment of one or more subscription sets to group *group name* **would result in duplicate table subscription target table names. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the assignment of subscription set(s) to the specified group would result in duplicate target table names. The target table names of table subscriptions assigned to subscription sets of a group must be unique.

User Response: Ensure table subscriptions assigned to subscription sets for the specified group contain unique target table names and try again.

DSYM088E: Unable to delete the instance of a DSY default adapter *adapter name***. Action ignored.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected an attempt to delete the specified DSY default adapter. The specified adapter can not be deleted as it is required by the DB2 Everyplace Mobile Devices Administration Center.

User Response: Do not attempt to remove any DSY default adapter.

DSYM089E: Invalid or duplicate adapter name *adapter name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified adapter because the name is invalid or another adapter with the same name exists.

User Response: Specify a unique, valid adapter name and try again.

DSYM090E: Unable to locate ID for adapter *adapter name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate the specified adapter.

User Response: Correct the error and try again.

DSYM091E: Update adapter *adapter name* **name failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified adapter because the name is invalid or it already exists.

User Response: Specify a unique, valid adapter name and try again.

DSYM092E: Update adapter *adapter name* **description failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified adapter description because the description is invalid.

User Response: Specify a valid adapter description and try again.

DSYM093E: Update adapter *adapter name* **signature failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified adapter signature because the signature is invalid.

User Response: Specify a valid adapter signature and try again.

DSYM095E: Update adapter *adapter name* **communication attributes failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified adapter communication attributes because one or more communication attributes are invalid.

User Response: Specify valid communication attributes and try again.

DSYM096E: Update adapter *adapter name* **file attributes failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified adapter file attributes because one or more file attributes are invalid.

User Response: Specify valid file attributes and try again.

DSYM098E: Unable to associate adapter with subscription *subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not associate an adapter with the specified subscription. Possible reason: The adapter is invalid or does not exist.

User Response: Verify that the adapter exists and is valid and try again.

DSYM099E: Unable to disassociate adapter from subscription *subscription name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not disassociate an adapter from the specified subscription. Possible reason: The adapter is invalid or does not exist.

User Response: Verify that the adapter exists and is valid and try again.

DSYM100I: DataPropagator table subscription *table subscription name* **created successfully. Additional steps be required before the table subscription can be used for synchronization. Refer to the DB2 Everyplace Sync Server Administration Guide for more information.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center successfully created the specified table subscription. However, additional steps be required before the specified subscription can be used for synchronization.

User Response: Refer to the *DB2 Everyplace Sync Server Administration Guide* for further information.

DSYM102E: Unable to delete adapter *adapter name* **as one or more subscriptions are currently using it. Action ignored.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected an attempt to delete the specified adapter. The specified adapter can not be deleted as it is still being used by one or more subscriptions.

User Response: Reassign all subscriptions using the specified adapter to use different adapters and try again (or refresh the adapter object and try again).

DSYM104E: Unable to connect to database *database name* **using driver** *driver name*, **user ID** *user ID*.

Explanation: A connection to the specified database could not be established.

User Response: Verify that the database name, driver, user ID and/or password is valid and that it is operational.

DSYM115W: More than *maximum tables* **available tables were found in the master database** *master database name*. **Only the first** *maximum tables* **tables will be displayed. Use the Filter button to limit the available tables result set.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that more than the specified maximum available tables at the specified master.

User Response: If the desired table is not listed, use the Filter button to limit the result set.

DSYM122E: Update custom subscription *custom subscription name* **other failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified custom subscription other because the other is invalid.

User Response: Specify a valid custom subscription other and try again.

DSYM124E: Unable to load adapter *adapter name* **customizer** *customizer class name*. **Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not load the specified adapter customizer. Possible reason: the adapter communication attributes command is invalid, the class is not found or is not specified in the CLASSPATH environment variable.

User Response: Verify that the the adapter communication attributes command is valid and verify that the command class exists and that it is specified in the CLASSPATH environment variable.

DSYM125E: Unable to load adapter *adapter name* **customizer** *customizer class name*, **exception** *exception*. **Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not load the specified adapter customizer. Possible reason: the adapter communication attributes command is invalid, the class is not found or is not specified in the CLASSPATH environment variable.

User Response: Verify that the the adapter communication attributes command is valid and verify that the command class exists and that it is specified in the CLASSPATH environment variable.

DSYM127E: No valid custom adapters found. Action canceled.

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not allow the custom subscription action because no custom (non-DSY) adapters were found.

User Response: Define at least one custom adapter and try again.

DSYM128E: Unable to locate a local database which could be used as a mirror database.

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate any local database(s) which could be used as a mirror database. At least one local database must be defined for use as a mirror database.

User Response: Ensure at least one local database is defined and try again.

DSYM132E: The AS/400 source database *database name* **is not supported for JDBC table subscriptions.**

Explanation: The specified database name is not supported by the DB2 Everyplace Sync Server for synchronization using a JDBC table subscription. DB2/400 V5R1 or later is required for DB2 Everyplace Sync Server support for a AS/400 source database.

User Response: Use DB2/400 V5R1 or later for JDBC table subscription support. If using an earlier version of DB2/400, refer to the subscription type DataPropragator table subscription.

DSYM133E: Unable to complete custom subscription *subscription name* **as the custom adapter reported a failure while processing the save request. Action canceled.**

Explanation: The custom adapter reported that the request to save failed.

User Response: Refer to the documentation for the custom adapter.

DSYM134E: Update adapter *adapter name* **class name failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified adapter class name because the class name is invalid.

User Response: Specify a valid adapter class name and try again.

DSYM135E: Duplicate source and mirror databases *database name* **found.**

Explanation: The table subscription is attempting to use the specified database as both the source and the mirror. The source and mirror databases must be different.

User Response: Specify a different source and mirror database name and try again.

DSYM136E: Update subscription *subscription name* **encryption level failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription because the encryption level is invalid.

User Response: Specify a valid encryption level and try again.

DSYM137E: Table *table name* **does not contain a primary key. Action canceled.**

Explanation: No primary key was found in the specified table. A table must have at least one primary key defined in order to be used as a table subscription.

User Response: Select a table with a primary key and try again.

DSYM138E: Update table subscription *table subscription name* **subcolumns failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified table subscription subcolumns because a subcolumn is invalid.

User Response: Specify a valid table subscription subcolumn and try again.

DSYM141E: Invalid or duplicate subscription name *subscription name*. **Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription because the name is invalid or another subscription with the same name exists.

User Response: Specify a unique, valid subscription name and try again.

DSYM142E: Unable to create subscription *subscription name*. **Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not create the specified subscription.

User Response: Correct the error and try again.

DSYM143E: Unable to locate ID for subscription *subscription name*. **Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate the specified subscription.

User Response: Correct the error and try again.

DSYM144E: Update subscription *subscription name* **name failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription because the name is invalid or it already exists.

User Response: Specify a unique, valid subscription name and try again.

DSYM145E: Update subscription *subscription name* **description failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription description because the description is invalid.

User Response: Specify a valid subscription description and try again.

DSYM146E: Update subscription *subscription adapter* **adapter failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not update the specified subscription because the adapter is invalid.

User Response: Specify a valid subscription adapter and try again.

DSYM149W: One or more user defined indexes were removed as they will result in duplicate default index(es) using primary keys.

Explanation: The DB2 Everyplace Mobile Devices Administration Center automatically removed one or more user defined indexes as they were duplicates of primary key indexes.

User Response: No action required.

DSYM150I: One or more imported indexes were removed as they would result in duplicate default index(es) using primary keys.

Explanation: The DB2 Everyplace Mobile Devices Administration Center automatically removed one or more imported indexes as they were duplicates of primary key indexes.

User Response: No action required.

DSYM153E: Unable to allow table changes to subscription *subscription name* **as the connection to database** *database name* **using driver** *driver name* **and user ID** *user ID* **failed. The define subscription button is disabled.**

Explanation: The define subscription button has been disabled because a connection to the specified database could not be established. Typically, this is the result of the password being changed.

User Response: Verify that the database name, driver, user ID and or password is valid and that it is operational by using the test connection button and apply any changes by clicking the OK button. After the changes are applied, try again.

DSYM155E: Unable to use mirror *mirror database name* **for source** *source database name* **as the mirror is already used by another subscription with a different source. Two different source databases not share the same mirror database.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the specified mirror was already in use by another subscription. Source databases not share the same mirror database.

User Response: Change the mirror database name and try again.

DSYM156E: Unable to determine if any DB2 Everyplace devices are installed. One or more DB2 Everyplace device types must be installed in order to create or modify a subscription of this type. Check your DSYIdflt.properties file and ensure that you have a DB2e.InstalledDeviceTypes entry or rerun the DB2 Everyplace installation program.

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not determine which DB2 Everyplace devices were installed. One or more DB2 Everyplace device types must be installed in order to create or modify a subscription of this type.

User Response: Check your DSYIdflt.properties file and ensure that you have a DB2e.InstalledDeviceTypes entry or rerun the DB2 Everyplace installation program.

DSYM157E: One or more DB2 Everyplace Sync Server servers were unable to block subscription *subscription name***. Action canceled.**

Explanation: One or more DB2 Everyplace Sync Server servers were unable to block the specified subscription. A subscription must be blocked on all alive servers in order to process the edit request.

User Response: The administrator should perform the following steps to ensure that the attempted action will succeed. 1. Ensure that all servers (or the server in a single server environment) are not currently running. 2. Ensure that in the MDAC Server view, all entries on the right pane show that each server is marked as offline. If they are not, right-click on each server and select ″Mark as offline″. 3. If the servlet engine listens for http requests on a port other than 8080, be sure that the property Server.Port is set to the corresponding value in the DSYGdflt.properties file. After adding this value, delete the server in the MDAC Server view if there is an entry with an incorrect value. Be aware that this port must be accessible from any machine that is running MDAC or MDAC scripting. 4. If more than one server is running on this machine be sure that Server.Name is defined to something unique in the DSYGdflt.properties file. After defining this value, delete the server in the MDAC Server view if there is an entry with an incorrect value. 5. If there is more than one IP address that the server is on, ensure that the one the servlet engine is listening on is defined in DSYGdflt.properties by the Server.IP variable. Again, delete any entries in the MDAC Server view if they contain an incorrect value. 6. To prevent the error in the future, be sure that all servers are shut down by a supported shutdown method, as opposed to killing the process or JVM. In the case of a failure that prevents proper shutdown, open MDAC and mark the servers that are no longer running as offline as describe above. After verifying all of the above, the server be restarted and/or the action be attempted again.

DSYM158E: Unable to connect to the LDAP server *WEBSERVICE_SOAP_ROUTER* **using login** *user name***.\n\nSpecify a valid user name and/or password and try again or modify the DSYLDAP.properties to change the LDAP default connection information.\n\nLDAP exception:** *exception***.**

Explanation: Unable to connect to the specified LDAP server using the specified user name and password.

User Response: Specify a valid user name and/or password and try again or modify the DSYLDAP.properties to change the LDAP default connection information.

DSYM159E: Unable to determine the LDAP server to connect to as no WEBSERVICE_SOAP_ROUTER was specified in the file com/ibm/mobileserivces/DSYLDAP.properties. Specify a valid WEBSERIVCE_SOAP_ROUTER in the com/ibm/mobileservices/DSYLDAP.properties file and try again.

Explanation: Unable to determine the LDAP server to connect to as no WEBSERVICE_SOAP_ROUTER was specified in the file com/ibm/mobileservices/ DSYLDAP.properties.

User Response: Specify a valid WEBSERIVCE_SOAP_ROUTER in the com/ibm/mobileservices/ DSYLDAP.properties file and try again.

DSYM160W: No groups with *DB2e group name prefix* **name prefix were found in the LDAP server** *WEBSERVICE_SOAP_ROUTER***.**

Explanation: No groups with DB2e* prefix (prefix is configurable, it is DB2e by default) were found at the specified LDAP server.

User Response: Define at least one DB2e* group.

DSYM161W: No users were found in the sync group *SYNCGROUP***, LDAP server** *WEBSERVICE_SOAP_ROUTER***.**

Explanation: No users were found in the specified sync group.

User Response: Specify a sync group containing one or more users and try again.

DSYM162W: One or more DB2 Everyplace installed device types do not support table encryption. Table data on these devices will not be encrypted.

Explanation: The DB2 Everyplace Mobile Devices Administration Center has detected that you have requested that the table be encrypted, but one or more DB2 Everyplace installed device types do not support encryption. Thus, table data on these unsupported encryption devices will not be encrypted.

User Response: No action required.

DSYM163E: Table *table name* **contains a primary key column with a length exceeding 255 or the total length of all primary key columns exceeds 1024. Action canceled.**

Explanation: DB2 Everyplace requires primary key column length not exceed 255 characters. DB2 Everyplace requires that the total length of all primary key columns not exceed 1024 characters.

User Response: Select a valid table and try again.

DSYM165E: One or more DB2 Everyplace Sync Server servers were not offline. All DB2 Everyplace Sync Servers must be marked as offline to create a DataPropagator subscription. Action canceled.

Explanation: One or more DB2 Everyplace Sync Server servers were not offline. All DB2 Everyplace Sync Servers must be marked as offline to create a DataPropagator subscription.

User Response: Mark all servers as offline and try again.

DSYM166W: The DataPropagator subscription *subscription name* **is currently read-only. The subscription be viewed, but any changes made will be ignored because one or more DB2 Everyplace Sync Server servers were not offline. All DB2 Everyplace Sync Servers must be marked as offline to edit a DataPropagator subscription.**

Explanation: One or more DB2 Everyplace Sync Server servers were not offline. All DB2 Everyplace Sync Servers must be marked as offline to edit a DataPropagator subscription.

User Response: Mark all servers as offline and try again.

DSYM167E: Source table *source table name* **has a syntax error in the source to mirror where clause text:** *where clause text*. **Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected a syntax error in the source to mirror (all rows needed) where clause in the specified table.

User Response: Validate the source to mirror (all rows needed) where clause for the specified table and try again.

DSYM168I: User *user name*, **device id** *device id*, **device type** *device type* **was reset.**

Explanation: The specified users' device was reset.

User Response: No action required.

DSYM169E: Update subscription *subscription name* **custom logic policy type** *policy type* **failed. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center can not update the specified subscription custom logic policy as the specified policy is invalid.

User Response: Specify a valid policy type and try again.

DSYM170E: One or more column names in table *table name* **exceeds** *maximum column name length* **characters.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not add the specified table as one or more column name lengths exeeceded the specified maximum column name length.

User Response: Specify a table with valid column name lengths and try again.

DSYM171E: DB2 Everyplace Mobile Devices Administration Center could not communicate with one or more servers and did not perform replication on the mirror database *mirror database name*, **reason:** *exception*. **Action canceled.**

Explanation: DB2 Everyplace Mobile Devices Administration Center could not communicate with one or more servers and did not perform replication on the specified mirror database for the specified exception. To prevent loss of data, the edit request was canceled.

User Response: The administrator should perform the following steps to ensure that the attempted action will succeed. 1. Ensure that all servers (or the server in a single server environment) are not currently running. 2. Ensure that in the MDAC Server view, all entries on the right pane show that each server is marked as offline. If they are not, right-click on each server and select "Mark as offline". 3. If the servlet engine listens for http requests on a port other than 8080, be sure that the property Server.Port is set to the corresponding value in the DSYGdflt.properties file. After adding this value, delete the server in the MDAC Server view if there is an entry with an incorrect value. Be aware that this port must be accessible from any machine that is running MDAC or MDAC scripting. 4. If more than one server is running on this machine be sure that Server.Name is defined to something unique in the DSYGdflt.properties file. After defining this value, delete the server in the MDAC Server view if there is an entry with an incorrect value. 5. If there is more than one IP address that the server is on, ensure that the one the servlet engine is listening on is defined in DSYGdflt.properties by the Server.IP variable. Again, delete any entries in the MDAC Server view if they contain an incorrect value. 6. To prevent the error in the future, be sure that all servers are shut down by a supported shutdown method, as opposed to killing the process or JVM. In the case of a failure that prevents proper shutdown, open MDAC and mark the servers that are no longer running as offline as describe above. After verifying all of the above, the server be restarted and/or the action be attempted again.

DSYM172E: Unable to delete subscription *subscription name*. **Action canceled.**

Explanation: DB2 Everyplace Mobile Devices Administration Center could not delete the specified subscription due to internal validations failing. To prevent loss of data, the delete request was canceled.

User Response: The administrator should perform the following steps to ensure that the attempted action will succeed. 1. Ensure that all servers (or the server in a single server environment) are not currently running. 2. Ensure that in the MDAC Server view, all entries on the right pane show that each server is marked as offline. If they are not, right-click on each server and select ″Mark as offline″. 3. To prevent the error in the future, be sure that all servers are shut down by a supported shutdown method, as opposed to killing the process or JVM. In the case of a failure that prevents proper shutdown, open MDAC and mark the servers that are no longer running as offline as describe above. After verifying all of the above, the server be restarted and/or the action be attempted again.

DSYM173E: Unable to delete subscription *subscription name***, reason:** *exception***. Action canceled.**

Explanation: DB2 Everyplace Mobile Devices Administration Center could not delete the specified subscription due to internal cleanup failing. To prevent loss of data, the delete request was halted.

User Response: The administrator should perform the following steps to ensure that the attempted action will succeed. 1. Ensure that all servers (or the server in a single server environment) are not currently running. 2. Ensure that in the MDAC Server view, all entries on the right pane show that each server is marked as offline. If they are not, right-click on each server and select ″Mark as offline″. 3. To prevent the error in the future, be sure that all servers are shut down by a supported shutdown method, as opposed to killing the process or JVM. In the case of a failure that prevents proper shutdown, open MDAC and mark the servers that are no longer running as offline as describe above. After verifying all of the above, the server be restarted and/or the action be attempted again.

DSYM174E: Target table name *target table name* **already used by another subscription for mirror** *mirror database name***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the specified target table name was already used by another table subscription for the specified mirror. The target table names within a mirror must be unique.

User Response: Specify a unique target table name and try again.

DSYM175E: Unable to support the Join Filter feature for subscription *subscription name***, target table** *target table name***, reason:** *reason***. Action canceled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the specified target table contained a Join Filter that is unsupported for the specified reason.

User Response: Correct the filter for the table and try again.

DSYM176E: Unable to locate a local database which could be used as a source database.

Explanation: The DB2 Everyplace Mobile Devices Administration Center could not locate any local database(s) which could be used as a source database. At least one local database must be defined for use as a source database.

User Response: Ensure at least one local database is defined and try again.

DSYM177I: The source table *table name* **contains one or more columns whose datatypes are not supported. The following columns are unavailable for use in the subscription:** *column name [datatype name/datatype]***.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the specified source table contained one or more columns that are not supported. The specified columns are unavailable for use in the subscription.

User Response: No action required.

DSYM178I: DB2 Everyplace Mobile Device Administration Center has detected that DB2 UDB Version 8.1 or later is installed. You are now running the DB2 UDB Version 7.2 Control Center which has DB2 Everyplace Sync Server support. Control Center specific objects will be hidden or disabled. Also, DPROPR support will be disabled in the DB2 Everyplace Mobile Device Administration Center. To adminstrate DPROPR, use the Replication Center and the DB2 Everyplace XML Scripting tool.

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that DB2 UDB Version 8.1 or later is installed. because DB2 Everyplace Sync Server does not fully support DB2 UDB Version 8.1 or later yet, Control Center specific objects will be hidden or disabled and DPROPR functionality will be disabled in the DB2 Everyplace Mobile Device Administration Center. To adminstrate DPROPR, use the Replication Center and the DB2 Everyplace XML Scripting tool.

User Response: To administrate your DB2 installation, use the Control Center. To adminstrate DPROPR, use the Replication Center and the DB2 Everyplace XML Scripting tool.

DSYM179E: Unable to retrieve a list of tables from source database *source database name* **using schema pattern** *schema pattern*, **table pattern** *table pattern*. **The list of tables to add be empty. Reason:** *exception*.

Explanation: The DB2 Everyplace Mobile Device Administration Center was unable to retrieve a list of tables from the specified source database due to the unexpected specified exception.

User Response: Correct the exception and try again.

DSYM180E: The source table *table name* **contains one or more columns whose datatypes are not supported and are nullable with no default value specified. The table cannot be used in a table subscription. Action cancelled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the specified source table contained one or more columns that are not supported and are nullable with no default value specified. The table cnanot be used in a table subscription.

User Response: Specify a table that does not contain unsupported datatypes and are not nullable with no default value specified.

DSYM181E: Unable to unblock subscription *subscription name* **until a replication on mirror database** *mirror database name* **is completed. Action cancelled.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the specified subscription could not be unblocked until a replication of the specified mirror database has been completed.

User Response: Perform a replication and try again.

DSYM182W: Unable to support the Join Filter feature for subscription *subscription name*, **target table** *target table name*, **reason:** *reason*. **Changes were committed.**

Explanation: The DB2 Everyplace Mobile Devices Administration Center detected that the specified target table contained a Join Filter that is unsupported for the specified reason. Changes to the subscription were committed.

User Response: Correct the filter for the table and try again.

DSYM183E: Target table *target table name* **is invalid. Action canceled.**

Explanation: The specified target table name is invalid. Target table names be a maximum of 18 characters in length.

User Response: Specify a valid target table name and try again.

DSYM185I: User *user name*, **device id** *device id*, **device type** *device type* **was deleted.**

Explanation: The specified users' device was deleted.

User Response: No action required.

DSYS000I: Complete synchronization request received; *operation type* **operation started for user** *user name*, **session** *session number*, **for subscription** *subscription name*.

Explanation: A complete synchronization request has been received from the specified user as the last message has been received. The DB2 Everyplace Sync Server will now begin the synchronization process.

User Response: No action required.

DSYS001I: Synchronization ended for user *user name*, **session** *session number*, **for subscription** *subscription name*.

Explanation: The synchronization process for the specified user has ended.

User Response: No action required.

DSYS002E: Synchronization failed for user *user name*, **session** *session number*, **for subscription** *subscription name*: *reason*.

Explanation: The synchronization process for the specified user has failed.

User Response: Contact your DB2 Everyplace Sync Server administrator.

DSYS003I: Replication started for database *source database name*.

Explanation: The replication process has begun for the specified database.

User Response: No action required.

DSYS004I: Replication ended for database *source database name*.

Explanation: The replication process has ended for the specified database.

User Response: No action required.

DSYS005E: Replication failed for database *database name*: *reason*.

Explanation: The replication process for the specified database failed.

User Response: Contact your DB2 Everyplace Sync Server administrator.

DSYS006I: Receiving synchronization request from user *user name*, **session** *session number*, **for subscription** *subscription name*.

Explanation: The specified user initiated the synchronization process and the first message has been received by the DB2 Everyplace Sync Server.

User Response: No action required.

DSYS007I: Restarting synchronization for user *user name*, **session** *session number*, **for subscription** *subscription name*, **from state** *state*.

Explanation: Synchronization was restarted for the specified user by the DB2 Everyplace Sync Server.

User Response: No action required.

DSYS008I: Synchronization started for user *user name*.

Explanation: Synchronization has started for the specified user.

User Response: No action required.

DSYS009I: Synchronization completed for user *user name*.

Explanation: Synchronization has completed for the specified user.

User Response: No action required.

DSYS010I: Starting synchronization.

Explanation: Synchronization is starting for the specified user. No specific subscription has yet been specified.

User Response: No action required.

DSYS011I: Receiving synchronization request for specified subscription.

Explanation: A synchronization request is being received for the specified subscription from the specified user. Server processing on behalf of this request has not yet begun.

User Response: No action required.

DSYS012I: Subscription synchronization in progress.

Explanation: A complete synchronization request has been received for the specified subscription from the specified user, and server processing on behalf of this request has begun.

User Response: No action required.

DSYS013I: Subscription synchronization completed.

Explanation: Server processing of the specified subscription synchronization has completed successfully.

User Response: No action required.

DSYS014E: Subscription synchronization failed.

Explanation: Server processing of the specified subscription synchronization did not complete successfully.

User Response: Contact your administrator.

DSYS015W: Previous session was incomplete, performing session cancellation recovery for user *user name*, **session** *session number*, **for subscription** *subscription name*.

Explanation: The messages from a previous subscription synchronization were never retrieved by the client. Session cancellation recovery was needed.

User Response: No action required.

DSYS016I: User *user name* **will be forced to refresh table** *table name* **on its next synchronization request.**

Explanation: The DB2 Everyplace Sync Server has determined that the specified User has not performed a synchronization of the specified table for a period of days defined by the property MaxSyncPeriod.Days. On its next synchronization, the User will receive notification that it must refresh the table. Forcing this refresh allows the DB2 Everyplace Sync Server to improve performance for more frequently-synchronizing Users.

User Response: No action required.

DSYS017I: User *user name* **upgraded device successfully.**

Explanation: The auto deployment servlet has determined that the user has successfully upgraded.

User Response: No action required.

DSYS018E: User *user name* **failed to upgrade device.**

Explanation: The auto deployment servlet has determined that the user failed to upgrade.

User Response: No action required.

DSYS019I: User *user name* **has begun to upgrade device.**

Explanation: The auto deployment servlet has determined that the user started the upgrade.

User Response: No action required.

DSYS020E: The directory specified in DSYDeploy.properties does not exist: *directory name***.**

Explanation: The directory specified in DSYDeploy.properties does not exist.

User Response: Administration must take action.

DSYS021E: User *security name* **is not a member of the required LDAP** *sync group* **group. Session aborted.**

Explanation: The specified user is not a member of the required LDAP group. A user must be a member of the SYNCGROUP specified in com/ibm/mobileservices/DSYLDAP.properties in order to synchronize.

User Response: Contact your administrator.

DSYS023E: User *security name* **belongs to more than one** *DB2e group name prefix* **prefixed LDAP group. Session aborted.**

Explanation: The specified user is a member of more than one group that has 'DB2e' prefix (prefix is configurable, it is 'DB2e' by default) which has subscriptions assigned to it. An LDAP user user not belong to multiple DB2e groups due to possible subscription conflicts.

User Response: Contact your administrator.

DSYS024W: The synchronization thread pool has reached maximum capacity.

Explanation: The server has begun to queue new synchronization request because the thread pool is full. This will cause increased response times from the server for new requests.

User Response: If the machine's resources are not fully maximized, then increasing the thread pool size is recommended. The thread pool size is determined based on the property ThreadPoolCount in DSYGdflt.properties.

DSYS025W: The synchronization connection pool has reached maximum capacity.

Explanation: The server has begun to wait for connections because all connections in the connection pool are currently in use. This will cause increased response times from the server for new requests.

User Response: If the machine's database resources are not fully maximized, then increasing the connection pool size is reccommended. The connection pool size is determined based on the property Jdbc.MaxConnections in the control database properties table.

DSYS026W: The device *device id* **in group** *group name* **has failed to authenticate because it is not enabled.**

Explanation: The device has failed to authenticate because it is not enabled.

User Response: Check to make sure that this device/user/group are enabled.

DSYS027W: The device *device id* **for user** *user name* **has failed to authenticate because it is not associated with a group.**

Explanation: The device has failed to authenticate because it is not assoicated with a group.

User Response: Check to make sure that this user is associated with a group.

DSYS028W: The data sent by user *user name***, device id** *device id* **cannot be processed. This can happen when the device is deleted or when the schema of a table to which this user subscribes is changed.**

Explanation: The device sent data which could not be processed by the server due to an administrative action performed for this user, device, or associated subscription.

User Response: Contact the user and have them re-apply the necessary changes.

DSYS029I: The user *user name* **with device id** *device id* **is synchronizing from an offline refresh image.**

Explanation: The user is synchronizing from an offline refresh image.

User Response: No action required.

DSYS030I: The user *user name* **with device id** *device id* **is synchronizing from a backup image.**

Explanation: The user is synchronizing from an backup image.

User Response: No action required.

DSYS031E: The backup image for user *user name* **with device** *device id* **cannot be used to synchronize until its previous sessions are processed.**

Explanation: A backup image cannot be used until all previous device sessions are complete or the administrator manually deletes device.

User Response: Run the command DSYINSTDIR/Server/bin/dsyrecovery or delete the device.

DSYS032W: The user *user name* **with device** *device id* **cannot synchronize because the subscription is blocked.**

Explanation: The user be blocked from synchronizing because the subscription is being edited, was manually blocked by the administrator, replication is currently executing or replication did not finish successfully.

User Response: Unblock the subscription using MDAC. If the subscription is not blocked and no replication is in progress, replicate manually. This will recover any failed replication and make the subscription available for synchronization after it has been edited.

DSYT001E: User *user name* **cannot override data filter** *data filter name* **because the data filter is not defined for this user''s group** *group name***.**

Explanation: Users can only use data filters that are defined in their groups to customize the data filter value based on current user. In this case, the XML script tried to refer to a data filter to specify customized filter value but no such data filter exists in the user's group.

User Response: Fix XML script to reference data filter names that are valid for the related group.

DSYT002E: Data filter *data filter name* **is specified more than once.**

Explanation: The XML script contains duplicate references for a data filter making the required action unclear.

User Response: Fix XML script to reference data filter names uniquely.

DSYT003E: Subscription *subscription name* **is not a jdbc subscription.**

Explanation: The XML script tries to change a subscription as if it were a JDBC subscription, where as the subscription type is different.

User Response: Fix XML script to match subscription type you are trying to make changes to.

DSYT004E: Could not create index *index name* **on table** *table name* **in mirror database.**

Explanation: The XML scripting tool failed to create the specified index in the mirror database. This be caused by invalid column references, table name or index already exist.

User Response: Fix XML script to specify correct index description.

DSYT005E: Subscription *subscription name* **is not an upload subscription.**

Explanation: The XML script tries to change a subscription as if it were an Upload subscription, where as the subscription type is different.

User Response: Fix XML script to match subscription type you are trying to make changes to.

DSYT006E: Adapter type does not match subscription type for subscription *subscription name***.**

Explanation: The XML script tried to associate a subscription with an adapter where the adapter and subscription types do not match.

User Response: Fix XML script to match the subscription type and adapter type that you are trying to associate the subscription with.

DSYT007E: Subscription *subscription name* **is not a file subscription.**

Explanation: The XML script tries to change a subscription as if it were a File subscription, where as the subscription type is different.

User Response: Fix XML script to match subscription type you are trying to make changes to.

DSYT008E: Could not unblock subscription *subscription name***.**

Explanation: The admin tool could not unblock a subscription that was blocked earlier to prevent synchronization while it was being changed.

User Response: A server that is down be marked as running, thus admin tool is not able to communicate with it, or a running server not be responding. There also be a network problem. Using admin GUI, check to make sure that subscription is not blocked if you wish synchronization to proceed.

DSYT009E: Subscription *subscription name* **is not a custom subscription.**

Explanation: The XML script tries to change a subscription as if it were a Custom subscription, where as the subscription type is different.

User Response: Fix XML script to match subscription type you are trying to make changes to.

DSYT010E: Conflicting include and exclude tags detected.

Explanation: The XML script had ambigious tags where it was not clear whether to include or exclude a referenced item.

User Response: Fix XML script to remove ambigious include or exclude tags.

DSYT011E: ''Order'' tags used in XML scripts must specify ascending order. XML Scripting tool found order *order index* **after processing order** *another order index***.**

Explanation: The XML script specified incorrect ordering of items such as Subscription Sets in Groups, or Subscriptions in Subscription Sets. When order of items are specified, they must appear in ascending order or the 'order' tag must be ommited in which case the order that the items appear will be used.

User Response: Fix XML script to remove or change ambigious 'Order' tags.

DSYT012E: Column *column name* **with data type** *data type name* **cannot be replicated. Data type is not supported.**

Explanation: The XML script specified a column with an unsupported data type.

User Response: Fix XML script to remove or change columns that have unsupported data types. Depending on the types of databases involved, some data types not be supported. Read documentation about unsupported data types and data type mappings for further information to better design your data synchronization system.

DSYT013E: Subscription *subscription name* **has no tables.**

Explanation: The XML script did not specify any tables for the table subscription.

User Response: Fix XML script to add tables to the subscription.

DSYT014I: Including table *table name* **to subscription** *subscription name***.**

Explanation: The XML scripting tool is including the table into the subscription to be replicated and synchronized.

User Response: No action required.

DSYT015E: Table *table name* **already exists in subscription** *subscription name***.**

Explanation: Table is already included in the subscription. A table cannot be added to a subscription more than once with the same target schema and table name.

User Response: Fix the XML Script to remove offending table description or change it's target name.

DSYT016E: A table named *table name* **does not exist in subscription** *subscription name***.**

Explanation: XML Scripting tool could not change the details of a table of a subscription because subscription does not have a table with that name.

User Response: Fix the XML Script to remove offending table description or change it's target name to refer to an existing table.

DSYT017I: Altering table *table name* **in subscription** *subscription name***.**

Explanation: XML Scripting tool is about to change the details of a table of a subscription.

User Response: No action required.

DSYT018E: Unexpected join filter value *Y or N or G***.**

Explanation: XML Script contains unknown character for join filter setting. It must be one of Y (enabled), N (disabled), or G (Use whatever the Global setting is).

User Response: Fix XML Script to use one of the valid join filter setting identifiers.

DSYT019I: 'JoinFilter' tag for subscription tables does not apply to upload subscriptions. It will be ignored.

Explanation: XML Script contains instruction for join filter setting for a table of an upload subscription where as this setting is used only for replicated subscriptions.

User Response: To avoid this message, fix XML Script to remove join filter settings for tables of upload subscriptions.

DSYT020I: Mirror only index is not applicable in this context. Index *index name* **is ignored.**

Explanation: XML Script contains instruction to add an index that should be created in the mirror database but subscription does not have a mirror or adding such index is not possible at this time.

User Response: To avoid this message, fix XML Script to remove invalid AddIndex tags.

DSYT021I: Excluding source database column *column name* **from target table** *table name***.**

Explanation: XML Scripting tool is excluding a column of a table. This is typically required if the source database has changed and column no longer exists. Normally you should not exclude any column of a replicated table.

User Response: No action required.

DSYT022I: Setting up synchronization and replication attributes of source column *column name***.**

Explanation: XML Scripting tool is processing the setup of a column of a table.

User Response: No action required.

DSYT023E: Source column *column name* **is being skipped in the XML script. Column need to have default value for the table replication to succeed. Source column name will be used for target name.**

Explanation: The source database table contains a column whose replication and synchronization properties are not specified for the Data DB2 Everyplace Sync Server. Replication process not be able to figure out how to replicate this column. Setup process will continue, however, DB2 Everyplace Sync Server fail trying to use this column.

User Response: Make sure all columns of the source table are listed in the XML Script with correct replication and synchronization settings.

DSYT024I: Blocking subscription *subscription name*.

Explanation: XML Scripting tool is requesting the servers to stop servicing synchronization requests for this subscription.

User Response: No action required.

DSYT025I: Unblocking subscription *subscription name*.

Explanation: XML Scripting tool is requesting the servers to continue servicing synchronization requests for this subscription.

User Response: No action required.

DSYT026E: The value of the element *xml element name* **should be a valid integer. Specified value is** *value*.

Explanation: XML Scripting tool is epxecting an integer, where as it is unable to interpret the specified value as a valid integer.

User Response: Fix the XML content to specify a valid integer where required.

DSYT027E: *SourceTableSpace* **tag is not valid when source database is not DB2.**

Explanation: It is possible to put a table in a Tablespace in DB2. This tag is used to specify the tablespace of a table in a DB2 database. It is not meaningful if source database is not DB2.

User Response: Fix the XML content to remove this tag.

DSYT028E: Could not block mirror database *database name*.

Explanation: The admin tool could not block a mirror database to prevent synchronization while it was being changed.

User Response: A server that is down be marked as running, thus admin tool is not able to communicate with it, or a running server not be responding, or the mirror database be down. There also be a network problem.

DSYT029E: Cannot change the name of an LDAP user.

Explanation: In LDAP environment, XML Tool cannot edit the name of a user.

User Response: You can try to remove the LDAP user in the LDAP server and add a user with a different name.

DSYT030E: There was an error creating an LDAP user and associating it with DB2 Everyplace Sync Server.

Explanation: XML Tool failed to create a user.

User Response: This be be because the user already exists, or one of it's properties were invalid. There be more information displayed as the root cause.

DSYT031E: There was an error creating an LDAP group and associating it with DB2 Everyplace Sync Server.

Explanation: XML Tool failed to create a group.

User Response: This be be because the group already exists, or one of it's properties were invalid. There be more information displayed as the root cause.

DSYT032E: Cannot change the name of an LDAP group.

Explanation: In LDAP environment, XML Tool cannot edit the name of a group.

User Response: You can try to remove the LDAP group in the LDAP server and add a group with a different security name.

DSYT033E: A valid name for the user must be provided.

Explanation: An invalid name was specified for a user.

User Response: Provide a valid username. In LDAP environment this name must satisfy LDAP requirements also.

DSYT034I: Updating control database.

Explanation: Changes are being written to DB2 Everyplace Sync Server control database.

User Response: No action required.

DSYT035E: A valid name for the group must be provided.

Explanation: An invalid name was specified for a group.

User Response: Provide a valid group name. In LDAP environment this name must satisfy LDAP requirements also.

DSYT036E: *group name* **is not a SyncGroup. A valid SyncGroup name that is specified as a SyncGroup in DSYLDAP.properties file must be provided as the SyncGroup of this user.**

Explanation: An invalid name was specified for a SyncGroup group. Valid SyncGroups are listed in DSYLDAP.properties file in LDAP enabled DB2 Everyplace Sync Server installations.

User Response: Provide a valid group name that is listed in DSYLDAP.properties as a SyncGroup.

DSYT037E: User''s group name *group name* **must start with** *db2e group name prefix***.**

Explanation: In LDAP enabled DB2 Everyplace Sync Server installations, to be able to synchronize, a user needs to belong to 1 'DB2e' group whose name starts with 'DB2e' (or whatever the prefix value is set in configuration properties). This 'DB2e' group is used by DB2 Everyplace Sync Server for synchronization purposes.

User Response: Provide a valid group name that starts with the designated prefix (usually 'DB2e') in addition to other groups this LDAP user already belong to.

DSYT038I: 'WhereClauseMirrorToMobile' and 'WhereClauseMasterToMirror' tags for subscription tables does not apply to upload subscriptions. These tags will be ignored.

Explanation: XML Script contains instruction to add a where clause setting for a table of an upload subscription where as this setting is used only for replicated subscriptions.

User Response: To avoid this message, fix XML Script to remove where clauses for tables of upload subscriptions.

DSYT039E: Create user failed -- possible invalid password length.

Explanation: The creation of the user failed, possibly due to an invalid password length.

User Response: Ensure the specified password conforms to the LDAP security policies as specified by the administrator.

DB2 Everyplace Update Tool error messages

This topic lists all of the error messages that can be generated by the DB2 Everyplace Update Tool.

Table 46 displays the error message and a possible remedy for the problem.

Table 46. Troubleshooting guide

Error message	Possible remedy
Authentication failed (invalid encryption key) - update aborted	Verify that the client settings match the user's settings defined in the Mobile Device Administration Center.
File size exceeds available memory	Delete any applications or files that are no longer needed on the device and try again.

