# IBM DB2 Universal Database Building Applications for UNIX** Environments Version 5.2

Document Number S10J-8161-01

IBM DB2 Universal Database

# Building Applications for UNIX** Environments

*Version 5.2*

S10J-8161-01

IBM DB2 Universal Database

# Building Applications for UNIX** Environments

*Version 5.2*

S10J-8161-01

Before using this information and the product it supports, be sure to read the general information under Appendix E, “Notices” on page 185.

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

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

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

© **Copyright International Business Machines Corporation 1993, 1998. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

# Contents



© Copyright IBM Corp. 1993, 1998

# About This Book

This book explains how to build applications using the DB2 Software Developer's Kits (DB2 SDKs) for DB2 Universal Database Version 5.2 on the following UNIX operating systems:

- AIX
- HP-UX
- SCO UnixWare 7
- Silicon Graphics IRIX
- Solaris

The book provides information to set up your environment for developing DB2 applications, and step-by-step instructions to compile, link, and run these applications in this environment.

Different programming interfaces can be used to develop your applications:

| | |
|---|---|
| **Embedded SQL** | Uses SQL statements that are precompiled before your program is compiled. |
| **DB2 Call Level Interface (CLI)** | Is a callable SQL interface based on the X/Open CLI specification, and is compatible with Microsoft Corporation's Open Database Connectivity (ODBC) interface. |
| **Java Database Connectivity (JDBC)** | Is a dynamic SQL API for Java. The JDBC API is included in the Java Development Kits available for supported platforms. |
| **DB2 Application Programming Interfaces (APIs)** | Use administrative functions in your applications to create administrative programs to manage DB2 databases. |

For information on these programming interfaces, and to decide which one best fits your needs, refer to the *Road Map to DB2 Programming*, especially chapter 2, "Deciding which Programming Interface to Use".

For more detailed information on each of the different programming interfaces, refer to:

- *Embedded SQL Programming Guide*

  Discusses how to code and design application programs that access DB2 family servers using embedded SQL and Java Database Connectivity (JDBC).

- *CLI Guide and Reference*

© Copyright IBM Corp. 1993, 1998

Explains how to code and design application programs that use the DB2 Call Level Interface and ODBC.

- *API Reference*

  Discusses how to code and design application programs that use DB2 Application Programming Interfaces.

You will find the following books useful for further related information, such as detailed product installation and setup:

- *Quick Beginnings for UNIX*

  Explains how to install the database manager, and the DB2 Software Developer's Kit (DB2 SDK) on server and client workstations.

- *Command Reference*

  Explains how to use the DB2 Command Line Processor (CLP).

- *Troubleshooting Guide*

  Helps you resolve application development problems involving DB2 clients and servers, as well as problems with related tasks in database administration and connectivity.

Important new information for those of the above books not re-issued since the release of DB2 Universal Database Version 5 is documented in *What's New*. This includes information for the *Embedded SQL Programming Guide* and the *CLI Guide and Reference*. For a complete list of the DB2 documentation library, see Appendix D, “How the DB2 Library Is Structured” on page 175.

**Notes:**

1. The examples in this book are provided "as is" without any warranty of any kind. The user, and not IBM, assumes the entire risk of quality, performance, and repair of any defects.
2. A revision bar, "|", on the left side of a page indicates that the line on the same level has been modified or added since the book was last issued with the release of DB2 Universal Database Version 5.

## Who Should Use This Book

You should use this book if you want to develop applications on one of the currently supported UNIX platforms for DB2 Universal Database Version 5.2. You may use embedded SQL, the DB2 CLI, or JDBC to access DB2 databases, or DB2 APIs to create administrative programs.

In order to use this book, you should know one or more of the supported programming languages on the UNIX platform you will be using. These languages are listed in “Supported Software by Platform” on page 2.

## How To Use This Book

The book is designed to allow easy access to the information needed to develop your applications. The first two chapters contain common information for users who will be developing either embedded SQL, DB2 CLI, JDBC , or DB2 API applications on any of these platforms, and should therefore be read by all users. Chapter 3 contains common information for all those who want to develop embedded SQL applications.

Each of Chapters 4 through 8 gives detailed information for developing embedded SQL applications on one of the supported platforms, except for embedded SQL for Java (SQLJ), which is discussed in Chapter 10.

The DB2 API script file for each supported compiler in Chapters 4 through 8 is noted after the first embedded SQL script file for the compiler is discussed, as these files share the same compile and link options.

Chapter 9 contains common information for all those developing DB2 CLI applications.

Chapter 10 contains common information for building DB2 Java programs. It contains the information needed for developing JDBC applications and applets for DB2. It also covers developing embedded SQL for Java (SQLJ) applications and applets for DB2.

To use this book, a user who wanted, for example, to develop embedded SQL applications on Solaris should read Chapters 1, 2, 3, and 8. A user who wanted to develop DB2 CLI applications on any of the platforms should read Chapters 1, 2, and 9. A user who wanted to develop Java applications or applets for DB2 on a supporting platform should read Chapters 1, 2, and 10.

Since DB2 API calls can be made from either the embedded SQL, CLI, or JDBC programming interfaces, a user who wanted to develop DB2 API applications using one of these interfaces should read the appropriate set of chapters given above.

## Highlighting Conventions

This book uses the following conventions:

*Italics* Indicate one of the following:

- Introduction of a new term
- Names or values that are supplied by the user
- References to another source of information
- General emphasis

UPPERCASE Indicates one of the following:

- API names
- Database manager data types
- Field names
- Key words
- SQL statements

| | |
|---|---|
| `Example text` | Indicates one of the following:<br>• Coding examples and code fragments<br>• Commands<br>• Examples of output, similar to what is displayed by the system<br>• Examples of specific data values<br>• Examples of system messages<br>• File and directory names<br>• Information that you are instructed to enter |
| **Bold** | Emphasizes a point. |

# Chapter 1. About the DB2 Software Developer's Kit

The DB2 Software Developer's Kit (DB2 SDK) provides the tools and environment you need to develop applications that access DB2 servers and application servers that implement the Distributed Relational Database Architecture (DRDA).

You can develop applications on a server or client that has the DB2 SDK installed. Your applications can also run on a server or client. To run your applications on a client, you must have the appropriate DB2 Client Application Enabler (DB2 CAE) installed. The DB2 CAE is installed from the DB2 Client Pack. See Chapter 2, “Setup” on page 21 for information about setting up your programming environment.

The DB2 SDKs for the UNIX platforms described in this book include the following:

- Precompilers for C, C++, COBOL, and Fortran (unless the language is not supported for that platform. Please see “Supported Software by Platform” on page 2 for details).
- Include files and code samples to develop applications that use embedded SQL.
- Programming libraries, include files, and code samples that use the DB2 Call Level Interface (DB2 CLI) to develop applications which are easily ported to ODBC and compiled with an ODBC SDK. The DB2 CAE contains an ODBC driver for DB2 that supports applications developed with Visigenic ODBC version 2.1.
- DB2 Java Database Connectivity (DB2 JDBC) support to develop Java applications and applets.
- DB2 embedded SQL for Java (SQLJ) support to develop Java embedded SQL applications and applets.
- On AIX, support to develop database applications that use the REXX language.
- Interactive SQL through the Command Line Processor (CLP) to prototype SQL statements or to perform ad hoc queries against the database.
- A documented API to enable other application development tools to implement precompiler support for DB2 directly within their products. For example, on AIX, IBM COBOL and PL/I use this interface. Information on the set of precompiler service APIs, and how to use them, is available from the anonymous FTP site, ftp://ftp.software.ibm.com. The PostScript file, called prepapi.psbin, is located in the directory /ps/products/db2/info. This file is in binary format. If you do not have access to this electronic forum and would like to get a copy of this document, you can call IBM Service as described in the Service Information Flyer.
- SQL92 and MVS Conformance Flagger: Identifies embedded SQL statements in applications that do not conform to the ISO/ANSI SQL92 Entry Level standard, or which are not supported by DB2 for MVS. If you migrate applications developed on a workstation to another platform, the Flagger saves you time by showing syntax incompatibilities. Refer to the *Command Reference* for information about the SQLFLAG option in the PRECOMPILE PROGRAM command.

© Copyright IBM Corp. 1993, 1998

## Supported Servers

You use the DB2 SDK to develop applications that will run on a specific platform. However, your applications can access remote databases on the following platforms:

- DB2 for AIX
- DB2 for HP-UX
- DB2 for OS/2
- DB2 for SCO OpenServer
- DB2 for SCO UnixWare 7
- DB2 for SINIX
- DB2 for Solaris
- DB2 for Windows NT
- Distributed Relational Database Architecture (DRDA)-compliant application servers, such as:
  - DB2 for OS/390
  - DB2 for OS/400
  - DB2 for VSE & VM (formerly SQL/DS for VM and VSE)
  - DRDA-compliant application servers from database vendors other than IBM.
- DB2 CLI applications that conform to ODBC can be ported to work under ODBC, provided an ODBC driver manager is available on the application platform.

**Note:** DB2 for SCO OpenServer and DB2 for SINIX are only available for DB2 Version 2.1.2.

## Supported Software by Platform

This section lists the compilers and related software supported by DB2 for the platforms described in this book. The compiler information assumes that you are using the DB2 precompiler for that platform, and not the precompiler support that may be built into one of the listed compilers. The exception is VisualAge for Basic; in this case, the precompiler is provided by VisualAge for Basic and not by DB2. For information on precompiler support built into any of the listed compilers, see that compiler's documentation.

Refer to the *Quick Beginnings for UNIX* book for information on the communication products supported by your platform's operating system.

**Notes:**

1. The **README** file for a supported platform may contain information on other compilers that are supported for that platform. The **README** file for a platform can be found in the directory in which the program files are installed.
2. **HP-UX**. If you are migrating DB2 from HP-UX Version 10 or earlier to HP-UX Version 11, your DB2 programs must be re-precompiled with DB2 on HP-UX Version 11 (if they include embedded SQL), and must be re-compiled. This includes all DB2 applications, stored procedures, user-defined functions and user exit programs. As well, DB2 programs that are compiled on HP-UX Version 11 may not run on HP-UX Version 10 or earlier. DB2 programs that are compiled and run on HP-UX Version 10 may connect remotely to HP-UX Version 11 servers.