Table 46. Troubleshooting guide (continued)

Error message	Possible remedy
Internal server error	This is an internal error that you need to report to IBM Software Support with the trace file.
Failed to open connection	Check your network connection and the SyncServer. Make sure that the host is connected and the server is running.
Failed to establish connection	Check your network connection and the SyncServer. Make sure that the host is connected and the server is running.
Failed to send request	Try to synchronize again when there is less traffic on the network or try to synchronize from a faster network.
Failed to receive reply	Try to synchronize again when there is less traffic on the network or try to synchronize from a faster network.
Timeout while receiving reply	Specify a larger timeout value or try to synchronize when there is less traffic on the network.
Failed to receive acknowledge	Try to synchronize again when there is less traffic on the network or try to synchronize from a faster network.
Failed to open Net library	Verify that the network library exists on the device. Try to reinstall the library.
Failed to resolve hostname	Verify that the hostname and the DNS addresses are correct.
Failed to allocate working buffer for transport	Delete any applications or files that are no longer needed on the device and try again.
Unknown network error	This is an internal error that you need to report to IBM Software Support with the trace file.
Failed to create target file	Verify that the target file is not being used by another application. If the target file is being used, unlock it and synchronize again.
No files received for update	This is an informational message stating that the server does not have an update for the mobile device.

Interfaces

This topic explains the interfaces provided by DB2 Everyplace.

ADO.NET

This topic explains the functions that are provided by the DB2 Everyplace .Net Data Provider.

Important: If you try to invoke a method or property that DB2 Everyplace does not support, DB2 Everyplace throws a `System.NotSupportedException`.

DB2eConnection members

Table 47 describes public static methods that are provided by DB2eConnection.

Table 47. Public static (shared) methods

Method	Description
ReleaseObjectPool	Indicates that the DB2 Everyplace environment handle can be released when the last underlying connection is released.

Table 48 describes public instance constructors that are provided by DB2eConnection.

Table 48. 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 49 describes public instance properties that are provided by DB2eConnection.

Table 49. 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 50 describes public instance methods that are provided by DB2eConnection.

Table 50. Public instance methods

Method	Description
BeginTransaction	Overloaded. Begins a transaction at the database.
BeginTransaction(IsolationLevel isoLevel)	Overloaded. Begins a transaction at the specified isolation level. Supported values: ReadCommitted ReadUncommitted RepeatableRead Serializable
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`.
GetBufferpoolSize	Gets the size of the bufferpool, in bytes.
Open	Opens a connection to a data source with the property settings specified by the `ConnectionString`.

Table 51 on page 155 describes public instance events that are provided by DB2eConnection.

Table 51. 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.

Table 52 describes ConnectionString properties that are provided by DB2eConnection.

Table 52. ConnectionString properties

Keyword	Type	Description
bufferpool_size	integer	The amount of memory, in bytes, that the DB2 Everyplace database should reserve for its bufferpools. If this value is not a multiple of 4K (4096 bytes), DB2 Everyplace rounds it down to the next smallest multiple of 4K.
Database	string	Database name.
User ID or UID	string	User ID.
Password or PWD	string	User password.
ENCODING	string	Specifies the database encoding. For example, to connect to a UTF-8 database, the encoding is UTF-8.
LOCK TIMEOUT	integer	Indicates the number of seconds to wait before rolling back a transaction when a lock cannot be obtained. The default value is 20.
SHARED DB ACCESS	boolean	A boolean value that indicates whether the database allows connections to share access. The default value is false.
IO WRITETHROUGH	boolean	Specifies whether changes to the database are pushed to storage media without delay or handed to the operating system.
TABLE CHECKSUM	boolean	A boolean value that indicates whether to enable checksum on a table or not.

DB2 Everyplace includes the following pre-defined bufferpool sizes:

Table 53. Pre-defined bufferpool size constants

Constant	Bufferpool size in bytes
SQL_BUFFERPOOL_SIZE_DEFAULT	The default value for the platform on which you are running DB2 Everyplace.
SQL_BUFFERPOOL_SIZE_64K	65 536
SQL_BUFFERPOOL_SIZE_128K	131 072
SQL_BUFFERPOOL_SIZE_256K	262 144
SQL_BUFFERPOOL_SIZE_512K	524 288
SQL_BUFFERPOOL_SIZE_1M	1 048 576

Table 53. Pre-defined bufferpool size constants (continued)

Constant	Bufferpool size in bytes
SQL_BUFFERPOOL_SIZE_2M	2 097 152
SQL_BUFFERPOOL_SIZE_4M	4 194 304
SQL_BUFFERPOOL_SIZE_8M	8 388 608

Important:

- The minimum value for SQL_ATTR_BUFFERPOOL_SIZE is SQL_BUFFERPOOL_SIZE_64K. If you call SQLSetConnectAttr() and specify a smaller value than SQL_BUFFERPOOL_SIZE_64K, SQlSetConnectAttr() returns SQLSTATE **HY**024.
- If the database engine cannot allocate as much memory as you specify in the SQL_ATTR_BUFFERPOOL_SIZE connection attribute, the engine will try to use a smaller bufferpool configuration. SQLConnect() will return SQLSTATE 01000.
- If there is not enough memory for the minimum bufferpool configuration, SQLConnect() will return SQLState 58004.
- You cannot change the size of the bufferpool if a connection to the database already exists. New connections will use the bufferpool size of the existing connection. SQLConnect() will return a warning.

DB2eCommand members

Table 54describes public instance constructors that are provided by DB2eCommand.

Table 54. 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 55 describes public instance properties that are provided by DB2eCommand.

Table 55. 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.

Table 55. Public instance properties (continued)

Property	Description
UpdatedRowSource	Gets or sets a value that specifies how the `Update` method should apply command results to the `DataRow`.

Table 56 describes public instance methods that are provided by DB2eCommand.

Table 56. Public instance methods

Method	Description
CreateParameter	Creates a new instance of a `DB2eParameter` object.
Dispose	Overloaded. Clean up.
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.

DB2eCommandBuilder members

Table 57. 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 58. 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 59. Public instance properties

Property	Description
DataAdapter	Gets or sets an `DB2eDataAdapter` object for which this `DB2eCommandBuilder` object will generate SQL statements.

Table 60. 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 61. Protected instance methods

Method	Description
Dispose	Overloaded.

DB2eDataAdapter members

Table 62 describes public instance constructors that are supported by DB2eDataAdapter.

Table 62. 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 63 on page 159 describes public instance properties that are provided by DB2eDataAdapter.

Table 63. 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 64 describes public instance methods provided by DB2eDataAdapter.

Table 64. 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 65 describes public instance events that are provided by DB2eDataAdapter.

Table 65. 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 66 describes public instance properties that are supported by DB2eDataReader.

Table 66. 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 67 describes public instance methods that are supported by DB2eDataReader.

Table 67. 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.
IsDBNull	Gets a value indicating whether the column contains non-existent or missing values.

Table 67. Public instance methods (continued)

Method	Description
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 68 lists properties that are provided by DB2eError.

Table 68. 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.

DB2eParameter members

Table 69. 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.
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 70. 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.

Table 70. Public instance properties (continued)

Property	Description
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 71. Public instance methods

Method	Description
ToString	Gets a string containing the `ParameterName`.

DB2eTransaction members

Table 72. Public instance properties

Property	Description
Connection	Gets the `DB2eConnection` object associated with a transaction or a null reference if the transaction is no longer valid.
IsolationLevel	Gets the isolation level of a transaction.

Table 73. Public instance methods

Method	Description
Commit	Commits the transaction.
Rollback	Rolls back the transaction.
Save(string)	Creates a savepoint in the transaction. The ROLLBACK statement can use a savepoint to rollback a portion of the transaction by using the savepoint name.
Rollback(string)	Rolls back the transaction to the specified savepoint name.

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 74. 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.
Binary	A fixed length binary (CHAR FOR BIT DATA). This maps to an array of type Byte[].
VarBinary	A variable length binary (VARCHAR FOR BIT DATA). This maps to an array of type Byte[].

Requirements

NameSpace: IBM.Data.DB2.DB2e Namespace

Assembly: IBM.Data.DB2.DB2e.dll

DB2 Call Level Interface (CLI)

This topic explains the functions that are provided by the DB2 Call Level Interface.

DB2 CLI function summary

Table 75 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 75. DB2 CLI function list

Function name	Purpose	Summary of differences
SQLAllocConnect	Obtains a connection handle.	
SQLAllocEnv	Obtains an environment handle.	
SQLAllocHandle	Obtains a handle.	
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.

Table 75. DB2 CLI function list (continued)

Function name	Purpose	Summary of differences
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.
SQLCancel	Prematurely ends the data-at-execution sequence for sending and retrieving long data in pieces and cancels a function called in a different thread.	
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.
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.

Table 75. DB2 CLI function list (continued)

Function name	Purpose	Summary of differences
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.
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.
SQLGetLength	Returns the length of a large object value.	
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.
SQLGetSubString	Returns a portion of a large object value.	
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.	

Table 75. DB2 CLI function list (continued)

Function name	Purpose	Summary of differences
SQLParamData	Gets next parameter for which a data value is needed.	
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.
SQLPutData	Returns the data values of parameters.	
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 167

"Key to DB2 CLI function descriptions"

Key to DB2 CLI function descriptions

Each function description contains the following topics:

Purpose

This topic 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 topic for the function.

Syntax

This topic 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 topic lists each function argument, 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 topic 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 topic instead.

Return codes

This topic 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 topic 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 topic indicates any differences or limitations between DB2 Everyplace CLI and ODBC that might affect an application.

See the DB2 Version 9.1 documentation 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

"DB2 CLI function summary" on page 163

"Data conversion by DB2 CLI functions"

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 76 shows the supported data conversions between C and SQL data types. The first column in Table 76 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 76. 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
CHAR FOR BIT DATA	SQL C BINARY	SQL C BLOB
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
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
VARCHAR FOR BIT DATA	SQL C BINARY	SQL C BLOB

Related reference

"Key to DB2 CLI function descriptions" on page 166

"DB2 CLI function summary" on page 163

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 166

"DB2 CLI function summary" on page 163

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 166

"DB2 CLI function summary" on page 163

SQLAllocHandle—Allocate handle

Purpose

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().

Specification

- DB2 CLI 5.0
- ODBC 3.0
- ISO CLI

Syntax

```
SQLRETURN   SQLAllocHandle   (SQLSMALLINT   HandleType,
                              SQLHANDLE     InputHandle,
                              SQLHANDLE    *OutputHandlePtr);
```

Function arguments

Table 77. 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.

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 78. 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.
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 166

"DB2 CLI function summary" on page 163

"SQLExecDirect—Execute a statement directly" on page 194

"SQLFreeHandle—Free handle resources" on page 209

SQLAllocStmt—Allocate a statement handle

In ODBC Version 3, *SQLAllocStmt()* was deprecated and replaced with SQLAllocHandle(); see "SQLAllocHandle—Allocate handle" on page 169 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 166

"DB2 CLI function summary" on page 163

SQLBindCol—Bind a column to an application variable

Purpose

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().

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

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 79 describes the arguments that are supported by SQLBindCol.

Table 79. 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.

Table 79. SQLBindCol arguments (continued)

Data type	Argument	Use	Description
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. SQLBindCol() allows SQL_C_LOB_LOCATOR as the *TargetType*. In doing so, the application will receive a LOB LOCATOR, which it can use to retrieve part of the data, all of the data or its length.
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 topic 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 80 describes the SQLSTATEs that are returned by SQLBindCol.

Table 80. 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.
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.

Table 80. SQLBindCol SQLSTATEs (continued)

SQLSTATE	Description	Explanation
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

"DB2 CLI function summary" on page 163

"Key to DB2 CLI function descriptions" on page 166

SQLBindParameter—Bind a parameter marker to a buffer

Purpose

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.

Specification

- DB2 CLI 2.1
- ODBC 2.0

Syntax

```
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 81. 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.

Table 81. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
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*.

Table 81. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*ParameterType*	input	SQL data type of the parameter. The supported types are: • SQL_BINARY • SQL_BLOB • SQL_CHAR • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR
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.

Table 81. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
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 • If *StrLen_or_IndPtr* contains SQL_DATA_AT_EXEC, then *ParameterValuePtr* is an application-defined 32-bit alue that is associated with this parameter. This 32-bit value is returned to the application via a subsequent SQLParamData() call.

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 82. 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.
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.

Table 82. SQLBindParameter SQLSTATEs (continued)

SQLSTATE	Description	Explanation
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 435
"Key to DB2 CLI function descriptions" on page 166
"DB2 CLI function summary" on page 163
"SQLExecDirect—Execute a statement directly" on page 194
"SQLExecute—Execute a statement" on page 195

SQLCancel function (CLI) - Cancel statement

Purpose

SQLCancel() can be used to prematurely terminate the data-at-execution sequence for sending and retrieving long data in pieces.

SQLCancel() can also be used to cancel a function that is called in a different thread.

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLCancel       (SQLHSTMT         StatementHandle);  /* hstmt */
```

Function arguments

Table 83. SQLCancel arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle

Usage

After SQLExecDirect() or SQLExecute() returns SQL_NEED_DATA to solicit for values for data-at-execution parameters, SQLCancel() can be used to cancel the data-at-execution sequence for sending and retrieving long data in pieces. SQLCancel() can be called any time before the final SQLParamData() in the sequence. After the cancellation of this sequence, the application can call SQLExecute() or SQLExecDirect() to re-initiate the data-at-execution sequence.

If no processing is being done on the statement, SQLCancel() has no effect. Applications should not call SQLCancel() to close a cursor, but rather SQLFreeStmt() should be used.

If an SQL statement is being executed when SQLCancel() is called on another thread to cancel the statement execution, it is possible that the execution succeeds and returns SQL_SUCCESS, while the cancel is also successful. In this case, DB2 CLI assumes that the cursor opened by the statement execution is closed by the cancel, so the application will not be able to use the cursor.

Return codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

Note: SQL_SUCCESS means that the cancel request was processed, not that the function call was canceled.

Diagnostics

Table 84. SQLCancel SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	Unable to allocate memory required to support execution or completion of the function.
HY013	Unexpected memory handling error.	Unable to access memory required to support execution or completion of the function.
HY018	Server declined cancel request.	The server declined the cancel request.
HY506	Error closing a file.	An error occurred when closing the temporary file generated by DB2 CLI when inserting LOB data in pieces using SQLParamData()/SQLPutData().

Restrictions

None.

Example

```
/* cancel the SQL_DATA_AT_EXEC state for hstmt */
cliRC = SQLCancel(hstmt);
```

SQLColumns - Get column information for a table

Purpose

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.

Specification

- DB2 CLI 2.1
- ODBC 1.0

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 85 describes the arguments that are supported by SQLColumns.

Table 85. SQLColumns arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLCHAR	*CatalogName*	Input	Buffer that can 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 can 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 can contain a *pattern-value* to qualify the result set by table name.
SQLSMALLINT	NameLength3	Input	Length of TableName.
SQLCHAR	*ColumnName*	Input	Buffer that can 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 might 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. "Columns returned by SQLColumns" lists the columns in the result set.

The TableName, and *ColumnName* arguments accept search patterns.

This function does not return information about 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. because 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 can 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 435.

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 437.

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 437

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 437

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 (for example, hfCURRENT 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.

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 86 describes the SQLSTATEs that are supported by SQLColumns.

Table 86. 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 259

SQLConnect—Connect to a data source

Purpose

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().

Specification

- DB2 CLI 2.1
- ODBC 1.0
- ISO CLI

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 87. 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.

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
```

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 "Encrypting local data" on page 75, "DB2eSYSUSERS" on page 67, and "GRANT" on page 408.

Return codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics

Table 88. 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
A DB2 Everyplace data source accepts connections from one process at a time. When more than one process tries to connect to the same data source at the same time, the requests are put into a queue through a mechanism called *connection serialization*.

Related reference

"Key to DB2 CLI function descriptions" on page 166

"DB2 CLI function summary" on page 163

"SQLAllocHandle—Allocate handle" on page 169

"SQLDisconnect—Disconnect from a data source" on page 191

SQLDescribeCol—Return a set of attributes for a column

Purpose

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.

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLDescribeCol   (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLCHAR      *FA R 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 89. 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 90. 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.)
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 166
"DB2 CLI function summary" on page 163
"SQLExecDirect—Execute a statement directly" on page 194
"SQLNumResultCols—Get number of result columns" on page 235

SQLDisconnect—Disconnect from a data source

Purpose

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().

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLDisconnect    (SQLHDBC          ConnectionHandle;) /* hdbc */
```

Function arguments

Table 91. 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 92. 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 166

"DB2 CLI function summary" on page 163

"SQLAllocHandle—Allocate handle" on page 169

"SQLConnect—Connect to a data source" on page 185

"SQLFreeHandle—Free handle resources" on page 209

SQLEndTran—Request a COMMIT or ROLLBACK

Purpose

SQLEndTran() requests a COMMIT or ROLLBACK operation for all active operations on all statements associated with a connection.

Specification

- DB2 CLI
- ODBC
- ISO CLI

Syntax

```
SQLRETURN   SQLEndTran      (SQLSMALLINT       HandleType,
                             SQLHANDLE         Handle,
                             SQLSMALLINT       Completion Type);
```

Function arguments

Table 93. 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 94. 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.
HY012	Invalid transaction code.	*CompletionType* is neither SQL_COMMIT nor SQL_ROLLBACK.
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 166

"DB2 CLI function summary" on page 163

"SQLSetConnectAttr—Set options related to a connection" on page 247

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 221 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 166

"DB2 CLI function summary" on page 163

SQLExecDirect—Execute a statement directly

Purpose

SQLExecDirect() directly executes the specified SQL statement. The statement can be executed only once.

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLExecDirect    (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLCHAR      *FAR StatementText,     /* szSqlStr */
                              SQLINTEGER        TextLength);       /* cbSqlStr */
```

Function arguments

Table 95. 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_NEED_DATA

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 96. 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.
40001	Transaction rollback	The transaction to which this SQL statement belonged was rolled back due to a deadlock or timeout.
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 166

"DB2 CLI function summary" on page 163

"SQLBindCol—Bind a column to an application variable" on page 172

SQLExecute—Execute a statement

Purpose

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().

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLExecute       (SQLHSTMT         StatementHandle);  /* hstmt */
```

Function arguments

Table 97. 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_NEED_DATA

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 96 on page 195), except for **HY**009 and **HY**090, and also include the SQLSTATEs in Table 98.

Table 98. 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().
HY501	Invalid DataSource name.	The specified data source name is not valid.
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.

Table 98. SQLExecute SQLSTATEs (continued)

SQLSTATE	Description	Explanation
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to a deadlock or timeout.
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.
59122	Loading library failed.	Some required libraries can not be found on the system.

Restrictions

None.

Related reference

“Key to DB2 CLI function descriptions” on page 166
“DB2 CLI function summary” on page 163
“SQLBindParameter—Bind a parameter marker to a buffer” on page 175
“SQLBindCol—Bind a column to an application variable” on page 172
“SQLExecDirect—Execute a statement directly” on page 194
“SQLPrepare—Prepare a statement” on page 239
“SQLFetch—Fetch next row”

SQLFetch—Fetch next row

Purpose

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, it performs the appropriate data transfer. SQLFetch() also performs data conversion if conversion is indicated when the column is bound. You can receive columns 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.

Specification

- DB2 CLI 1.1
- ODBC 1.0

Syntax

```
SQLRETURN   SQLFetch        (SQLHSTMT         StatementHandle);  /* hstmt */
```

Function arguments

Table 99. 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 topic).

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 100. 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.
24501	Cursor not open.	A FETCH is not valid because no result set has been generated.
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 166

"DB2 CLI function summary" on page 163

"SQLBindCol—Bind a column to an application variable" on page 172

"SQLExecDirect—Execute a statement directly" on page 194

"SQLGetData—Get data from a column" on page 217

SQLFetchScroll—Fetch row set and return data for all bound columns

Purpose

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.

Specification

- DB2 CLI 5.0
- ODBC 3.0

Syntax

```
SQLRETURN   SQLFetchScroll  (
                 SQLHSTMT          StatementHandle,
                 SQLSMALLINT       FetchOrientation,
                 SQLINTEGER        FetchOffset);
```

Function arguments

Table 101. 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 topic.

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 topics 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 102. 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 103. SQL_FETCH_PRIOR rules:

Condition	First row of new row set
Before start	Before start

Table 103. SQL_FETCH_PRIOR rules: (continued)

Condition	First row of new row set
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 104. 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 topic.

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 105. 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 106. SQL_FETCH_FIRST rules:

Condition	First row of new row set
Any	1

SQL_FETCH_LAST rules

Table 107. 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 197.

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 172.

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 108. 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 108. 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 166
"DB2 CLI function summary" on page 163
"SQLBindCol—Bind a column to an application variable" on page 172
"SQLDescribeCol—Return a set of attributes for a column" on page 188
"SQLExecDirect—Execute a statement directly" on page 194
"SQLFetch—Fetch next row" on page 197
"SQLExecute—Execute a statement" on page 195
"SQLNumResultCols—Get number of result columns" on page 235
"SQLSetStmtAttr—Set options related to a statement" on page 252

SQLForeignKeys—Get the list of foreign key columns

Purpose

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.

Specification

- DB2 CLI 2.1
- ODBC 1.0

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 109. 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 110. 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 166

"DB2 CLI function summary" on page 163

"SQLPrimaryKeys—Get primary key columns of a table" on page 241

SQLFreeConnect—Free connection handle

In ODBC Version 3, SQLFreeConnect() was deprecated and replaced with SQLFreeHandle(); see "SQLFreeHandle—Free handle resources" on page 209 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 166

"DB2 CLI function summary" on page 163

SQLFreeEnv—Free environment handle

In ODBC Version 3, SQLFreeEnv() was deprecated and replaced with SQLFreeHandle(); see "SQLFreeHandle—Free handle resources" on page 209 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 166

"DB2 CLI function summary" on page 163

SQLFreeHandle—Free handle resources

Purpose

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.

Specification

- DB2 CLI 5.0
- ODBC 3.0
- ISO CLI

Syntax

```
SQLRETURN   SQLFreeHandle    (SQLSMALLINT       HandleType,
                              SQLHANDLE         Handle);
```

Function arguments

Table 111. 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 112. 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 112. 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 166

"DB2 CLI function summary" on page 163

"SQLAllocHandle—Allocate handle" on page 169

SQLFreeStmt—Free (or reset) a statement handle

Purpose

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.

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLFreeStmt      (SQLHSTMT         StatementHandle,   /* hstmt */
                              SQLUSMALLINT     Option);           /* fOption */
```

Function arguments

Table 113. 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, SQL_RESET_PARAMS, or SQL_CLOSE.

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.

SQL_CLOSE
The cursor (if any) associated with the statement handle (StatementHandle) is closed and all pending results are discarded. The application can reopen the cursor by calling SQLExecute() with the same or different values in the application variables (if any) that are bound to StatementHandle. If no cursor has been associated with the statement handle, this option has no effect (no warning or error is generated).

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 114. 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 114. 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 166

"DB2 CLI function summary" on page 163

"SQLAllocHandle—Allocate handle" on page 169

"SQLBindCol—Bind a column to an application variable" on page 172

SQLGetConnectAttr—Get current setting of a connection attribute

Purpose

SQLGetConnectAttr() returns the current setting of a connection attribute.

Specification

- DB2 CLI 5.0
- ODBC 3.0
- ISO CLI

Syntax

```
SQLRETURN   SQLGetConnectAttr(SQLHDBC          ConnectionHandle,
                              SQLINTEGER       Attribute,
                              SQLPOINTER       ValuePtr,
                              SQLINTEGER       BufferLength,
                              SQLINTEGER       *StringLengthPtr);
```

Function arguments

Table 115 describes arguments supported by SQLGetConnectAttr.

Table 115. 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 115. 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" on page 247.

- SQL_ATTR_AUTOCOMMIT (DB2 CLI/ODBC)
- SQL_ATTR_CONNECTION_DEAD (DB2 CLI/ODBC)
- SQL_ATTR_DATABASE_ACCESS (DB2 Everyplace)
- SQL_ATTR_LOGIN_TIMEOUT (DB2 CLI/ODBC)
- SQL_ATTR_FILENAME_FORMAT (DB2 Everyplace)
- SQL_ATTR_IO_MODE (DB2 Everyplace)
- SQL_ATTR_LOCK_TIMEOUT (DB2 Everyplace)
- SQL_ATTR_TABLE_CHECKSUM (DB2 Everyplace)
- SQL_ATTR_TEMP_DIR (DB2 Everyplace)
- SQL_ATTR_TXN_ISOLATION (DB2 CLI/ODBC)
- SQL_ATTR_BUFFERPOOL_SIZE (DB2 CLI/ODBC)

Depending on the attribute, an application might not need to establish a connection before calling SQLGetConnectAttr().

Return codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics

Table 116 describes SQLSTATEs that are returned by SQLGetConnectAttr.

Table 116. 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.)
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 about 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.

SQLGetCursorName—Get cursor name

Purpose

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.

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLGetCursorName (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      *FAR CursorName,        /* szCursor */
                SQLSMALLINT       BufferLength,      /* cbCursorMax */
                SQLSMALLINT  *FAR NameLengthPtr);    /* pcbCursor */
```

Function arguments

Table 117. 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 CUR. Cursor names are always 128 bytes or less and are always unique within a connection.

Return codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics

Table 118. 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.
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 166

"DB2 CLI function summary" on page 163

"SQLExecDirect—Execute a statement directly" on page 194

SQLGetData—Get data from a column

Purpose

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 or LOB locators on each SQLFetch() or SQLFetchScroll() call. An application can either bind LOBs with SQLBindCol() or use SQLGetData() to retrieve LOBs, but both methods cannot be used together. SQLGetData() can also be used to retrieve large data values in pieces.

SQLFetch() must be called before SQLGetData().

After calling SQLGetData() for each necessary column, SQLFetch() is called to retrieve the next row.

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

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 119. 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 119. 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_BLOB_LOCATOR • 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. SQLGetData() allows SQL_C_BLOB_LOCATOR as the *TargetType*. In doing so, the application will receive a LOB LOCATOR, which it can use to retrieve part of the data, all of the data or its length.
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 119. 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 120. 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.
24504	Invalid cursor state.	The cursor identified in the UPDATE, DELETE, SET™, or GET statement is not positioned on a row.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
54028	Max LOB handles reached.	The maximum number of concurrent LOB handles has been reached.
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().
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 166

"DB2 CLI function summary" on page 163

"SQLBindCol—Bind a column to an application variable" on page 172

SQLGetDiagRec—Get multiple fields settings of diagnostic record

Purpose

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.

Specification

- DB2 CLI 5.0
- ODBC 3.0

Syntax

```
SQLRETURN   SQLGetDiagRec    (SQLSMALLINT       HandleType,
                              SQLHANDLE         Handle,
                              SQLSMALLINT       RecNumber,
                              SQLCHAR           *SQLState,
                              SQLINTEGER        *NativeErrorPtr,
                              SQLCHAR           *MessageText,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *TextLengthPtr);
```

Function arguments

Table 121. 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.

Table 121. SQLGetDiagRec arguments (continued)

Data type	Argument	Use	Description
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 445 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 166

"DB2 CLI function summary" on page 163

SQLGetFunctions

SQLGetFunctions reports whether a specific function is supported by the current database server. It is provided by the DB2 Everyplace ODBC driver.

Specification

ODBC 1.0

Syntax

```
SQLRETURN SQLGetFunctions(SQLHDBC ConnectionHandle,
                          SQLUSMALLINT FunctionId,
                          SQLUSMALLINT * SupportedPtr);
```

Function arguments

Table 122. SQLGetFunctions arguments

Data type	Argument	Use	Description
SQLHDBC	ConnectionHandle	Input	The connection handle.
SQLUSMALLINT	FunctionID	Input	A `#define` value that identifies the ODBC function of interest: • SQL_API_ODBC3_ALL_FUNCTIONS is used by an ODBC 3.x application to determine support of ODBC 3.x and earlier functions. • SQL_API_ALL_FUNCTIONS is used by an ODBC 2.x application to determine support of ODBC 2.x and earlier functions.
SQLUSMALLINT	SupportedPtr	Output	SupportedPtr can point to a single value or an array of values, depending on the value of FunctionId, as shown in the following table. Arrays returned in SupportedPtr use zero-based indexing.

Table 123. Return value of SupportedPtr argument

Value of FunctionId	Return value of SupportedPtr
`#define` value for a single ODBC function	Points to a single SQLUSMALLINT value. The value is SQL_TRUE if the specified function is supported by the driver; otherwise, the value is SQL_FALSE.
SQL_API_ODBC3_ALL_FUNCTIONS	Points to a SQLSMALLINT array with a number of elements equal to SQL_API_ODBC3_ALL_FUNCTIONS_SIZE. The Driver Manager treats this array as a 4,000-bit bitmap that can be used to determine whether an ODBC 3.x or earlier function is supported.

Table 123. Return value of SupportedPtr argument (continued)

Value of FunctionId	Return value of SupportedPtr
SQL_API_ALL_FUNCTIONS	Points to an SQLUSMALLINT array of 100 elements. The array is indexed by `#define` values used by FunctionId to identify each ODBC function; some elements of the array are unused and reserved for future use. An array element's value is SQL_TRUE if it identifies an ODBC 2.x or earlier function supported by the driver. It is SQL_FALSE if it does not identify an ODBC function, or if it identifies an ODBC function that the driver does not support.

SQLGetLength function (CLI) - Retrieve length of a string value

Purpose

SQLGetLength() is used to retrieve the length of a large object value, referenced by a large object locator that has been returned from the server (as a result of a fetch, or an SQLGetSubString() call) during the current transaction.

Specification

- DB2 CLI 2.1

Syntax

```
SQLRETURN  SQLGetLength   (SQLHSTMT          StatementHandle,  /* hstmt */
                           SQLSMALLINT       LocatorCType,
                           SQLINTEGER        Locator,
                           SQLINTEGER        *StringLength,
                           SQLINTEGER        *IndicatorValue);
```

Function arguments

Table 124. SQLGetLength arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
SQLSMALLINT	*LocatorCType*	input	The C type of the source LOB locator. This might be: • SQL_C_BLOB_LOCATOR
SQLINTEGER	*Locator*	input	Must be set to the LOB locator value.
SQLINTEGER *	*StringLength*	output	The length of the returned information in *rgbValue* in bytes[a] if the target C buffer type is intended for a binary string variable and not a locator value. If the pointer is set to NULL then the SQLSTATE **HY**009 is returned.
SQLINTEGER *	*IndicatorValue*	output	Always set to zero.

Usage

SQLGetLength() can be used to determine the length of the data value represented by a LOB locator. It is used by applications to determine the overall length of the referenced LOB value so that the appropriate strategy to obtain some or all of the LOB value can be chosen.

The statement handle must not have been associated with any prepared statements or catalog function calls.

Return codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics

Table 125. SQLGetLength SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The combination of *LocatorCType* and *Locator* is not valid.
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.	Unable to allocate memory required to support execution or completion of the function.
HY003	Program type out of range.	*LocatorCType* is SQL_C_BLOB_LOCATOR
HY009	Invalid argument value.	Pointer to *StringLength* was NULL.
HY010	Function sequence error.	The specified *StatementHandle* is not in an *allocated* state. The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.
HY013	Unexpected memory handling error.	Unable to access memory required to support execution or completion of the function.
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.
0F001	The LOB token variable does not currently represent any value.	The value specified for *Locator* has not been associated with a LOB locator.

Restrictions

None.

Example

```
/* get the length of the whole BLOB data */
cliRC = SQLGetLength(hstmtLocUse,
                     SQL_C_BLOB_LOCATOR,
                     blobLoc,
                     &blobLen,
                     &ind);
```

SQLGetInfo—Get general information

Purpose

SQLGetInfo() returns general information (including supported data conversions) about the DBMS to which the application is connected.

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLGetInfo       (
                SQLHDBC          ConnectionHandle,   /* hdbc */
                SQLUSMALLINT      InfoType,      /* fInfoType */
                SQLPOINTER        InfoValuePtr,    /* rgbInfoValue */
                SQLSMALLINT       BufferLength,        /* cbInfoValueMax */
                SQLSMALLINT     *FAR StringLengthPtr,        /* pcbInfoValue */
```

Function arguments

Table 126. 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.
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.

- If the *InfoType* is not listed in Information Returned By SQLGetInfo, SQLGetInfo() returns SQLState HY096.

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 V`*`m.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_DEFAULT_TXN_ISOLATION (32-bit mask)

The default transaction isolation level supported. One of the following masks is returned:

- SQL_TXN_SERIALIZABLE = Data affected by pending transaction is not available to other transactions (repeatable read, phantoms are not possible) This is equivalent to IBM's Repeatable Read level.
- SQL_TXN_REPEATABLE_READ = A transaction can add or remove rows matching the search condition or a pending transaction (repeatable read, but phantoms are possible) This is equivalent to IBM's Read Stability level.
- SQL_TXN_READ_COMMITTED = Row read by transaction 1 can be altered and committed by transaction 2 (non-repeatable read and phantoms are possible) This is equivalent to IBM's Cursor Stability level.
- SQL_TXN_READ_UNCOMMITTED = Changes are immediately perceived by all transactions (dirty read, non-repeatable read, and phantoms are possible). This is equivalent to IBM's Uncommitted Read level.

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_TXN_ISOLATION_OPTION (32-bit mask)

The transaction isolation levels available at the currently connected database server. The following masks are used in conjunction with the flag to determine which options are supported:

- SQL_TXN_SERIALIZABLE = Data affected by pending transaction is not available to other transactions (repeatable read, phantoms are not possible) This is equivalent to IBM's Repeatable Read level.
- SQL_TXN_REPEATABLE_READ = A transaction can add or remove rows matching the search condition or a pending transaction (repeatable read, but phantoms are possible) This is equivalent to IBM's Read Stability level.

- SQL_TXN_READ_COMMITTED = Row read by transaction 1 can be altered and committed by transaction 2 (non-repeatable read and phantoms are possible) This is equivalent to IBM's Cursor Stability level.
- SQL_TXN_READ_UNCOMMITTED = Changes are immediately perceived by all transactions (dirty read, non-repeatable read, and phantoms are possible). This is equivalent to IBM's Uncommitted Read level.

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 166

"DB2 CLI function summary" on page 163

SQLGetStmtAttr—Get current setting of a statement attribute

Purpose

SQLGetStmtAttr() returns the current setting of a statement attribute.

Specification

- DB2 CLI 5.0
- ODBC 3.0
- ISO CLI

Syntax

```
SQLRETURN   SQLGetStmtAttr      (
                SQLHSTMT            StatementHandle,
                SQLINTEGER          Attribute,
                SQLPOINTER          ValuePtr,
                SQLINTEGER          BufferLength,
                SQLINTEGER         *StringLengthPtr);
```

Function arguments

Table 127 describes the types of arguments that are supported by SQLGetStmtAttr.

Table 127. 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*.

Table 127. SQLGetStmtAttr arguments (continued)

Data type	Argument	Use	Description
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 252.

- 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 128 describes the types of SQLSTATEs that are returned by SQLGetStmtAttr.

Table 128. 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.
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 166

"DB2 CLI function summary" on page 163
"SQLSetConnectAttr—Set options related to a connection" on page 247
"SQLSetStmtAttr—Set options related to a statement" on page 252

SQLGetSubString function (CLI) - Retrieve portion of a string value

Purpose

SQLGetSubString() is used to retrieve a portion of a large object value, referenced by a large object locator that has been returned from the server (returned by a fetch or a previous SQLGetSubString() call) during the current transaction.

Specification

- DB2 CLI 2.1

Syntax

```
SQLRETURN  SQLGetSubString   (
             SQLHSTMT          StatementHandle,   /* hstmt */
             SQLSMALLINT       LocatorCType,
             SQLINTEGER        SourceLocator,
             SQLUINTEGER       FromPosition,
             SQLUINTEGER       ForLength,
             SQLSMALLINT       TargetCType,
             SQLPOINTER        DataPtr,           /* rgbValue */
             SQLINTEGER        BufferLength,      /* cbValueMax */
             SQLINTEGER        *StringLength,
             SQLINTEGER        *IndicatorValue);
```

Function arguments

Table 129. SQLGetSubString arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
SQLSMALLINT	*LocatorCType*	input	The C type of the source LOB locator. This can be: • SQL_C_BLOB_LOCATOR
SQLINTEGER	*Locator*	input	*Locator* must be set to the source LOB locator value.
SQLUINTEGER	*FromPosition*	input	For BLOBs, this is the position of the first byte to be returned by the function.
SQLUINTEGER	*ForLength*	input	This is the length of the string to be returned by the function. For BLOBs, this is the length in bytes. If *FromPosition* is less than the length of the source string but *FromPosition* + *ForLength* - 1 extends beyond the end of the source string, the result is padded on the right with the necessary number of characters (X'00' for BLOBs).
SQLSMALLINT	*TargetCType*	input	The C data type of the *DataPtr*. This can be: • SQL_C_BINARY
SQLPOINTER	*DataPtr*	output	Pointer to the buffer where the retrieved string value or a LOB locator is to be stored.
SQLINTEGER	*BufferLength*	input	Maximum size of the buffer pointed to by *DataPtr* in bytes.

Table 129. SQLGetSubString arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	*StringLength*	output	The length of the returned information in *DataPtr* in bytes[a] if the target C buffer type is intended for a binary string variable and not a locator value. If the pointer is set to NULL, nothing is returned.
SQLINTEGER *	*IndicatorValue*	output	Always set to zero.

Usage

SQLGetSubString() is used to obtain any portion of the string that is represented by the LOB locator.

SQLGetSubString() can be used as an alternative to SQLGetData() for getting LOB data in pieces. In this case a column is first bound to a LOB locator, which is then used to fetch the LOB as a whole or in pieces.

The statement handle must not have been associated with any prepared statements or catalog function calls.

Return codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics

Table 130. SQLGetSubString SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The amount of data to be returned is longer than *BufferLength*. The actual length of data available for return is stored in *StringLength*.
07006	Invalid conversion.	The value specified for *TargetCType* was not SQL_C_BINARY. The value specified for *TargetCType* is inappropriate for the source.
22011	A substring error occurred.	*FromPosition* is greater than the of length of the source string.
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.	Unable to allocate memory required to support execution or completion of the function.
HY003	Program type out of range.	*LocatorCType* is not SQL_C_BLOB_LOCATOR.
HY009	Invalid argument value.	The value specified for *FromPosition* or for *ForLength* was not a positive integer.
HY010	Function sequence error.	The specified *StatementHandle* is not in an *allocated* state. The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.
HY013	Unexpected memory handling error.	Unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value of *BufferLength* was less than 0.

Table 130. SQLGetSubString SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.
0F001	No locator currently assigned	The value specified for *Locator* is not currently a LOB locator.

Restrictions

Only SQL_C_BINARY is supported for *TargetCType* and *LocatorCType* must be SQL_C_BLOB_LOCATOR.

Example

```
/* get the LOB locator */
sqlrc = SQLGetData(hstmt1, 1, SQL_C_BLOB_LOCATOR, (SQLPOINTER) &loc1, 0, &ind1);

/* get the length of the BLOB */
sqlrc = SQLGetLength(hstmt2, SQL_C_BLOB_LOCATOR, loc1, &len, NULL);

/* retrieve the first 26 bytes from the LOB locator into the variable data1 */
sqlrc = SQLGetSubString(hstmt2,SQL_C_BLOB_LOCATOR, loc1, 1, 26, SQL_C_BINARY,
                        data1, 52, &bufPos1, NULL);
```

SQLNumParams - Get number of parameters in anSQL statement

Purpose

SQLNumParams() returns the number of parameter markers in an SQL statement.

Specification

- DB2 CLI 2.1
- ODBC 1.0

Syntax

```
SQLRETURN   SQLNumParams     (SQLHSTMT          StatementHandle,
                              SQLSMALLINT  FAR  *ParameterCountPtr);
```

Function arguments

Table 131 describes the arguments that are supported by SQLNumParams.

Table 131. 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 132 describes the SQLSTATEs that are returned by SQLNumParams.

Table 132. 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

"SQLPrepare—Prepare a statement" on page 239

"SQLBindParameter—Bind a parameter marker to a buffer" on page 175

SQLNumResultCols—Get number of result columns

Purpose

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.

Specification

- DB2 CLI 1.1
- ODBC 1.0

Syntax

```
SQLRETURN   SQLNumResultCols (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLSMALLINT  FAR  *ColumnCountPtr);  /* pccol */
```

Function arguments

Table 133. SQLNumResultCols arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
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 134. 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 166
"DB2 CLI function summary" on page 163
"SQLBindCol—Bind a column to an application variable" on page 172
"SQLDescribeCol—Return a set of attributes for a column" on page 188
"SQLExecDirect—Execute a statement directly" on page 194
"SQLGetData—Get data from a column" on page 217

SQLParamData function (CLI) - Get next parameter for which a data value is needed

Purpose

SQLParamData() is used in conjunction with SQLPutData() to send long data in pieces. It can also be used to send fixed-length data at execution time.

Specification

- DB2 CLI 2.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLParamData (
               SQLHSTMT          StatementHandle, /* hstmt */
               SQLPOINTER        *ValuePtrPtr );  /* prgbValue */
```

Function arguments

Table 135. SQLParamData arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLPOINTER *	*ValuePtrPtr*	output	Pointer to a buffer in which to return the address of the *ParameterValuePtr* buffer specified in SQLBindParameter() (for parameter data) or the address of the *TargetValuePtr* buffer specified in SQLBindCol() (for column data), as contained in the SQL_DESC_DATA_PTR descriptor record field.