3. **Micro Focus COBOL**. Any existing applications precompiled with DB2 Version 2.1.1 or earlier and compiled with Micro Focus COBOL should be re-precompiled with the current version of DB2, and then recompiled with Micro Focus COBOL. If these applications built with the earlier versions of the IBM precompiler are not re-precompiled, there is a possibility of database corruption if abnormal termination occurs.
4. **SCO OpenServer**. If you are migrating from DB2 for SCO OpenServer to DB2 for SCO UnixWare 7, any existing DB2 applications precompiled on SCO OpenServer should be re-precompiled on SCO UnixWare 7, and all DB2 applications compiled on SCO OpenServer should be recompiled on SCO UnixWare 7.
5. **VisualAge for Basic**. The product includes DB2 functions for embedded and static SQL, stored procedures, and User-Defined Functions (UDFs). It includes sample applications that connect to DB2 with embedded SQL, CLI and ODBC. The precompiler support for DB2 is provided by VisualAge for Basic. Refer to the VisualAge for Basic documentation for more information, especially for the versions of DB2 supported, and for details about the sample applications provided by the product.

## AIX

The DB2 SDK for AIX supports the following operating system:

| | |
|---|---|
| **AIX/6000** | Version 4.1.4 and later |

The DB2 SDK for AIX supports the following programming languages and compilers:

| | |
|---|---|
| **Basic** | IBM VisualAge for Basic Version 1 |
| **C** | IBM C for AIX Version 3.1 |
| **C++** | IBM C Set++ for AIX Version 3.1 |
| **COBOL** | IBM COBOL Set for AIX Version 1.1 |
| | Micro Focus COBOL Version 3.2.46 |
| | Micro Focus COBOL Version 4.0.20 (PRN 12.03 or higher) |
| | Micro Focus COBOL Version 4.1.10 (PRN 13.04 or higher) |
| **Fortran** | IBM XL Fortran for AIX Versions 4.1 and 5.1 |
| **Java** | Java Development Kit (JDK) Version 1.1 for AIX from IBM |
| **REXX** | IBM AIX REXX/6000 AISPO Product Number: 5764-057 |

## HP-UX

The DB2 SDK for HP-UX supports the following operating systems:

| | |
|---|---|
| **HP-UX** | Version 10.10 with Patch Levels: PHCO_6134, PHKL_5837, PHKL_6133, PHKL_6189, PHKL_6273, and PHSS_5956; |
| | Version 10.20 |
| | Version 11.0 |

The DB2 SDK for HP-UX supports the following programming languages and compilers:

| | |
|---|---|
| **C** | HP C/HP-UX Version A.10.13, with Patch Level PHSS_5743 |
| | HP C Compiler version A.11.00.00 (for HP-UX Version 11) |
| **C++** | HP-UX C++ Version A.10.03.60, with Patch Level PHSS_5883 |
| | HP C++ Version A.12.00 |
| **COBOL** | Micro Focus COBOL Version 3.2 |
| **Fortran** | HP Fortran/9000 Version 10.0 |
| | HP-UX f77 B.11.00.01 (for HP-UX Version 11) |
| **Java** | HP-UX Developer's Kit for Java Release 1.1 from Hewlett-Packard |

## SCO UnixWare 7

The DB2 SDK for SCO UnixWare 7 supports the following operating system:

**SCO UnixWare 7**

The DB2 SDK for SCO UnixWare 7 supports the following programming languages and compilers:

| | |
|---|---|
| **C** | Optimizing C Compilation System (CCS) Version 3.2 |
| **C++** | C++ Compilation System Version 3.0 |
| **COBOL** | Micro Focus COBOL Version 4.1 |
| **Java** | Java Development Kit (JDK) Version 1.1.3 for SCO UnixWare 7 from The Santa Cruz Operation |

## Silicon Graphics IRIX

The DB2 SDK for Silicon Graphics IRIX supports the following operating system:

| | |
|---|---|
| **Silicon Graphics IRIX** | Version 6.x (all levels of Version 6) |

The DB2 SDK for Silicon Graphics IRIX supports the following programming languages and compilers:

| | |
|---|---|
| **C** | MIPSpro C Compiler 7.2 |
| **C++** | MIPSpro C++ 7.2 |
| **Fortran** | MIPSpro Fortran-77 7.2 |
| **Java** | Java Development Environment 3.1 (Sun JDK 1.1.5) and Java Execution Environment 3.1 (Sun JRE 1.1.5) from Silicon Graphics, Inc. |

## Solaris

The DB2 SDK for Solaris supports the following operating system:

| | |
|---|---|
| **Solaris** | Version 2.5.1 and Version 2.6 |

The DB2 SDK for Solaris supports the following programming languages and compilers:

| | |
|---|---|
| **C** | SPARCompiler C Versions 3.0.1 and 4.2 |
| **C++** | SPARCompiler C++ Versions 4.0.1 and 4.2, and IBM C Set++ for Solaris Version 1.1.1 |
| **COBOL** | Micro Focus COBOL Version 3.2 |
| **Fortran** | SPARCompiler Fortran Version 3.0.1 |
| **Java** | Java Development Kit (JDK) Version 1.1.4 for Solaris from Sun Microsystems |

## Sample Programs

The DB2 SDK comes with sample programs. The file extensions for each supported language, and the directories where the programs can be found on the supported platforms, are given in Table 1 on page 6. In addition, the locations and extensions for other sample programs can be found in Table 2 on page 6.

The sample programs providing examples of embedded SQL (except for Java), and DB2 API calls are shown in Table 3 on page 10. Log Management User Exit programs are shown in Table 4 on page 15. Command Line Processor (CLP) programs provided by DB2 are shown in Table 5 on page 16.

Java JDBC sample programs are shown in Table 6 on page 16. Java SQLJ sample programs are shown in Table 7 on page 17. Object Linking and Embedding (OLE) sample programs are shown in Table 8 on page 17. The sample programs demonstrating DB2 CLI calls are shown in Table 9 on page 17.

You can use the sample programs to learn how to code your applications.

**Note:** Not all sample programs have been ported to all the supported programming languages.

*Table 1. Sample Program File Extensions and Locations*

| Language | | Embedded SQL Programs | Non-embedded SQL Programs |
|---|---|---|---|
| C | File Ext. | `.sqc` | `.c` |
| | Directory | `samples/c` | `samples/c`<br>`samples/cli` (CLI programs) |
| C++ | File Ext. | `.sqC` (UNIX)<br>`.sqx` (Windows & OS/2) | `.C` (UNIX)<br>`.cxx` (Windows & OS/2) |
| | Directory | `samples/cpp` | `samples/cpp` |
| COBOL | File Ext. | `.sqb` | `.cbl` |
| | Directory | `samples/cobol`<br>`samples/cobol_mf` | `samples/cobol`<br>`samples/cobol_mf` |
| Fortran | File Ext. | `.sqf` | `.f` (UNIX)<br>`.for` (OS/2) |
| | Directory | `samples/fortran` | `samples/fortran` |
| JAVA | File Ext. | `.sqlj` | `.java` |
| | Directory | `samples/java` | `samples/java` |
| REXX | File Ext. | `.cmd` | `.cmd` |
| | Directory | `samples/rexx` | `samples/rexx` |

*Table 2. Other Samples, their Extensions and Locations*

| Sample Group | | |
|---|---|---|
| CLP | File Ext. | `.db2` |
| | Directory | `samples/clp` |
| OLE | File Ext. | `.bas` (Microsoft Visual Basic)<br>`.CPP` (Microsoft Visual C++) |
| | Directory | `samples\ole\msvb` (Microsoft Visual Basic)<br>`samples\ole\msvc` (Microsoft Visual C++) |
| User Exit | File Ext. | `.cad` (Windows & OS/2)<br>`.cadsm` (UNIX)<br>`.cdisk` (UNIX)<br>`.ctape` (UNIX) |
| | Directory | `samples/c` |

**Note:**

**Embedded SQL Programs** require precompilation, except for REXX embedded SQL programs where the embedded SQL statements are interpreted when the program is run.

**Directory Delimiters** — On UNIX are /. On OS/2 and Windows platforms, are \. In the tables, the UNIX delimiters are used unless the directory is only available on Windows and/or OS/2.

**IBM COBOL samples** — Are only supplied on the OS/2, AIX, Windows NT and Windows 95 platforms in the `cobol` subdirectory.

**Micro Focus Cobol Samples** — Are supplied on all platforms except the Macintosh and Silicon Graphics IRIX. The 16-bit Micro Focus COBOL examples are supplied in the `cobol_16` subdirectory on OS/2, and the `cobol` subdirectory on Windows 3.1. For all other platforms, the Micro Focus COBOL samples are in the `cobol_mf` subdirectory.

**Fortran Samples** — Are only supplied on the AIX, HP-UX, Silicon Graphics IRIX, Solaris, and OS/2 platforms.

**Java Samples** — Are Java Database Connectivity (JDBC) applications, applets, stored procedures and UDFs, and embedded SQL (SQLJ) applications, applets, stored procedures and UDFs. Java samples are available on the AIX, HP-UX, SCO UnixWare 7, Silicon Graphics IRIX, Solaris, OS/2, Windows NT and Windows 95 platforms.

**REXX Samples** — Are only supplied on the AIX, OS/2, Windows NT and Windows 95 platforms.

**CLP Samples** — Are Command Line Processor scripts that execute SQL statements.

**OLE Samples** — Are for Object Linking and Embedding (OLE) in Microsoft Visual Basic and Microsoft Visual C++, supplied on the Windows NT and Windows 95 platforms only.

**User Exit samples** — Are Log Management User Exit programs used to archive and retrieve database log files. The files must be renamed with a `.c` extension and compiled as C language programs.

You can find the sample programs in the `samples` subdirectory of the directory where DB2 has been installed. There is a subdirectory for each supported language. The following examples show you how to locate the samples written in C or C++ on each supported platform.

- On UNIX platforms.

  You can find the C source code for embedded SQL and DB2 API programs in `sqllib/samples/c` under your database instance directory; the C source code for

DB2 CLI programs is in sqllib/samples/cli. For additional information about the sample programs in Table 3 on page 10 and Table 9 on page 17, refer to the README file in the appropriate samples subdirectory under your database manager instance. The README file will contain any additional samples that are not listed in this book.

- On OS/2, Windows NT, and Windows 95 platforms.

  You can find the C source code for embedded SQL and DB2 API programs in %DB2PATH%\samples\c under the DB2 install directory; the C source code for DB2 CLI programs is in %DB2PATH%\samples\cli. The variable %DB2PATH% determines where DB2 is installed. Depending on which drive DB2 is installed, %DB2PATH% will point to *drive*:\sqllib. For additional information about the sample programs in Table 3 on page 10 and Table 9 on page 17, refer to the README file in the appropriate %DB2PATH%\samples subdirectory. The README file will contain any additional samples that are not listed in this book.

- On Windows 3.1.

  You can find the C source code for embedded SQL and DB2 API programs in %DB2PATH%\samples\c; the C source code for DB2 CLI programs is in %DB2PATH%\samples\cli. The db2.ini file, which stores the DB2 settings, defines the value for %DB2PATH%, which by default points to *drive*:\sqllib\win. The value of %DB2PATH%, as referenced in the db2.ini file, is only recognized within the DB2 environment. For additional information about the sample programs in Table 3 on page 10 and Table 9 on page 17, refer to the README files in these subdirectories. The README files will contain any additional samples that are not listed in this book.