Usage

SQLParamData() returns SQL_NEED_DATA if there is at least one SQL_DATA_AT_EXEC parameter for which data still has not been assigned. This function returns an application-provided value in *ValuePtrPtr* supplied by the application during a previous SQLBindParameter() call. SQLPutData() is called one or more times (in the case of long data) to send the parameter data. SQLParamData() is called to signal that all the data has been sent for the current parameter and to advance to the next SQL_DATA_AT_EXEC parameter. SQL_SUCCESS is returned when all the parameters have been assigned data values and the associated statement has been executed successfully. If any errors occur during or before actual statement execution, SQL_ERROR is returned.

If SQLParamData() returns SQL_NEED_DATA, then only SQLPutData() or SQLCancel() calls can be made. All other function calls using this statement handle will fail. In addition, all function calls referencing the parent connection handle of *StatementHandle* will fail if they involve changing any attribute or state of that connection; that is, that following function calls on the parent connection handle are also not permitted:

- SQLSetConnectAttr()
- SQLEndTran()

Should they be invoked during an SQL_NEED_DATA sequence, these functions will return SQL_ERROR with SQLSTATE of **HY**010 and the processing of the SQL_DATA_AT_EXEC parameters will not be affected.

Return codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NEED_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NEED_DATA

Diagnostics

SQLParamData() can return any SQLSTATE returned by the SQLPrepare(), SQLExecDirect(), and SQLExecute() functions. In addition, the following diagnostics can also be generated:

Table 136. SQLParamData SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	Transfer of data between DB2 CLI and the application variables would result in incompatible data conversion.
22026	String data, length mismatch	The SQL_NEED_LONG_DATA_LEN information type in SQLGetInfo() was 'Y' and less data was sent for a long parameter (the data type was SQL_LONGVARCHAR, SQL_LONGVARBINARY, or other long data type) than was specified with the *StrLen_or_IndPtr* argument in SQLBindParameter(). The SQL_NEED_LONG_DATA_LEN information type in SQLGetInfo() as 'Y' and less data was sent for a long column (the data type was SQL_LONGVARCHAR, SQL_LONGVARBINARY, or other long data type) than was specified in the length buffer corresponding to a column in a row of data that was updated with SQLSetPos().
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to a deadlock or timeout.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by SQLGetDiagRec() in the argument *MessageText* 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.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, SQLCancel() was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	SQLParamData() was called out of sequence. This call is only valid after an SQLExecDirect() or an SQLExecute(), or after an SQLPutData() call. Even though this function was called after an SQLExecDirect() or an SQLExecDirect() call, there were no SQL_DATA_AT_EXEC parameters (left) to process.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY092	Option type out of range.	The *FileOptions* argument of a previous SQLBindFileToParam() operation was not valid.
HY506	Error closing a file.	Error encountered while trying to close a temporary file.
HY509	Error deleting a file.	Error encountered while trying to delete a temporary file.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

Restrictions

None.

Example

```
/* get next parameter for which a data value is needed */
cliRC = SQLParamData(hstmt, (SQLPOINTER *)&valuePtr);
```

SQLPrepare—Prepare a statement

Purpose

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().

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLPrepare       (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLCHAR      FAR  *StatementText,    /* szSqlStr */
                              SQLINTEGER        TextLength);       /* cbSqlStr */
```

Function arguments

Table 137 describes the types of arguments supported by SQLPrepare.

Table 137. 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 195.

Refer to the topic on the PREPARE statement in the DB2 Version 9.1 documentation for information about 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 138 describes the types of SQLSTATEs returned by SQLPrepare.

Table 138. 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 166
"DB2 CLI function summary" on page 163
"SQLBindParameter—Bind a parameter marker to a buffer" on page 175
"SQLDescribeCol—Return a set of attributes for a column" on page 188
"SQLExecDirect—Execute a statement directly" on page 194
"SQLExecute—Execute a statement" on page 195
"SQLNumResultCols—Get number of result columns" on page 235

SQLPrimaryKeys—Get primary key columns of a table

Purpose

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.

Specification

- DB2 CLI 2.1
- ODBC 1.0

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 139 describes the types of arguments that are supported by SQLPrimaryKeys.

Table 139. 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 140 describes the types of SQLSTATEs that are returned by SQLPrimaryKeys.

Table 140. 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 166

"DB2 CLI function summary" on page 163

"SQLForeignKeys—Get the list of foreign key columns" on page 205

SQLPutData function (CLI) - Passing data value for a parameter

Purpose

SQLPutData() is called following an SQLParamData() call returning SQL_NEED_DATA to supply parameter data values. This function can be used to send large parameter values in pieces.

Specification

- DB2 CLI 2.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLPutData       (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLPOINTER        DataPtr,           /* rgbValue */
               SQLINTEGER        StrLen_or_Ind);    /* cbValue */
```

Function arguments

Table 141. SQLPutData arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLPOINTER	*DataPtr*	Input	Pointer to the actual data, or portion of data, for a parameter. The data must be in the form specified in the SQLBindParameter() call that the application used when specifying the parameter.
SQLINTEGER	*StrLen_or_Ind*	Input	The length of *DataPtr*. Specifies the amount of data sent in a call to SQLPutData() . The amount of data can vary with each call for a given parameter. The application can also specify SQL_NTS or SQL_NULL_DATA for *StrLen_or_Ind*. *StrLen_or_Ind* is ignored for all fixed length C buffer types, such as date, time, timestamp, and all numeric C buffer types. For cases where the C buffer type is SQL_C_CHAR or SQL_C_BINARY, or if SQL_C_DEFAULT is specified as the C buffer type and the C buffer type default is SQL_C_CHAR or SQL_C_BINARY, this is the number of bytes of data in the *DataPtr* buffer.

Usage

The application calls SQLPutData() after calling SQLParamData() on a statement in the SQL_NEED_DATA state to supply the data values for an SQL_DATA_AT_EXEC parameter. Long data can be sent in pieces via repeated calls to SQLPutData(). DB2 CLI generates a temporary file for each SQL_DATA_AT_EXEC parameter to which each piece of data is appended when SQLPutData() is called. The path in which DB2 CLI creates its temporary files can be set using the TEMPDIR keyword in the db2cli.ini file. If this keyword is not set, DB2 CLI attempts to write to the path specified by the environment variables TEMP or TMP. After all the pieces of data for the parameter have been sent, the application calls SQLParamData() again to proceed to the next SQL_DATA_AT_EXEC parameter, or, if all parameters have data values, to execute the statement.

SQLPutData() cannot be called more than once for a fixed length C buffer type, such as SQL_C_LONG.

After an SQLPutData() call, the only legal function calls are SQLParamData(), SQLCancel(), or another SQLPutData() if the input data is character or binary data. As with SQLParamData(), all other function calls using this statement handle will fail. In addition, all function calls referencing the parent connection handle of *StatementHandle* will fail if they involve changing any attribute or state of that connection; that is, the following function calls on the parent connection handle are also not permitted:

- SQLSetConnectAttr()
- SQLEndTran()

Should they be invoked during an SQL_NEED_DATA sequence, these functions will return SQL_ERROR with SQLSTATE of **HY**010 and the processing of the SQL_DATA_AT_EXEC parameters will not be affected.

If one or more calls to SQLPutData() for a single parameter results in SQL_SUCCESS, attempting to call SQLPutData() with *StrLen_or_Ind* set to SQL_NULL_DATA for the same parameter results in an error with SQLSTATE of **22**005. This error does not result in a change of state; the statement handle is still in a *Need Data* state and the application can continue sending parameter data.

Return codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics

Some of the following diagnostics conditions might also be reported on the final SQLParamData() call rather than at the time the SQLPutData() is called.

Table 142. SQLPutData SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The data sent for a numeric parameter was truncated without the loss of significant digits. Timestamp data sent for a date or time column was truncated. Function returns with SQL_SUCCESS_WITH_INFO.
22001	String data right truncation.	More data was sent for a binary or char data than the data source can support for that column.
22003	Numeric value out of range.	The data sent for a numeric parameter caused the whole part of the number to be truncated when assigned to the associated column. SQLPutData() was called more than once for a fixed length parameter.
22005	Error in assignment.	The data sent for a parameter was incompatible with the data type of the associated table column.
22007	Invalid datetime format.	The data value sent for a date, time, or timestamp parameters was invalid.
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. 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.

Table 142. SQLPutData SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, SQLCancel() was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY009	Invalid argument value.	The argument *DataPtr* was a NULL pointer, and the argument *StrLen_or_Ind* was neither 0 nor SQL_NULL_DATA.
HY010	Function sequence error.	The statement handle *StatementHandle* must be in a need data state and must have been positioned on an SQL_DATA_AT_EXEC parameter via a previous SQLParamData() call.
HY019	Non-character and non-binary data sent in pieces.	The SQL_DATA_AT_EXEC parameter is none of the following data types: • BLOB • CHAR • VARCHAR • CHAR FOR BIT DATA • VARCHAR FOR BIT DATA
HY020	Attempt to concatenate a null value.	The SQL_DATA_AT_EXEC parameter was set to NULL by a previous call to SQLPutData().
HY090	Invalid string or buffer length.	The argument *DataPtr* was not a NULL pointer, and the argument *StrLen_or_Ind* was less than 0, but not equal to SQL_NTS or SQL_NULL_DATA.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

Restrictions

An additional value for *StrLen_or_Ind*, SQL_DEFAULT_PARAM, was introduced in ODBC 2.0, to indicate that the procedure is to use the default value of a parameter, rather than a value sent from the application. Because DB2 stored procedure arguments do not support default values, specification of this value for StrLen_or_Ind argument will result in an error when the CALL statement is executed. It happens because the SQL_DEFAULT_PARAM value is considered an invalid length.

ODBC 2.0 also introduced the SQL_LEN_DATA_AT_EXEC(*length*) macro to be used with the *StrLen_or_Ind argument*. The macro is used to specify the sum total length of the entire data that would be sent for character or binary C data via the subsequent SQLPutData() calls. Because the DB2 ODBC driver does not need this information, the macro is not needed. An ODBC application calls SQLGetInfo() with the SQL_NEED_LONG_DATA_LEN option to check if the driver needs this information. The DB2 ODBC driver will return 'N' to indicate that this information is not needed by SQLPutData().

Example

```
SQLCHAR buffer[BUFSIZ];
size_t n = BUFSIZ;

/* ... */

/* passing data value for a parameter */
cliRC = SQLPutData(hstmt, buffer, n);
```

related links

SQLRowCount—Get row count

Purpose

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.

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLRowCount      (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLINTEGER   FAR  *RowCountPtr);     /* pcrow */
```

Function arguments

Table 143 describes the types of arguments that are supported by SQLRowCount.

Table 143. 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 144 describes the types of SQLSTATEs that are returned by SQLRowCount.

Table 144. SQLRowCount SQLSTATEs

SQLSTATE	Description	Explanation
40003	Communication link failure.	The communication link between the application and data source failed before the function completed.
08S01		
58004	Unexpected system failure.	Unrecoverable system error.

Table 144. SQLRowCount SQLSTATEs (continued)

SQLSTATE	Description	Explanation
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 about 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 166
"DB2 CLI function summary" on page 163
"SQLExecDirect—Execute a statement directly" on page 194
"SQLExecute—Execute a statement" on page 195
"SQLNumResultCols—Get number of result columns" on page 235

SQLSetConnectAttr—Set options related to a connection

Purpose

Use the SQLSetConnectAttr() function to set connection options for the DB2 Everyplace database.

Specification

- DB2 CLI
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLSetConnectAttr   (SQLHDBC        ConnectionHandle,
                                 SQLINTEGER     Attribute,
                                 SQLPOINTER     ValuePtr,
                                 SQLINTEGER     StringLength);
```

Function arguments

Table 145 describes the types of arguments that are supported by SQLSetConnectAttr.

Table 145. SQLSetConnectAttr arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle.
SQLINTEGER	*Attribute*	input	Option to set.

Table 145. SQLSetConnectAttr 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, *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.
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

SQLSetConnectAttr() supports the following attributes:

SQL_ATTR_AUTOCOMMIT (DB2 CLI/ODBC)

A 32-bit integer value that specifies the mode type. The supported values are:

- SQL_AUTOCOMMIT_ON = DB2 Everyplace automatically commits each statement. This is the default.

  In autocommit mode, all updates that are performed by a statement are made persistent automatically after the statement is executed. Statement-level atomicity is guaranteed.

- SQL_AUTOCOMMIT_OFF = The application must manually, explicitly commit or rollback a transaction. You can commit or roll back a transaction by calling SQLEndTran(). For more information about using SQLEndTran(), see "SQLEndTran—Request a COMMIT or ROLLBACK" on page 192.

  In transaction 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 transaction mode, transactions can issue any SQL statement except for the REORG TABLE statement, which can only be issued in autocommit mode.

SQL_ATTR_BUFFERPOOL_SIZE

An integer that specifies the amount of memory, in bytes, that the DB2 Everyplace database should reserve for its bufferpools. If this value is not a multiple of 4K (4096 bytes), DB2 Everyplace rounds it down to the next smallest multiple of 4K.

DB2 Everyplace includes the following pre-defined bufferpool sizes:

Table 146. Pre-defined bufferpool size constants

Constant	Bufferpool size in bytes
SQL_BUFFERPOOL_SIZE_DEFAULT	The default value for the platform on which you are running DB2 Everyplace.
SQL_BUFFERPOOL_SIZE_64K	65 536
SQL_BUFFERPOOL_SIZE_128K	131 072
SQL_BUFFERPOOL_SIZE_256K	262 144
SQL_BUFFERPOOL_SIZE_512K	524 288
SQL_BUFFERPOOL_SIZE_1024K	1 048 576
SQL_BUFFERPOOL_SIZE_2048K	2 097 152
SQL_BUFFERPOOL_SIZE_4096K	4 194 304
SQL_BUFFERPOOL_SIZE_8172K	8 388 608
SQL_BUFFERPOOL_SIZE_1M	1 048 576
SQL_BUFFERPOOL_SIZE_2M	2 097 152
SQL_BUFFERPOOL_SIZE_4M	4 194 304
SQL_BUFFERPOOL_SIZE_8M	8 388 608

Important:

- The minimum value for SQL_ATTR_BUFFERPOOL_SIZE is SQL_BUFFERPOOL_SIZE_64K. If you call SQLSetConnectAttr() and specify a smaller value than SQL_BUFFERPOOL_SIZE_64K, SQlSetConnectAttr() returns SQLSTATE **HY**024.
- If the database engine cannot allocate as much memory as you specify in the SQL_ATTR_BUFFERPOOL_SIZE connection attribute, the engine will try to use a smaller bufferpool configuration. SQLConnect() will return SQLSTATE 01000.
- If there is not enough memory for the minimum bufferpool configuration, SQLConnect() will return SQLState 58004.
- You cannot change the size of the bufferpool if a connection to the database already exists. New connections will use the bufferpool size of the existing connection. SQLConnect() will return a warning.

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_DATABASE_ACCESS (DB2 Everyplace)

A 32-bit integer value that defines the database access mode. The supported values are:

- SQL_DB_EXCLUSIVE - a DB2 Everyplace data source accepts only one connection at a time. This is the default value.
- SQL_DB_SHARED - a DB2 Everyplace data source accepts multiple connections from one process at a time.

SQL_ATTR_FILENAME_FORMAT (DB2 Everyplace)

A 32-bit integer specifies whether the DB2 Everyplace 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. SQLSetConnectAttr returns SQL_ERROR with SQLState HY000 if it cannot change the filename format because the catalog files already exist. If an application connects to a path where DB2 Everyplace catalog files already exist, any attempts to change filename format will fail.

Catalog files are created during the very first CREATE TABLE statement. You cannot change the filename format after the catalog files are created. If an application connects to a path and attempts to change filename format after the first CREATE TABLE statement, SQLSetConnectAttr returns SQL_ERROR.

The supported attribute values are:

- SQL_FILENAME_FORMAT_LONG - files will be created in long filename format. This is the default.
- SQL_FILENAME_FORMAT_83- files will be created in 8.3 filename format.

SQL_ATTR_IO_MODE (DB2 Everyplace)

I/O flushes that update the database are handed to the operating system or pushed directly to the storage media after a COMMIT (in manual commit mode) or after completion of the SQL statement (in autocommit mode). An attribute changes this behavior on Windows and Linux on x86 only.

- SQL_IO_BUFFERED -Changes are sent to the operating system. You can recover the data if an application stops unexpectedly, but you might not be able to recover the data if the operating system stops. Applications using this mode (the default) will perform considerably faster.
- SQL_IO_WRITETHROUGH -Changes are sent directly to the storage media. You can recover the data if either the application or the operating system stops unexpectedly. This mode increases reliability but decreases performance. Use this mode when data integrity is important or if hardware or operating system failure is a concern.

SQL_ATTR_LOCK_TIMEOUT (DB2 Everyplace)

A 32-bit integer value corresponding to the number of seconds to wait for a lock to be released. The default value is 20 seconds.

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_LOGIN_TXN_ISOLATION (DB2 CLI/ODBC)

A 32-bit bitmask that sets the transaction isolation level for the current connection referenced by ConnectionHandle.The following values are accepted by DB2 CLI:

- SQL_TXN_READ_UNCOMMITTED - Dirty reads, non-repeatable reads, and phantom reads are possible.

- SQL_TXN_READ_COMMITTED - Dirty reads are not possible. Non-repeatable reads and phantom reads are possible. This is the default.
- SQL_TXN_REPEATABLE_READ - Dirty reads and reads that cannot be repeated are not possible. Phantoms are possible.
- SQL_TXN_SERIALIZABLE - Transactions can be serialized. Dirty reads, non-repeatable reads, and phantoms are not possible.
- In IBM terminology:
- SQL_TXN_READ_UNCOMMITTED is Uncommitted Read
- SQL_TXN_READ_COMMITTED is Cursor Stability
- SQL_TXN_REPEATABLE_READ is Read Stability
- SQL_TXN_SERIALIZABLE is Repeatable Read.

You cannot specify this option while there is an open cursor on any statement handle or an outstanding transaction for this connection. A SQL_ERROR is returned on the function call (SQLSTATE HY011) in this situation.

SQL_ATTR_TABLE_CHECKSUM

A 32-bit integer value that defines whether the DB2 Everyplace database engines should use checksums to verify that none of the files that DB2 Everyplace uses to store a database has been altered or corrupted. You can change this attribute only before establishing a database connection. Any changes you make to this attribute will only take effect when connecting to a directory that does not yet contain any catalog tables. After the first CREATE TABLE statement all the database files will be stored with checksums enabled. The supported values are:

- SQL_TABLE_CHECKSUM_OFF - DB2 Everyplace stores files without checksum information. This is the default value.
- SQL_TABLE_CHECKSUM_ON - DB2 Everyplace stores files with checksum information.

SQL_ATTR_TEMP_DIR (DB2 Everyplace)

A null-terminated ASCII character string that specifies the path of a writable temporary directory. DB2 Everyplace cannot access read-only databases unless this path is valid.

Set this attribute before connecting to the database. If you are using multiple connections to access different read-only databases simultaneously, specify a different temporary directory for each connection. You can retrieve the value of this attribute with the SQLGetConnectAttr() function before and after you establish a connection.

If you do not explicitly set a temporary directory, DB2 Everyplace uses the default path. Table 147 describes the default temporary directories for each platform.

Table 147. Default temporary directories

Platform	Directory
Linux and Neutrino	/tmp
Palm	main memory
Symbian	C:\
Windows	path specified by the TEMP or TMP environment variable
Windows CE	\

Example: To connect to a read only database and using a temporary directory of c:\temp, use:

```
SQLSetConnectAttr(hdbc, SQL_ATTR_TEMP_DIR, "c:\\temp\\", SQL_NTS);
```

Return codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

- SQL_INVALID_HANDLE

Diagnostics

Table 148 describes the types of SQLSTATEs that are returned by SQLSetConnectAttr.

Table 148. 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.
HY011	Operation invalid at this time.	The argument Attribute was SQL_ATTR_TXN_ISOLATION and a transaction or cursor was open.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to internal resources.
HY024	Invalid attribute value.	Given the specified *Attribute* value, an invalid value is specified in *ValuePtr*.
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

- Transactions were added to DB2 Everyplace to allow consistent updating and insertion of several related records in a number of tables. DB2 Everyplace writes changes to the data tables after the application commits the transaction. Use the SQL_ATTR_IO_MODE parameter to tell DB2 Everyplace whether updates to the database are handed to the operating system or pushed directly to the storage media.
- If the application stops prematurely without committing the current transaction, the updates within that transaction are rolled back automatically.
- After the SQLEndTran() function 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* updated with the information from that transaction.
  - 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 166

"DB2 CLI function summary" on page 163

"SQLEndTran—Request a COMMIT or ROLLBACK" on page 192

SQLSetStmtAttr—Set options related to a statement

Purpose

SQLSetStmtAttr() sets attributes related to a statement.

Specification

- DB2 CLI 2.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLSetStmtAttr   (SQLHSTMT         StatementHandle,
                              SQLINTEGER       Attribute,
                              SQLPOINTER       ValuePtr,
                              SQLINTEGER       StringLength);
```

Function arguments

Table 149. SQLSetStmtAttr arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement 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, 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_IS_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 150. 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 151. 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 152. 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 166

"DB2 CLI function summary" on page 163

SQLSetCursorName—Set cursor name

Purpose

SQLSetCursorName() associates a cursor name with the statement handle. This function is optional because DB2 CLI implicitly generates a cursor name. The implicit cursor name is available after the dynamic SQL has been prepared on the statement handle.

Specification

- DB2 CLI 1.1
- ODBC 1.0
- ISO CLI

Syntax

```
SQLRETURN   SQLSetCursorName (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      *FAR CursorName,        /* szCursor */
                SQLSMALLINT       NameLength,      /* cbCursorMax */);
```

Function arguments

Table 153. SQLSetCursorName arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLCHAR *	*CursorName*	output	Cursor name
SQLSMALLINT	*NameLength*	input	Length of buffer *CursorName*

Usage

DB2 CLI always generates and uses and internally-generated cursor name when a query is prepared or executed directly. SQLSetCursorName() allows an application-defined cursor name to be used. DB2 CLI maps this name to the internal name. The name will remain associated with the statement handle until the handle is dropped or another SQLSetCursorName() is called on this statement handle.

Cursor names most follow these rules:

- All cursor names within the connection must be unique.
- Each cursor name must be less than or equal to 128 bytes in length.
- Because internally-generated names begin with CUR, the application must not input a cursor name starting with CUR in order to avoid conflicts with internal names.

For efficient processing applications should not include any leading or trailing spaces in the *CursorName* buffer. If the *CursorName* buffer contains a delimited identifier, applications should position the first double quote as the first character in the *CursorName* buffer.

Return codes

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics

Table 154. SQLSetCursorName SQLSTATEs

SQLSTATE	Description	Explanation
34000	Invalid cursor name	The cursor name that is specified by the argument *CursorName* was invalid. The cursor name either beings with CUR or violates the cursor naming rules.
HY009	Invalid argument value	The cursor name that is specified by the argument *CursorName* is a null pointer.
58004	Unexpected system failure.	Unrecoverable system error.

Restrictions

None.

SQLTables - Get table information

Purpose

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.

Specification

- **DB2 CLI** 2.1
- **ODBC** 1.0

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 155 describes the types of arguments that are supported by the SQLTables function.

Table 155. SQLTables arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.

Table 155. SQLTables arguments (continued)

Data type	Argument	Use	Description
SQLCHAR	*CatalogName*	Input	Buffer that can 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 can 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 can 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: 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 156 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. because 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 156 describes the types of columns that are returned by the SQLTables function.

Table 156. 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 156. Columns Returned By SQLTables (continued)

Column Name	Data type	Description
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 157 describes the types of SQLSTATEs that are returned by the SQLTables function.

Table 157. 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 225

SQLState messages reported by CLI

Table 158 describes the SQLState messages reported by CLI and the function that returns each type of message.

Table 158. 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.)

Table 158. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
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 *FetchOffset* 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.)
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).

Table 158. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
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.
08003	SQLGetConnectAttr	The connection does not exist	The connection does not exist.
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.
0F001	SQLGetLength		The value specified for Locator has not been associated with a LOB locator.
0F001	SQLGetSubString		The value specified for Locator has not been associated with a LOB locator.
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.
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*
22011	SQLGetSubstring	Substring error	A substring error occurred; for example, an argument of SUBSTR is out of range.
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.

Table 158. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
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.
54028	SQLGetData	Max LOB handles reached	The maximum number of concurrent LOB handles has been reached.
58004	SQLBindCol	Unexpected system failure	Unrecoverable system error.
58004	SQLBindParameter	Unexpected system failure	Unrecoverable system error.
58004	SQLConnect	Unexpected system failure	Unrecoverable system error.
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.

Table 158. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
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.
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.

Table 158. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
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.
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	SQLExecute	Invalid argument value	See API for detailed explanation.
HY009	SQLForeignKeys	Invalid argument value	See API for detailed explanation.
HY009	SQLGetLength	Invalid argument value	See API for detailed explanation.
HY009	SQLGetSubString	Invalid argument value	See API for detailed explanation.
HY009	SQLNumResultCols	Invalid argument value	*StatementText* is a null pointer.
HY009	SQLPrepare	Invalid argument value	See API for detailed explanation.
HY009	SQLPutData	Invalid argument value	See API for detailed explanation.
HY009	SQLSetCursorName	Invalid argument value	See API for detailed explanation.
HY010	SQLDescribeCol	Function sequence error	The function is called prior to calling SQLPrepare() or SQLExecDirect() for the *StatementHandle* .

Table 158. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
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*.
HY 011	SQLSetConnectAttr	Operation invalid at this time	The argument attribute was SQL_ATTR_TXN_ISOLATION and a transaction or cursor was open.
HY012	SQLEndTran	Invalid transaction code	See API for detailed explanation.
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.

Table 158. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
HY013	SQLExecDirect	Unexpected memory handling error	DB2 CLI is unable to access memory required to support execution or completion of the function.
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.
HY019	SQLPutData	Non-character and non-binary data set in pieces.	See API for detailed explanation.
HY020	SQLPutData	Attempt to concatenate a null value.	See API for detailed explanation.
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.

Table 158. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
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
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).
HY096	SQLGetInfo	Information type out of range.	See API for detailed explanation.
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* .

Table 158. SQLState messages reported by CLI (continued)

SQLSTATE	CLI function name	Description	Explanation
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.
HYT00	SQLConnect	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.

Related reference

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"Summary of SQLState class codes" on page 445

DB2 Everyplace Sync Client Interface

This topic explains the functions that are provided by the DB2 Everyplace Sync Client.

Java Sync API supported operating systems

Java Sync APIs are available on the following operating systems:

- Windows (Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows 2003)
- 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 IBM Sync Java interface in order to integrate DB2 Everyplace mobile database and DB2 Everyplace 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 <DSYPATH>\Clients\javadoc directory, where <DSYPATH> is the directory where DB2 Everyplace is installed.

Related concepts

"Sample JDBC database engine applications" on page 26
This topic describes the DB2eAppl.java and the DB2eJavaCLP.java sample applications for the DB2 Everyplace database engine.

Related tasks

"Developing DB2 Everyplace Java applications" on page 22
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.

Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2

This topic summarizes the major changes made to the DB2 Everyplace 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 DB2 Everyplace Sync 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 DB2 Everyplace 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-ISYN) 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 DB2 Everyplace 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 DB2 Everyplace Sync Client API Version 7.2.1 is still supported through an API wrapper (the isync 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: \Program 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 (isync) library does need to be installed if choose to use DB2 Everyplace Sync Client API Version 8.1.

Table 159 lists the major differences between the functions in the DB2 Everyplace Sync Client C-API Version 8.1 and the DB2 Everyplace Sync Client Version 7.2.

Table 159. The DB2 Everyplace 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 DB2 Everyplace Sync Client C/C++ application" on page 19

Related tasks

"Developing DB2 Everyplace Sync Client applications using C/C++" on page 18
This topic provides an overview of how to develop DB2 Everyplace Sync Client applications using C/C++ based on the IBM DB2 Everyplace Sync Client C API.

Related reference

"Key to DB2 Everyplace Sync Client C-API function descriptions" on page 282

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary"
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

DB2 Everyplace Sync Client C-API function summary

DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

Table 160. DB2 Everyplace Sync Client C-API function list

Function name	Purpose
iscGetVersion	Gets the version number of the DB2 Everyplace Sync Client C-API.

Table 161 describes the IBM Service API functions supported by DB2 Everyplace and includes the purpose of each function.

Table 161. 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 162 describes the IBM Configuration API functions supported by DB2 Everyplace and includes the purpose of each function.

Table 162. 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 163 describes the IBM Sync Engine API functions supported by DB2 Everyplace and includes the purpose of each function.

Table 163. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related tasks

"Developing DB2 Everyplace Sync Client applications using C/C++" on page 18
This topic provides an overview of how to develop DB2 Everyplace Sync Client applications using C/C++ based on the IBM DB2 Everyplace Sync Client C API.

Related reference

"Key to DB2 Everyplace Sync Client C-API function descriptions" on page 282

"DB2 Everyplace Sync Client C-API data types"

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

DB2 Everyplace Sync Client C-API data types

Table 164 lists the new data types defined by the DB2 Everyplace Sync Client C-API. When calling the C-API functions, make sure that the argument type complies with the prototype of the functions.

Table 164. Data types for IBM DB2 Everyplace 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

Table 164. Data types for IBM DB2 Everyplace Sync Client C-API (continued)

Data type	Description
isy_TCHAR	Character type
isy_BOOL	Boolean type
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.

Table 164. Data types for IBM DB2 Everyplace Sync Client C-API (continued)

Data type	Description

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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related tasks

"Developing DB2 Everyplace Sync Client applications using C/C++" on page 18
This topic provides an overview of how to develop DB2 Everyplace Sync Client applications using C/C++ based on the IBM DB2 Everyplace Sync Client C API.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"Key to DB2 Everyplace Sync Client C-API function descriptions" on page 282

DB2 Everyplace Sync Client C-API constants and error codes

The following tables list the constants and the error codes that are defined by the DB2 Everyplace Sync Client C-API. The error codes are also found in the isyncore.h file.

The following table lists the buffers and their size limits.

Table 165. Buffer size limits

Name	Value	Description
ISCLEN_SubsSetID	30	Maximum length in characters of subscription set ID
ISCLEN_SubsSetName	20	Maximum length in characters of subscription set name
ISCLEN_SubsName	20	Maximum length in characters of subscription name
ISCLEN_Table	20	Maximum length in characters of the table name
ISCLEN_Trace	2	Deprecated. Maximum length in characters of the ISCPREF_Trace preference value when it is passed into the iscEngineSetPref() function. The ISCPREF_Trace value can be set to either `Y` or `N`.
ISCLEN_Timeout	12	Deprecated. Maximum length in characters of the ISCPREF_Timeout preference value when it is passed into the iscEngineSetPref() function. The ISCPREF_Timeout preference value is expressed in seconds.

The following table lists the preferences for the Sync engine and their values.

Table 166. Sync engine preferences

Name	Value	Description
ISCPREF_Trace	1	Deprecated. This is the preference ID for setting detailed tracing. You can set the preference value to `ISCCONST_TraceON` or `ISCCONST_TraceOFF`.
ISCPREF_Timeout	2	Deprecated. This is the preference ID for setting the timeout length for receiving messages. The preference value is a string in seconds.
ISCCONST_TraceON	y	Deprecated. This is the preference value for ISCPREF_Trace. Detailed trace is turned on.
ISCCONST_TraceOFF	n	Deprecated. This is the preference value for ISCPREF_Trace. Detailed trace is turned off.
ISCCONST_TimeoutNever	-1	No timeout.
ISCCONST_TimeoutMinimum	0	Timeout is the minimum value

The following table lists the conflict operations and their causes.

Table 167. Conflict operations

Name	Value	Description
ISCCONST_OpDelete	1	Delete operation caused a conflict row
ISCCONST_OpInsert	2	Insert operation caused a conflict row.
ISCCONST_OpUpdate	3	Update operation caused a conflict row.

The following table lists the properties that are passed into the iscServiceOpenEx() function.

Table 168. Properties passed into iscServiceOpenEx()

Name	Value	Description
ISCPROP_SyncUser	isync.user	(Required) DB2 Everyplace Sync Server username. Maximum length is 254 characters.
ISCPROP_SyncPassword	isync.password	(Required) DB2 Everyplace Sync Server password. Maximum length is 254 characters.
ISCPROP_SyncEncoding	isync.encoding	(Optional) Codepage of synchronized data. Maximum length is 22 characters.
ISCPROP_SyncMessageSize	isync.messagesize	(Optional) Message size to and from DB2 Everyplace Sync Server in bytes. Maximum length is 8 characters.
ISCPROP_SyncMessageDump	isync.messagedump	(Optional) Dump transport messages in the DB2 Everyplace Sync Client trace. The value is either`yes` or `no`.
ISCPROP_SyncProxy	isync.proxy	(Optional) Synchronization through proxy. The value has the format `address:port`. Maximum length is 48 characters.
ISCPROP_SyncTrace	isync.trace	(Optional) Trace setting. The value is either `detailed` or `default`.

Table 168. Properties passed into iscServiceOpenEx() (continued)

Name	Value	Description
ISCPROP_SyncTimeout	isync.timeout	(Optional) Timeout setting. The value is an integer and is expressed in seconds. Maximum length is 8 characters.
ISCPROP_SyncCreateImage	isync.createimage	(Optional) Create a distributable client image upon successful synchronization. Value is `yes` or `no`.
ISCPROP_SyncAllowBackup	isync.allowbackup	(Optional) Allow synchronization from a backed-up client image. The value is either `yes` or `no`.
ISCPROP_SyncRetryInterval	isync.retryinterval	(Optional) The length of time that a client sleeps between retry requests when the server is busy. The value is in seconds.
ISCPROP_SyncDBUrl	isync.db.url	(Optional) URL of the target database.
ISCPROP_SyncDBUser	isync.db.user	(Optional) User to access any encrypted tables in the client database.
ISCPROP_SyncDBPassword	isync.db.password	(Optional) Password to access any encrypted tables in the client database.
ISCPROP_TargetDBUrl	target.db.url	(Optional) URL of target database.
ISCPROP_TargetDBUser	target.db.user	(Optional) User to access any encrypted tables in the target database.
ISCPROP_TargetDBPassword	target.db.password	(Optional) Password to access any encrypted tables in the target database.
ISCPROP_DB2eWriteThrough	db2e.writethrough	(Optional) Feature for that causes changes to be directly written to the storage media. Value is `yes` or `no`.

The following table lists the return codes for the DB2 Everyplace Sync Client API and their values.

Table 169. DB2 Everyplace Sync Client API return codes

Name	Value
ISCRTN_Empty	2
ISCRTN_True	1
ISCRTN_False	0
ISCRTN_Ready	1
ISCRTN_Succeeded	0
ISCRTN_Failed	-1
ISCRTN_Canceled	-2
ISCRTN_NotFound	-3
ISCRTN_UnknownID	-4
ISCRTN_ValTooLong	-5
ISCRTN_ValTruncated	-6
ISCRTN_OutOfMemory	-7
ISCRTN_ResourceBusy	-8
ISCRTN_NotPermitted	-9

The following table lists the return codes for the Sync listener and their values.

Table 170. Sync listener return codes

Name	Value
ISCRTNCB_Default	0
ISCRTNCB_Done	1
ISCRTNCB_ReplyNo	2
ISCRTNCB_ReplyYes	3

The following table lists the event types and their values.

Table 171. Event types

Name	Value
ISCEVTTYPE_Info	1
ISCEVTTYPE_Conflict	2
ISCEVTTYPE_Query	3
ISCEVTTYPE_Retry	4
ISCEVTTYPE_Error	5
ISCEVTTYPE_Fatal	6

The following table lists the different types of subscriptions and their values.

Table 172. Subscription types

Name	Value
ISCSUBSTYPE_Config	100
ISCSUBSTYPE_File	101
ISCSUBSTYPE_DB2e	102
ISCSUBSTYPE_Custom	10000

The following tables lists the informational events and their values.

Table 173. Informational events

Name	Value
ISCEVT_InfGeneral	1000
ISCEVT_InfSyncStarted	1001
ISCEVT_InfPrepMsg	1002
ISCEVT_InfSendMsg	1003
ISCEVT_InfWaitMsg	1004
ISCEVT_InfApplyMsg	1005
ISCEVT_InfCancelingSync	1006
ISCEVT_InfSubsSetStarted	1007
ISCEVT_InfSyncingSubs	1008
ISCEVT_InfSubsSetFailed	1009
ISCEVT_InfSubsSetCanceled	1010
ISCEVT_InfSubsSetSucceeded	1011

Table 173. Informational events (continued)

Name	Value
ISCEVT_InfSyncSucceeded	1012
ISCEVT_InfSyncFailed	1013
ISCEVT_InfSyncCanceled	1014
ISCEVT_InfSyncProg	1015
ISCEVT_InfNoNewChange	1016
ISCEVT_InfLoginFailed	1017
ISCEVT_InfSyncMsgSent	1018
ISCEVT_InfStats	1019
ISCEVT_InfDBTimeStats	1020
ISCEVT_InfDBRowSent	1021
ISCEVT_InfDBRowRecv	1022
ISCEVT_InfDBBuildTimeStats	1023

The following table lists the different conflict events and their values.

Table 174. Conflict events

Name	Value
ISCEVT_CftReject	2000

The following table lists the error events and their values.

Table 175. Error events. During an error event, the synchronization process might continue to the next subscription set if one exists.

Name	Value
ISCEVT_ErrOpenAdapter	300
ISCEVT_ErrLoadAdapter	301
ISCEVT_ErrCloseAdapter	302
ISCEVT_ErrAuthenticateKey	306
ISCEVT_ErrClientCryptoFailed	307
ISCEVT_ErrEncryptNotAvail	308
ISCEVT_ErrEncryptLibOpen	309
ISCEVT_ErrSubsNotFound	311
ISCEVT_ErrSubsNotAvail	312
ISCEVT_ErrSubsDefAltered	316
ISCEVT_ErrAllocResource	400
ISCEVT_ErrConnectData	401
ISCEVT_ErrDisconnectData	402
ISCEVT_ErrNoData	403
ISCEVT_ErrSyncDisabled	417
ISCEVT_ErrServerException	418
ISCEVT_ErrMessageFormat	412

Table 175. Error events (continued). During an error event, the synchronization process might continue to the next subscription set if one exists.

Name	Value
ISCEVT_ErrNotFound	413
ISCEVT_ErrEndOfData	414
ISCEVT_ErrDataTooLong	415
ISCEVT_ErrReadOnly	420
ISCEVT_ErrOperation	421
ISCEVT_ErrUnauthorized	423
ISCEVT_ErrNotAvailable	424
ISCEVT_ErrNotSupported	425
ISCEVT_ErrSubsTargetDir	426
ISCEVT_ErrCloseNetLib	608
ISCEVT_ErrOutOfMemory	610
ISCEVT_ErrInternal	698

The following table lists the fatal events and their values.

Table 176. Fatal events. A fatal event causes the synchronization process to stop immediately.

Name	Value
ISCEVT_FatSyncCfgAbort	303
ISCEVT_FatAuthenticateFailed	304
ISCEVT_FatIncompVersion	310
ISCEVT_FatInvalidSession	313
ISCEVT_FatSyncGroup	314
ISCEVT_FatRegisterDevice	315
ISCEVT_FatCreateImage	317
ISCEVT_FatServerForbidden	611
ISCEVT_FatServerNotFound	612
ISCEVT_FatServerNotAvail	614
ISCEVT_FatServer	613
ISCEVT_FatServerBusy	616
ISCEVT_FatNetOpenConn	600
ISCEVT_FatNetConnect	601
ISCEVT_FatNetSend	602
ISCEVT_FatNetReceive	603
ISCEVT_FatNetTimeout	604
ISCEVT_FatOpenNetLib	606
ISCEVT_FatResolveHost	609
ISCEVT_FatProtocolNotSupported 615 ISCEVT_FatNetUnknown	699

The following table lists the retry events and their values.

Table 177. Retry events

Name	Value
ISCEVT_TryServerBusy	4616
ISCEVT_TryNetConnect	4601
ISCEVT_TryNetSend	4602
ISCEVT_TryNetReceive	4603
ISCEVT_TryNetTimeout	4604

The following table lists the different query events and their values.

Table 178. Query events

Name	Value
ISCEVT_QueCancel	5000
ISCEVT_QueLogin	5001
ISCEVT_QueCancelUponError	5002

"iscEngineSetPref() - sets the preferences of the synchronization engine" on page 312
"iscEngineGetPref() - retrieves the current preference setting" on page 314
"iscServiceOpenEx() - open a new service handle based on a property array" on page 286
"iscEngineListenerPF() - defines the prototype for use with iscEngineSetListener" on page 305

Key to DB2 Everyplace Sync Client C-API function descriptions

Descriptions for each DB2 Everyplace Sync Client C-API function contain the following topics:

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 DB2 Everyplace Sync Client C-API function.

References
Lists related DB2 Everyplace Sync Client C-API functions.

Note: There is no Diagnostics topic in the DB2 Everyplace Sync Client C-API.

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related tasks

"Developing DB2 Everyplace Sync Client applications using C/C++" on page 18
This topic provides an overview of how to develop DB2 Everyplace Sync Client applications using C/C++ based on the IBM DB2 Everyplace Sync Client C API.

Related reference

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"DB2 Everyplace Sync Client C-API data types" on page 274

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

iscGetVersion() - gets the version number of the DB2 Everyplace Sync Client C-API

Purpose

iscGetVersion() gets the version number of the DB2 Everyplace Sync Client C-API.

Syntax