- On Macintosh.

  You can find the sample programs in the DB2:samples: folder. There are sub-folders for sample programs written in C and CLI. For additional information about the sample programs in Table 3 on page 10 and Table 9 on page 17, refer to the README file in the DB2:samples: folder. The README file will contain any additional samples that are not listed in this book.

The sample programs directory is typically read-only on most platforms. Before you alter or build the sample programs, copy them to your working directory. On the Macintosh, copy them to your working folder.

**Note:** The sample programs that are shipped with DB2 Universal Database have dependencies on the English version of the sample database and the associated table and column names. If the sample database has been translated into another national language on your version of DB2 Universal Database, you need to update the name of the sample database, and the names of the tables and the columns coded in the supplied sample programs, to the names used in the translated sample database. Otherwise, you will experience problems running the sample programs as shipped.

Currently, the `sample` database is translated into the following languages:

- Brazilian Portuguese
- French
- Korean
- Norwegian
- Simplified Chinese

In Table 3 on page 10, 'Yes', in the *Embedded SQL* column, indicates that the program contains embedded SQL. A blank indicates that the program does not contain embedded SQL, and thus no precompiling is required.

*Table 3 (Page 1 of 6). Sample Programs Showing Embedded SQL and APIs*

| Sample Program Name | Embedded SQL | Program Description |
|---|---|---|
| adhoc | Yes | Demonstrates dynamic SQL and the SQLDA structure to process SQL commands interactively. SQL commands are input by the user, and output corresponding to the SQL command is returned. |
| advsql | Yes | Demonstrates the use of advanced SQL expressions like CASE, CAST, and scalar full selects. |
| asynrlog | Yes | Demonstrates the use of the following API:<br>`ASYNCHRONOUS LOG READ` |
| backrest | | Demonstrates the use of the following APIs:<br>`BACKUP DATABASE`<br>`RESTORE DATABASE`<br>`ROLL FORWARD DATABASE` |
| blobfile | Yes | Demonstrates the manipulation of a Binary Large Object (BLOB), by reading a BLOB value from the sample database and placing it in a file, the contents of which can be displayed using an external viewer. |
| bindfile | Yes | Demonstrates the use of the BIND API to bind an embedded SQL application to a database. |
| calludf | Yes | Demonstrates the use of the library of User-Defined Functions (UDFs) created by `udf` for the `sample` database tables. |
| client | | Demonstrates the use of the following APIs:<br>`SET CLIENT`<br>`QUERY CLIENT` |
| columns | Yes | Demonstrates the use of a cursor that is processed using dynamic SQL. This program lists all the entries in the system table, SYSIBM.SYSTABLES, under a desired schema name. |
| cursor | Yes | Demonstrates the use of a cursor using static SQL. |
| d_dbconf | | Demonstrates the use of the following API:<br>`GET DATABASE CONFIGURATION DEFAULTS` |
| d_dbmcon | | Demonstrates the use of the following API:<br>`GET DATABASE MANAGER CONFIGURATION DEFAULTS` |
| db2mon | | Demonstrates how to use the Database System Monitor APIs, and how to process the output data buffer returned from the Snapshot API. |
| dbauth | Yes | Demonstrates the use of the following API:<br>`GET AUTHORIZATIONS` |
| dbcat | | Demonstrates the use of the following APIs:<br>`CATALOG DATABASE`<br>`CLOSE DATABASE DIRECTORY SCAN`<br>`GET NEXT DATABASE DIRECTORY ENTRY`<br>`OPEN DATABASE DIRECTORY SCAN`<br>`UNCATALOG DATABASE` |
| dbcmt | | Demonstrates the use of the following APIs:<br>`CHANGE DATABASE COMMENT` |

*Table 3 (Page 2 of 6). Sample Programs Showing Embedded SQL and APIs*

| Sample Program Name | Embedded SQL | Program Description |
|---|---|---|
| dbconf | | Demonstrates the use of the following APIs:<br>CREATE DATABASE<br>DROP DATABASE<br>GET DATABASE CONFIGURATION<br>RESET DATABASE CONFIGURATION<br>UPDATE DATABASE CONFIGURATION |
| dbinst | | Demonstrates the use of the following APIs:<br>ATTACH TO INSTANCE<br>DETACH FROM INSTANCE<br>GET INSTANCE |
| dbmconf | | Demonstrates the use of the following APIs:<br>GET DATABASE MANAGER CONFIGURATION<br>RESET DATABASE MANAGER CONFIGURATION<br>UPDATE DATABASE MANAGER CONFIGURATION |
| dbsnap | | Demonstrates the use of the following API:<br>DATABASE SYSTEM MONITOR SNAPSHOT |
| dbstart | | Demonstrates the use of the following API:<br>START DATABASE MANAGER |
| dbstat | Yes | Demonstrates the use of the following APIs:<br>REORGANIZE TABLE<br>RUN STATISTICS |
| dbstop | | Demonstrates the use of the following APIs:<br>FORCE USERS<br>STOP DATABASE MANAGER |
| db_udcs | | Demonstrates the use of the following APIs in order to simulate the collating behaviour of a DB2 for OS/390 CCSID 500 (EBCDIC International) collating sequence:<br>CREATE DATABASE<br>DROP DATABASE |
| dcscat | | Demonstrates the use of the following APIs:<br>ADD DCS DIRECTORY ENTRY<br>CLOSE DCS DIRECTORY SCAN<br>GET DCS DIRECTORY ENTRY FOR DATABASE<br>GET DCS DIRECTORY ENTRIES<br>OPEN DCS DIRECTORY SCAN<br>UNCATALOG DCS DIRECTORY ENTRY |
| delet | Yes | Demonstrates static SQL to delete items from a database. |
| dmscont | | Demonstrates the use of the following APIs in order to create a database with more than one database managed storage (DMS) container:<br>CREATE DATABASE<br>DROP DATABASE |
| dynamic | Yes | Demonstrates the use of a cursor using dynamic SQL. |

*Table 3 (Page 3 of 6). Sample Programs Showing Embedded SQL and APIs*

| Sample Program Name | Embedded SQL | Program Description |
|---|---|---|
| ebcdicdb | | Demonstrates the use of the following APIs in order to simulate the collating behaviour of a DB2 for OS/390 CCSID 037 (EBCDIC US English) collating sequence:<br>CREATE DATABASE<br>DROP DATABASE |
| expsamp | Yes | Demonstrates the use of the following APIs:<br>EXPORT<br>IMPORT<br>in conjunction with a DRDA database. |
| fillcli | Yes | Demonstrates the client-side of a stored procedure that uses the SQLDA to pass information specifying which table the stored procedure populates with random data. |
| fillsrv | Yes | Demonstrates the server-side of a stored procedure example that uses the SQLDA to receive information from the client specifying the table that the stored procedure populates with random data. |
| impexp | Yes | Demonstrates the use of the following APIs:<br>EXPORT<br>IMPORT |
| inpcli | Yes | Demonstrates stored procedures using either the SQLDA structure or host variables. This is the client program of a client/server example. (The server program is called inpsrv.) The program fills the SQLDA with information, and passes it to the server program for further processing. The SQLCA status is returned to the client program. This program shows the invocation of stored procedures using an embedded SQL CALL statement. |
| inpsrv | Yes | Demonstrates stored procedures using the SQLDA structure. This is the server program of a client/server example. (The client program is called inpcli.) The program creates a table (PRESIDENTS) in the sample database with the information received in the SQLDA. The server program does all the database processing and returns the SQLCA status to the client program. |
| joinsql | Yes | An example using advanced SQL join expressions. |
| largevol | Yes | Demonstrates parallel query processing in a partitioned environment, and the use of an NFS file system to automate the merging of the result sets. |
| lobeval | Yes | Demonstrates the use of LOB locators and deferring the evaluation of the actual LOB data. |
| lobfile | Yes | Demonstrates the use of LOB file handles. |
| lobloc | Yes | Demonstrates the use of LOB locators. |
| lobval | Yes | Demonstrates the use of LOBs. |

*Table 3 (Page 4 of 6). Sample Programs Showing Embedded SQL and APIs*

| Sample Program Name | Embedded SQL | Program Description |
|---|---|---|
| makeapi | Yes | Demonstrates the use of the following APIs:<br>BIND<br>PRECOMPILE PROGRAM<br>START DATABASE MANAGER<br>STOP DATABASE MANAGER |
| migrate | | Demonstrates the use of the following API:<br>MIGRATE DATABASE |
| monreset | | Demonstrates the use of the following API:<br>RESET DATABASE SYSTEM MONITOR DATA AREAS |
| monsz | | Demonstrates the use of the following APIs:<br>ESTIMATE DATABASE SYSTEM MONITOR BUFFER SIZE<br>DATABASE SYSTEM MONITOR SNAPSHOT |
| nodecat | | Demonstrates the use of the following APIs:<br>CATALOG NODE<br>CLOSE NODE DIRECTORY SCAN<br>GET NEXT NODE DIRECTORY ENTRY<br>OPEN NODE DIRECTORY SCAN<br>UNCATALOG NODE |
| openftch | Yes | Demonstrates fetching, updating, and deleting of rows using static SQL. |
| outcli | Yes | Demonstrates stored procedures using the SQLDA structure. This is the client program of a client/server example. (The server program is called outsrv.) This program allocates and initializes a one variable SQLDA, and passes it to the server program for further processing. The filled SQLDA is returned to the client program along with the SQLCA status. This program shows the invocation of stored procedures using an embedded SQL CALL statement. |
| outsrv | Yes | Demonstrates stored procedures using the SQLDA structure. This is the server program of a client/server example. (The client program is called outcli.) The program fills the SQLDA with the median SALARY of the employees in the STAFF table of the sample database. The server program does all the database processing (finding the median). The server program returns the filled SQLDA and the SQLCA status to the client program. |
| qload | Yes | Demonstrates the use of the following API:<br>LOAD QUERY |
| rebind | Yes | Demonstrates the use of the following API:<br>REBIND PACKAGE |
| rechist | | Demonstrates the use of the following APIs:<br>CLOSE RECOVERY HISTORY FILE SCAN<br>GET NEXT RECOVERY HISTORY FILE ENTRY<br>OPEN RECOVERY HISTORY FILE SCAN<br>PRUNE RECOVERY HISTORY FILE ENTRY<br>UPDATE RECOVERY HISTORY FILE ENTRY |