```
isy_UINT32 iscGetVersion();
```

Function arguments

None.

Usage

iscGetVersion() is used to retrieve the version number of the DB2 Everyplace 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 DB2 Everyplace Sync Client C-API version number.

Restrictions

None.

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"Key to DB2 Everyplace Sync Client C-API function descriptions" on page 282
"DB2 Everyplace Sync Client C-API data types" on page 274
"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271
"iscEngineGetInfo() - gets general information about the synchronization engine" on page 303

iscServiceOpen() - opens a new service handle

Purpose

iscServiceOpen() opens a new service handle. To use non-default values for properties such as timeout, see iscServiceOpenEx().

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 179 lists the valid arguments used with the iscServiceOpen() function.

Table 179. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"iscServiceOpenEx() - open a new service handle based on a property array" on page 286

"iscServiceClose() - closes an opened service handle"

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"DB2 Everyplace Sync Client C-API data types" on page 274

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

iscServiceClose() - closes an opened service handle

Purpose

iscServiceClose() closes an opened service handle.

Syntax

```
isy_INT32 iscServiceClose(
    HISCSERV  hServ);
```

Function arguments

Table 180 lists the valid argument used with the iscServiceClose() function.

Table 180. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"iscServiceOpen() - opens a new service handle" on page 284

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"DB2 Everyplace Sync Client C-API data types" on page 274

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

iscServiceOpenEx() - open a new service handle based on a property array

Purpose

iscServiceOpenEx() opens a new service handle that is based on a property array.

Syntax

```
isy_INT32 iscServiceOpenEx(
    isy_CONST isy_TCHAR* URL,
    ISCPROPERTY*         property,
    isy_INT32            propertyNum,
    HISCSERV*            phServ);
```

Function arguments

Table 181 lists arguments for the `iscServiceOpenEx()` function.

Table 181. iscServiceOpenEx() arguments

Data type	Argument	Use	Description
isy_CONST isy_TCHAR	URL	input	Server information as a URL string

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;`
			
			The following properties are available:
			
			isync.user
			DB2 Everyplace Sync Client user name. This property is required.
			
			isync.password
			DB2 Everyplace Sync Client password. This property is required.
			
			isync.encoding
			Character encoding of the target data. See Table 182 on page 288 for a complete list of supported encodings.
			
			Windows and Palm have default encodings of Cp1252. Windows Unicode, Windows CE, and Symbian should only use ″UnicodeLittle″ (default). Linux and QNX Neutrino should use ″UTF-8″ (default).

Table 181. iscServiceOpenEx() arguments (continued)

Data type	Argument	Use	Description
ISCPROPERTY (continued)			**isync.messagesize** Message size (in bytes) Default values for Palm and Windows CE: 64 000″ Default values for other platforms: ″128 000″ The actual value might be smaller, depending on available free memory of the mobile device. Minimum message size for all platforms: ″4 096″ Maximum message size for Palm: ″64 000″ Maximum message size for non-Palm: ″5 000 000″ **isync.proxy** Proxy IP address and port number **isync.trace** DB2 Everyplace Sync Client trace level (″default″ or ″detailed″) **isync.timeout** Timeout length in seconds for message communication. The default value is 60 seconds. If you specify 0, DB2 Everyplace will use the smallest timeout that is possible. If you specify -1, DB2 Everyplace will not use a timeout. **target.db.url** Default target database for relational data. The path must be a relative path. If you specify a directory, it must end in a slash or backslash, depending on the platform. If omitted, the default target database is under the current directory. The application can overrule this default target database for each subscription by responding to the ISCEVT_QueSubsTarget callback. **target.db.username** User name for the default target database that is specified by the ″target.db.url″ property. **target.db.password** Password for the default target database that is specified by the ″target.db.url″ property. **isync.db.url** Client control database for keeping synchronization statistics and trace data. The path must be a relative path. If you specify a directory, it must end in a slash or backslash, depending on the platform. If you do not specify a directory, the DB2 Everyplace Sync Client uses the default target database.

Table 181. iscServiceOpenEx() arguments (continued)

Data type	Argument	Use	Description
ISCPROPERTY (continued)			**isync.db.username** User name for the client control database that is specified by the ″isync.db.url″ property. **isync.db.password** Password for the client control database that is specified by the ″isync.db.url″ property. **db2e.writethrough** Use this property to enable or disable the writethrough feature. When you set this property is to ″yes,″ changes are sent directly to the storage media. You can recover the data if either the application or the operating system stops unexpectedly. This mode increases reliability but decreases performance. DB2 Everyplace does not offer media recovery. If you set the property to ″no″, DB2 Everyplace sends changes to the operating system. You can recover data if an application stops unexpectedly, but you might not be able to recover data if the operating system stops. Applications using this mode perform considerably faster.
isy_INT32	propertyNum	input	Number of properties.
HISCSERV	phServ	output	Handle to a service.

Table 182. Encodings that are supported by the isync.encoding property

Encoding type	Description
ASCII	ASCII
Big5	Big5 CP950 Chinese (Taiwan)
Cp1250	Windows Eastern European
Cp1251	Windows Cyrillic
Cp1252	Windows Latin-1
Cp1253	Windows Greek
Cp1254	Windows Turkish
Cp1255	Windows Hebrew
Cp1256	Windows Arabic
Cp1257	Windows Baltic, UDB/Latvia, Lithuanian
Cp1258	Windows Vietnamese
EUC_JP	EUC_JP Japanese
EUC_KR	EUC_KR Korean
GB18030	GB18030 Simplified Chinese (mainland)
GB2312	Chinese (mainland)
ISO2022JP	ISO2022JP Japanese
ISO2022KR	ISO2022KR Korean

Table 182. Encodings that are supported by the isync.encoding property (continued)

Encoding type	Description
ISO8859_1	ISO8859_1 Latin1 including Albanian and Catalan
ISO8859_2	ISO8859_2 Czech, Hungarian, Polish, Croatian, Romanian, Slovak, Slovenian
ISO8859_4	ISO8859_4 Estonian, Latvian, Lithuanian
ISO8859_5	ISO8859_5 Russian, Turkish, Bulgarian, Belarusian, Macedonian, Serbian, Ukranian, Kazakh
ISO8859_6	ISO8859_6 Arabic
ISO8859_7	ISO8859_7 Greek
ISO8859_8	ISO8859_8 Hebrew
ISO8859_9	ISO8859_9 UDB/Turkish
SJIS	Shift-JIS Japanese
TIS620	TIS620 Thai
UnicodeLittle	UNICODE Little-Endian
UTF8	UTF-8

Usage

iscServiceOpenEx() is used to request a new handle for a specific service from a server with settings that are represented as a property array. The server is identified by a string, which can contain the protocol, the host name (or IP address), and the port number. If the DB2 Everyplace Sync Server is configured for Secure Socket Layer (SSL), specify "https://" for the protocol; otherwise, specify "http://". You can omit the port number. The default port for SSL is 443. The default port for non-ssl is 80. 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[12] = {
           {"isync.user", "myUserName"},
           {"isync.password", "myPassword"},
           {"isync.encoding", "ISO8859_1"},
           {"isync.trace", "detailed"},
           {"isync.timeout", "120"}, // 2 minutes
           {"isync.messagesize", "65536"}, // 64K Bytes
           {"isync.db.url", "CtrlDB"}, // client control database
           {"isync.db.username", "dbUserName"}, // username to CtrlDB
           {"isync.db.password", "dbPassword"}, // password to CtrlDB
           {"target.db.url", "SyncDB"}, // default target database
           {"target.db.username", "dbUserName"},// username to SyncDB
           {"target.db.password", "dbPassword"}}// password to SyncDB
rc = iscServiceOpenEx("http://localhost.mycom.com:8080",
        properties, 12, &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 : Other error

Related reference

"iscConfigClose() - closes an opened config store connection" on page 291

iscConfigOpen() - opens a connection to the configuration store

Purpose

iscConfigOpen() opens a connection to the configuration store.

Syntax

```
isy_INT32 iscConfigOpen(
    HISCSERV   hServ,
    isy_TCHAR *path,
    HISCCONF  *phConf);
```

Table 183 lists the default path that DB2 Everyplace uses if the *path argument is an empty string.

Table 183. Default paths for iscConfigOpen()

Platform	Default path
Palm	Main memory
Windows	Current directory
Linux	Current directory
QNX Neutrino	Current directory
Windows CE	\
Symbian	C:\

Function arguments

Table 184 lists the valid arguments that are used with the iscConfigOpen() function.

Table 184. iscConfigOpen() arguments

Data type	Argument	Use	Description
HISCSERV	*hServ*	input	Service handle
isy_TCHAR*	*path*	input	Path of the working directory
HISCCONF*	*phConf*	output	Configuration connection

Usage

iscConfigOpen() opens a connection to the configuration store that is specified in the given path for a specific service. Upon success, a configuration connection (HISCCONF) returns through *phServ. Otherwise, *phServ is NULL, and DB2 Everyplace returns an error code. If this is a new service (either a new host or a new port), a new empty configuration is created for that service.

Return codes

Table 185 describes the codes that are returned by iscConfigOpen().

Table 185. Codes that are returned by iscConfigOpen()

Code	Description
ISCRTN_Succeeded	OK

Table 185. Codes that are returned by iscConfigOpen() (continued)

Code	Description
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	Other error

Restrictions

None.

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigClose() - closes an opened config store connection"

iscConfigClose() - closes an opened config store connection

Purpose

iscConfigClose() closes an opened config store connection.

Syntax

```
isy_INT32 iscConfigClose(
    HISCCONF    hConf);
```

Function arguments

Table 186 lists the valid argument used with the iscConfigClose() function.

Table 186. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related tasks

"Developing DB2 Everyplace Sync Client applications using C/C++" on page 18
This topic provides an overview of how to develop DB2 Everyplace Sync Client applications using C/C++ based on the IBM DB2 Everyplace Sync Client C API.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigOpen() - opens a connection to the configuration store" on page 290

iscConfigPurge() - empties subscription information from config store

Purpose

iscConfigPurge() empties all of the subscription information from the config store.

Syntax

```
isy_INT32 iscConfigPurge(
    HISCCONF    hConf);
```

Function arguments

Table 187 lists the valid argument used with the iscConfigPurge() function.

Table 187. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigResetSubsSet() - resets a subscription set" on page 297

iscConfigOpenCursor() - gets a cursor

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 188 lists the valid arguments used with the iscConfigOpenCursor() function.

Table 188. iscConfigOpenCursor() arguments

Data type	Argument	Use	Description
HSYNCCONF	*hConf*	input	Config connection
HISCCSR*	*phCursor*		

iscConfigCloseCursor() - disposes an opened cursor

Purpose

iscConfigCloseCursor() disposes an opened cursor.

Syntax

```
isy_INT32 iscConfigCloseCursor(
    HISCCONF    hConf,
    HISCCSR     hCursor);
```

Function arguments

Table 189 lists the valid arguments used with the iscConfigCloseCursor() function.

Table 189. 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 because this can cause unexpected errors.

Return codes

- ISCRTN_Succeeded : OK
- ISCRTN_Failed : Otherwise

Restrictions

None.

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related tasks

"Developing DB2 Everyplace Sync Client applications using C/C++" on page 18
This topic provides an overview of how to develop DB2 Everyplace Sync Client applications using C/C++ based on the IBM DB2 Everyplace Sync Client C API.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigOpenCursor() - gets a cursor" on page 293

"iscConfigGetNextSubsSet() - moves cursor to the next subscription set and gets its description"

iscConfigGetNextSubsSet() - moves cursor to the next subscription set and gets its description

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 190 lists the valid arguments used with the iscConfigGetNextSubsSet() function.

Table 190. iscConfigGetNextSubsSet() arguments

Data type	Argument	Use	Description
HISCCONF	*hConf*	input	Config connection
HISCCSR	*hCursor*	input	Cursor for iterating subscription sets

Table 190. iscConfigGetNextSubsSet() arguments (continued)

Data type	Argument	Use	Description
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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigSubsSetIsReset() - preforms a query if a set is in reset mode" on page 299

"iscConfigSubsSetIsEnabled() - queries if a set is enabled for synchronization" on page 298

iscConfigEnableSubsSet() - enables a subscription set in the config for synchronization

Purpose

iscConfigEnableSubsSet() enables a subscription set in the config for synchronization.

Syntax

```
isy_INT32 iscConfigEnableSubsSet(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments

Table 191 lists the valid arguments used with the iscConfigEnableSubsSet() function.

Table 191. 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

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigDisableSubsSet() - disables a subscription set"

"iscConfigSubsSetIsEnabled() - queries if a set is enabled for synchronization" on page 298

iscConfigDisableSubsSet() - disables a subscription set

Purpose

iscConfigDisableSubsSet() disables the synchronization on a subscription set.

Syntax

```
isy_INT32 iscConfigDisableSubsSet(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments

Table 192 lists the valid arguments used with the iscConfigDisableSubsSet() function.

Table 192. 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

disables a subscription set

None.

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigEnableSubsSet() - enables a subscription set in the config for synchronization" on page 296

"iscConfigSubsSetIsEnabled() - queries if a set is enabled for synchronization" on page 298

iscConfigResetSubsSet() - resets a subscription set

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 193 lists the valid arguments used with the iscConfigResetSubsSet() function.

Table 193. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigSubsSetIsReset() - preforms a query if a set is in reset mode" on page 299

iscConfigSubsSetIsEnabled() - queries if a set is enabled for synchronization

Purpose

iscConfigSubsSetIsEnabled() queries if a subscription set is enabled for synchronization.

Syntax

```
isy_INT32 iscConfigSubsSetIsEnabled(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments

Table 194 lists the valid arguments used with the iscConfigSubsSetIsEnabled() function.

Table 194. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigSubsSetIsReset() - preforms a query if a set is in reset mode"

iscConfigSubsSetIsReset() - preforms a query if a set is in reset mode

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 195 lists the valid arguments used with the iscConfigSubsSetIsReset() function.

Table 195. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscConfigSubsSetIsEnabled() - queries if a set is enabled for synchronization" on page 298

iscConfigGetSubsSetStatus() - gets the synchronization status of a subscription set

Purpose

iscConfigGetSubsSetStatus() gets the synchronization status of a subscription set.

Syntax

```
isy_INT32 iscConfigGetSubsSetStatus(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

Function arguments

Table 196 lists the valid arguments used with the iscConfigGetSubsSetStatus() function.

Table 196. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscEngineSync() - launches a synchronization session" on page 315

"iscConfigSubsSetIsEnabled() - queries if a set is enabled for synchronization" on page 298

iscEngineOpen() - opens a handle to the synchronization engine

Purpose

iscEngineOpen() opens a handle to the synchronization engine.

Syntax

```
isy_INT32 iscEngineOpen(
    HISCCONF     hConf,
    HISCENG     *phEngine);
```

Function arguments

Table 197 lists the valid arguments used with the `iscEngineOpen()` function.

Table 197. 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() because multiple calls open multiple handles to the synchronization engine and might cause consistency problems.

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscEngineClose() - closes an opened handle to the synchronization engine"

iscEngineClose() - closes an opened handle to the synchronization engine

Purpose

iscEngineClose() closes an opened handle to the synchronization engine.

Syntax

```
isy_INT32 iscEngineClose(
   HISCENG    hEngine);
```

Function arguments

Table 198 lists the valid argument used with the iscEngineClose() function.

Table 198. 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

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscEngineOpen() - opens a handle to the synchronization engine" on page 301

iscEngineGetInfo() - gets general information about the synchronization engine

Purpose

iscEngineGetInfo() gets general information about the synchronization engine.

Syntax

```
isy_INT32 iscEngineGetInfo(
    HISCENG       hEngine,
    isy_TCHAR    *info,
    isy_INT32     infoLen);
```

Function arguments

Table 199 lists the valid arguments used with the iscEngineGetInfo() function.

Table 199. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscGetVersion() - gets the version number of the DB2 Everyplace Sync Client C-API" on page 283

iscEngineSetListener() - registers the user-defined listener function with the synchronization engine

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 200 lists the valid arguments used with the iscEngineSetListener() function.

Table 200. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscEngineSync() - launches a synchronization session" on page 315

iscEngineListenerPF() - defines the prototype for use with iscEngineSetListener

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 201 lists the valid arguments used with the `iscEngineSetListenerPF` function type.

Table 201. 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-defined 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 that contains various information about that event.

Table 202 lists all the fields in the event structure and the purpose of each field.

Table 202. 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.

Table 202. iscEngineListenerPF event fields (continued)

Field	Description
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 203 lists all the event codes by category of event type.

Table 203. 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.

Table 203. iscEngineListenerPF event codes (continued). Event type: ISCEVTTYPE_Info

Event code	Event info (argc)	Description
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 204. 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 205. iscEngineListenerPF event codes. Event type: ISCEVTTYPE_Retry

Event code	Event info (argc)	Description
ISCEVT_TryNetConn (4601)	NULL	Failed to connect to the server, ask if should try again.
ISCEVT_TrySendRequest (4602)	NULL	Failed to send requests to the server, ask if should try again.
ISCEVT_TryRecvReply (4603)	NULL	Failed to receive replies from the server, ask if should try again.
ISCEVT_TryRecvTimeout (4604)	NULL	Timed out while receiving replies from the server, ask if should try again.
SCEVT_TryRecvAck (4605)	NULL	Server is busy, ask if should try again.

Table 206. 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)

Table 206. iscEngineListenerPF event codes (continued). Event type: ISCEVTTYPE_Query

Event code	Event info (argc)	Description
ISCEVT_QueLogin (5002)	ISCLISTENARG(3)	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).
	info->argv[0]	
	info->argv[1]	Target name of data source
	info->argv[2]	Blank buffer for holding the user name
		Blank buffer for holding the password
ISCEVT_QueSubsTarget (5003)	ISCLISTENARG(1)	Database information requested by an adapter. The listener can provide the requested information in the event info and return ISCRTNCB_Done or return ISCRTNCB_Default to use the default target directory.
	info->argv[0]	
		Directory for subscription.

Table 207. 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	The server does not recognize the format of the client message.
ISYNCEVT_ErrClientCryptoFailed (307)	NULL	The DB2 Everyplace Sync Client could not successfully encrypt the outgoing message or decrypt the received message.
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 because 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)	Requested data not found.
	info->argv[0]	Target name of data source
	info->argv[1]	Data name
ISCEVT_ErrEndOfData (414)	NULL	Unexpected end-of-data.

Table 207. iscEngineListenerPF event codes (continued). Event type: ISCEVTTYPE_Error

Event code	Event info (argc)	Description
ISCEVT_ErrDataTooLong (415)	ISCLISTENARG(3)	Data is too long and is truncated.
	info->argv[0]	Target name of data source
	info->argv[1]	Data name
	info->argv[2]	Data element name (if not empty)
ISCEVT_ErrSyncDisabled (417)	NULL	Server reported that the user is not enabled.
ISCEVT_ErrServerException (418)	NULL	Server reported exceptions.
ISCEVT_ErrReadOnly (420)	ISCLISTENARG(2)	Attempted to upload changes to read-only data.
	info->argv[0]	
	info->argv[1]	
ISCEVT_ErrOperation (421)	NULL	Attempted to upload impermissible operations to the server.
ISCEVT_ErrUnauthorized (423)	NULL	Not authorized to access the target data.
ISCEVT_ErrNotAvailable (424)	ISCLISTENARG(2)	Requested data not available.
	info->argv[0]	Target name of data source
	info->argv[1]	Data name
ISCEVT_ErrNotSupported (425)	ISCLISTENARG(3)	Requested data is not supported.
	info->argv[0]	Target name of data source
	info->argv[1]	Data name
	info->argv[2]	Data element name (if not empty)
ISCEVT_ErrSubsTargetDir (426)	NULL	The target database (path) provided in the ISCEVT_QueSubsTarget event is invalid, for example, an absolute path.
ISCRTN_ErrCloseNetLib (608)	NULL	Failed to close the network library
ISCEVT_ErrOutOfMemory (610)	NULL	Out of memory.
ISCEVT_ErrInternal (698)	ISCLISTENARG(1)	Other internal errors occurred.
	info->argv[0]	Error state (as a string).

Table 208. 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 DB2 Everyplace Sync Client version.
ISCEVT_FatInvalidSession (313)	NULL	Attempt to synchronize from a backed-up client.
ISCEVT_FatSyncGroup (314)	NULL	User is not associated with any group with the synchronization privilege
ISCEVT_FatRegisterDevice (315)	NULL	Failed to register the device for the user.
ISCEVT_FatCreateImage (317)	NULL	Attempt to create client image without a valid device ID
ISCEVT_FatNetOpenConn (600)	NULL	Failed to open a connection to the server.

Table 208. iscEngineListenerPF event codes (continued). Event type: ISCEVTTYPE_Fatal

Event code	Event info (argc)	Description
ISCEVT_FatNetConnect(601)	NULL	Failed to connect to the server
ISCEVT_FatNetSend(602)	NULL	Failed to send messages to the server.
ISCEVT_FatNetReceive(603)	NULL	Failed to receive messages from the server.
ISCEVT_FatNetTimeout(604)	NULL	Timed out on receiving the messages from 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_FatServerBusy (616)	NULL	The DB2 Everyplace Sync Server is busy.
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.
- 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscEngineSync() - launches a synchronization session" on page 315

"iscEngineSetListener() - registers the user-defined listener function with the synchronization engine" on page 304

iscEngineSetPref() - sets the preferences of the synchronization engine

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 209 lists the valid arguments used with the iscEngineSetPref() function.

Table 209. iscEngineSetPref() arguments

Data type	Argument	Use	Description
HISCENG	*hEngine*	input	Handle to the synchronization engine

Table 209. iscEngineSetPref() arguments (continued)

Data type	Argument	Use	Description
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. Use iscServiceOpenEx with respective properties for the trace and timeout settings.

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscEngineGetPref() - retrieves the current preference setting"

iscEngineGetPref() - retrieves the current preference setting

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 210 lists the valid arguments used with the iscEngineGetPref() function.

Table 210. 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.
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. Use iscServiceOpenEx with respective properties for the trace and timeout settings.

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

"DB2 Everyplace Sync Client C-API data types" on page 274

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"iscEngineSetPref() - sets the preferences of the synchronization engine" on page 312

iscEngineSync() - launches a synchronization session

Purpose

iscEngineSync() launches a synchronization session.

Syntax

```
isy_INT32 iscEngineSync(
    HISCENG       hEngine);
```

Function arguments

Table 211 lists the valid argument used with the iscEngineSync() function.

Table 211. 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 DB2 Everyplace 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

Release all connections to all involved databases before invoking the IBM DB2 Everyplace Sync Client API, because during synchronization, the sync engine opens an exclusive connection to the target database. During synchronization, attempts to connect to the database that is being used by the sync engine will fail.

Related concepts

"The sample DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"iscEngineSyncConfig() - launches a synchronization session that synchronizing only the configuration"

"iscConfigPurge() - empties subscription information from config store" on page 292

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"DB2 Everyplace Sync Client C-API data types" on page 274

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

iscEngineSyncConfig() - launches a synchronization session that synchronizing only the configuration

Purpose

iscEngineSyncConfig() launches a synchronization session that synchronizing only the configuration.

Syntax

```
isy_INT32 iscEngineSyncConfig(
    HISCENG        hEngine);
```

Function arguments

Table 212 lists the valid argument used with the iscEngineSyncConfig() function.

Table 212. 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 DB2 Everyplace Sync Client C/C++ application" on page 19
This example illustrates how to use a select number of DB2 Everyplace Sync Client API functions for building an application.

Related reference

"iscConfigPurge() - empties subscription information from config store" on page 292

"iscEngineSync() - launches a synchronization session" on page 315

"DB2 Everyplace Sync Client C-API function summary" on page 273
DB2 Everyplace Sync Client C-API function list describes the DB2 Everyplace Sync Client C-API functions supported by DB2 Everyplace and includes the purpose of each function.

"DB2 Everyplace Sync Client C-API data types" on page 274

"Comparisons between DB2 Everyplace Sync Client C-API Version 8.1 and Version 7.2" on page 271

JDBC Interface

This topic explains the methods that are provided by the JDBC Interface.

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 about 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 319 and "DB2eConnection class" on page 319 for more information.

The JDBC Optional Package for CDC/Foundation Profile (JSR 169) can be used to run DB2 Everyplace Java applications. However, DB2 Everyplace does not support all of the methods defined in JSR 169 (for example, DB2 Everyplace does not support the CLOB type, so it does not support the Clob interface).

Related tasks

"Developing DB2 Everyplace Java applications" on page 22
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.

Related reference

"Blob interface" on page 321

"Connection interface" on page 323

"DatabaseMetaData interface" on page 325

"Driver interface" on page 335

"DB2eConnection class" on page 319

"DB2eStatement class" on page 319

"PreparedStatement interface" on page 337

"ResultSet interface" on page 339

"ResultSetMetaData interface" on page 344

"SQLState messages reported by JDBC" on page 441
"Statement interface" on page 345

Restrictions for table subscriptions

DB2 Everyplace has several restrictions on table subscriptions. Consider the following limitations when you plan and deploy your mobile applications.

- If you are using a source database that is running on Informix Dynamic Server, each table in a subscription should be created by using an explicit schema qualifier.
- If you are adding a table to a subscription, its source and target schema names, column names, and table names cannot be keywords, reserved words, or special registers in SQL, DB2 Version 9.1 and the source database that you are using.
- If the structure of a source table needs to be changed with either an ALTER TABLE statement or DROP TABLE and CREATE TABLE statements, follow these steps:
  1. Remove the table from all subscriptions.
  2. Execute the ALTER TABLE statement or the DROP TABLE and CREATE TABLE statements.
  3. Add the table back to the subscriptions.
- The DB2 Everyplace Sync Client does not support encryption for more than one target database.
- The IBM Cloudscape client does not support all features that the DB2 Everyplace client supports. For example, multiple server, ordering of subscription sets/subscriptions/tables, and local data encryption are not supported.
- Referential integrity is not supported for DataPropagator table subscriptions.
- Always replicate DataPropagator table subscriptions on the mirror database server. This means that if replication needs to be performed during administrative actions, the Mobile Devices Administration Center must be run on the mirror database server.
- A DataPropagator subscription cannot use the same mirror database that is used by a JDBC subscription.
- DB2 Everyplace does not support database object names that must be enclosed in double quotes.
- In general, the maximum size of a row in a table is limited by the data source. Adding the table to a JDBC or DataPropagator subscription further restricts the maximum row size. The additional restriction on the maximum row size is approximately 125 bytes.
- The DB2 Everyplace Sync Server does not synchronize times or timestamps with an hour of 24 correctly due to differences in how time is represented in Java and a data source. A time of ″24:00:00″ is converted to ″00:00:00″ and causes the data saved in the mobile database to differ from the data saved in the source database. You should avoid using such an hour in your applications.
- A mirror database can replicate with only one source database. DB2 Everyplace does not allow a mirror database to be associated with multiple source databases.
- DB2 Everyplace converts table and column names from the source database to uppercase when applying them in the mirror and target databases. In rare cases, this can result in an inability to create a table due to non-uniqueness of the table name or column name. In other rare cases it can result in a conversion of a character to multiple characters, as in the German "sharp s" (ß), which will be converted to "SS".
- When a JDBC subscription is created on DB2 Version 9.1 for z/OS, DB2 Everyplace creates tables and indices on these tables in the source database. DB2 Everyplace creates the indices with the default attributes. Thus, DB2 Version 9.1 for z/OS assigns the index data sets to the default storage groups and gives them default space attributes. Because this might affect the performance of mass inserts, you can improve performance by altering the attributes of the indices that start with the prefix DSY.

  Example: To alter the attributes of the DSYI19099874167 index to the following parameters:
  - Maximum addressability of each data set: 256 megabytes
  - Storage group for the data set: DB2ERSTG
  - Minimum primary space allocation for the data set: 80 000 kilobytes

– Minimum secondary space allocation data set: 40 000 kilobytes

issue the following command:

```
ALTER INDEX DSYI19099874167 PIECESIZE 256M using
  STOGROUP DB2ERSTG
  PRIQTY 80000
  SECQTY 40000
```

Referential integrity constraints

If your source tables have referential integrity constraints, observe the following restrictions to avoid synchronization and replication failures due to referential integrity constraint violations:

- Do not update the primary keys on the client database or mobile device.
- Do not subscribe tables which have parent-child relationships that involve cycles (for example self-loops).
- When you create subscriptions, add tables in parent-to-child order.
- Referential integrity relationships cannot cross any subscription boundaries.
- Do not subscribe to tables that have triggers, unless the tables are the target of an upload subscription

com.ibm.db2e.jbdc Interface

This topic explains the classes provided by the JDBC com.ibm.db2e.jbdc package.

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.

com.ibm.db2e.jdbc package

public class **DB2eConnection**

implements Connection

The following table lists the methods in the DB2eConnection class.

Table 213. DB2eConnection class methods

Method return type	Method
`int`	**getLockTimeout()**Returns the number of seconds before a lock request times out.
`int`	**getBufferpoolSize()** Returns the size, in bytes, of the DB2 Everyplace connection bufferpool.
`boolean`	**isEnabledFilenameFormat83()** Returns true if the database engine creates filenames in 8.3 format. Returns false if it creates filenames in long format.
`boolean`	**isEnabledIOWritethrough()**Returns true if the engine pushes changes to disk without delay.
`boolean`	**isEnabledSharedDatabaseAccess()** Returns true if the connection allows shared access to tables.
`boolean`	**isEnabledTableChecksum()** Returns true if the database uses files with checksums.

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 163 for more information.

com.ibm.db2e.jdbc package

public class **DB2eStatement**

implements Statement

Table 214 lists the methods in the DB2eStatement class.

Table 214. 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 22
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.

Related reference

"SQLSetStmtAttr—Set options related to a statement" on page 252

"SQLGetStmtAttr—Get current setting of a statement attribute" on page 229

"SQLState messages reported by JDBC" on page 441

"Overview of DB2 Everyplace JDBC support" on page 317

Java.sql Interface

This topic explains the packages provided by the java.sql package.

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 215 lists the methods in the Blob interface that are supported by DB2 Everyplace.

Table 215. 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.
`long`	**position(Blob pattern, long start)** Retrieves the byte position in the BLOB value designated by this Blob object at which pattern begins.
`long`	**position(byte[] pattern, long start)** Retrieves the byte position at which the specified byte array pattern begins within the BLOB value that this Blob object represents.
`OutputStream`	**setBinaryStream(long pos)** Retrieves a stream that can be used to write to the BLOB value that this Blob object represents.
`void`	**setBinaryStream(int parameterIndex, InputStream x, int length)**
`int`	**setBytes(long pos, byte[] bytes)** Writes the given array of bytes to the BLOB value that this Blob object represents, starting at position pos, and returns the number of bytes written.

Table 215. Blob interface methods (continued)

Method return value type	Method
`int`	**setBytes(long pos, byte[] bytes, int offset, int len)** Writes all or part of the given byte array to the BLOB value that this Blob object represents and returns the number of bytes written.
`void`	**truncate(long len)** Truncates the BLOB value that this Blob object represents to be len bytes in length.

Related tasks

"Developing DB2 Everyplace Java applications" on page 22
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.

Related reference

"SQLState messages reported by JDBC" on page 441

"Overview of DB2 Everyplace JDBC support" on page 317

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 remote query and stored procedure adapter" on page 49 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 216 lists the methods in the CallableStatement interface that are supported by DB2 Everyplace.

Table 216. CallableStatement interface methods

Method return value type	Method
@ `BigDecimal`	**getBigDecimal(int parameterIndex)** JDBC 2.0 Gets the value of a JDBC decimal parameter as a `BigDecimal` object in the Java programming language.
`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.

Table 216. CallableStatement interface methods (continued)

Method return value type	Method
`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.
`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 22
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.

Related reference

"SQLState messages reported by JDBC" on page 441

"Overview of DB2 Everyplace JDBC support" on page 317

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 217 lists the methods in the Connection interface that are supported by DB2 Everyplace.

Table 217. Connection interface methods

Method return value type	Method
`void`	**clearWarnings()** Clears all warnings reported for this `Connection` object.

Table 217. Connection interface methods (continued)

Method return value type	Method
`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 because the previous commit or rollback permanent and releases any database locks currently held by the `Connection`.
`Blob`	**createBlob()** Creates a `Blob` object. The object that is returned contains no data. To add data to the `Blob` object, use the `setBinaryStream` and `setBytes` methods of the `Blob` interface.
`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`	**getAutoCommit()** JDBC 4.0 Retrieves the current autocommit mode for this Connection object.
`DatabaseMetaData`	**getMetaData()** Gets the metadata regarding this `Connection`'s database.
`int`	**getTransactionIsolation()** Gets the transaction isolation level of this `Connection` object.
`SQLWarning`	**getWarnings()** Returns the first warning reported by calls on this `Connection`.
`boolean`	**isClosed()** Tests to see if a `Connection` is closed.
`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.
`Savepoint`	**setSavepoint()** Creates an unnamed savepoint in the current transaction and returns the new `Savepoint` object that represents it.
`Savepoint`	**setSavepoint(String name)** Creates a savepoint with the given name in the current transaction and returns the new `Savepoint` object that represents it.

Table 217. Connection interface methods (continued)

Method return value type	Method
`void`	**rollback()** Drops all changes made because the previous commit or rollback and releases any database locks currently held by this `Connection` object.
`void`	**releaseSavepoint(Savepoint savepointname)** Removes the given `Savepoint` object from the current transaction.
`void`	**rollback(Savepoint savepointname)** Drops all changes made after the given `Savepoint` object was set.
`void`	**setAutoCommit(boolean autoCommit)** Sets this `Connection`'s autocommit mode.
`void`	**setTransactionIsolation(int level)** Attempts to change the transaction isolation level of this `Connection` object.

Related tasks

"Developing DB2 Everyplace Java applications" on page 22
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.

Related reference

"SQLState messages reported by JDBC" on page 441

"Overview of DB2 Everyplace JDBC support" on page 317

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 218 lists the fields in the DatabaseMetaData interface that are supported by DB2 Everyplace.

Table 218. 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.

Table 218. DatabaseMetaData fields (continued)

Field type	Field
`static int`	**columnNullableUnknown** Indicates that the nullability of columns is unknown.

Table 219 lists the methods in the DatabaseMetaData interface that are supported by DB2 Everyplace.

Table 219. DatabaseMetaData interface methods

Method return value type	Method
`boolean`	**allProceduresAreCallable()** Can the current user call all the procedures returned by the method getProcedures?
`boolean`	**allTablesAreSelectable()** Can the current user use all the tables returned by the method getTables in a SELECT statement?
`boolean`	**dataDefinitionCausesTransactionCommit()** Can a data definition statement within a transaction force the transaction to commit?
`boolean`	**dataDefinitionIgnoredInTransactions()** Does this database ignore a data definition statement within a transaction?
`boolean`	**deletesAreDetected(int type)** Can a visible row delete be detected by calling the method ResultSet.rowDeleted?
`boolean`	**doesMaxRowSizeIncludeBlobs()** Does the return value for the method getMaxRowSize include the SQL data types LONGVARCHAR and LONGVARBINARY?
`String`	**getCatalogSeparator()** Gets the String that this database uses as the separator between a catalog and table name.
`ResultSet`	**getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)** Gets a description of table columns available in the specified catalog. The ResultSet returned by this method is based on the JDK 1.3 specification and has 18 columns.
`Connection`	**getConnection()** JDBC 2.0 Gets the connection that produced this metadata object.
`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.
`int`	**getDatabaseMajorVersion()** JDBC 3.0 Gets the database's major version number.

Table 219. DatabaseMetaData interface methods (continued)

Method return value type	Method
`int`	**getDatabaseMinorVersion()** JDBC 3.0 Gets the database's minor version number.
`String`	**getDatabaseProductName()** Gets the name of this database product.
`String`	**getDatabaseProductVersion()** Gets the version of this database product.
`int`	**getDefaultTransactionIsolation()** Gets the database's default transaction isolation level.
`int`	**getDriverMajorVersion()** Gets the JDBC driver's major version number.
`int`	**getDriverMinorVersion()** Gets the JDBC driver's minor version number.
`String`	**getDriverName()** What is the Gets the name of this JDBC driver.
`String`	**getDriverVersion()** Gets the version of this JDBC driver.
`String`	**getIdentifierQuoteString()** Gets 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).
`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).
`int`	**getJDBCMajorVersion()** JDBC 3.0 Gets the JDBC driver's major version number.
`int`	**getJDBCMinorVersion()** JDBC 3.0 Gets the JDBC driver's minor version number.
`int`	**getMaxBinaryLiteralLength()** Gets the maximum amount of hex characters in an inline binary literal.
`int`	**getMaxCatalogNameLength** Gets the maximum number of characters in a catalog name.
`int`	**getMaxCharLiteralLength()** Gets the maximum length for a character literal.

Table 219. DatabaseMetaData interface methods (continued)

Method return value type	Method
int	**getMaxColumnNameLength()** Gets the limit of the column name length.
int	**getMaxColumnsInGroupBy()** Gets the maximum number of columns in a GROUP BY clause.
int	**getMaxColumnsInIndex()** Gets the maximum number of columns allowed in an index.
int	**getMaxColumnsInOrderBy()** Gets the maximum number of columns in an ORDER BY clause.
int	**getMaxColumnsInSelect()** Gets the maximum number of columns in a SELECT statement.
int	**getMaxColumnsInTable** Gets the maximum number of columns this database allows in a table.
int	**getMaxConnections()** Gets the maximum amount of active connections to this database at one time.
int	**getMaxCursorNameLength** Gets the maximum number of characters that this database allows in a cursor name.
int	**getMaxIndexLength()** Gets the maximum length of an index (in bytes).
int	**getMaxProcedureNameLength** Gets the maximum number of characters that this database allows in a procedure name.
int	**getMaxRowSize()** Gets the maximum length of a single row.
int	**getMaxSchemaNameLength** Gets the maximum number of characters that this database allows in a schema name.
int	**getMaxStatementLength()** Gets the maximum length of a SQL statement.
int	**getMaxStatements()** Gets the maximum amount of active statements that can be opened at one time on this database.
int	**getMaxTableNameLength()** Gets the maximum length of a table name.
int	**getMaxTablesInSelect()** Gets the maximum number of tables in a SELECT statement.
int	**getMaxUserNameLength()** What is the maximum length of a user name?

Table 219. DatabaseMetaData interface methods (continued)

Method return value type	Method
`ResultSet`	**getPrimaryKeys(String catalog, String schema, String table)** Gets a description of a table's primary key columns.
`int`	**getResultSetHoldability** JDBC 3.0 What is the default holdability of this ResultSet object?
`String`	**getSearchStringEscape()** Gets the string that can be used to escape wildcard characters.
`int`	**getSQLStateType()** JDBC 3.0 Returns a value to indicate whether the SQLSTATE returned by SQLException.getSQLState is X/Open (now known as Open Group) SQL CLI or SQL99.
`ResultSet`	**getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)** Gets a description of tables available in a catalog. The ResultSet returned by this method is based on the JDK 1.3 specification and has 5 columns.
`String`	**getURL()** Gets the URL for this database.
`String`	**getUserName()** Gets the user name as it is known to the database.
`boolean`	**insertsAreDetected(int type)** Can a visible row insert be detected by calling the method ResultSet.rowInserted?
`boolean`	**isCatalogAtStart()** Does a catalog appear at the start of a fully-qualified table name?
`boolean`	**isReadOnly()** Is this database in read-only mode?
`boolean`	**nullPlusNonNullIsNull()** Does this database support concatenations between NULL and non-NULL values being NULL?
`boolean`	**nullsAreSortedAtEnd()** Are NULL values sorted at the end regardless of sort order?
`boolean`	**nullsAreSortedAtStart()** Are NULL values sorted at the start regardless of sort order?
`boolean`	**nullsAreSortedHigh()** Are NULL values sorted high?
`boolean`	**nullsAreSortedLow()** Are NULL values sorted low?
`boolean`	**othersDeletesAreVisible(int type)** Are deletes made by others visible?

Table 219. DatabaseMetaData interface methods (continued)

Method return value type	Method
boolean	**othersInsertsAreVisible(int type)** Are inserts made by others visible?
boolean	**othersUpdatesAreVisible(int type)** Are updates made by others visible?
boolean	**ownDeletesAreVisible(int type)** Are a result set's own deletes visible?
boolean	**ownInsertsAreVisible(int type)** Are a result set's own inserts visible?
boolean	**ownUpdatesAreVisible(int type)** Are a result set's own updates visible?
boolean	**storesLowerCaseIdentifiers()** Does this database treat mixed-case unquoted SQL identifiers as case insensitive and store them in lower case?
boolean	**storesLowerCaseQuotedIdentifiers()** Does this database treat mixed-case quoted SQL identifiers as case insensitive and store them in lower case?
boolean	**storesMixedCaseIdentifiers()** Does this database treat mixed-case unquoted SQL identifiers as case insensitive and store them in mixed case?
boolean	**storesMixedCaseQuotedIdentifiers()** Does this database treat mixed-case quoted SQL identifiers as case insensitive and store them in mixed case?
boolean	**storesUpperCaseIdentifiers()** Does this database treat mixed-case unquoted SQL identifiers as case insensitive and store them in upper case?
boolean	**storesUpperCaseQuotedIdentifiers()** Does this database treat mixed-case quoted SQL identifiers as case insensitive and store them in upper case?
boolean	**supportsAlterTableWithAddColumn()** Does this database support ALTER TABLE with add column?
boolean	**supportsAlterTableWithDropColumn()** Does this database support ALTER TABLE with drop column?
boolean	**supportsANSI92EntryLevelSQL()** Does this database support the ANSI92 entry-level SQL grammar?
boolean	**supportsANSI92FullSQL()** Does this database support the intermediate ANSI92 SQL grammar?
boolean	**supportsANSI92IntermediateSQL()** Does this database support the full ANSI92 SQL grammar?

Table 219. DatabaseMetaData interface methods (continued)

Method return value type	Method
boolean	**supportsBatchUpdates()** Does this database support batch updates?
boolean	**supportsCatalogsInDataManipulation()** Can a catalog name be used in a data manipulation statement?
boolean	**supportsCatalogsInIndexDefinitions()** Can a catalog name be used in an index definition statement?
boolean	**supportsCatalogsInPrivilegeDefinitions()** Can a catalog name be used in an privilege definition statement?
boolean	**supportsCatalogsInProcedureCalls()** Can a catalog name be used in a procedure call statement?
boolean	**supportsColumnAliasing()** Returns true if column aliasing is supported.
boolean	**supportsCatalogsInTableDefinitions()** Can a catalog name be used in a table definition statement?
boolean	**supportsConvert( )** Does this database support the CONVERT function?
boolean	**supportsConvert(int fromType, int toType)** Does database support the CONVERT function for two specified SQL types?
boolean	**supportsCoreSQLGrammar()** Does this database support the ODBC Core SQL grammar?
boolean	**supportsCorrelatedSubqueries()** Does this database support correlated subqueries?
boolean	**supportsDataDefinitionAndDataManipulationTransactions()** Does this database support both data definition and data manipulation statements within a transaction?
boolean	**supportsDataManipulationTransactionsOnly()** Does this database support only data manipulation statements within a transaction?
boolean	**supportsDifferentTableCorrelationNames()** When table correlation names are supported, must they be different from the names of the tables?
boolean	**supportsExpressionsInOrderBy** Does this database support expressions in ORDER BY lists?
boolean	**supportsExtendedSQLGrammar()** Does this database support the ODBC Extended SQL grammar?
boolean	**supportsFullOuterJoins()** Returns true if full nested outer joins are supported.

Table 219. DatabaseMetaData interface methods (continued)

Method return value type	Method
`boolean`	**supportsGetGeneratedKeys()** JDBC 3.0 Can generated keys be retrieved after a statement has been executed?
`boolean`	**supportsGroupBy()** Does this database support the GROUP BY clause?
`boolean`	**supportsGroupByBeyondSelect()** Does this database allow a GROUP BY clause that includes columns not included in the SELECT statement if all the columns in the SELECT statement are included in the GROUP BY clause?
`boolean`	**supportsGroupByUnrelated()** Does this database support allow a GROUP BY clause that includes a column that is not in the SELECT statement?
`boolean`	**supportsIntegrityEnhancementFacility()** Does this database support the SQL Integrity Enhancement Facility?
`boolean`	**supportsLikeEscapeClause()** Does this database support specifying a LIKE escape clause?
`boolean`	**supportsLimitedOuterJoins()** Does this database provide limited support for outer joins?
`boolean`	**supportsMinimumSQLGrammar()** Does this database support the ODBC Minimum SQL grammar?
`boolean`	**supportsMixedCaseIdentifiers()** Returns true if the database treats mixed-case, unquoted SQL identifiers as case sensitive and stores them in mixed case.
`boolean`	**supportsMixedCaseQuotedIdentifiers()** Returns true if the database treats mixed-case, quoted SQL identifiers as case sensitive and stores them in mixed case.
`boolean`	**supportsMultipleOpenResultSets()** JDBC 3.0 Can a CallableStatement object return multiple ResultSet objects simultaneously?
`boolean`	**supportsMultipleResultSets()** Does this database support getting multiple ResultSet objects from a single call to the execute method?
`boolean`	**supportsMultipleTransactions()** Does this database allow multiple transactions to be open at the same time on different connections?
`boolean`	**supportsNamedParameters()** JDBC 3.0 Does this database support named parameters for callable statements?
`boolean`	**supportsNonNullableColumns()** Returns true if columns can be defined as non-nullable.

Table 219. DatabaseMetaData interface methods (continued)

Method return value type	Method
`boolean`	**supportsOpenCursorsAcrossCommit()** Returns true if the database supports keeping cursors open across commits; otherwise, returns false.
`boolean`	**supportsOpenCursorsAcrossRollback()** Does this database support keeping cursors open across rollbacks?
`boolean`	**supportsOpenStatementsAcrossCommit()** Does this database support keeping statements open across commits?
`boolean`	**supportsOpenStatementsAcrossRollback()** Does this database support keeping statements open across rollbacks?
`boolean`	**supportsOrderByUnrelated()** Returns true if an ORDER BY clause can use columns not in the SELECT statement.
`boolean`	**supportsOuterJoins()** Returns true if some form of outer join is supported.
`boolean`	**supportsPositionedDelete()** Returns true if positioned DELETE is supported.
`boolean`	**supportsPositionedUpdate()** Returns true if positioned UPDATE is supported.
`boolean`	**supportsResultSetConcurrency(int type, int concurrency)** Does this database support a specified concurrency type in combination with a specified result set type?
`boolean`	**supportsResultSetHoldability(int holdability)** JDBC 3.0 Does this database support the specified result set holdability?
`boolean`	**supportsResultSetType(int type)** JDBC 2.0 Returns true if the database supports the given result set type.
`boolean`	**supportsSavepoints()** JDBC 3.0 Does this database support save points?
`boolean`	**supportsSchemasInDataManipulation()** Can a schema name be used in a data manipulation statement?
`boolean`	**supportsSchemasInIndexDefinitions()** Can a schema name be used in an index definition statement?
`boolean`	**supportsSchemasInPrivilegeDefinitions()** Can a schema name be used in privilege definition?
`boolean`	**supportsSchemasInProcedureCalls()** Can a schema name be used in a procedure call?
`boolean`	**supportsSchemasInTableDefinitions()** Returns true if the schema name can be used in a table definition statement.

Table 219. DatabaseMetaData interface methods (continued)

Method return value type	Method
boolean	**supportsSelectForUpdate()** Does this database support SELECT FOR UPDATE statements?
boolean	**supportsStatementPooling()** JDBC 3.0 Does this database support statement pooling?
boolean	**supportsStoredProcedures()** Does this database support stored procedure calls that use the stored procedure escape syntax?
boolean	**supportsSubqueriesInComparisons()** Does this database support subqueries in comparison expressions?
boolean	**supportsSubqueriesInExists()** Does this database support subqueries in EXISTS expressions?
boolean	**supportsSubqueriesInIns()** Does this database support subqueries in IN expressions?
boolean	**supportsSubqueriesInQuantifieds()** Does this database support subqueries in quantified expressions?
boolean	**supportsTableCorrelationNames()** Does this database support table correlation names?
boolean	**supportsTransactions()** Returns true if transactions are supported. If not, the isolation level is TRANSACTION_NONE.
boolean	**supportsTransactionIsolationLevel(int level)** Returns true if the database supports the transaction isolation level specified in *level*.
boolean	**supportsUnion()** Does this database support SQL UNION?
boolean	**supportsUnionAll()** Does this database support SQL UNION ALL?
boolean	**updatesAreDetected(int type)** Can a visible row update be detected by calling the method ResultSet.rowUpdated?
boolean	**usesLocalFilePerTable()** Does this database use a file for each table?
boolean	**usesLocalFiles()** Does this database store tables in a local file?

Related tasks

"Developing DB2 Everyplace Java applications" on page 22
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.

Related reference

"SQLState messages reported by JDBC" on page 441
"Overview of DB2 Everyplace JDBC support" on page 317

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 220 lists the methods in the Driver interface that are supported by DB2 Everyplace.

Table 220. 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, java.util.Hashtable info)** DB2 Everyplace overloaded method for class library configurations that do not support java.util.Properties. See connect(String url, Properties info) for supported key/value pairs.

Table 220. Driver interface methods (continued)

Method return value type	Method
`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: DB2e_ENCODING Value: character encoding Restriction: This is not supported on Palm OS. • Key: ENABLE_DELETE_PHYSICAL_REMOVE Value: true, false Default: false • Key: ENABLE_DIRTY_BIT_SET_BY_APPLICATION Value: true, false Default: false • Key: ENABLE_FILENAME_FORMAT_83 Value: true, false Default: false • Key: ENABLE_IO_WRITETHROUGH Value: true, false Default (Windows and Linux x86 platforms only): false Default (Other platforms): true **Restriction:** The ENABLE_IO_WRITETHROUGH key affects only Windows and Linux x86 platforms. It has no effect on other platforms. • Key: ENABLE_READ_INCLUDE_MARKED_DELETE Value: true, false Default: false • Key: ENABLE_REORG Value: true, false Default: true • Key: ENABLE_SHARED_DATABASE_ACCESS Value: true, false Default: false • Key: ENABLE_TABLE_CHECKSUM Value: true, false Default: false • Key: LOCK_TIMEOUT Value: number of seconds to wait for a lock to be obtained before rolling back a transaction. Default: 20 • Key: LOGIN_TIMEOUT Value: number of seconds to wait for a login request to complete before returning control to the application Default: 0 Restriction: This is not supported on Palm OS. • Key: password Value: user password • Key: user Value: user name

Table 220. Driver interface methods (continued)

Method return value type	Method
`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.

Example: The following example illustrates how to create a Properties object and how to use the setProperty() method.

```
Properties props = new Properties();
props.setProperty("ENABLE_REORG", "false");
props.setProperty("LOCK_TIMEOUT", "200");
props.setProperty("ENABLE_SHARED_DATABASE_ACCESS" , "true");
props.setProperty("ENABLE_IO_WRITETHROUGH","true");
props.setProperty("ENABLE_TABLE_CHECKSUM", "true");
Connection con = DriverManager.getConnection(url, props);
```

Related tasks

"Developing DB2 Everyplace Java applications" on page 22
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.

Related reference

"SQLState messages reported by JDBC" on page 441

"Overview of DB2 Everyplace JDBC support" on page 317

"SQLSetConnectAttr—Set options related to a connection" on page 247

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.

Note: The setter methods for setting IN parameter values must specify types that are compatible with the defined SQL type of the input parameter. For instance, if the IN parameter has SQL type INTEGER, then the method setInt should be used.

java.sql package

public interface **PreparedStatement**

extends Statement

Table 221 on page 338 lists the methods in the PreparedStatement interface that are supported by DB2 Everyplace.

Table 221. PreparedStatement interface methods

Method return value type	Method
`void`	**clearParameters()** Clears the current parameter values immediately.
@ `void`	**close()** @ Releases a *PreparedStatement*'s database and JDBC resources immediately instead of @ waiting for them to be automatically released.
`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`	**setBinaryStream(int parameterIndex, InputStream x, int length)** Sets the designated parameter to a `java.lang.InputStream` value with a length of `x` bytes.
`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.

Table 221. PreparedStatement interface methods (continued)

Method return value type	Method
`void`	**setNull (int parameterIndex, int sqlType)** Sets the designated parameter to SQL NULL.
`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 22
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.

Related reference

"SQLState messages reported by JDBC" on page 441

"Overview of DB2 Everyplace JDBC support" on page 317

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. The JDBC driver converts the underlying data to the Java type specified in the getter method and returns a suitable Java value.

java.sql package

public interface **ResultSet**

Table 222 lists the fields in the ResultSet interface that are supported by DB2 Everyplace.

Table 222. ResultSet interface fields

Field type	Field
`static int`	**CONCUR_READ_ONLY** The constant indicating the concurrency mode for a ResultSet object that can 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 223 lists the methods in the ResultSet interface that are supported by DB2 Everyplace.

Table 223. 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.

Table 223. ResultSet interface methods (continued)

Method return value type	Method
`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.
`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.**
`InputStream`	**getBinaryStream(int columnIndex)** Gets the value of the designated column in the current row of this ResultSet object as a binary stream of bytes.
`InputStream`	**getBinaryStream(String columnName)** Gets the value of the designated column in the current row of this ResultSet object as a binary stream of bytes.
`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. The driver first gets the value of the column as a Java short. If the value is equal to 1, true is returned. Otherwise, false is returned.
`boolean`	**getBoolean(String columnName)** Gets the value of a column in the current row as a Java boolean. The driver first gets the value of the column as a Java short. If the value is equal to 1, true is returned. Otherwise, false is returned.
`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.

Table 223. ResultSet interface methods (continued)

Method return value type	Method
`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.
`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.

Table 223. ResultSet interface methods (continued)

Method return value type	Method
`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.
`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.

Table 223. ResultSet interface methods (continued)

Method return value type	Method
`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 22
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.

Related reference

"SQLSetStmtAttr—Set options related to a statement" on page 252

"SQLState messages reported by JDBC" on page 441

"Overview of DB2 Everyplace JDBC support" on page 317

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 224 lists the fields in the ResultSetMetaData interface that are supported by DB2 Everyplace.

Table 224. 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.

Table 224. ResultSetMetaData interface fields (continued)

Field type	Field
`static int`	**columnNullableUnknown** The constant indicating that the nullability of a column's values is unknown.

Table 225 lists the methods in the ResultSetMetaData interface that are supported by DB2 Everyplace.

Table 225. 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 22
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.

Related reference

"SQLState messages reported by JDBC" on page 441

"Overview of DB2 Everyplace JDBC support" on page 317

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 226 lists the fields in the Statement interface that are supported by DB2 Everyplace.

Table 226. 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.

Table 227 lists the methods in the Statement interface that are supported by DB2 Everyplace.

Table 227. 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`	**cancel()**
`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.

Table 227. Statement interface methods (continued)

Method return value type	Method
`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.

Related tasks

"Developing DB2 Everyplace Java applications" on page 22
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.

Related reference

"SQLState messages reported by JDBC" on page 441

"Overview of DB2 Everyplace JDBC support" on page 317

Javax.sql Interface

This topic explains the packages provided by the javax.sql package.

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 mobile 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 228 and Table 229 on page 348 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 228. Standard DataSource properties supported by DB2 Everyplace

Property Name	Type	Description	Access Methods
description	String	description of this data source	String getDescription() void setDescription(String Description)
password	String	database password	String getPassword() void setPassword(String password)
user	String	user's account name	String getUser() void setUser(String user)

Table 229 on page 348 lists the supported properties for the DataSource interface that are specific to DB2 Everyplace.

Table 229. DB2 Everyplace-specific properties for the DataSource interface

Property Name	Type	Description	Access Methods
deletePhysicalRemove	boolean	enable/disable physically removing records	boolean isDeletePhysicalRemove() void setDeletePhysicalRemove(boolean enable)
dirtyBitSetByApplication	boolean	enable/disable application to set dirty bit	boolean isDirtyBitSetByApplication() void setDirtyBitSetByApplication(boolean enable)
encoding	String	character encoding	String getEncoding() void setEncoding(String encoding)
filenameFormat83	boolean	either short (8.3 format) or long filenames	boolean isFilenameFormat83() void setFilenameFormat83(boolean enable)
ioMode	int	push changes directly to storage media or let the OS handle them.	int getIoMode() void setIoMode(int mode)
lockTimeout	int	lock timeout in seconds (default = 20)	int getLockTimeout() void setLockTimeout(int seconds)
readIncludeMarkedDelete	boolean	enable/disable read logically deleted records	boolean isReadIncludeMarkedDelete() void setReadIncludeMarkedDelete(boolean enable)
reorg	boolean	enable/disable reorganization	boolean isReorg() void setReorg(boolean enable)
sharedDatabaseAccess	boolean	either shared or exclusive access to database	boolean isSharedDatabaseAccess() void setSharedDatabaseAccess(boolean enable)
tableChecksum	boolean	enable/disable checksums for database files	boolean isEnabledTableChecksum() void setEnabledTableChecksum(boolean enable)
url	String	data source	String getUrl() void setUrl(String url)

Table 230 lists the methods in the DataSource interface that are supported by DB2 Everyplace.

Table 230. DataSource interface methods

Method return value type	Method
`int`	**getBufferpoolSize()** Gets the size of the connection bufferpool in bytes.
`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.

Table 230. DataSource interface methods (continued)

Method return value type	Method
`java.io.PrintWriter`	**getLogWriter()** Retrieves the log writer for this DataSource object.
`int`	**setBufferpoolSize(int size)** Sets the amount of memory, in bytes, that the DB2 Everyplace database should reserve for its bufferpools. If this value is not a multiple of 4K (4096 bytes), DB2 Everyplace rounds it down to the next smallest multiple of 4K.
`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.

DB2 Everyplace includes the following pre-defined bufferpool sizes:

Table 231. Pre-defined bufferpool size constants

Constant	Bufferpool size in bytes
SQL_BUFFERPOOL_SIZE_DEFAULT	The default value for the platform on which you are running DB2 Everyplace.
SQL_BUFFERPOOL_SIZE_64K	65 536
SQL_BUFFERPOOL_SIZE_128K	131 072
SQL_BUFFERPOOL_SIZE_256K	262 144
SQL_BUFFERPOOL_SIZE_512K	524 288
SQL_BUFFERPOOL_SIZE_1024K	1 048 576
SQL_BUFFERPOOL_SIZE_2048K	2 097 152
SQL_BUFFERPOOL_SIZE_4096K	4 194 304
SQL_BUFFERPOOL_SIZE_8172K	8 388 608
SQL_BUFFERPOOL_SIZE_1M	1 048 576
SQL_BUFFERPOOL_SIZE_2M	2 097 152
SQL_BUFFERPOOL_SIZE_4M	4 194 304
SQL_BUFFERPOOL_SIZE_8M	8 388 608

Important:

- The minimum value for SQL_ATTR_BUFFERPOOL_SIZE is SQL_BUFFERPOOL_SIZE_64K. If you call SQLSetConnectAttr() and specify a smaller value than SQL_BUFFERPOOL_SIZE_64K, SQlSetConnectAttr() returns SQLSTATE **HY**024.
- If the database engine cannot allocate as much memory as you specify in the SQL_ATTR_BUFFERPOOL_SIZE connection attribute, the engine will try to use a smaller bufferpool configuration. SQLConnect() will return SQLSTATE 01000.
- If there is not enough memory for the minimum bufferpool configuration, SQLConnect() will return SQLState 58004.

- You cannot change the size of the bufferpool if a connection to the database already exists. New connections will use the bufferpool size of the existing connection. SQLConnect() will return a warning.

National language support (NLS)

This topic 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 multibyte character sets, and Unicode. Both Unicode and non-Unicode (ANSI) are supported on all Windows operating systems.

A note about character encodings: The DB2 Everyplace Sync Client and database support local code-page encodings on Win32, Palm, Linux, and Neutrino platforms. For the DB2 Everyplace Sync Client, the character encoding can be specified with the code page file config-isyn or the isync.encoding property in iscServiceOpenEx(). This property tells the DB2 Everyplace Sync Server to convert the data on the source to the specified encoding for the client. For the DB2 Everyplace database, the system's default character encoding is used. It is important to note that for synchronized data to be displayed correctly on the client, you must check that the DB2 Everyplace Sync Client encoding matches the system's encoding. For example, if the DB2 Everyplace Sync Client encoding is set to UTF-8 and the operating system default is set to CP1252, special characters might appear corrupted when querying the client database.

Related concepts

"Character encoding in Java applications" on page 25
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. You can dynamically specify the data encoding format.

Related reference

Encodings that are supported by the isync.encoding property

DB2 Everyplace NLS support by operating system

Table 232 lists which operating systems and corresponding languages have NLS support.

Table 232. NLS support

Language	Windows	WinCE	Linux	Palm OS	Symbian OS	Neutrino
English	CP1252/ ISO8859-1/ UCS-2	UCS-2	CP1252/ ISO8859-1/ UTF-8	CP1252/ ISO8859-1	UCS-2	UTF-8
French	CP1252/ ISO8859-1/ UCS-2	UCS-2	CP1252/ ISO8859-1/ UTF-8	CP1252/ ISO8859-1	UCS-2	UTF-8
German	CP1252/ ISO8859-1/ UCS-2	UCS-2	CP1252/ ISO8859-1/ UTF-8	CP1252/ ISO8859-1	UCS-2	UTF-8
Italian	CP1252/ ISO8859-1/ UCS-2	UCS-2	CP1252/ ISO8859-1/ UTF-8	CP1252/ ISO8859-1	UCS-2	UTF-8
Spanish	CP1252/ ISO8859-1/ UCS-2	UCS-2	CP1252/ ISO8859-1/ UTF-8	CP1252/ ISO8859-1	UCS-2	UTF-8

Table 232. NLS support (continued)

Language	Windows	WinCE	Linux	Palm OS	Symbian OS	Neutrino
Simplified Chinese	Codepage/ UCS-2	UCS-2	Codepage/ UTF-8	Codepage • Install enabler	N/A	UTF-8
Traditional Chinese	CP950/ UCS-2	UCS-2 • Install enabler for Pocket PC	CP950/ UTF-8	CP950 • Install enabler Acer S60 has a built-in Traditional Chinese Palm OS.	N/A	UTF-8
Korean	CP1363/ UCS-2	UCS-2 • Install enabler	CP970/ UTF-8	CP1363 • Install enabler	N/A	UTF-8
Japanese	CP943/ UCS-2	UCS-2	CP954/ UTF-8	CP943	N/A	UTF-8
Hebrew	N/A	N/A	N/A	ISO8859-8/ CP1255 • Install enabler	N/A	N/A
Czech	ISO8859-2/ CP1250/ UCS-2	UCS-2 • Install enabler	N/A	ISO8859-2/ CP1250/ • Install enabler	UCS-2	N/A
Arabic	N/A	N/A	N/A	ISO8859-6/ CP1256 • Install enabler	N/A	N/A
Brazilian Portugese	CP1252/ ISO8859-1/ UCS-2	UCS-2	N/A	CP1252/ ISO8859-1	N/A	N/A
Hungarian	ISO8859-2/ CP1250/ UCS-2	UCS-2	N/A	ISO8859-2/ CP1250	N/A	N/A
Polish	ISO8859-2/ CP1250/ UCS-2	UCS-2	N/A	ISO8859-2/ CP1250	N/A	N/A
Slovak	ISO8859-2/ CP1250/ UCS-2	UCS-2	N/A	ISO8859-2/ CP1250	N/A	N/A

On WinCE and Symbian operating systems, only Unicode (UCS-2) is supported. On Palm OS, QNX Neutrino, and Linux operating systems, locale information is used to determine the correct code page. On Windows platforms, Unicode applications use Unicode (the application uses DB2 Everyplace Unicode APIs), and, other applications use code page. DB2 Everyplace does not provide code page conversion functions. DB2 Everyplace mobile 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 DB2 Everyplace mobile database are responsible for interpreting the character data correctly.

On Linux systems, the user is responsible for setting the locale setting correctly. (See the man pages for setlocale for an introduction to locale names on Linux systems.) For example, you can export environment variable LC_CTYPE to ″ja_JP.UTF-8″, then call `setlocale(LC_CTYPE, "")` inside the application. UTF-8 encoded strings are processed in any locale with DB2 Everyplace by specifying UTF-8 in the locale name. For example: de_DE.UTF-8.

On Palm OS, the presence of language enablers is also used to determine the code page.

DB2 Everyplace detects the currently used encoding format by examining the currently set or available locale.

Related concepts

“Character encoding in Java applications” on page 25
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. You can dynamically specify the data encoding format.

Related reference

“DB2 Everyplace Unicode support” on page 353

“DB2 Everyplace language enablers”

Encodings that are supported by the isync.encoding property

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 233. 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
Korean	• HANME 2.0 for Palm OS • HANTIP 2.01for Palm OS CessHan for Casio E-115 1.0 on Windows CE

Related concepts

“Character encoding in Java applications” on page 25
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. You can dynamically specify the data encoding format.

Related reference

“DB2 Everyplace Unicode support” on page 353

“DB2 Everyplace NLS support by operating system” on page 350

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 DB2 Version 9.1 might require more space when the string is downloaded and stored in a DB2 Everyplace Unicode database.

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.

@ DB2eCLP

@ DB2eCLP is a tool that allows you to directly issue SQL statements from a command-line interface.

@ This tool is an application that interacts with DB2 Everyplace through a command-line interface. It is
@ used for the DB2 Everyplace mobile database on mobile devices and not used by the Sync Server.

@ To find this tool, navigate to the following directory: <DSYPATH> \Clients\ *platform*
@ \database*lang**proc*\DB2eCLP.

@ DB2eCLP commands

@ This topic presents the commands that you can use on DB2eCLP.

@ You can issue SQL statements directly from a command-line interface. For example:

```
@ SELECT * FROM PHONEBOOK
```

@ On some platforms, each statement must end in a semicolon. For example:

```
@ SELECT * FROM PHONEBOOK;
```

@ DB2eCLP also supports some extended commands:

@ **$file** *[input file] [output file]*
@ Runs SQL statements from an input file, and writes the result to an output file. This command is
@ not supported on Palm OS systems. For all other platforms, you can specify the full path.

@ **AUTOCOMMIT OFF|ON**
@ Specifies whether the application commits each statement by default (the engine default is ON).
@ When autocommit mode is on (true), each statement is treated as a single, complete transaction.
@ AUTOCOMMIT OFF changes the transaction mode to manual, which enables applications to
@ either roll back or commit work.

@ **BLASTDB**
@ Drops all user tables in the database.

@ **CONNECT TO** *arg1*
@ Automatically disconnects the application from the current connection and reconnects the
@ application to a local database (*arg1*). The specification is in the SQLConnect() CLI call. The
@ delimiter for the paths for CLI-SQLConnect is either \ (backslash) or / (slash). Both delimiters are
@ understood on all platforms and are mapped to the appropriate delimiter when the file system is
@ accessed, which allows databases to reside in different directories. For example,

```
@ connect to c:\temp\
@ create table t (a int)
@ insert into t values (10)
@ select * from t
```

@ **CONNECT TO** *arg1* **USER** *arg2* **USING** *arg3*
@ Connects the application to a local database (**arg1**) using the specified user name (*arg2*) and
@ password (*arg3*). This information is needed to access encrypted tables. If the application is
@ already connected to another database, that connection is dropped.

@ A directory name can include a space. For example, C:\System\program files\ is a valid
@ directory structure, but you must use the same directory structure that exists on your machine.

@ **DBCHECK outputfile**
@ Runs the data integrity check tool and writes the result to an output file in the database directory.
@ This command is supported on Linux and Windows 32-bit operating systems only.

@ **DESCRIBE SELECT**
@ Describe the type, column, and name length of the data that is returned by a SELECT statement.
@ For example:

```
@ DESCRIBE SELECT * FROM PHONEBOOK
```

@ **DISABLE APPLICATION SET DIRTY**
@ Disables setting dirty bit by DB2eCLP.

@ **DISABLE LONG FILENAME**
@ Creates files in 8.3 file name format.

@ **DISABLE PHYSICAL DELETE**
@ Disables physical delete mode (default).

@ **DISABLE READ DELETED**
@ Disables reading deleted rows.

@ **DISABLE REORG**
@ Disables table reorganization.

@ **ENABLE APPLICATION SET DIRTY**
@ Enables setting dirty bit by DB2eCLP.

@ **ENABLE LONG FILENAME**
@ Creates files in long file name format (default).

@ **ENABLE PHYSICAL DELETE**
@ Enables physical delete mode. Deleted rows will no longer be readable.

@ **ENABLE READ DELETED**
@ Enables reading deleted rows.

@ **ENABLE REORG**
@ Enables table reorganization automatically (default).

@ HELP Lists all available commands.

@ **LIST COLUMNS**
@ List all user table columns in the database.

@ **LIST INDEX**
@ List all indexes ordered by table name, index name, and column order.

@ **LIST TABLES**
@ List all user tables in the database.

@ **VERSION**
@ Prints the DB2 Everyplace mobile database version string. This command returns the same string
@ as the SQLGetInfo() function.

@ Importing and exporting data using the DB2eCLP

@ The DB2eCLP for Palm OS, Symbian OS, Windows CE, Windows platforms, Neutrino, and embedded
@ Linux supports importing data from a file to DB2 Everyplace and exporting DB2 Everyplace data to a
@ file.

@ Importing and exporting data on the Palm OS uses the memo files on the device.

@ • **To import data from a file on a mobile device to DB2 Everyplace:**

@ 1. Type `IMPORT FROM` *`file_name`* `OF DEL INSERT INTO` *`table_name`* `[(column list)]` where *file_name* is
@ the name of the file to import from.

@ On Palm OS, *file_name* is the name of the memo to import from. The file name must appear on the
@ first line of the memo. Palm memos have a limitation of storing 4K bytes text. *table_name* is the
@ name of an existing table to import into. For example, to import data from a file named mydata.txt
@ to an existing table named mytable, type:

@ `IMPORT FROM mydata.txt OF DEL INSERT INTO mytable`

@ • **To export data from DB2 Everyplace to a file:**

@ 1. Type `EXPORT TO` *`file_name`* `OF DEL` *`stmt`* where *file_name* is the name of the file to write the data to.
@ *stmt* is the SELECT statement to select the data to export. For example, to export all data from the
@ table named mytable to a file named myfile.txt, type:

@ `EXPORT TO myfile.txt OF DEL SELECT * FROM mytable`

@ DB2 Everyplace for Palm OS uses a set of command-line tools for Windows and a Palm OS application
@ to import and export data as PDB files. The Import/Export tools include the following executable files,
@ which are installed on the Windows workstation. Download the utilities package of these files from
@ theDB2 Everyplace Support Web page, under Tools and Samples for DB2 Everyplace.

@ **CSV2DB2e.exe**
@ This file imports data from a Comma Separated Values file (with file extension of .csv) into a
@ DB2 Everyplace table. A DB2e table (for example, named PERSON) is represented by two files,
@ DSY_PERSON and DSY_iPERSON. The DSY_PERSON file contains the data, and
@ DSY_iPERSON contains the indexing information.

@ **DB2e2PDB.exe**
@ This program converts a DB2 Everyplace table into Palm OS PDB format. It then copies the
@ files into the user's directory and informs the HotSync program that there are files to install. If
@ more than one user is defined in the system, a user list will be displayed so that you can select
@ the intended user.

@ **PDB2DB2e.exe**
@ PDB2DB2e.exe converts Palm OS PDB files from the user backup area into a DB2 Everyplace
@ table. If more than one user is defined in the system, a user list will be displayed so that you
@ can select the intended user.

@ **DB2e2CSV.exe**
@ DB2e2CSV exports a DB2 Everyplace table into a CSV file. It also makes use of the DB2
@ Everyplace system catalog files named DB2eSYSTABLES and DB2eSYSCOLUMNS.

@ **PalmImport.bat**
@ PalmImport.bat combines the operations of CSV2DB2e.exe and DB2e2PDB.exe.

@ **PalmExport.bat**
@ PalmExport.bat combines the operations of PDB2DB2e.exe and DB2e2CSV.exe.

@ The Import/Export tools include the following Palm OS application:

@ **DB2eImport.prc**
@ This program registers the DB2 Everyplace files transferred by the HotSync program to the
@ local DB2 Everyplace system.

@ • To import data to the Palm OS device:

@ 1. Create two files with the same name (for example, VNPERSON), one with a .csv extension and one
@ with a .sch extension. The .csv file contains the data, and the .sch file contains the schema for the
@ table to be imported. Note that the file name cannot contain any blank spaces. Sample .csv and .sch
@ files are located in the DemoImport\ folder. The files are named VNPERSON.csv and
@ VNPERSON.sch.

@ Table 234 lists the supported data types and their representation in the CSV file.

@ *Table 234. Data types and their representation in a CSV file*

@ Data type	Represented as
@ integer (or int)	1234
@ smallint	1234
@ decimal(n,p)	12.34
@ char(n)	″John″
@ varchar(n)	″John″
@ date	yyyymmdd
@ time	″14.05.48″
@ timestamp	″2001-05-01-16.16.51.000000″

@
@ To represent a null value for a column, enter nothing between the commas of the CSV file. For
@ example, three integer columns with a null in the second column would be represented as `1,,3` in
@ the CSV file and become `1, null, 3` in the database.

@ 2. Start the import tool, PalmImport.bat, providing the schema of the table as a parameter. The
@ schema should be in an associated .sch file. Use the following syntax to start the Import tool:

@ `PalmImport.bat` *`path_name file_name`* where *path_name* is the path to the CSV file, and *file_name* is the CSV file name in uppercase without an extension. The CSV file name cannot be enclosed in double quotation marks or contain any blank spaces. For example:

```
PalmImport.bat DemoImport VNPERSON
```

@ The imported tables are automatically added to the Palm Install Tool to be installed after the next HotSync operation.

@ 3. Install the DB2eImport.prc to the Palm OS device using the Palm Install Tool.

@ 4. Perform a HotSync operation to complete the installation of the imported tables and DB2eImport.prc.

@ 5. Start the DB2eImport program on the Palm OS device to complete the import.

@ • To export data from the Palm OS device:

@ 1. Perform a HotSync operation to back up the Palm OS device. Always back up and remove the DB2 Everyplace files (files starting with DSY) from the Palm OS user's backup directory before starting a HotSync operation. The backup area on Windows workstations is usually located at *PalmDir**user_name*\Backup where *PalmDir* is the directory where the Palm OS software is installed, and *user_name* is the user name of the Palm OS user.

@ 2. Start the export program, PalmExport.bat, with the following syntax:

```
PalmExport.bat path_name file_name
```

@ where *path_name* is the output path, and *file_name* is the DB2 Everyplace table name in uppercase. The DB2 Everyplace table name cannot be enclosed in double quotation marks or contain any blank spaces. For example:

```
PalmExport.bat DemoExport VNPERSON
```

@ The resulting file is in the same path as the source file.

@ When an error is encountered, the Import/Export tools report the number of records processed.

@ A sample using the Import/Export tools is included in the package. The batch files PalmExport.bat and @ PalmImport.bat provide examples of how to use the CSV2DB2e.exe and DB2e2CSV.exe tools.

@ **Related concepts**

@ "DB2eCLP" on page 353
@ DB2eCLP is a tool that allows you to directly issue SQL statements from a command-line interface.

@ **Related reference**

@ "DB2eCLP commands" on page 354
@ This topic presents the commands that you can use on DB2eCLP.

DB2 Everyplace sample applications

You can use the DB2 Everyplace sample applications to learn how to create your own applications to interface with the DB2 Everyplace engine.

The following table indicates which sample applications are supported on the various mobile devices.

Table 235. DB2 Everyplace sample applications by target operating system

Target platform	Sample applications
Palm	• VNurse • DB2eAppl • DB2 Sync

Table 235. DB2 Everyplace sample applications by target operating system (continued)

Target platform	Sample applications
Symbian 7	• DB2 Sync • DB2eJavaCLP
Windows CE	• VNurse • DB2eAppl • DB2eJavaCLP • DB2 Sync
Windows	• DB2eAppl • DB2eJavaCLP • DB2 Sync Console
Linux and Neutrino	• DB2 Sync Console • DB2eJavaCLP

The following table describes where the DB2 Everyplace sample applications are located and which edition of DB2 Everyplace includes each sample.

Table 236. DB2 Everyplace sample applications by version and location

Application	Description	Directory
VNurse	Up and running sample	<DSYPATH> \Clients\ *platform* \database*lang*\Samples\VNurse
DB2 Sync C • db2sync_console . exe • db2sync.exe • db2sync.prc • DB2Sync.sis • db2sync_console Java and ISync.NET • ISyncSample • DB2SyncConsole	Up and running synchronization samples (binaries)	For Windows systems: <DSYPATH> \Clients\ win32\ sync*lang*\ *[Unicode/non-Unicode}*\db2sync_console.exe For WinCE systems: <DSYPATH> \Clients\ wince\sync*lang**version*\ *proc*\db2sync.exe For Palm OS systems: <DSYPATH> \Clients\ palmos \sync*lang*\db2sync.prc For Linux and Neutrino:$DSYINSTDIR \Clients\ *platform*\sync*proc* \db2sync_console For Symbian OS Version 7 systems: <DSYPATH> \Clients\ symbian7\sync*lang*\DB2SYNC.APP For Java: <DSYPATH> \Clients\ clientapisample\Java_API For ISync.NET: <DSYPATH> \Clients\ clientapisample\NMP
JDBCSample • DB2eAppl • DB2eJavaCLP	JDBC sample application	For Windows and WinCE:<DSYPATH>\Clients\ *platform*\database\jdbc\ For Symbian OS Version 7:<DSYPATH>\Clients\ symbian7\database\ *lang* \JDBCSample\classes\ For Palm OS:<DSYPATH>\Clients\ palmos\database\jdbc\cldc\sample\ For Linux:$DSYINSTDIR\Clients\ linux\database\jdbc\

Important: DB2 Everyplace Database Edition does not include the DB2 Sync sample application.

Synchronizing using DB2 Sync

This topic explains how to set up the necessary software and mobile devices in order to test synchronization using DB2 Sync.

The DB2 Sync sample application

The DB2 Sync sample application works with the DB2 Everyplace Sync Server to synchronize data and applications between mobile devices and enterprise data sources. Before you configure DB2 Sync, you need to define a user, group, subscription, and subscription set in the Mobile Devices Administration Center.

DB2 Sync is a synchronization application with a graphical interface. It is available for Palm OS, Windows CE, and the Symbian OS mobile devices. DB2 Sync Console, a command-line program, is also provided to test synchronization on Windows, Neutrino, and Linux mobile devices. Both the applications are open source and demonstrate how to use the DB2 Everyplace Sync Client C-API.

Important: Read the license agreement before you use the source code.

During every synchronization, the client software checks whether the client's configuration (the subscription sets and subscriptions that are assigned to the client's group) must be updated. For example, when you refresh a file that is referenced in a subscription, DB2 Everyplace sets a flag that indicates that the subscription has changed. The updated subscription is downloaded to all subscribed users on the next synchronization.

Configuring Server settings on DB2 Sync for a Palm OS device

This topic describes how to configure the Server settings field that is necessary for successful synchronization.

Be sure that you have installed DB2 Everyplace on the mobile device and that the mobile device or emulator is set up for network connections. For more information, see *Installation and User's Guide* and the DB2 Sync menu options topic.

1. Locate the DB2 Sync application on the mobile device or emulator.
2. Start **DB2 Sync**.
3. Select **Server Settings** from the menu.
4. Tap the drop-down list next to URL.
5. Select **-enter-** and enter the DB2 Everyplace Sync Server URL. The default port used by the DB2 Everyplace Sync Server basic application server is 8080. If the DB2 Everyplace Sync Server is configured on a port other than 8080, you must specify the port number in the URL. Each new server URL that you enter is stored in the drop-down list.
6. Type a user ID in the **User** field. The user name that you enter must be defined in the DB2 Everyplace Mobile Devices Administration Center. For information about creating a user in the Mobile Devices Administration Center, see *Sync Server Administration Guide*.
7. Type a password in the **Password** field. The password that you enter must be defined in the DB2 Everyplace Mobile Devices Administration Center. For information about creating a user in the Mobile Devices Administration Center, see *Sync Server Administration Guide*.
8. Select the **Save Password** check box if you want to save the password for subsequent uses.

DB2 Sync menu options

This topic describes the menu options for configuring DB2 Sync.

In the upper left-hand corner of the DB2 Sync dialog, click the **DB2 Sync** menu. A drop-down menu appears with the following options:

- Subs sets
- Server settings

- Client settings
- Network settings
- About DB2 sync

Subscription sets

Note: Configuring subscription sets is optional for successful synchronization.

You can view the subscription sets to which the DB2 Everyplace Sync Client subscribes by selecting the **Subscription Sets** option from the menu. In the subscription set panel, the check box next to a subscription set indicates whether the subscription set is enabled for synchronization. Thus, you can disable synchronization on subscription sets by clearing a check box. The command buttons and their actions in the panel are explained as follows:

OK After changing the synchronization options of the subscriptions, you can save the changes by clicking **OK**.

Cancel
If you want to discard the changes, click **Cancel**.

Details
When you click the **Details** button, you can see the detailed information about the selected subscription set. In addition, if you want to discard the mobile device data and perform a refresh of a subscription set, you can do this by selecting the **Reset** check box. The next time that you synchronize, the DB2 Everyplace Sync Client refreshes the data on that subscription set by dropping the data and re-fetching the source data from the server.

Purge This button drops the current subscription set information. The next time that you synchronize, the DB2 Everyplace Sync Client refreshes the subscription set information, and then refreshes each subscription set.

Important: If you made changes to the local tables but have not synchronized these changes with the source database, you will lose those changes when you perform a purge operation.

Server settings

Note: Configuing server settings is necessary for successful synchronization.

When you click the **Server Settings** menu option, the Setting dialog opens. In this dialog, you can configure the URL, user ID and password of the DB2 Sync settings.

Client settings

Note: Configuring client settings is optional for successful synchronization.

You can configure the settings that are specific to the client by selecting the **Client Settings** option from the menu. The client settings that can be configured are as follows:

Trace Tap the drop-down list next to **Trace**, and select **Detailed** to generate detailed tracing information when you encounter problems during synchronization.

Memory slot
Tap the drop-down list next to **Memory slot**, and select the target memory expansion card where you want to save the client information and synchronized data. The default **Device** setting is the main memory on the mobile device. When a memory slot other than **Device** is selected, you can also specify the exact target path in the **Target Path** field, for which the default is the root.

Table 237 lists the default paths where DB2 Sync stores client information and synchronized data for each platform.

Table 237. Default paths for DB2 Everyplace Sync Client information and synchronized data

Platform	Default installation path
Palm	Main memory
Windows	Current directory
Linux	Current directory
QNX Neutrino	Current directory
Windows CE	\
Symbian	C:\System\Data\Sync\

Network settings

Note: Configuring network settings is optional for successful synchronization.

You can configure the network settings by selecting the **Network Settings** option from the menu. The network settings that can be configured are as follows:

Timeout
In this drop-down list, you can specify a timeout duration for synchronization. Select a longer duration if the server is busy or if there is a large amount of data to be synchronized. If you frequently encounter timeout error messages during synchronization, contact the system administrator for the proper setting of this value. The default value is 1 minute.

Network Speed
Tap the drop-down list next to **Network Speed**, and select the proper network speed. This setting allows the DB2 Everyplace Sync Client to adjust the actual message size when communicating with the DB2 Everyplace Sync Server to achieve the best network performance.

Use Proxy
If the system uses a proxy server, select the **Use Proxy** check box and enter the IP address and the port number of the proxy server.

About DB2 Sync

When you select **About DB2 Sync**, DB2 Sync displays the mobile device information, including the version number and the build date for the DB2 Everyplace client and device ID.

Synchronizing data using DB2 Sync

Prerequisites:

Before you use the DB2 Sync application, you need to configure the DB2 Sync application as described in "Configuring Server settings on DB2 Sync for a Palm OS device" on page 359.

1. **To synchronize data using DB2 Sync:**
   a. Start the mobile device.
   b. Start the DB2 Sync application. The system displays the main DB2 Sync window.
   c. Tap **Synchronize**. The synchronization begins. The panel will display the status and the progress of the synchronization. You can cancel a synchronization at any time by clicking the **Cancel** button. When the synchronization ends, the following messages will appear to indicate the synchronization succeeds, fails, or get cancelled respectively.
      - Synchronization succeeded

- Synchronization failed
- Synchronization cancelled

If the synchronization fails, you can click the **Log** button to see the cause (error) of the failure. If the synchronization is successful, then you can go on to verify that the synchronized data is correct.

2. Verify the data on the client.
   a. Start DB2eCLP on the mobile device.
   b. Enter an SQL statement that selects all the records from the table that you recently synchronized.
3. Verify the data on the data source using the DB2 Command Line Processor:
   a. Open the DB2 CLP on the source database.
   b. Enter an SQL statement to select all the records of the subscribed table.

   Note: If you are using the standalone version of DB2 Version 9.1, you can also verify the data on the source database using the DB2 Version 9.1 Control Center. Right-click the subscribed table and select **Sample Contents** to browse the contents of the table.
4. Compare the contents of the table on the mobile device to that of the data source.

Related concepts

"The DB2 Sync sample application" on page 359

"DB2 Sync menu options" on page 359
This topic describes the menu options for configuring DB2 Sync.

Related tasks

"Configuring Server settings on DB2 Sync for a Palm OS device" on page 359
This topic describes how to configure the Server settings field that is necessary for successful synchronization.

The Visiting Nurse sample application

The Visiting Nurse sample application provides an example of how DB2® Everyplace® can increase the productivity of an employee.

This sample application is designed for nurses who visit patients at their homes. If they did not have this DB2 Everyplace sample application, the nurses would have to take notes on paper, then later transcribe their notes into a database on a workstation in their offices. After performing an initial synchronization with the server, the visiting nurses can:

- Access a patient's general information, such as name, address, phone number, and medical condition.
- Collect a patient's medical status, such as blood pressure, pulse rate, temperature, and weight.
- Get an automatic time and date stamp on the new medical record.
- Access a list of people to contact in case of an emergency.

At the end of the day, the visiting nurse can synchronize the data on the mobile device with a central database to:

- Update the central database with the patient status.
- Obtain a list of patients to visit the next day.

Installing the Visiting Nurse sample application

Follow these steps to install the Visiting Nurse sample application on a mobile device.

Prerequisite: Before you can install the Visiting Nurse sample application, you must install the synchronization application that came with your mobile device.

To install the Visiting Nurse sample application on a mobile device:

1. Locate the Visiting Nurse sample application for your platform. The application is located in the following directory: <DSYPATH>\Clients\<*platform*>\database\<*language*>\Samples\VNurse

   where

   <platform>
   the operating system of the mobile device that you are using

   <language>
   the language code that you are using.

   Example: The language code for United States English is en_US.
2. Copy the Visiting Nurse sample application files onto your mobile device by using the synchronization application that came with your mobile device.

   Palm OS devices

   a. Connect the mobile device to a workstation. Use the documentation that came with your mobile device to ensure that the mobile device is connected correctly.
   b. On the workstation, start the HotSync synchronization application. Use the Install Tool to install the files for the Visiting Nurse application.
   c. Perform a HotSync function to complete the installation.

   Windows CE devices

   a. Connect the mobile device to a workstation. Use the documentation that came with your mobile device to ensure that the mobile device is connected correctly.
   b. On the workstation, start the ActiveSync synchronization application. Use the Install Tool to install the files for the Visiting Nurse sample application.
   c. Synchronize the device to complete the installation.

You are now ready to run the Visiting Nurse sample application.

Running the Visiting Nurse application

You can view the Visiting Nurse sample application on your mobile device. The examples in this topic show how the Visiting Nurse sample application looks on the Palm OS emulator or mobile device.

To run the Visting Nurse application:

1. Tap the **Nurse** icon to start the Visiting Nurse sample application. The Schedule window opens with a list of the patients to visit that day.

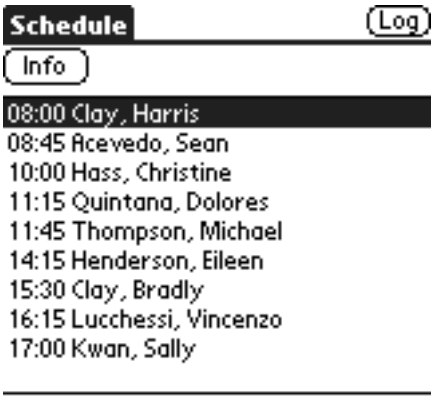


Figure 2. The Schedule window

2. Select a patient's name from the list and tap **Info** to view general information about the patient.

Figure 3. The Person Information window

3. Enter a new medical record:
   a. Tap **Records**. The Medical Record List window opens with a list of all of the records that have previously been created for the patient.

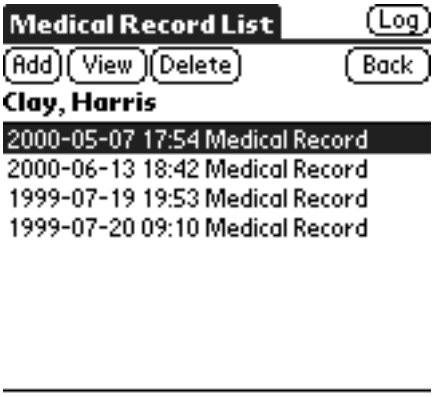


Figure 4. The Medical Record List window

   b. Tap **Add**. The Medical Record window opens.

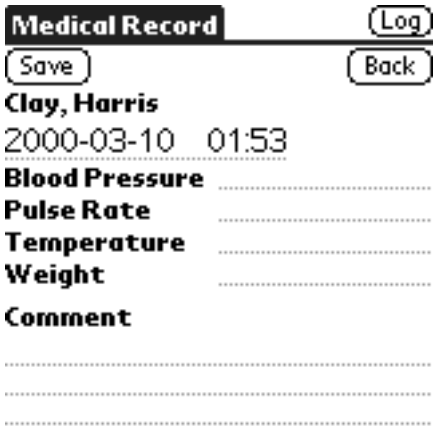


Figure 5. The Medical Record window

c. Fill in the patient's vital statistics and tap **Save** to save the medical record. The medical record will be saved with the current date and time. Tap **Back** to return to Person Information window.

4. View the patient's emergency contact list.
   a. Tap **Contacts**. The Emergency Contact List window opens with a list of the patient's emergency contacts.
   b. View information about a contact person by selecting the person's name from the list and tapping **Info**.

Related tasks

"Installing the Visiting Nurse sample application" on page 362
Follow these steps to install the Visiting Nurse sample application on a mobile device.

Related reference

"Visiting Nurse sample application tables"
This topic contains a description of each sample application table for the Visiting Nurse sample application.

Visiting Nurse sample application tables

This topic contains a description of each sample application table for the Visiting Nurse sample application.

VNSCHEDULE

Contains the nurse's appointments. This table contains information such as patient ID and time of the appointment. The table schema is:

```
CREATE TABLE VNSchedule (PatientID Char(9) NOT NULL,
                         Time_C Time PRIMARY KEY)
```

VNPERSON

Contains data about the patients. This table contains information such as name, Social Security Number, address, and phone numbers. The Social Security Number is used as the primary key. The table schema is:

```
CREATE TABLE VNPerson (ID Char(9) PRIMARY KEY,
                       Name Varchar(40),
                       Address Varchar(50),
                       City Varchar(30),
                       HomePhone Varchar(20),
                       WorkPhone Varchar(20),
                       MobilePhone Varchar(20))
```

VNMEDICALRECORD

Contains the patients' medical records. This table contains information such as blood pressure, pulse rate, and temperature. The medical record ID is used as the primary key. The table schema is:

```
CREATE TABLE VNMedicalRecord (RecordID Integer PRIMARY KEY,
                              Date_C Date,
                              Time_C Time,
                              PatientID Char(9) NOT NULL,
                              BloodPressure Char(7),
                              PulseRate Smallint,
                              Temperature Decimal(4,1),
                              Weight Decimal(5,2),
                              Comment Varchar(100))
```

VNCONTACT

Contains the list of emergency contacts for each patient. This table contains information such as the patient Social Security Number, emergency contact name, and relationship to the patient. The table schema is:

```
CREATE TABLE VNContact (PatientID Char(9) NOT NULL,
                        ContactID Char(9) NOT NULL,
                        Relationship Varchar(20),
                        PRIMARY KEY (PatientID, ContactID))
```

VNSIGNATURE

Contains binary signature data. This table is empty when the sample application is run for the first time. This table is used with the Visiting Nurse Plus application. The table schema is:

```
CREATE TABLE VNSignature (RecordID Integer not null PRIMARY KEY,
                          NurseName Varchar(40),
                          Signature Blob(2000))
```

Related concepts

"DB2eCLP" on page 353
DB2eCLP is a tool that allows you to directly issue SQL statements from a command-line interface.

Related tasks

"Running the Visiting Nurse application" on page 363
You can view the Visiting Nurse sample application on your mobile device. The examples in this topic show how the Visiting Nurse sample application looks on the Palm OS emulator or mobile device.

Java sample applications

This topic explains how to utilize DB2 Everyplace in Java applications.

Compiling and running the DB2 Sync Console sample Java synchronization application

Prerequisites:

- Install and configure the DB2 Everyplace Sync Server.
- Install the DB2 Everyplace Sync Client binaries on the device. See "DB2 Everyplace sample applications" on page 357 for the location of the binaries.
- If you are using a IBM Cloudscape Version 10 client, install IBM Cloudscape Version 10 on the device.

DB2 Sync Console is a Java sample application to demonstrate the use of the DB2 Everyplace Sync Client Java API. The DB2 Sync Console consists of seven files:

DB2SyncConsole.java

DB2SyncConstants.java

DB2SyncConsoleListener.java

db2sync_db2.properties

db2sync_db2e.properties

db2sync_db2j.properties

db2sync_cloudscape.properties

1. Compile the DB2 Sync Console application. This requires the isync4j.jar file, which is one of the DB2 Everyplace Sync Client binaries.
   a. Open a command prompt.
   b. Type the following command: `javac -target 1.1 -classpath isync4j.jar *.java`
2. Set up the environment. The path environment must be setup so that the DB2 Everyplace Sync Client binaries can be located.
   a. For Windows: Set your PATH variable to include the folder where the DB2 Everyplace Sync Client binaries are located.
   b. For Linux or Neutrino: Export the LD_LIBRARY_PATH to include the folder where the DB2 Everyplace Sync Client binaries are located.
3. Run the sample. DB2 Sync Console 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 `db2sync_db2e.properties` file by entering the following command:

```
java -classpath isync4j.jar DB2SyncConsole db2sync_db2e.properties
```

If you get an UnsupportedEncodingException when running the sample with J9, include charconv.zip in the classpath from \ive\runtimes\common\ive\lib.

- To use the Java DB2j client for IBM Cloudscape Version 10, include the DB2j DB2 Everyplace Sync Client jar file, the IBM Cloudscape Version 10 jar file, and pass in the `db2sync_cloudscape.properties` when running DB2 Sync Console. For example, enter:

  ```
  java -classpath %CLOUDSCAPE_HOME%\lib\derby.jar;db2jisync.jar DB2SyncConsole
  db2sync_cloudscape.properties
  ```

- To use the Java DB2j client for IBM Cloudscape Version 5.1, include the DB2j DB2 Everyplace Sync Client jar file, the IBM Cloudscape Version 5.1 jar file, and pass in the `db2sync_db2j.properties` when running DB2 Sync Console. For example, enter:

  ```
  java -classpath %CLOUDSCAPE_HOME%\lib\db2j.jar;db2jisync.jar DB2SyncConsole
  db2sync_db2j.properties
  ```

The application starts with a text menu containing the following options:

a. Perform Synchronization
b. Enable, Disable or Reset Subscription Sets
c. Change Server Settings
d. View The Log
e. About DB2 Everyplace Sync Client
f. Exit

4. Specify option 3 to configure the server settings. This option will allow you to specify the DB2 Everyplace Sync Server's IP address, your sync user name and password, and other options.
5. Specify option 1 to perform the synchronization.

Related concepts

"Overview of DB2 Everyplace Java synchronization providers" on page 23
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 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 mobile devices.

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 "DB2 Everyplace sample applications" on page 357 for information about where the samples are located.

The sample program ISyncSample.java demonstrates how to code a DB2 Everyplace Sync Client application for DB2 Everyplace native synchronization provider.

The major steps of the ISyncSample.java sample application are:

1. Import the DB2 Everyplace synchronization packages.

   ```
   import com.ibm.mobileservices.isync.*;
   import com.ibm.mobileservices.isync.event.*;
   ```

   For Trap-based synchronization provider,

   ```
   import com.ibm.mobileservices.isync.db2e.sti.*;
   ```

2. Implement the eventIssued method of the ISyncListener interface for event notification during synchronization.
3. Get an instance of DB2 sync provider.
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 return code and exception for status of the synchronization.
9. Close and free all resources allocated by the synchronization provider.

```
// Example 1:  ISync Java - Simple API usage
//

 // Step 1:  import the DB2 Everyplace 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 a DB2e sync provider
            //
            provider = ISyncManager.getISyncProvider("isync:db2e");

            // 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

Related reference

"IBM Java Sync APIs" on page 270

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

1. Install WebSphere® Studio Device Developer (WebSphere Studio Device Developer) 5.7, which includes the J9 2.2 Java Virtual Machine. WebSphere Studio Device Developer uses the J9 VM, which might 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. You can download an evaluation version of WebSphere Studio Device Developer from http://www.ibm.com/software/wireless/WebSphere Studio Device Developer/
2. Prepare your target and development environment according to WebSphere Studio Device Developer documentation. Verify your WebSphere Studio Device Developer installation by building and running WebSphere Studio Device Developer sample applications.
3. Install DB2 Everyplace on your target device.

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

1. Create a WebSphere Studio Device Developer project and add jar files to the build path for `DB2eAppl.java`.
2. Import DB2eAppl.java into WebSphere Studio Device Developer.
3. Run DB2eAppl.java. The steps vary depending on your operating system.
   - "Running DB2eAppl.java on Windows" on page 375
   - "Running DB2eAppl.java on Windows CE" on page 376
   - "Running DB2eAppl.java on QNX Neutrino or embedded Linux" on page 379
   - "Running DB2eAppl.java on Symbian" on page 380

Related tasks

"Compiling and running sample Java applications on Palm OS targets" on page 373

Installing WCE Tooling for WebSphere Studio Device Developer 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 371.

1. In WebSphere Studio Device Developer, click **Help → Software Updates → Update Manager** to open the Install/Update Perspective
2. Use the Update Manager to install the following features:
   - SMF Bundle Development Kit 5.7.0
   - Extension Services 5.7.0
   - JSR 169 (BETA) for Extension Services 5.7.0

   The DB2 Everyplace sample project only needs the JSR 169 feature, however, the other two features are requirements for the JSR 169 feature.
3. In the Feature Updates View of the Install/Update Perspective, click **Sites to Visit IBM Micro Environment Toolkit for WebSphere Studio Extension Services For WebSphere Everyplace**.
4. For each feature listed above, select the feature and click Install in the Preview View.
5. Follow the installation instructions to install the feature.

Return to "Compiling and running sample Java applications on non-Palm OS targets" on page 371.

Creating a WebSphere Studio Device Developer project and adding jar files to the build path for DB2eAppl.java for non-Palm OS targets:

Prerequisites:

The DB2eAppl sample project for non-Palm OS targets uses the jclFoundation J9 class library. For java.sql package support, it uses the J9 jdbc.jar defined by JSR 169.

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 371.

1. In WebSphere Studio Device Developer, click **Window → Open Perspective → Java** to switch to the Java Perspective.
2. Create the project for the sample:
   a. Click **File → New → Other**.
   b. On the Select page of the New Project window, select J2ME in the left pane, select J2ME project in the right pane, and click **Next**.
   c. On the J2ME Project page of the New Project window, type `DB2Everyplace-Sample` for the project name and click **Next**.
   d. On the Library selection page of the New Project window, expand 2.2.0, select JCL Foundation 1.0, then click **Next**.
   e. On the Java Settings page of the New Project window, click the **Libraries** tab.
3. Add db2ejdbc.jar to the build path:
   a. Click **Add External JARs**.
   b. In the JAR Selection window, browse to <DSYPATH>\Clients\Win32\database\jdbc\db2ejdbc.jar, where <DSYPATH> is the root installation directory and then click **Open**.
4. Back on the Java Settings page, add jdbc.jar to the build path.
   a. Click **Add Variable**.
   b. In the New Variable Classpath Enty window, select the ESWE_BUNDLES variable and click **Extend**.
   c. In the Variable Extension window, select jdbc.jar and click **OK**.
5. 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 371

Importing DB2eAppl.java into WebSphere Studio Device Developer 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 371.

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 **From directory** field, browse to the <DSYPATH>\Clients\Win32\database\jdbc folder, where <DSYPATH> is the root installation directory and then click **OK**.
4. Back on the File system page of the Import window, select the **DB2eAppl.java** check box in the right pane and click **Finish**. After importing DB2eAppl.java, you should see one error in the Tasks View.
5. Expand the src folder for the project in the Package Explorer View until you see DB2eAppl.java.
6. Double-click DB2eAppl.java to open the file in an editor. The DB2eAppl.java sample was designed to be able to get a connection using either the DriverManager class or the DataSource interface. The sample fails to compile in this environment though, because the jdbc.jar on the project classpath does not support DriverManager (according to the JSR 169 specification).
7. To fix this the above problem, replace the following lines of code:

```
try {
 Class.forName("java.sql.DriverManager");
 Class.forName("com.ibm.db2e.jdbc.DB2eDriver");
 con = DriverManager.getConnection(url); //Step 2a
}
catch (ClassNotFoundException e) {
 com.ibm.db2e.jdbc.DB2eDataSource ds = new com.ibm.db2e.jdbc.DB2eDataSource();
 ds.setUrl(url);
 con = ds.getConnection(); //Step 2b
}
```

   with these lines of code:

```
com.ibm.db2e.jdbc.DB2eDataSource ds = new com.ibm.db2e.jdbc.DB2eDataSource();
ds.setUrl(url);
con = ds.getConnection(); //Step 2b
```

Return to “Compiling and running sample Java applications on non-Palm OS targets” on page 371

Compiling and running sample Java applications on Palm OS targets

Prerequisites:

1. Install WebSphere Studio Device Developer (WebSphere Studio Device Developer) 5.7, which includes the J9 2.2 Java Virtual Machine. If your target is Palm OS, you *must* use the J9 VM that comes with WebSphere Studio Device Developer. The evaluation version of WebSphere Studio Device Developer can be downloaded from http://www.ibm.com/software/pervasive/products/WebSphere Studio Device Developer/.
2. Prepare your target and development environment according to WebSphere Studio Device Developer documentation. Verify your WebSphere Studio Device Developer installation by building and running WebSphere Studio Device Developer sample applications.
3. Install DB2 Everyplace on your target device.

To compile and run sample Java applications on Palm OS targets:

1. Create a WebSphere Studio Device Developer project for DB2eAppl.java.
2. Add the DB2 Everyplace JDBC Driver and java.sql package to the build path.
3. Import DB2eAppl.java into WebSphere Studio Device Developer.

4. Run DB2eAppl.java on a Palm OS emulator.

Related tasks

"Compiling and running sample Java applications on non-Palm OS targets" on page 371

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

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 373.

1. Right-click the **DB2Everyplace-Sample-midp20** 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 <DSYPATH>\Clients\PalmOS\database\JDBC\midp20\DB2eJDBC.jar, then click **Open**.

   Note: <DSYPATH> is the root installation directory for DB2 Everyplace.
4. Back in the Properties window, click **OK**.

Return to "Compiling and running sample Java applications on Palm OS targets" on page 373.

Creating a WebSphere Studio Device Developer project for DB2eAppl.java for Palm OS targets:

You can use the DB2 Everyplace JDBC Driver for Palm OS with the jclMidp20 J9 class library. The DB2eAppl sample project for Palm OS uses the jclMidp20 J9 class library. because the J9 jdbc.jar defined by JSR 169 is not compatible with jclMidp20, support for the java.sql package on Palm OS is included in DB2eJDBC.jar and DB2eJDBC.prc in the <DSYPATH>\Clients\palmos\database\JDBC\midp20 directory, where <DSYPATH> is the root installation directory for DB2 Everyplace. DB2eJDBC.jar is used on the classpath when compiling JDBC applications, and DB2eJDBC.prc, which includes the java.sql package, is installed on the Palm simulator or device.

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.

1. In WebSphere Studio Device Developer, click **Window** → **Open** → **Perspective** → **Java** to switch to the Java Perspective.
2. Click **File** → **New** → **Other**.
3. Create a project for the sample on Palm:
   a. On the Select page of the New Project window, select J2ME in the left pane, select MIDlet Suite in the right pane, then click Next.
   b. On the MIDlet Suite Creation page of the New Project window, enter the following and click Next.
      - Project (MIDP only): DB2Everyplace-Sample-midp20
      - MIDlet Suite Name: DB2eApplSuite
      - MIDlet Name: DB2eAppl
      - MIDlet class Name: DB2eAppl
   c. On the Select J9 page of the New Project window, select ive-2.2 and click Finish.

Return to "Compiling and running sample Java applications on Palm OS targets" on page 373.

Importing DB2eAppl.java into WebSphere Studio Device Developer for Palm OS:

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 373.

1. In the Package Explorer View in the Java Perspective, right-click the *src* folder in the **DB2Everyplace-Sample-midp20** 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** for the **From directory** field.
4. Browse to the <DSYPATH>\Clients\PalmOS\database\JDBC\midp20\sample folder, where <DSYPATH> is the root installation directory of DB2 Everyplace and then click **OK**.
5. Select the **DB2eAppl.java** check box in the right pane, then click **Finish**.
6. Click **Yes** when asked if you want to overwrite DB2eAppl.java in the DB2Everyplace-Sample-midp20\src\ folder.

Return to "Compiling and running sample Java applications on Palm OS targets" on page 373.

The DB2eAppl sample application

This topic explains how to run DB2eAppl Java code on different platforms.

Running DB2eAppl.java on Windows

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: <DSYPATH>\Clients\Win32\database\x86, where <DSYPATH> is the root installation directory for DB2 Everyplace.
2. Using the set command, include the following file in your CLASSPATH system variable: <DSYPATH>\Clients\Win32\database\jdbc\db2ejdbc.jar

Note: If WebSphere Studio Device Developer is open, you will need to restart it for these changes to be reflected in WebSphere Studio Device Developer.

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 371.

1. Build DB2eAppl.java
   a. In the Package Explorer view in the Java Perspective, right-click the DB2Everyplace-Sample project, then click Device Developer Builds.
   b. In the Configure builds window, click **Add**.
   c. In the Create new build window, keep the defaults and click **Next**.
   d. In the General build settings frame of the Create new JXE window, keep the defaults and click **Next**.
   e. In the Target platform frame of the Create new JXE window, select J9 for Windows x86 for the **Platform** field and click **Next**.
   f. In the Contents frame of the Create New JXE window, keep the defaults and click **Finish**.
   g. Back in the Configure builds window, select winx86 Jxe Build and click **Run**.
   h. In the Jxelink warnings window, click **Details**. The warnings indicate that referenced classes in the java.sql package were not found. Ignore these warnings, because those classes are included in the DB2eJDBC.prc file.
   i. Click **OK** to close the Jxelink warnings window.
   j. Click **Close** to close the Progress Information window, which should say `jxelink prc BUILD SUCCESSFUL`.
   k. Click **Close** to close the Configure builds window.

2. Run DB2eAppl.java.
   a. Click **Run** → **Run**. The Run window opens.
   b. In the Run 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. In the Main panel, complete the following steps:
      1) Click **Browse** for the **Project** field. In the Project Selection window, select DB2Everyplace-Sample, then click **OK**.
      2) Click **Search** for the **Main class** field. In the Choose Main Type window, select DB2eAppl, then click **OK**.
   e. In the Classpath panel, uncheck the **Use default class path** checkbox
   f. In the User classes panel, select <WebSphere Studio Device Developer57>/WebSphere Studio Device Developer5.0/technologies/eswe/bundlefiles/jdbc.jar and click **Remove**.
   g. In the Bootstrap classes panel, click **Add External JARs**.
   h. In the Jar Selection window, browse to <WebSphere Studio Device Developer57>/WebSphere Studio Device Developer5.0/technologies/eswe/bundlefiles/jdbc.jar and click **Open**.
   i. Back in the Launch Configurations window, click **Apply**, then click **Run**. You should see output for the sample application in the WebSphere Studio Device Developer Console.

Return to "Compiling and running sample Java applications on non-Palm OS targets" on page 371.

Running DB2eAppl.java on Windows CE

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:

   <DSYPATH>\Clients\WinCE\database*proc**ver*\db2ejdbc.dll, where <DSYPATH> is the root installation directory.

   <DSYPATH>\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. Copy the following file to the *Program Files**J9* directory on your device where *Program Files**J9* is the installation directory of J9:

   <*WebSphere Studio Device Developer57*>\WebSphere Studio Device Developer5.0\technologies\eswe\bundlefiles\jdbc.jar

   where *proc* is the processor type and *ver* is the version number of the Windows CE operating system on your device.
3. 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

   \Program Files\J9\jdbc.jar

   Alternatively, you can update the DB2eAppl shortcut generated by WebSphere Studio Device Developer: `256#"\Program Files\J9\FOUN10\bin\j9.exe" "-Xbootclasspath/a:\Program Files\J9\jdbc.jar" "-classpath" "\Temp\DB2eAppl.jxe;\Windows\db2ejdbc.jar" "-jcl:foun10" "DB2eAppl"`

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 371.

To run DB2eAppl.java on Windows CE:

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 **Location of J9 runtime OR Emulator** field. In the Browse for Folder on Device window, select \Program Files\J9\FOUN10 (assuming that you installed J9 under that directory of your device), then click **OK**.
      3) Click **Browse** for the **Location to install** field. In the Browse for Folder on Device window, select Temp, then click **OK**.
      4) Click **Browse** for the **Location for shortcut** 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 view in the Java Perspective, right-click the DB2Everyplace-Sample project and select **Device Developer Builds**.
   b. In the Configure builds window, click **Add**.
   c. In the Create new build window, keep the defaults and click **Next**.
   d. In the General build settings frame of the Create new JXE window, keep the defaults and click **Next**.
   e. In the Target platform frame of the Create new JXE window, select J9 for Windows Mobile 2003 ARM for the **Platform** field and click **Next**.
   f. In the Contents frame of the Create New JXE window, keep the defaults and click **Finish**.
   g. Back in the Configure builds window, select wm2003arm Jxe Build and click **Run**.
   h. In the Jxelink warnings window, click **Details**. The warnings indicate that referenced classes in the java.sql package were not found. Ignore these warnings, because those classes are included in the DB2eJDBC.prc file.
   i. Click **OK** to close the Jxelink warnings window.
   j. Click **Close** to close the Progress Information window, which should say `jxelink BUILD SUCCESSFUL`.
   k. Click **Close** to close the Configure builds window.
3. Run DB2eAppl.java:
   a. Click **Run** → **Run**. The Run window opens.
   b. In the Run window, select Java on Device on the left pane and click **New**.
   c. In the configuration that appears in the right pane, type `DB2eAppl PocketPC` in the **Name** field.
   d. In the Java Application panel, complete the following steps:
      1) Click **Browse** for the **Project** field.
      2) In the Project Selection window, select DB2Everyplace-Sample and click **OK**.
      3) Select **DB2 Everyplace PocketPC Handheld** from the drop-down list for the **Device or JRE** field.
      4) Select **DB2eAppl.jxe (wm2003arm Jxe Build)** for the **Java Application** field.
   e. Click **Apply** and 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 371.

Running DB2eAppl.java on a Palm OS simulator

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:

<DSYPATH>\Clients\PalmOS\database\JDBC\midp20\DB2eJDBC.prc , where <DSYPATH> is the root installation directory for DB2 Everyplace.

<DSYPATH>\Clients\PalmOS\database\JDBC\ midp20\DB2eJDBCNatives.prc

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 373.

1. Configure the Palm OS simulator:
   a. Click **Devices** → **Configure**.
   b. In the Device Configurations window, select Palm OS 5 Simulator in the left pane, then click **New**.
   c. In the configuration that appears on the right, enter the following information:
      1) In the **Device name** field, type `DB2 Everyplace Palm OS 5 Simulator`.
      2) In the Palm OS Simulator executable field, browse to <*PalmSimulator*>\Release\PalmSim.exe, where <*PalmSimulator*> is the directory where you installed the Palm Simulator.
      3) In the Palm OS Simulator ROM file field, browse to <*PalmSimulator*>\Release\ NTFull_enUS.rom
      4) In the Run arguments field, type

         `-storagesnapshotfile:` *`file.ssf`*where *file.ssf* is a .ssf 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, right-click the DB2Everyplace-Sample-midp20 project and select **Device Developer Builds**.
   b. In the Configure builds window, click **Add**.
   c. In the Create new build window, select **Palm OS 5 database (PRC file)** and click **Next**.
   d. In the General build settings frame of the New PRC file window, keep the default values and click **Next**.
   e. In the Palm application settings frame of the New PRC file window, type `DB2e` in the **Creator ID** field and click **Next**.
   f. In the Contents frame of the New PRC file window, keep the default values and click **Finish**.
   g. In the Configure builds window, select **Palm OS 5 Build** and click **Run**.
   h. In the Jxelink warnings window, click **Details**. The warnings indicate that referenced classes in the java.sql package were not found. Ignore these warnings, because those classes are included in the DB2eJDBC.prc file.
   i. Click **OK** to close the Jxelink warnings window.
   j. Click **Close** to close the Progress Information window, which should say `jad2prc BUILD SUCCESSFUL`.
   k. Click **Close** to close the Configure builds window.
3. Run DB2eAppl.java.
   a. Click **Run** → **Run** from the main menu. The Run window opens.
   b. Select **MIDlet Suite** on the left pane and click **New**.
   c. In the configuration that appears in the right pane, type `DB2eAppl midp20` in the **Name** field.
   d. In the MIDlet Suite panel, enter the following information:
      1) In the Project field, browse to DB2Everyplace-Sample-midp20.