*Table 3 (Page 5 of 6). Sample Programs Showing Embedded SQL and APIs*

| Sample Program Name | Embedded SQL | Program Description |
|---|---|---|
| recursql | Yes | Demonstrates the use of advanced SQL recursive queries. |
| regder | | Demonstrates the use of the following APIs:<br>REGISTER<br>DEREGISTER |
| restart | | Demonstrates the use of the following API:<br>RESTART DATABASE |
| sampudf | Yes | Demonstrates the use of User-Defined Types (UDTs) and User-Defined Functions (UDFs). The UDFs declared in this program are all sourced UDFs. |
| setact | | Demonstrates the use of the following API:<br>SET ACCOUNTING STRING |
| setrundg | | Demonstrates the use of the following API:<br>SET RUNTIME DEGREE |
| static | Yes | Uses static SQL to retrieve information. |
| sws | | Demonstrates the use of the following API:<br>DATABASE MONITOR SWITCH |
| tabscont | | Demonstrates the use of the following APIs:<br>TABLESPACE CONTAINER QUERY<br>OPEN TABLESPACE CONTAINER QUERY<br>FETCH TABLESPACE CONTAINER QUERY<br>CLOSE TABLESPACE CONTAINER QUERY<br>SET TABLESPACE CONTAINER QUERY |
| tabspace | | Demonstrates the use of the following APIs:<br>TABLESPACE QUERY<br>SINGLE TABLESPACE QUERY<br>OPEN TABLESPACE QUERY<br>FETCH TABLESPACE QUERY<br>GET TABLESPACE STATISTICS<br>CLOSE TABLESPACE QUERY |
| tabsql | Yes | Demonstrates the use of advanced SQL table expressions. |
| tblcli | | Demonstrates a call to a table function (client-side) to display weather information for a number of cities. |
| tblsrv | | Demonstrates a table function (server-side) that processes weather information for a number of cities. |
| thdsrver | Yes | Demonstrates the use of posix threads APIs for thread creation and management. The program maintains a pool of contexts. A generate_work function is executed from main, and creates dynamic SQL statements that are executed by worker threads. When a context becomes available, a thread is created and dispatched to do the specified work. The work generated consists of statements to delete entries from either the STAFF or EMPLOYEE tables of the sample database. This program is only available on UNIX platforms. |

*Table 3 (Page 6 of 6). Sample Programs Showing Embedded SQL and APIs*

| Sample Program Name | Embedded SQL | Program Description |
|---|---|---|
| tload | Yes | Demonstrates the use of the following APIs:<br>EXPORT<br>QUIESCE TABLESPACE FOR TABLES<br>LOAD |
| trigsql | Yes | An example using advanced SQL triggers and constraints. |
| udf | Yes | Creates a library of User-Defined Functions (UDFs) made specifically for the sample database tables, but can be used with tables of compatible column types. |
| updat | Yes | Uses static SQL to update a database. |
| util | | Demonstrates the use of the following APIs:<br>GET ERROR MESSAGE<br>GET SQLSTATE MESSAGE<br>INSTALL SIGNAL HANDLER<br>INTERRUPT<br>This program also contains code to output information from an SQLDA. |
| varinp | Yes | An example of variable input to Embedded Dynamic SQL statement calls using parameter markers. |

*Table 4 (Page 1 of 2). Log Management User Exit Sample Programs.*

| Sample File Name | File Description |
|---|---|
| db2uext2.cadsm | This is a sample User Exit utilizing ADSTAR DSM ( ADSM ) APIs to archive and retrieve database log files. The sample provides an audit trail of calls (stored in a separate file for each option) including a timestamp and parameters received. It also provides an error trail of calls in error including a timestamp and an error isolation string for problem determination. These options can be disabled. The file must be renamed db2uext2.c and compiled as a C program. Available on supported UNIX platforms only. The Windows NT version is db2uext2.cad. The OS/2 version is db2uexit.cad. |
| db2uext2.cad | This is the Windows NT version of db2uext2.cadsm. The file must be renamed db2uext2.c and compiled as a C program. |
| db2uexit.cad | This is the OS/2 version of db2uext2.cadsm. The file must be renamed db2uexit.c and compiled as a C program. |
| db2uext2.cdisk | This is a sample User Exit utilizing the AIX system copy command to archive and retrieve database log files. The sample provides an audit trail of calls (stored in a separate file for each option) including a timestamp and parameters received. It also provides an error trail of calls in error including a timestamp and an error isolation string for problem determination. These options can be disabled. The file must be renamed db2uext2.c and compiled as a C program. Available on supported UNIX platforms only. |

*Table 4 (Page 2 of 2). Log Management User Exit Sample Programs.*

| Sample File Name | File Description |
|---|---|
| db2uext2.ctape | This is a sample User Exit utilizing the SUN system tape commands to archive and retrieve database log files. All limitations of the SUN system tape commands are limitations of this user exit. The sample provides an audit trail of calls (stored in a separate file for each option) including a timestamp and parameters received. It also provides an error trail of calls in error including a timestamp and an error isolation string for problem determination. These options can be disabled. The file must be renamed db2uext2.c and compiled as a C program. Available on supported UNIX platforms only. |