2) For the Device or JRE field, select **DB2 Everyplace Palm OS 5 Simulator**.
3) For the **MIDlet Suite** field, select DB2Everyplace-Sample-midp20.prc (Palm OS 5 Build build).
4) Click **Apply** and then **Run**.

Note: A Palm simulator should start DB2eAppl. You should see output for the sample application on the Palm simulator screen. If you do not see output for the sample application, check the j9stdout.txt and j9stderr.txt files for errors. The files are in the same directory that your .ssf file is in.

Return to "Compiling and running sample Java applications on Palm OS targets" on page 373.

Running DB2eAppl.java on QNX Neutrino or embedded Linux

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.

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 371.

1. Build DB2eAppl.java.
   a. In the Package Explorer view in the Java Perspective, right-click the DB2Everyplace-Sample project, and select **Device Developer Builds**.
   b. In the Configure builds window, **click Add**.
   c. In the Create new build window, keep the defaults and **click Next**.
   d. In the General build settings frame of the Create new JXE window, keep the defaults and click **Next**.
   e. In the Target platform frame of the Create new JXE window, select the appropriate platform for the Platform field and click **Next**.
   f. In the Contents frame of the Create New JXE window, keep the defaults and click **Finish**.
   g. In the Configure builds window, select the appropriate build and click **Run**.
   h. In the Jxelink warnings window, click **Details**. The warnings should be about certain types and classes not being found. Ignore these warnings, because these types and classes will not be accessed by the JDBC driver at runtime.
   i. Click **OK** to close the Jxelink warnings window.
   j. Click **Close** to close the Progress Information window, which should say `jxelink BUILD SUCCESSFUL`.
   k. Click **Close** to close the Configure builds window.
2. Run DB2eAppl.java.
   a. Copy the appropriate DB2eAppl.jxe file to your device from \DB2Everyplace-Sample\<*target*> in your workspace directory for WebSphere Studio Device Developer 5.7, where <*target*> represents the target device and processor type.
   b. Start the application by using the following command:

```
j9 -Xbootclasspath/a:/j9/lib/jdbc.jar -classpath /DB2e/DB2eAppl.jxe:/DB2e/db2ejdbc.jar
-jcl:foun10 DB2eAppl
```

Return to "Compiling and running sample Java applications on non-Palm OS targets" on page 371.

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 371.

Some Symbian devices come with a JVM. To run a text-based Java application (for example, the sample Java programs), you must 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 371.

Sample application code

This topic contains the code in Java and C for the sample application. The code requires a connection string for the SQLConnect() function to connect to the remote data source.

There are two formats for the connection string:

http://*IPAddr:port*/db2e/servlet/ com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample

http://*IPAddr:port*/db2e/agent?DB=mysample

where *IPAddr:port* is the IP address and port number of the server.

For example: http://*192.168.0.11:8080*/db2e/servlet/ com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample

```
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,"/db2e/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;
}
```

Sample application code in Java:

The following Java code has the same function as the C sample.

```
import java.sql.*;

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

   if (args.length != 3){
             System.out.println("Usage java MyClient AccountName Cmd Amount");
             System.out.println("  cmd 1 : Query balance.");
             System.out.println("  cmd 2 : Transfer from Saving to Checking.");
             System.out.println("  cmd 3 : Transfer from Checking to Saving.");
             System.exit(-1);
        }

        String driver = "com.ibm.db2e.jdbc.DB2eDriver";
        String url = "jdbc:db2e:http://9.30.40.21:8080/db2e/servlet/
             com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample";
        String sql = "Call db2e.MYPROC(?,?,?,?,?)";

        try{
            Class.forName(driver);
            //*************************************************************
            //* Connect to remote database
            //*************************************************************
            Connection con = DriverManager.getConnection(url,"userex","userex");

            CallableStatement cst =  con.prepareCall(sql);

            //*************************************************************
            //* Register the output parameters
            //*************************************************************
            cst.registerOutParameter (4, Types.INTEGER);
            cst.registerOutParameter (5, Types.INTEGER);

            //*************************************************************
            //* Set input parameters
            //*************************************************************
            cst.setString (1, args[0]);
            cst.setString (2, args[1]);
            cst.setString (3, args[2]);

            //*************************************************************
            //* Call the Remote Stored Procedure
            //*************************************************************
            cst.execute ();
```

```
            //*************************************************************
            //* Retrieve output
            //*************************************************************
            System.out.println("\nSaving = " + cst.getInt(4));
            System.out.println("\nChecking = " + cst.getInt(5));

            cst.close();
            con.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();
        }
    }
}
```

Related concepts

"The remote query and stored procedure adapter" on page 49
DB2 Everyplace includes a remote query and stored procedure adapter. This adapter enables DB2 Everyplace application to use the DB2 Everyplace Sync Server architecture to call a stored procedure located at a remote data source.

Related tasks

"Creating the Custom subscription for the sample application" on page 54

"Using the remote query and stored procedure adapter" on page 50
The remote query and stored procedure adapter supports the Windows 32-bit (non-Unicode), Windows CE, Symbian, and Palm OS client platforms. The remote query and stored procedure adapter require stored procedures to be registered to DB2.

Related reference

"Creating a stored procedure using the sample application" on page 51

"Testing the remote query and stored procedure adapter" on page 54

SQL support in DB2 Everyplace

This topic explains the SQL features that are supported by DB2 Everyplace.

Supported SQL statements in DB2 Everyplace

Supported executable SQL statements can be issued interactively from the mobile device by using the DB2eCLP, or they can be used in applications to access data in a DB2 Everyplace mobile database. The following table lists the SQL statements supported by DB2 Everyplace.

Table 238. Supported SQL statements

SQL statement	Function
ALTER TABLE	Modifies a table by adding one or more columns or by changing the length of one or more VARCHAR columns.
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.

Table 238. Supported SQL statements (continued)

SQL statement	Function
DATE	Returns a date from a value.
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.
LOCK TABLE	Acquires a shared or exclusive table lock on a specified table
REORG TABLE	Removes or reduces the wasted storage associated with the specified table.
REVOKE	Revokes a user's encryption privileges.
ROLLBACK	Backs out of the database changes that were made within a unit of work or a savepoint.
SAVEPOINT	Sets a savepoint within a transaction.
SELECT	Specifies a result table queried from one or more tables.
TIME	Returns a time from a value.
TIMESTAMP	Returns a timestamp from a value.
UPDATE	Updates the values of one or more columns in one or more rows of a table.

"SQLState messages reported by SQL" on page 441 lists 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 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

ALTER TABLE

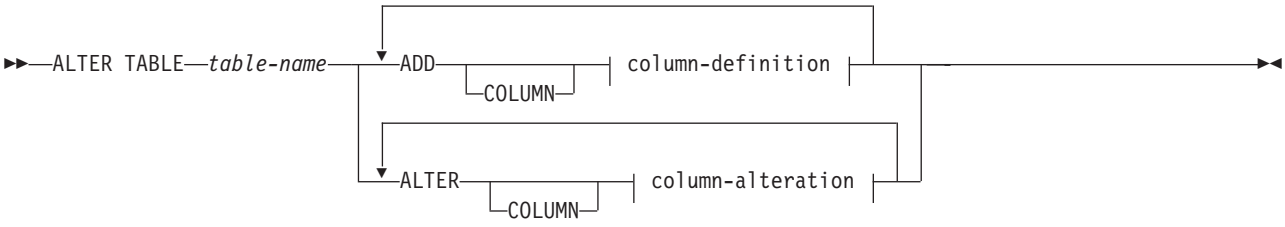
The ALTER TABLE statement modifies existing tables by:

- Adding one or more columns to a table.
- Changing the length of one or more VARCHAR columns.

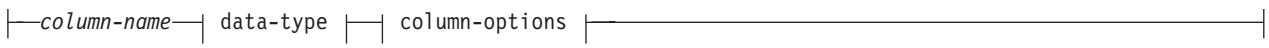
Invocation

This statement can be used in an application using DB2 CLI functions or issued through the DB2eCLP application.

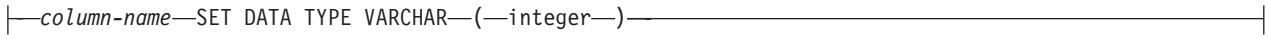
Syntax



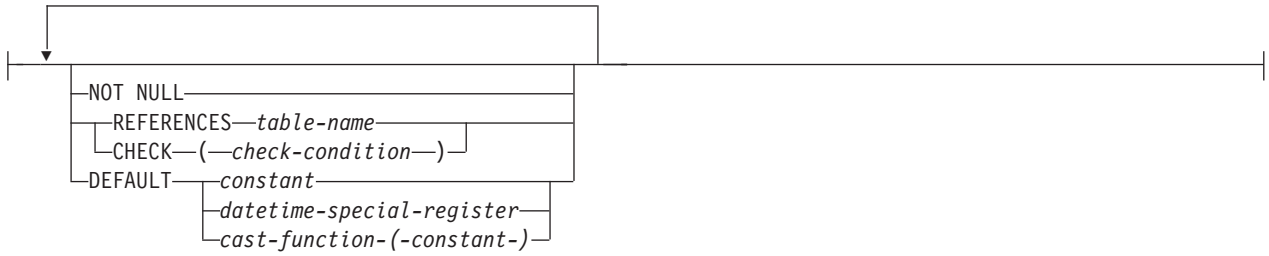
column-definition::



column-alteration::



column-options::



Description

table-name

Specifies the table to alter. The name can be up to 128 bytes long. The name must identify a table in the catalog.

You must use delimited identifiers (with double quotation marks) 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.

column-name

Specifies a column of the table. The name can be up to 128 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 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 supported by the CREATE TABLE statement.

Is one of the types supported by the CREATE TABLE statement.

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.

REFERENCES *table-name*
See the description of REFERENCES in the next topic.

CHECK (*check-condition***)**
See the description of CHECK in the next topic.

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 a 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.

  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).

cast-function
Specifies the cast-function as the default value for the column. This form of a default value can only be used with columns that are defined as a BLOB or datetime (DATE, TIME or TIMESTAMP) data type.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the data type. If the cast-function is BLOB, the constant must be a string constant.

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 32767 bytes for this combined check constraint.

Rules

- The actual total of byte counts of a row must not be greater than 65,536.
- 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

- System tables cannot be altered by the user. Any attempt will result in SQLSTATE 42832.
- Adding a primary key column is not supported. Any attempt will result in SQLSTATE 42601. You can however, add columns with default values, check, and referential constraints.
- Like the CREATE TABLE statement, an ALTER TABLE statement can be rolled back in a transaction.
- When altering the length of a column, the column to be altered must be a VARCHAR type and the specified length must be equal to or greater than the existing column length; otherwise, SQLSTATE 42837.
- Combining ADD and ALTER COLUMN clauses in an ALTER TABLE statement will result in a syntax error.
- A REORG operation can get invoked after an ALTER TABLE ADD COLUMN statement has been executed successfully. This depends on the table size and the user table reorg threshold level.
- Columns should be created using uppercase names. Mixed case and lowercase names might cause errors to occur with some languages.
- Byte counts for data: The following list contains the byte counts of columns by data type. This count might change with each release of DB2 Everyplace. 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

Data type	Column byte count
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

The following example shows some of the ways that ALTER TABLE can be used.

```
CREATE TABLE t1 (c1 INT PRIMARY KEY NOT NULL, c2 VARCHAR(10));

CREATE TABLE t2 (c1 DATE);

CREATE TABLE t3 (c1 TIME, c2 INT PRIMARY KEY NOT NULL);

ALTER TABLE t2 ADD COLUMN c2 INT REFERENCES t1;

ALTER TABLE t2 ADD c3 INT CHECK (c3 > 1) DEFAULT 10 ADD c4 DECIMAL(5,2) NOT NULL;

ALTER TABLE t2 ADD c5 TIMESTAMP DEFAULT CURRENT TIMESTAMP ADD COLUMN c6 CHAR(20)
DEFAULT 'xyz' ADD c7 INT REFERENCES t3;

CREATE TABLE t4 (c1 INT, c2 VARCHAR(2), c3 VARCHAR(10));

ALTER TABLE t1 ALTER c2 SET DATA TYPE VARCHAR(20)
ALTER c3 SET DATA TYPE VARCHAR(100);
```

CALL

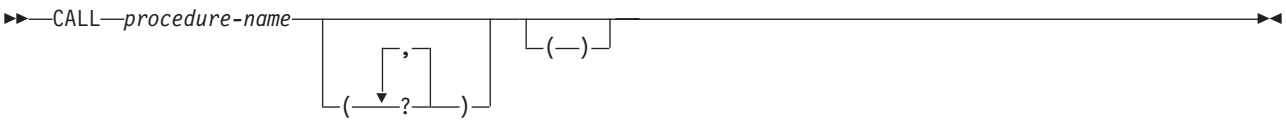
Invokes a stored procedure defined with the Remote Query and Stored Procedure Adapter for the DB2 Everyplace Sync Server. A stored procedure executes at the location of the remote database and returns data to the DB2 Everyplace client application.

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 Custom 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 Serveris required to use the CALL statement in DB2 Everyplace applications. DB2 Everyplace does not support local stored procedures.

Important: When you retrieve a result set using stored procedure, the entire result set must fit on the mobile device.

Example

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`. A Custom subscription is defined at the DB2 Everyplace Sync Serverwith 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

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

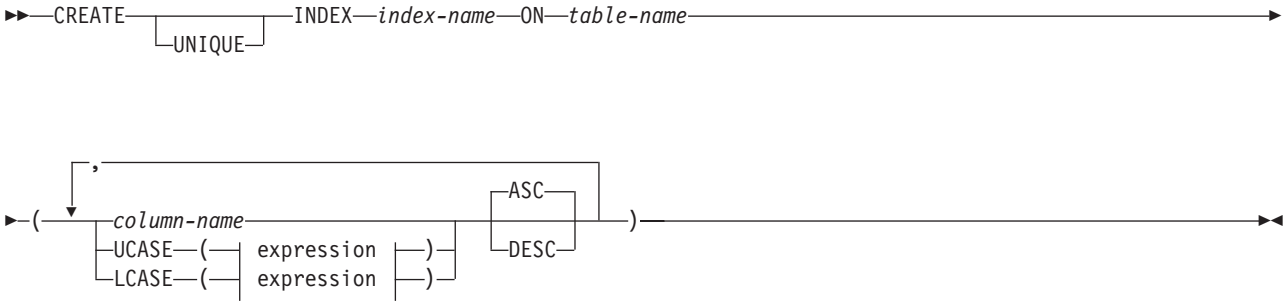
CREATE INDEX

The CREATE INDEX statement creates 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 DB2eCLP.

Syntax



Description

UNIQUE

If the table specified by ON table-name exists, UNIQUE prevents the table from containing two or more rows with the same value of the index key. DB2 Everyplace enforces the uniqueness at the end of each SQL statement that updates rows or inserts new rows. DB2 Everyplace also enforces the uniqueness during the execution of the CREATE INDEX statement. The index will not be created if the table already contains rows that have duplicate key values.

When you use the UNIQUE option, null values are treated as any other values. For example, if the key is a single column that can contain null values, that column can contain no more than one null value.

Important: Do not create unique indices for synchronization purposes. Using multiple mobile devices with poorly distributed unique values can cause a uniqueness constraint violation.

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/UCASE functions take a string of characters as input and return a string in which all the characters are converted to lowercase and uppercase characters, respectively. 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 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.

The alphabetic characters of the argument are translated based on the value of the LC_CTYPE locale in effect for the statement. For example, characters a-z are translated to A-Z, and characters with diacritical marks are translated to their LCASE/UCASE equivalent, if any. Characters that cannot be converted will remain unconverted in the string.

Important: Version 8 of DB2 Everyplace used an algorithm to determine the LCASE and UCASE forms of Latin1 characters. Version 9 of DB2 Everyplace uses operating system functions to support characters outside of the Latin1 character set. As a result, the behavior of applications that use LCASE/UCASE on Linux, Neutrino, Palm, and Symbian platforms might change when these programs are run on DB2 Everyplace version 9. SELECT queries that use LCASE/UCASE in WHERE clauses might return different results. If the SELECT queries use indexes created with LCASE/UCASE, the indexes might need to be dropped and recreated after migration to produce proper results.

Example: You might have information in a column that is a mixture of UCASE and LCASE data. To make searches easier, you might want DB2 Everyplace to treat all data as a single case. Use the following SQL command to create an index the entries of the JOB column in the EMPLOYEE table as UCASE data in ascending form.

```
CREATE INDEX IDX_JOB ON EMPLOYEE (UCASE(JOB) ASC);
```

The following table shows the default locales that the LCASE/UCASE functions use on each platform.

Table 239. Default locales that are used by the LCASE/UCASE functions

Operating System	Locale
Linux	Default operating system locale. Can be overridden with setlocale() function.
Palm OS	Default operating system locale
QNX Neutrino	C
Symbian OS	Default operating system locale
Windows	Default operating system locale. Can be overridden with _tsetlocale() function.
Windows CE	Default operating system locale

Restriction: On Windows systems, you must link the application with the same C Runtime Library as DB2e.dll. In Visual Studio, select **Settings** → **C/C++ Code Generation** → **Use Runtime Library** → **Multithreaded DLL**.

Restriction: On QNX Neutrino systems, the LCASE and UCASE functions can only translate ASCII characters.

The following examples show how to override the system locale to German on Windows and Linux systems.

Windows systems

```
#include <locale.h>
_tsetlocale(LC_CTYPE, TEXT("German"));
```

Linux systems

```
#include <locale.h>
setlocale(LC_CTYPE, "de_DE.UTF-8");
```

You can also export the environment variable LC_CTYPE to your target locale, for example:

```
export LC_CTYPE = "en_US.UTF-8"
```

After you export the locale to an environment variable, you can set that locale in your application by using the following code:

```
#include <locale.h>
setlocale(LC_CTYPE, "");
```

Rules

- The CREATE INDEX statement can contain a maximum of 8 columns.
- 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

- 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 "The dirty bit" on page 254 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

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

CREATE TABLE

The CREATE TABLE statement defines a table.

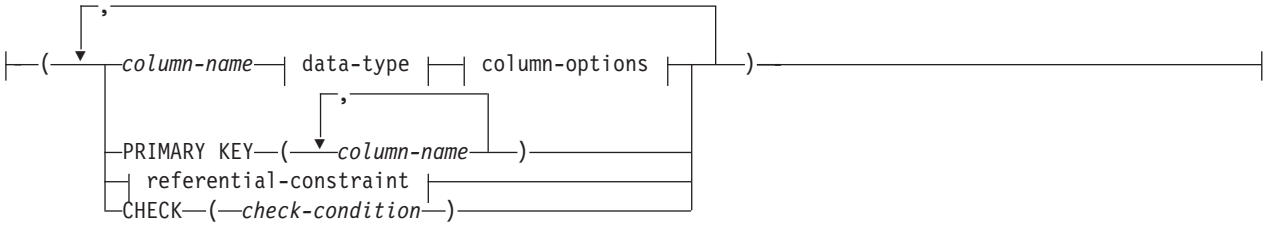
The definition must include its name and the names and attributes of its columns. The definition can also include other attributes of the table, such as its primary key.

Invocation

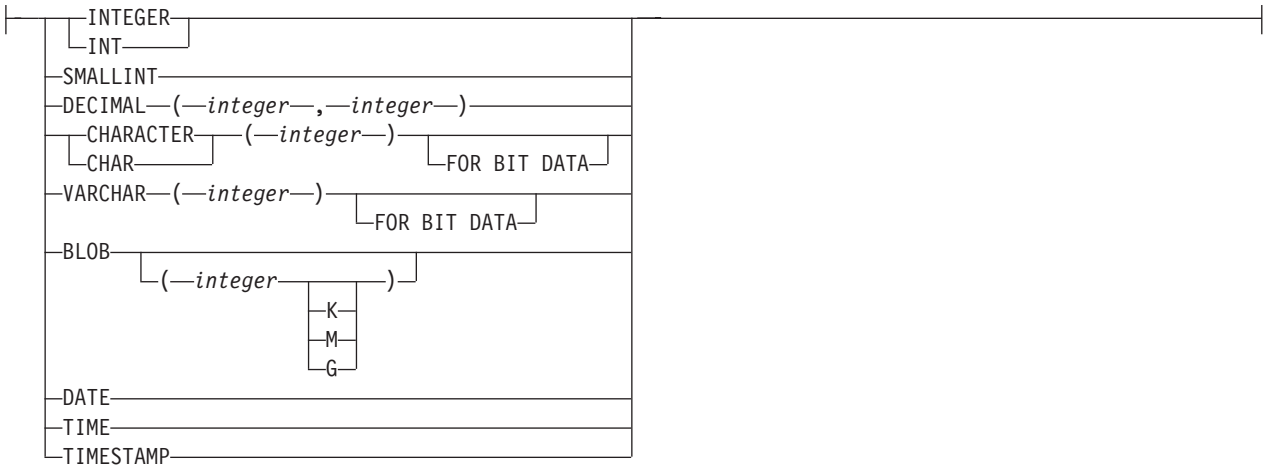
This statement can be used in an application program using the DB2 CLI functions or issued through the DB2eCLP.

Syntax

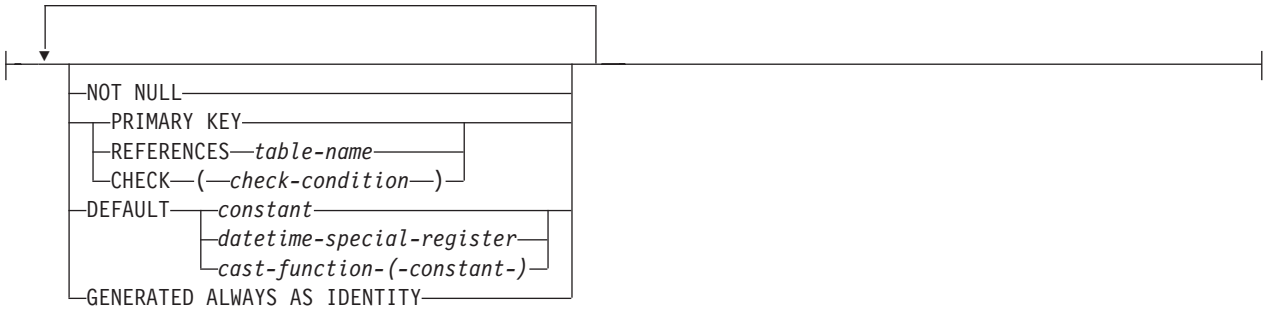
element-list:



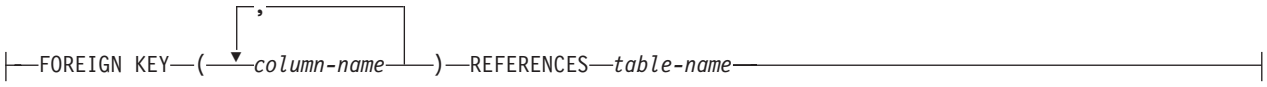
data-type:



column-options:



referential-constraint:



Description

table-name

Names the table. The name can be up to 128 bytes long. The table name can include Double Byte Character Set characters. The name must not identify a table in the catalog. The name must be unique for the mobile device.

DB2 Everyplace converts the names of the tables to uppercase before it stores the names 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.

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. Before you can encrypt a table, use the GRANT statement to grant yourself the permission to create encrypted tables. For more information, see “GRANT” on page 408.

You can encrypt a table only when you create it. You cannot add or remove encryption on an existing table.

column-name
Names a column of the table. The name can be up to 128 bytes long. The column name can include DBCS characters. The name cannot be qualified and the same name cannot be used for more than one column of the table.

DB2 Everyplace converts the names of the columns to uppercase before it stores the names 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.

data-type
Is one of the types in the following list::

INTEGER or INT
A four-byte signed integer in the range of 2147483647 to -2147483648.

SMALLINT
A two-byte signed integer in the range of -32768 to 32767.

DECIMAL*(precision-integer, scale-integer)*
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)* **or CHARACTER***(integer)*
A fixed-length character string of length *integer*, which can range from 1 to 32767.

FOR BIT DATA
Specifies that the contents of the column are to be treated as bit (binary) data. The length can range from 1 byte to 254 bytes. DB2 Everyplace does not perform code page conversions when it exchanges data with other systems. DB2 Everyplace performs the comparisons in binary, irrespective of the database collating sequence. CHAR FOR BIT DATA maps to the following data types:

VARCHAR*(integer)*
A varying-length character string of maximum length *integer*, which can range from 1 to 32767.

FOR BIT DATA
Specifies that the contents of the column are to be treated as bit (binary) data. The length can range from 1 byte to 32672 bytes. DB2 Everyplace does not perform code page conversions when it exchanges data with other systems. DB2 Everyplace performs the comparisons in binary, irrespective of the database collating sequence. VARCHAR FOR BIT DATA maps to the following data types:

BLOB or BINARY LARGE OBJECT*(integer [K | M | G])*

The short forms *K*, *M* and *G* can be used as length modifiers in table creations.

- *K* stands for kilobyte (1024 bytes)
- *M* stands for megabyte (1024 K)
- *G* stands for gigabytes (1024 M)

For a binary large object string of the specified maximum length in bytes.

The length can be in the range of 1 byte to 2 147 483 647 bytes.

If *integer* by itself is specified, that is the maximum length.

If *integer K* (in either upper- or lowercase) is specified, the maximum length is 1 024 times *integer*. The maximum value for integer is 2 097 152. If a multiple of K, M or G that calculates out to 2 147 483 648 is specified, the actual value used is 2 147 483 647 (or 2 gigabytes minus 1 byte), which is the maximum length for a LOB column.

If *integer M* is specified, the maximum length is 1 048 576 times *integer*. The maximum value for *integer* is 2 048.

If *integer G* is specified, the maximum length is 1 073 741 824 times *integer*. The maximum value for *integer* is 2.

If the length specification is omitted, a length of 1 048 576 (1 megabyte) is assumed.

Any number of spaces is allowed between the *integer* and K, M, or G, and a space is not required.

For example:

```
CREATE TABLE t1 (c1 BLOB(5M));  // 5 M BLOB
CREATE TABLE t2 (c1 blob( 20 k ), c2 blob (1G));  // 20 K and 1 G BLOB
CREATE TABLE t3 (c1 BLOB (1024)); // 1 K BLOB
```

DATE
A date. An input value can be in one of the following formats: MM/DD/YYYY, YYYY-MM-DD, or DD.MM.YYYY. DB2 Everyplace prints the date value only in the ISO format, YYYY-MM-DD. The year of a date value can range from 0001 to 9999.

The special register CURRENT DATE also produces the current date in ISO format.

TIME
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. DB2 Everyplace prints the time value only in the ISO format, HH:MM:SS.

The special register CURRENT TIME also produces the current time in ISO format.

TIMESTAMP
A timestamp. An input value must be in the following format: YYYY-MM-DD-HH.MM.SS.ZZZZZZ. DB2 Everyplace prints the timestamp value 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. These column options can be any of the following options:

NOT NULL
Prevents the column from containing null values.

If you do not specify NOT NULL, the column can contain null values. The default value of the column is either the null value or the value provided by the DEFAULT clause.

PRIMARY KEY
Provides a shorthand method of defining a primary key composed of a single column. If you specify PRIMARY KEY in the definition of column C, the effect is the same as if you specify PRIMARY KEY(C) as a separate clause.

REFERENCES *table-name*
see the description of *referential-constraint*.

CHECK (*check-condition***)**
see the description of *referential-constraint*.

DEFAULT
Provides a default value in the event that a value is not supplied on an INSERT statement.

If you do not include DEFAULT from a column-definition, DB2 Everyplace uses the null value as the default for the column. If you define a column NOT NULL, then the column does not have a valid default. If the default clause cannot be specified, DB2 Everyplace returns SQLState 42623.

constant
Specifies a constant as the default value for the column. The specified constant must:

- Represent a value that could be assigned to the column

  Example: `CREATE TABLE t1 (c1 INT DEFAULT 100, c2 VARCHAR(10) FOR BIT DATA DEFAULT x'FFFF');`
- Not have non-zero digits beyond the scale of the column data type if the constant is a decimal constant.

  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.

Example: If you specify CURRENT DATE, the data type of the column must be DATE.

cast-function
Specifies the cast-function as the default value for the column. You can use this form of a default value only with columns that are defined as a BLOB or datetime (DATE, TIME or TIMESTAMP) data type.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the data type. If the cast-function is BLOB, the constant must be a string constant.

GENERATED ALWAYS AS IDENTITY
When creating a table, you can specify a column as "GENERATED ALWAYS AS IDENTITY". The value of this column is generated by DB2 Everyplace each time you perform an INSERT or INSERT with sub–SELECT. This column must be a numeric type (INTEGER, SMALLINT, or DECIMAL). 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 an IDENTITY column reaches its maximum value, further INSERT statements 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:

- The precision and scale of the data type
- The maximum value allowed for IDENTITY column: 2.15* (10∧18) (19 decimal digits)

The smaller of these restrictions is the range limit. For an IDENTITY column of a DECIMAL type, the fractional part of the value is always 0. DB2 Everyplace increases the integral part of the type by 1 each time.

You can only define the IDENTITY specification 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 is 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 “Notes” 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

- If you specify too many columns in the table definition, DB2 Everyplace returns SQLSTATE 54011. See “DB2 Everyplace limits” on page 105 for information about the maximum number of columns per table.
- 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 is not 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 can contain character data. NOT NULL means that the column cannot contain a null value. VARCHAR means that the column can 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)
```

COMMIT

The COMMIT statement terminates a unit of work and commits the database changes that were made by that unit of work.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Syntax

Description

The unit of work in which the COMMIT statement is executed is terminated and a new unit of work is initiated. All changes made by the following statements executed during the unit of work are committed: ALTER, CREATE, DROP, GRANT, LOCK TABLE, REVOKE, and the data change statements (INSERT, DELETE, UPDATE).

All locks acquired by the unit of work subsequent to its initiation are released.

Open cursors remain open, and the cursor is positioned before the next logical row of the result set. All LOB locators are freed. Note that this is true even when the locators are associated with LOB values retrieved via a cursor.

All savepoints that were set within the transaction are released.

Notes

Each application process should explicitly end its unit of work before it terminates. If the application program ends normally without a COMMIT or ROLLBACK statement, the database engine will implicitly roll back the last active unit of work.

Example

Commit alterations to the database made since the last commit point.

```
COMMIT WORK
```

Related reference

| “COMMIT” on page 401
| The COMMIT statement terminates a unit of work and commits the database changes that were made
| by that unit of work.

DATE

The DATE function returns a date from a value.

Invocation

This statement can be used in an application using DB2 CLI functions or issued through the DB2eCLP.

Syntax

Description

expression
Specifies a value. The argument must be a date, timestamp, or a valid string representation of a date or timestamp.

Rules

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example

Assume that the column RECEIVED (of data type TIMESTAMP) has a data value of ‘2003-12-25-17.12.30.000000’.

- DATE(RECEIVED) is ‘2003-12-25’ (of data type DATE).
- DATE(’2003-12-25’) is ‘2003-12-25’ (of data type DATE).
- DATE(’25.12.2003’) is ‘2003-12-25’ (of data type DATE).

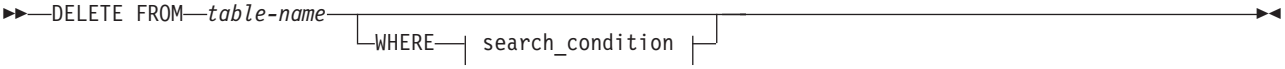
DELETE

The DELETE statement deletes one or more rows from a table.

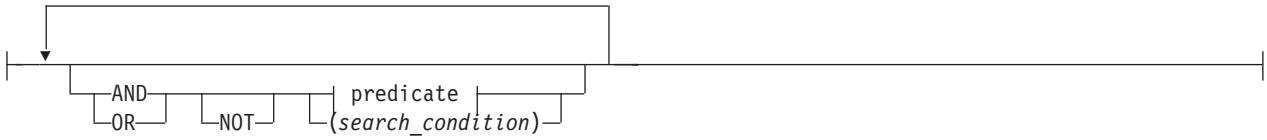
Invocation

This statement can be used in an application program using the DB2 CLI functions or issued through the DB2eCLP.

Syntax



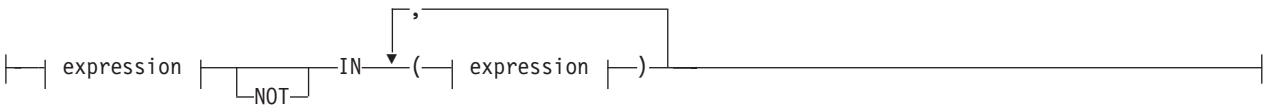
search_condition:



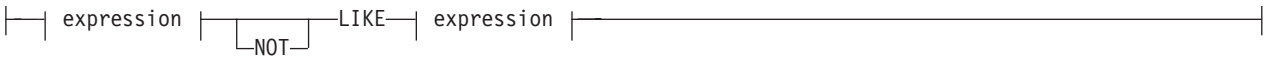
predicate:

basic predicate:

IN predicate:



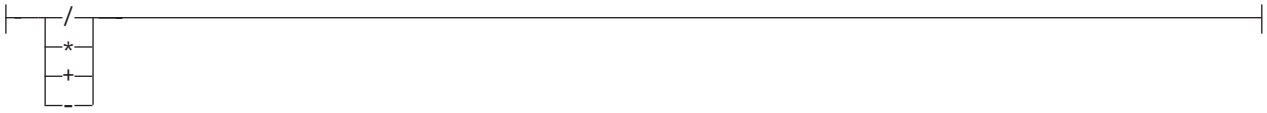
LIKE predicate:



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

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

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 DB2eCLP.

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

- Dropping tables or indexes when a table is in use (that is, when a statement handle is active on a query that uses the table or index) will invalidate the corresponding statement handles.

Example

Drop table EMPLOYEE.

```
DROP TABLE EMPLOYEE
```

Related reference

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

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:

- Windows (Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, and Windows 2003)
- Linux

Invocation

This statement can be used in an application program using the DB2 CLI functions or issued through the DB2eCLP.

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(128), index_name char(128),
sort_temp char(1), expl_timestamp timestamp, remarks varchar(300))
```

  Table 240 describes DB2ePLANTABLE columns.

Table 240. 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

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

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 about how to do this.)

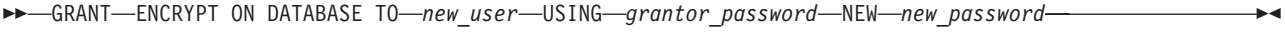
When GRANT statements are issued against other users, the access rights for those users take effect on subsequent connect calls, unless the connection is modifying its own access rights.

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 DB2eCLP.

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.
- While executing GRANT in a manual transaction, SELECT and DML statements will be blocked until that transaction is committed or rolled back.

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

“Supported SQL statements in DB2 Everyplace” on page 383

“Data type compatibility for assignments and comparisons” on page 435

“DB2 Everyplace supported parameter markers” on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

“SQLState listing” on page 445
This topic will help you interpret error messages generated from SQL or CLI.

“Summary of SQLState class codes” on page 445

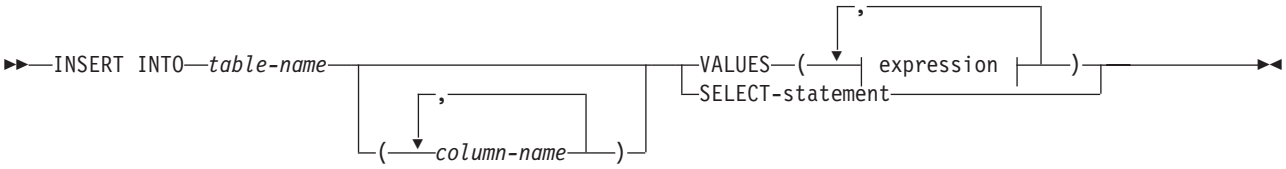
INSERT

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 DB2eCLP.

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 can 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 Version 9.1 documentation.

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.

3. You cannot insert values into an Oracle ROWID column. If you attempt to insert values into this type of column, DB2 Everyplace returns SQLSTATE 428C9.

Related reference

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

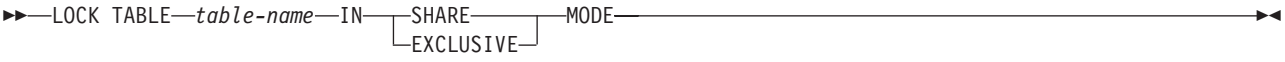
LOCK TABLE

The LOCK TABLE statement allows a user to explicitly acquire a shared or exclusive table lock on a specified table. The table lock lasts until the end of the current transaction. You cannot lock system tables with this statement.

Invocation

This statement can be used in an application using DB2 CLI functions or issued through the DB2eCLP.

Syntax



Description

table-name
Specifies the table to lock. The name can be up to 128 bytes long. The name must identify a table in the catalog.

You must use delimited identifiers (with double quotation marks) 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.

Notes

- You cannot use this statement to lock system tables. Any attempt will result in SQLSTATE 42832.
- DB2 Everyplace provides a timeout mechanism that applications can use to resolve deadlocks. If an application cannot obtain a lock within a specified amount of time, the database engine rolls back the transaction and returns SQLSTATE 40001. The default lock timeout is 20 seconds.

Example

The following code obtains an exclusive lock on the table EMP.

```
LOCK TABLE EMP IN EXCLUSIVE MODE
```

RELEASE SAVEPOINT

Use the RELEASE SAVEPOINT statement when you no longer want to have the named savepoint maintained. After you invoke this statement, you can no longer rollback to that savepoint.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Syntax

Description

savepoint-name

Specifies the savepoint that is to be released. DB2 Everyplace also releases all savepoints that might be nested within the named savepoint. Rollback to that savepoint, or any savepoint nested within it, is no longer possible. If the named savepoint does not exist in the current savepoint level (see the ″Rules″ section in the description of the SAVEPOINT statement), DB2 Everyplace returns SQLSTATE 3B001. The specified savepoint-name cannot begin with 'SYS' (SQLSTATE 42939).

Notes

Once you release a savepoint, you can reuse the name in another SAVEPOINT statement, regardless of whether the UNIQUE keyword was specified on an earlier SAVEPOINT statement specifying this same savepoint name.

Example

Release a savepoint named SAVEPOINT1.

```
RELEASE SAVEPOINT SAVEPOINT1
```

Related reference

"SAVEPOINT" on page 417
Use the SAVEPOINT statement to set a savepoint within a transaction.

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 DB2eCLP.

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

- REORG TABLE can only be performed in autocommit mode (autocommit=on or autocommit=true).
- Calling REORG TABLE inside a transaction will result in a SQLState 42887.
- 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 (for example, 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 (for example, 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.
- Implicit reorganizations of the system catalog tables after a DROP TABLE statement are triggered only when autocommit mode is on.
- No other activity should be occurring in the database while a REORG TABLE statement is being executed. If any other statements are executed against the database while a data reorganization is in progress, a SQLSTATE of 57011 will be returned.

Examples

The following command reorganizes the VNNURSE table using the default values.

```
REORG TABLE VNNURSE
```

Related reference

“Supported SQL statements in DB2 Everyplace” on page 383

“Data type compatibility for assignments and comparisons” on page 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

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 DB2eCLP.

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.
- When revoke statements are issued against other users, the access rights for those users take effect on subsequent CONNECT calls, unless the connection is modifying its own access rights.
- 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.
- While executing REVOKE in a manual transaction, SELECT and DML statements will be blocked until that transaction is committed or rolled back.
- When a SELECT privilege is revoked from a user, the effect takes effect on the next SELECT statement.

Example

The currently connected, authenticated user removes encryption privileges from user ″jsk″:

```
REVOKE ENCRYPT ON DATABASE FROM "jsk"
```

Related reference

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

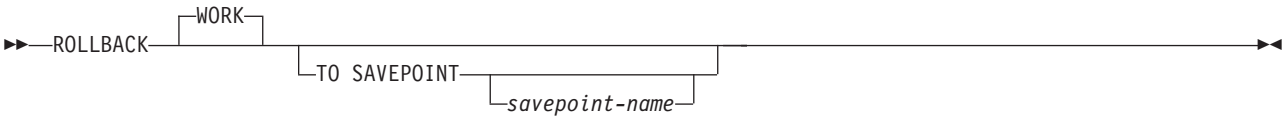
ROLLBACK

The ROLLBACK statement is used to back out of the database changes that were made within a unit of work or a savepoint.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Syntax



Description

The unit of work in which the ROLLBACK statement is executed is terminated and a new unit of work is initiated. All changes made to the database during the unit of work are backed out.

The generation of sequence and identity values is not under transaction control. Values generated by inserting rows into a table that has an identity column are independent of issuing the ROLLBACK statement.

TO SAVEPOINT
Specifies that a partial rollback (ROLLBACK TO SAVEPOINT) is to be performed. If no savepoint is active in the current savepoint level (see the "Rules" section in the description of the SAVEPOINT statement), DB2 Everyplace returns an error with SQLSTATE 3B502. After a successful rollback, the savepoint continues to exist, but any nested savepoints are released and no longer exist. The nested savepoints, if any, are considered to have been rolled back and then released as part of the rollback to the current savepoint. If a savepoint-name is not provided, rollback occurs to the most recently set savepoint within the current savepoint level.

If this clause is omitted, the ROLLBACK statement rolls back the entire transaction. Furthermore, savepoints within the transaction are released.

savepoint-name
Specifies the savepoint to which DB2 Everyplace reverts during the rollback operation. DB2 Everyplace reverses all data and schema changes that were made since the savepoint was set. After a successful rollback operation, the savepoint continues to exist.

When you use the *savepoint-name* argument, ensure that you abide by the following restrictions:

- The specified *savepoint-name* cannot begin with 'SYS'. If you specify a savepoint-name that begins with 'SYS', DB2 Everyplace returns SQLSTATE 42939.
- If the *savepoint-name* does not exist, DB2 Everyplace returns an error with SQLSTATE 3B001.

Notes

- All locks held are released on a ROLLBACK of the unit of work. All open cursors are closed. All LOB locators are freed.
- If the program terminates abnormally, the unit of work is implicitly rolled back.
- The impact on cursors resulting from a ROLLBACK TO SAVEPOINT depends on the statements within the savepoint:
  - If the savepoint contains DDL or DML on which a cursor is dependent, the cursor is marked invalid. If you try to use this cursor, DB2 Everyplace returns an error with SQLSTATE 57007.
  - If the cursor is referenced in the savepoint, the cursor remains open and is positioned before the next logical row of the result set.
  - Otherwise, the cursor remains opened and positioned and is not affected by the ROLLBACK TO SAVEPOINT statement.
- Dynamically prepared statement names are still valid, although the statement may be implicitly prepared again, as a result of DDL operations that are rolled back within the savepoint.
- After a ROLLBACK TO SAVEPOINT statement, all locks are retained if the savepoint specified by savepoint-name was created with the ON ROLLBACK RETAIN LOCKS option. Otherwise, all locks acquired after the savepoint will be released.
- All LOB locators are preserved following a ROLLBACK TO SAVEPOINT operation.

Example

Delete the alterations that have been made since the last commit or rollback.

```
ROLLBACK WORK
```

Related reference

"COMMIT" on page 401
The COMMIT statement terminates a unit of work and commits the database changes that were made by that unit of work.

"SAVEPOINT"
Use the SAVEPOINT statement to set a savepoint within a transaction.

SAVEPOINT

Use the SAVEPOINT statement to set a savepoint within a transaction.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Syntax

Description

savepoint-name
Specifies the name of a savepoint. The specified savepoint-name cannot begin with 'SYS' (SQLSTATE 42939). If a savepoint by this name has already been defined as UNIQUE within this savepoint level, an error is returned (SQLSTATE 3B501).

UNIQUE
Specifies that the application does not intend to reuse this savepoint name while the savepoint is active within the current savepoint level. If savepoint-name already exists within this savepoint level, an error is returned (SQLSTATE 3B501).

ON ROLLBACK RETAIN CURSORS
Specifies system behavior upon rollback to this savepoint with respect to cursors opened after the SAVEPOINT statement. This clause indicates that, whenever possible, the cursors are unaffected by a ROLLBACK TO SAVEPOINT operation. For situations where the cursors are affected by the rollback to savepoint, see "ROLLBACK". Unless this clause is specified, all open cursors will be closed upon a ROLLBACK TO SAVEPOINT statement that specifies this savepoint.

ON ROLLBACK RETAIN LOCKS
Specifies system behavior upon rollback to this savepoint with respect to locks acquired after the setting of the savepoint. Locks acquired since the savepoint are not tracked, and are not rolled back (released) upon rollback to the savepoint. If you do not specify this clause, on the next rollback event DB2 Everyplace releases all locks that were acquired after the savepoint.

Rules

- A new savepoint level starts whenever a new transaction starts. A savepoint level ends when the corresponding transaction ends. Whenever a savepoint level ends, all savepoints contained within it are released.
- The following rules apply to actions within a savepoint level:
  - Savepoints can only be referenced within the savepoint level in which they are established. You cannot release, destroy, or roll back to a savepoint established outside of the current savepoint level.
  - All active savepoints established within the current savepoint level are automatically released when the savepoint level ends.
  - The uniqueness of savepoint names is only enforced within the current savepoint level. The names of savepoints that are active in other savepoint levels can be reused in the current savepoint level without affecting those savepoints in other savepoint levels.

Notes

- Omitting the UNIQUE clause specifies that savepoint-name can be reused within the savepoint level by another savepoint. If a savepoint of the same name already exists within the savepoint level, the existing savepoint is destroyed and a new savepoint with the same name is created at the current point in processing. The new savepoint is considered to be the last savepoint established by the application. Note that the destruction of a savepoint through the reuse of its name by another savepoint simply destroys that one savepoint and does not release any savepoints established after the destroyed savepoint. These subsequent savepoints can only be released by means of the RELEASE SAVEPOINT statement, which releases the named savepoint and all savepoints established after the named savepoint.
- If the UNIQUE clause is specified, savepoint-name can only be reused after an existing savepoint with the same name has been released.

Example

Use this example to perform a rollback operation for nested savepoints. First, create a table named DEPARTMENT. Insert a row before starting SAVEPOINT1; insert another row and start SAVEPOINT2; then, insert a third row and start SAVEPOINT3.:

```
CREATE TABLE DEPARTMENT (DEPTNO   CHAR(6),
                         DEPTNAME VARCHAR(20),
                         MGRNO    INTEGER)

INSERT INTO DEPARTMENT VALUES ('A20', 'MARKETING', 301)

SAVEPOINT SAVEPOINT1 ON ROLLBACK RETAIN CURSORS
```

```
INSERT INTO DEPARTMENT VALUES ('B30', 'FINANCE', 520)

SAVEPOINT SAVEPOINT2 ON ROLLBACK RETAIN CURSORS

INSERT INTO DEPARTMENT VALUES ('C40', 'IT SUPPORT', 430)

SAVEPOINT SAVEPOINT3 ON ROLLBACK RETAIN CURSORS

INSERT INTO DEPARTMENT VALUES ('R50', 'RESEARCH', 150)
```

At this point, the DEPARTMENT table exists with rows A20, B30, C40, and R50. If you now issue the command:

```
ROLLBACK TO SAVEPOINT SAVEPOINT3
```

row R50 is no longer in the DEPARTMENT table. If you then issue:

```
ROLLBACK TO SAVEPOINT SAVEPOINT1
```

the DEPARTMENT table still exists, but the rows inserted since SAVEPOINT1 was established (B30 and C40) are no longer in the table.

Related reference

"ROLLBACK" on page 416
The ROLLBACK statement is used to back out of the database changes that were made within a unit of work or a savepoint.

"RELEASE SAVEPOINT" on page 413
Use the RELEASE SAVEPOINT statement when you no longer want to have the named savepoint maintained. After you invoke this statement, you can no longer rollback to that savepoint.

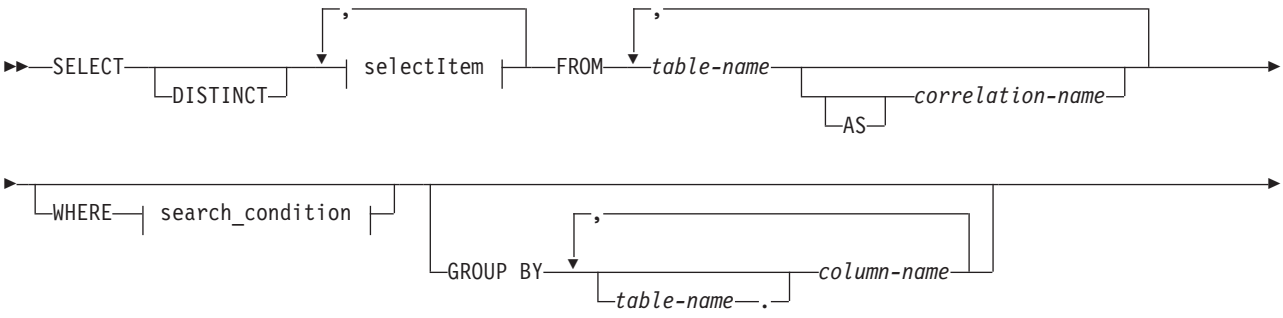
SELECT

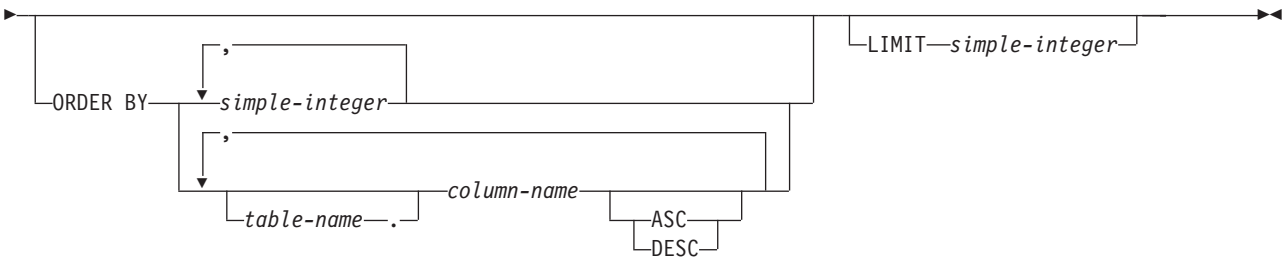
The SELECT statement is a form of query.

Invocation

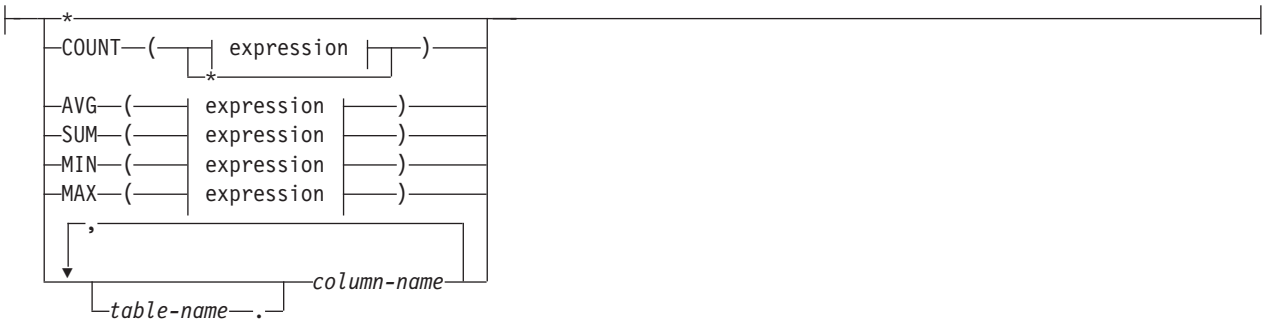
This statement can be used in an application program using the DB2 CLI functions or issued through the DB2eCLP.

Syntax

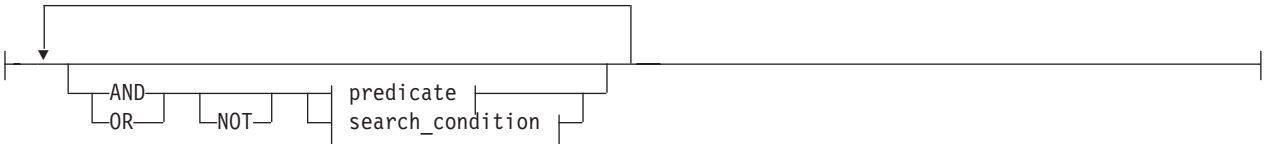




selectItem:



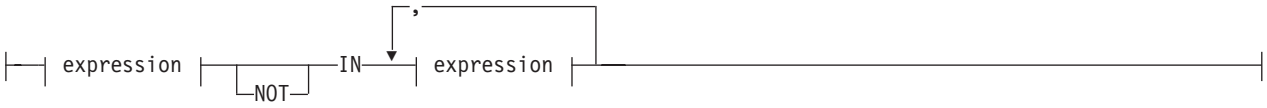
search_condition:



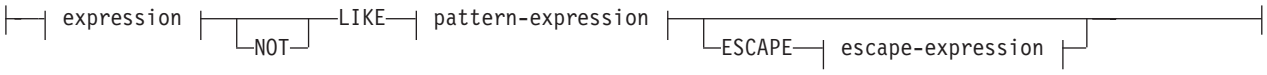
predicate:

basic predicate:

IN predicate:

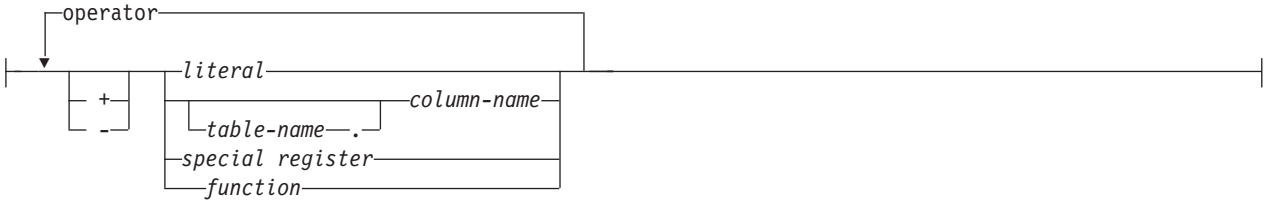


LIKE predicate:



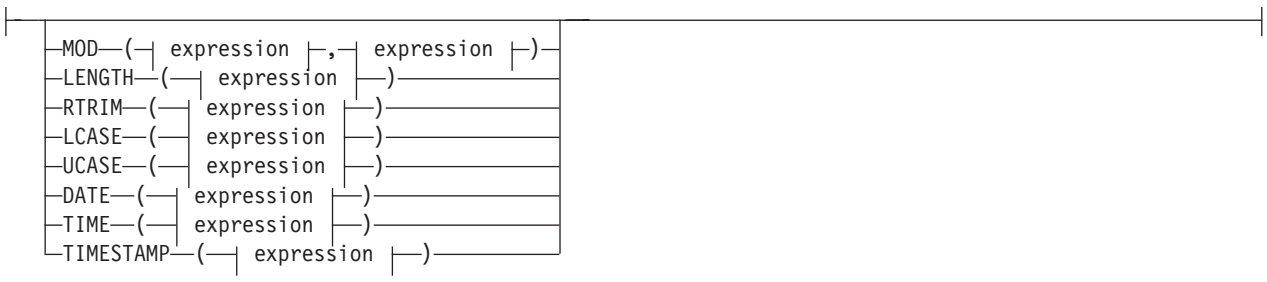
NULL predicate:

expression:



operator:

function:



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, RTRIM, LCASE, UCASE, DATE, TIME, and TIMESTAMP.

Arithmetic operations on CHAR, VARCHAR, BLOB(n) data types are not supported. Only subtraction is allowed on DATE, TIME, and TIMESTAMP.

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.

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.

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). Only subtraction is supported for DATE, TIME and TIMESTAMP.

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 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

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.

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/UCASE functions take a string of characters as input and return a string in which all the characters are converted to lowercase and uppercase characters, respectively. 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 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.

The alphabetic characters of the argument are translated based on the value of the LC_CTYPE locale in effect for the statement. For example, characters a-z are translated to A-Z, and characters with diacritical marks are translated to their LCASE/UCASE equivalent, if any. Characters that cannot be converted will remain unconverted in the string.

Important: Version 8 of DB2 Everyplace used an algorithm to determine the LCASE and UCASE forms of Latin1 characters. Version 9 of DB2 Everyplace uses operating system functions to support characters outside of the Latin1 character set. As a result, the behavior of applications that use LCASE/UCASE on Linux, Neutrino, Palm, and Symbian platforms might change when these programs are run on DB2 Everyplace version 9. SELECT queries that use LCASE/UCASE in WHERE clauses might return different results. If the SELECT queries use indexes created with LCASE/UCASE, the indexes might need to be dropped and recreated after migration to produce proper results.

Example: To ensure that the characters in the value of column JOB in the EMPLOYEE table are returned in LCASE characters, use the following SQL statement:

```
SELECT LCASE(JOB)
     FROM EMPLOYEE
     WHERE EMPNO = '000020';
```

The following table shows the default locales that the LCASE/UCASE functions use on each platform.

Table 241. Default locales that are used by the LCASE/UCASE functions

Operating System	Locale
Linux	Default operating system locale. Can be overridden with setlocale() function.
Palm OS	Default operating system locale
QNX Neutrino	C
Symbian OS	Default operating system locale
Windows	Default operating system locale. Can be overridden with _tsetlocale() function.
Windows CE	Default operating system locale

Restriction: On Windows systems, you must link the application with the same C Runtime Library as DB2e.dll. In Visual Studio, select **Settings** → **C/C++ Code Generation** → **Use Runtime Library** → **Multithreaded DLL**.

Restriction: On QNX Neutrino systems, the LCASE and UCASE functions can only translate ASCII characters.

The following examples show how to override the system locale to German on Windows and Linux systems.

Windows systems

```
#include <locale.h>
_tsetlocale(LC_CTYPE, TEXT("German"));
```

Linux systems

```
#include <locale.h>
setlocale(LC_CTYPE, "de_DE.UTF-8");
```

You can also export the environment variable LC_CTYPE to your target locale, for example:

```
export LC_CTYPE = "en_US.UTF-8"
```

After you export the locale to an environment variable, you can set that locale in your application by using the following code:

```
#include <locale.h>
setlocale(LC_CTYPE, "");
```

DATE
The DATE function returns a date from an expression that evaluates to a DATE, TIMESTAMP, or string data type.

TIME
The TIME function returns a time from an expression that evaluates to a TIME, TIMESTAMP, or string data type.

TIMESTAMP
The TIMESTAMP function returns a date from an expression that evaluates to a TIMESTAMP or string data type.

special register
The special registers CURRENT DATE, CURRENT TIME, and CURRENT TIMESTAMP can be used to produce the current date, current time, and current timestamp.

basic predicate
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

pattern-expression
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 two-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).

To match string data that contains percent or underscore characters, DB2 Everyplace supports the ESCAPE clause in LIKE predicates.

escape-expression

This optional argument specifies a character that changes the meaning of the underscore and percent characters in an expression, allowing the LIKE predicate to match values that contain percent and underscore characters. DB2 Everyplace does not support a default escape character. Use *escape-expression* to indicate the escape character.

The value of *escape-expression* can be specified by a constant or a host variable with the restriction that the result of the expression must be one character containing exactly one byte, and must not be the percent or underscore character itself (SQLState 22019). If *escape-expression* is invalid, DB2 Everyplace returns SQLState 22025.

An underscore, percent sign, or escape character can represent a literal occurrence of itself when *pattern-expression* contains an escape character. This is true when the character in question is preceded by an odd number of successive escape characters. It is not true otherwise.The following examples show the effect of successive occurrences of the escape character which, in this case, is the backslash.

Table 242. Escape pattern samples

Pattern string	Actual pattern
\%	Percent sign
\\%	Backslash followed by zero or more arbitrary characters
\\\%	Backslash followed by a percent sign

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.

operator
Can be one of the following operators

+ Add

- Subtract

* Multiple

/ Divide by

|| 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.

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
```

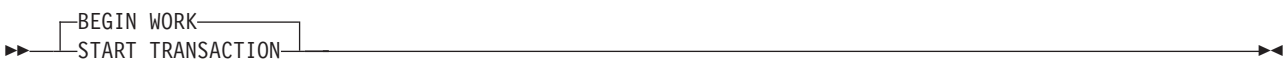
START TRANSACTION

The START TRANSACTION statement starts a new unit of work.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Syntax



Description

When you issue START TRANSACTION, DB2 Everyplace treats subsequent statements as a new unit of work. If this statement is executed within a unit of work, an error is returned (SQLState 25501).

Rules

A START TRANSACTION statement cannot be executed within a unit of work.

Notes

This statement explicitly starts a new unit of work for a database connection that uses auto-commit mode.

Example

Start a new unit of work for a database connection that currently uses auto-commit mode.

```
START TRANSACTION
```

Related reference

"COMMIT" on page 401
The COMMIT statement terminates a unit of work and commits the database changes that were made by that unit of work.

"ROLLBACK" on page 416
The ROLLBACK statement is used to back out of the database changes that were made within a unit of work or a savepoint.

TIME

The TIME function returns a time from a value.

Invocation

This statement can be used in an application using DB2 CLI functions or issued through the DB2eCLP.

Syntax

Description

expression
Specifies a value. The argument must be a time, timestamp, or a valid string representation of a time or timestamp.

Rules

The result of the function is a time: the time part of the argument. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example

Assume that the column RECEIVED (of data type TIMESTAMP) has a data value of '2003-12-25-17.12.30.000000'.

- TIME(RECEIVED) is '17:12:30' (of data type TIME).
- TIME('17.12.30') is '17:12:30' (of data type TIME).
- TIME('05:12 PM') is '17:12:00' (of data type TIME).

Related reference

"Subtraction rules for DATE, TIME, and TIMESTAMP" on page 439

TIMESTAMP

The TIMESTAMP function returns a timestamp from a value.

Invocation

This statement can be used in an application using DB2 CLI functions or issued through the DB2eCLP.

Syntax

Description

expression
Specifies a value. The argument must be a timestamp or a valid string representation of a timestamp.

Rules

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Notes

The DB2 Everyplace JDBC getString() method called on a TIMESTAMP column returns a timestamp in ISO format "YYYY-MM-DD-HH.MM.SS.ZZZZZZ". This is different from the JDBC timestamp escape format "YYYY-MM-DD HH:MM:SS.FFFFFFFFF". DB2 Version 9.1 JDBC returns a timestamp in the format "YYYY-MM-DD HH:MM:FFFFFF".

Example

```
If HOUR(TS2) <= HOUR(TS1)

then HOUR(RESULT) = HOUR(TS1) - HOUR(TS2).

If HOUR(TS2) > HOUR(TS1)

then HOUR(RESULT) = 24 + HOUR(TS1) - HOUR(TS2)

and DAY(TS2) is incremented by 1.
```

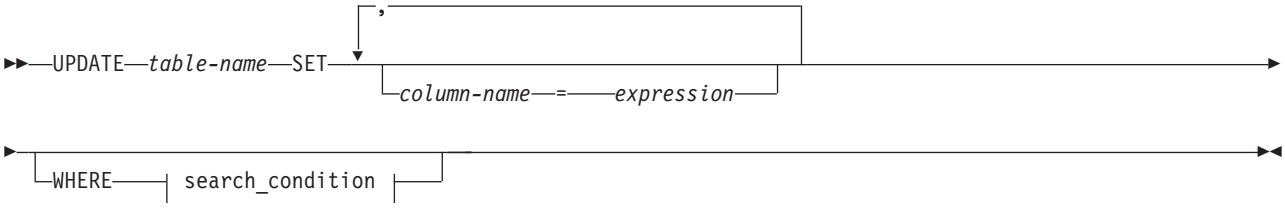
UPDATE

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 DB2eCLP.

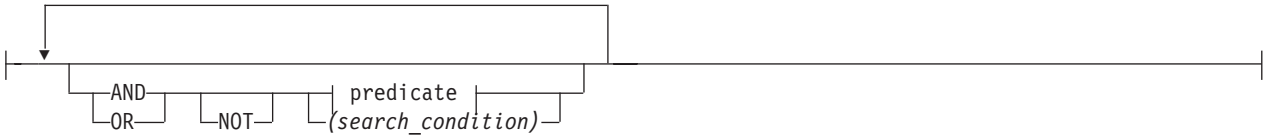
Syntax



expression:

operator:

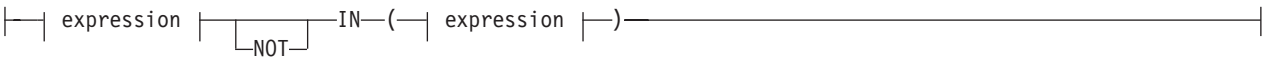
search_condition:



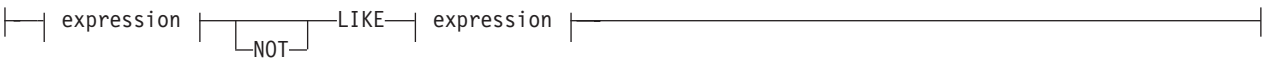
predicate:

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 "SELECT" on page 419.

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 Version 9.1.
- UPDATE never applies to logically deleted records.
- You cannot update values into an Oracle ROWID column. If you attempt to update values in this type of column, DB2 Everyplace returns SQLSTATE 428C9.

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 "The dirty bit" on page 254 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

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"Summary of SQLState class codes" on page 445

Supported data types for stored procedures

DB2 Everyplace supports calling stored procedures on a remote DB2 server through the CLI or JDBC interface. The client application uses the CALL statement to run the remote stored procedure. The CALL statement names the procedure to be called and specifies its parameters. The following types are supported: INTEGER, SMALLINT, DECIMAL, CHAR, VARCHAR, DATE, TIME, TIMESTAMP and BLOB.

Related concepts

"The remote query and stored procedure adapter" on page 49
DB2 Everyplace includes a remote query and stored procedure adapter. This adapter enables DB2 Everyplace application to use the DB2 Everyplace Sync Server architecture to call a stored procedure located at a remote data source.

"Restrictions for result sets" on page 55

Related tasks

"Creating the Custom subscription for the sample application" on page 54

"Using the remote query and stored procedure adapter" on page 50
The remote query and stored procedure adapter supports the Windows 32-bit (non-Unicode), Windows CE, Symbian, and Palm OS client platforms. The remote query and stored procedure adapter require stored procedures to be registered to DB2.

Related reference

"Sample application code" on page 380
This topic contains the code in Java and C for the sample application. The code requires a connection string for the SQLConnect() function to connect to the remote data source.

"Creating a stored procedure using the sample application" on page 51

"Testing the remote query and stored procedure adapter" on page 54

DB2 Everyplace supported parameter markers

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

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.

Table 243. 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

SQL symbolic and default data types

Table 244 describes the symbolic and default data types for each SQL data type.

Table 244. 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 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 245 on page 436 through Table 247 on page 436.

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 245. 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 246. 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 247. 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	
TIMESTAMP			X

Related reference

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"Supported SQL statements in DB2 Everyplace" on page 383

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. Table 248 defines the precision for each SQL data type.

Table 248. 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.
SQL_SMALLINT	5. The *cbParamDef* argument of SQLBindParameter() is ignored for this data type.
SQL_INTEGER	10. The *cbParamDef* argument of SQLBindParameter() is ignored for this data type.
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	10 (the number of characters in the yyyy-mm-dd format). The *cbParamDef* argument of SQLBindParameter() is ignored for this data type.
SQL_TIME	8 (the number of characters in the hh:mm:ss format). The *cbParamDef* argument of SQLBindParameter() is ignored for this data type.
SQL_TIMESTAMP	26 (The number of characters in the "yyyy-mm-dd-hh.mm.ss.ffffff" format used by the TIMESTAMP 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. Table 249 defines the scale for each SQL data type.

Table 249. 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 might be different than the number of bytes required to store the data on the data source.

Table 250defines the length for each SQL data type.

Table 250. 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. because 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. Table 251 defines the display size for each SQL data type.

Table 251. 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).

Subtraction rules for DATE, TIME, and TIMESTAMP

The only arithmetic operation that can be performed on datetime values is subtraction. The subtraction operator can be used with datetime values only when both operands are dates, or both operands are times, or both operands are timestamps.

DATE

The result of subtracting one date (DATE2) from another (DATE1) is a date duration that specifies the number of years, months, and days between the two dates. The data type of the result is DECIMAL(8,0). For example, the result of DATE('3/15/2000') – DATE('12/31/1999') is 00000215 (a duration of 0 years, 2 months, and 15 days).

If DATE1 is greater than or equal to DATE2, DATE2 is subtracted from DATE1. If DATE1 is less than DATE2, however, DATE1 is subtracted from DATE2, and the sign of the result is made negative. The following procedural description clarifies the steps involved in the operation result = DATE1 - DATE2.

```
 If DAY(DATE2) <= DAY(DATE1)

then DAY(RESULT) = DAY(DATE1) - DAY(DATE2).

If DAY(DATE2) > DAY(DATE1)

then DAY(RESULT) = N + DAY(DATE1) - DAY(DATE2)

where N = the last day of MONTH(DATE2).

MONTH(DATE2) is then incremented by 1.

If MONTH(DATE2) <= MONTH(DATE1)

then MONTH(RESULT) = MONTH(DATE1) - MONTH(DATE2).
```

```
If MONTH(DATE2) > MONTH(DATE1)

then MONTH(RESULT) = 12 + MONTH(DATE1) - MONTH(DATE2).

YEAR(DATE2) is then incremented by 1.

YEAR(RESULT) = YEAR(DATE1) - YEAR(DATE2).
```

TIME

The result of subtracting one time (TIME2) from another (TIME1) is a time duration that specifies the number of hours, minutes, and seconds between the two times. The data type of the result is DECIMAL(6,0). For example, the result of TIME('11:02:26') – TIME('00:32:56') is 102930 (a duration of 10 hours, 29 minutes, and 30 seconds).

If TIME1 is greater than or equal to TIME2, TIME2 is subtracted from TIME1. If TIME1 is less than TIME2, however, TIME1 is subtracted from TIME2, and the sign of the result is made negative. The following procedural description clarifies the steps involved in the operation result = TIME1 - TIME2.

```
If SECOND(TIME2) <= SECOND(TIME1)

then SECOND(RESULT) = SECOND(TIME1) - SECOND(TIME2).

If SECOND(TIME2) > SECOND(TIME1)

then SECOND(RESULT) = 60 + SECOND(TIME1) - SECOND(TIME2).

MINUTE(TIME2) is then incremented by 1.

If MINUTE(TIME2) <= MINUTE(TIME1)

then MINUTE(RESULT) = MINUTE(TIME1) - MINUTE(TIME2).

If MINUTE(TIME1) > MINUTE(TIME1)

then MINUTE(RESULT) = 60 + MINUTE(TIME1) - MINUTE(TIME2).

HOUR(TIME2) is then incremented by 1.

HOUR(RESULT) = HOUR(TIME1) - HOUR(TIME2).
```

TIMESTAMP

The result of subtracting one timestamp (TS2) from another (TS1) is a timestamp duration that specifies the number of years, months, days, hours, minutes, seconds, and microseconds between the two timestamps. The data type of the result is DECIMAL(20,6).

If TS1 is greater than or equal to TS2, TS2 is subtracted from TS1. If TS1 is less than TS2, however, TS1 is subtracted from TS2 and the sign of the result is made negative. The following procedural description clarifies the steps involved in the operation result = TS1 - TS2:

```
If MICROSECOND(TS2) <= MICROSECOND(TS1)

then MICROSECOND(RESULT) = MICROSECOND(TS1) - MICROSECOND(TS2).

If MICROSECOND(TS2) > MICROSECOND(TS1)

then MICROSECOND(RESULT) = 1000000 + MICROSECOND(TS1) - MICROSECOND(TS2)

and SECOND(TS2) is incremented by 1.
```

The seconds and minutes part of the timestamps are subtracted as specified in the rules for subtracting times. The date part of the timestamps is subtracted as specified in the rules for subtracting dates.

Related reference

"DATE" on page 402

"SELECT" on page 419
The SELECT statement is a form of query.

"TIME" on page 429

"TIMESTAMP" on page 430

SQLState messages in DB2 Everyplace

This topic explains the SQLstate return values for API specifications.

SQLState messages reported by JDBC

Table 252 describes the messages that are only returned by JDBC. The JDBC driver can also return the messages listed under "SQLState messages reported by SQL" and "SQLState messages reported by CLI".

Table 252. 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.
58030	I/O error.	An I/O error has occurred.
HY001	Memory allocation failure.	DB2 Everyplace is unable to allocate memory that is required to support execution or completion of the function.
HY009	Invalid argument value.	The argument value for start was not greater than 0.
HY013	Unexpected memory handling error.	DB2 Everyplace is unable to access memory that is required to support execution or completion of the function.
HY090	Invalid string or buffer length.	Search pattern is null.
S1010	Function sequence error.	`CallableStatement` get method is called without first calling `registerOutParameter`.
XJ010	Savepoint cannot be set.	A savepoint cannot be set because savepoints are not supported in auto-commit mode.
XJ011	Invalid savepoint name.	Null for savepoint name is not allowed.
XJ013	No ID for named savepoints.	No ID information available for named savepoints.
XJ014	No name for un-named savepoints.	No name information is available for un-named savepoints.
XJ081	Invalid parameter value passed to a JDBC interface method.	One of the parameters that was passed to a JDBC method has an invalid value.

SQLState messages reported by SQL

Table 253 on page 442 lists all the SQLSTATEs for the SQL statements reported by the DB2 Everyplace SQL engine. The SQLSTATEs reported by DB2 CLI are listed in Table 158 on page 261.

Table 253. 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.
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.
22019	Invalid escape character in LIKE predicate.	The LIKE predicate has an invalid escape character.
22025	Invalid escape sequence.	The LIKE predicate string pattern contains an invalid escape sequence.
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.
23522	The range of values for an identity column or sequence is exhausted.	The range of values for an identity column or sequence is exhausted.
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.
24504	Invalid cursor state.	The cursor identified in the UPDATE, DELETE, SET, or GET statement is not positioned on a row.
25501	Statement not allowed in context.	The statement is only allowed as the first statement in a unit of work.
3B001	Savepoint cannot be set.	A savepoint cannot be set because savepoints are not supported in auto-commit mode.

Table 253. SQLSTATE messages reported by SQL (continued)

SQLSTATE	Description	Explanation
3B002	Maximum number of savepoints reached.	The maximum number of savepoints has been reached.
3B501	Savepoint already exists.	A savepoint with the same name already exists, and this name cannot be reused.
3B502	Savepoint does not exist.	A savepoint with the specified name does not exist.
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.
42604	An invalid numeric or string constant has been detected.	A numeric or string constant is invalid.
42606	An invalid hexadecimal constant has been detected.	A hexadecimal constant is invalid.
42610	Invalid use of a parameter marker.	The statement contains a parameter marker that is not valid. See Table 15 on page 49 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.
42623	A DEFAULT clause cannot be specified.	A DEFAULT clause cannot be specified.
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.
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.

Table 253. SQLSTATE messages reported by SQL (continued)

SQLSTATE	Description	Explanation
42815	The data type, length, scale, value, or CCSID is invalid.	The data type, length, scale, value, or CSSID is invalid.
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.
42837	Column cannot be altered.	The column cannot be altered, because its attributes are not compatible with the currrent column attributes.
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.
428C1	Only one ROWID column can be specified for a table.	Only one ROWID column can be specified for a table.
428C9	A ROWID column cannot be specified as the target column of an INSERT or UPDATE.	A ROWID column cannot be specified as the target column of an INSERT or UPDATE.
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.
54002	A string constant is too long.	A string constant 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.
54011	Too many columns were specified for a table or view.	Too many columns were specified for a table or view.
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.

Table 253. SQLSTATE messages reported by SQL (continued)

SQLSTATE	Description	Explanation
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 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.
58005	Internal system error (stop).	A severe system error occurred.

Related reference

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"SQLState listing"
This topic will help you interpret error messages generated from SQL or CLI.

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

"Summary of SQLState class codes"

SQLState listing

This topic 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 441, "SQLState messages reported by CLI" on page 261, and "SQLState messages reported by JDBC" on page 441 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 Version 9.1 installed:

1. To open the command line processor, select **Start → Programs → DB2 → Command Line Processor** .
2. On the command line, type `? [SQLSTATE]`.

Related reference

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"Summary of SQLState class codes"

Summary of SQLState class codes

The first two characters of the SQLState messages in Table 158 on page 261 are in **bold** to indicate the class code. These class codes are *summarized* in Table 254.

Table 254. 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

Table 254. SQLState class codes (continued)

Code	Class
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
3B	Invalid savepoint
34	Invalid cursor name
38	External function exception
39	External function call exception
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

"Supported SQL statements in DB2 Everyplace" on page 383

"Data type compatibility for assignments and comparisons" on page 435

"SQLState listing" on page 445
This topic will help you interpret error messages generated from SQL or CLI.

"DB2 Everyplace supported parameter markers" on page 49
DB2 Everyplace supports only untyped parameter markers, which can be used in selected locations of an SQL statement. This topic lists the restrictions on parameter marker usage.

Glossary

Special characters

$DSYINSTDIR
Refers to the directory where DB2 Everyplace is installed on a Linux or UNIX computer.

<DSYPATH>
Refers to the directory where DB2 Everyplace is installed on a Windows computer.

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 control database to ensure that the user is authorized to use the DB2 Everyplace 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 Mobile Devices Administration Center.

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 Mobile Devices Administration Center object containing synchronization error messages and their descriptions.

M

master database
See *source database* .

MDAC
See *Mobile Devices Administration Center*.

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 DB2 Everyplace 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.

Mobile Devices Administration Center (MDAC)
A graphical interface that allows you to create, edit, and view synchronization objects and their relationships to each other. The Mobile Devices Administration Center also allows you to view synchronization status of individual clients and error messages.

O

object Anything that can be created or manipulated with SQL, for example, tables, views, indexes, or packages. 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

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.

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 Mobile Devices Administration Center 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 Mobile Devices Administration Center 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.

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
USA

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 Corporation
J46A/G4
555 Bailey Avenue

San Jose, CA 95141-1003
USA

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.

AIX
AS/400
Cloudscape
DataPropagator
DB2
DB2 Connect
DB2 Universal Database
Everyplace
IBM
Microsoft
iSeries
OS/390
WebSphere
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies:

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names, which may be denoted by a double asterisk (**) may be trademarks or service marks of others.

Index

Special characters

A

B

C

D

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Program Number: 5724–D04

Printed in USA

SC18-9996-00