*Table 5. Command Line Processor (CLP) Sample Programs.*

| Sample File Name | File Description |
|---|---|
| const | Creates a table with a CHECK CONSTRAINT clause. |
| cte | Demonstrates a common table expression. The equivalent sample program demonstrating this advanced SQL statement is tabsql. |
| flt | Demonstrates a recursive query. The equivalent sample program demonstrating this advanced SQL statement is recursql. |
| join | Demonstrates an outer join of tables. The equivalent sample program demonstrating this advanced SQL statement is joinsql. |
| stock | Demonstrates the use of triggers. The equivalent sample program demonstrating this advanced SQL statement is trigsql. |
| testdata | Uses DB2 built-in functions such as RAND() and TRANSLATE() to populate a table with randomly generated test data. |
| thaisort | This script is particularly for Thai users. Thai sorting is by phonetic order requiring pre-sorting/swapping of the leading vowel and its consonant, as well as post-sorting in order to view the data in the correct sort order. The file implements Thai sorting by creating UDF functions presort and postsort, and creating a table; then it calls the functions against the table to sort the table data. To run this program, you first have to build the user-defined function program, udf, from the C source file, udf.c. |

*Table 6. Java Database Connectivity (JDBC) Sample Programs*

| Sample Program Name | Program Description |
|---|---|
| DB2Appl.java | A JDBC application that queries the sample database using the invoking user's privileges. |
| DB2Applt.java | A JDBC applet that queries the sample database using a user and server specified as applet parameters. |
| DB2Applt.html | An HTML file that embeds the applet sample program, DB2Applt. It needs to be customized with server and user information. |
| DB2Stp.java | A Java stored procedure that updates the EMPLOYEE table on the server, and returns new salary and payroll information to the client. |
| DB2Udf.java | A Java UDF that demonstrates several tasks, including integer division, manipulation of Character Large OBjects (CLOBs), and the use of Java instance variables. |

*Table 7. Embedded SQL for Java (SQLJ) Sample Programs*

| Sample Program Name | Program Description |
|---|---|
| App.sqlj | An SQLJ application that uses static SQL to retrieve and update data from the EMPLOYEE table of the sample database. |
| Applt.sqlj | An SQLJ applet that queries the sample database using a user and server specified as applet parameters. |
| Applt.html | An HTML file that embeds the applet sample program, Applt. It needs to be customized with server and user information. |
| Stp.sqlj | An embedded SQL (SQLJ) stored procedure that updates the EMPLOYEE table on the server, and returns new salary and payroll information to the client. |
| CatUdf.sqlj | An SQLJ program that demonstrates cataloging Java UDFs and creating a sample table, udftest, for testing them. |
| Udf.sqlj | An SQLJ program that demonstrates calling Java UDFs against the sample table, udftest. |
| DropUdf.sqlj | An SQLJ program that demonstrates dropping Java UDFs and the sample table, udftest. |

*Table 8. Object Linking and Embedding (OLE) Sample Programs*

| Sample Program Name | Program Description |
|---|---|
| sales | Demonstrates rollup queries on a Microsoft Excel sales spreadsheet (implemented in Visual Basic). |
| names | Queries a Lotus Notes address book (implemented in Visual Basic). |
| inbox | Queries Microsoft Exchange inbox e-mail messages through OLE/Messaging (implemented in Visual Basic). |
| invoice | An OLE automation user-defined function that sends Microsoft Word invoice documents as e-mail attachments (implemented in Visual Basic). |
| ccounter | A counter OLE automation user-defined function (implemented in Visual C++). |
| salarysrv | An OLE automation stored procedure that calculates the median salary of the STAFF table of the sample database (implemented in Visual Basic). |
| salaryclt | A client program that invokes the median salary OLE automation stored procedure salarysrv (implemented in Visual Basic and in Visual C++). |

*Table 9 (Page 1 of 4). Sample CLI Programs in DB2 Universal Database*

| Sample Program Name | Program Description |
|---|---|
| Utility files used by most CLI samples | |
| samputil.c | Utility functions used by most samples |
| samputil.h | Header file for samputil.c, included by most samples |
| General CLI Samples | |
| adhoc.c | Interactive SQL with formatted output (was typical.c) |

*Table 9 (Page 2 of 4). Sample CLI Programs in DB2 Universal Database*

| Sample Program Name | Program Description |
|---|---|
| async.c ** | Run a function asynchronously (based on fetch.c) |
| basiccon.c | Basic connection |
| browser.c | List columns, foreign keys, index columns or stats for a table |
| colpriv.c | List column Privileges |
| columns.c | List all columns for table search string |
| compnd.c | Compound SQL example |
| datasour.c | List all available data sources |
| descrptr.c ** | Example of descriptor usage |
| drivrcon.c | Rewrite of basiccon.c using SQLDriverConnect |
| duowcon.c | Multiple DUOW Connect type 2, syncpoint 1 (one phase commit) |
| embedded.c | Show equivalent DB2 CLI calls, for embedded SQL (in comments) |
| fetch.c | Simple example of a fetch sequence |
| getattrs.c | List some common environment, connection and statement options/attributes |
| getcurs.c | Show use of SQLGetCursor, and positioned update |
| getdata.c | Rewrite of fetch.c using SQLGetData instead of SQLBindCol |
| getfuncs.c | List all supported functions |
| getfuncs.h | Header file for getfuncs.c |
| getinfo.c | Use SQLGetInfo to get driver version and other information |
| getsqlca.c | Rewrite of adhoc.c to use prepare/execute and show cost estimate |
| lookres.c | Extract string from resume clob using locators |
| mixed.sqc | CLI sample with functions written using embedded SQL (Note: This file must be precompiled ) |
| multicon.c | Multiple connections |
| native.c | Simple example of calling SQLNativeSql, and SQLNumParams |
| prepare.c | Rewrite of fetch.c, using prepare/execute instead of execdirect |
| proccols.c | List procedure parameters using SQLProcedureColumns |
| procs.c | List procedures using SQLProcedures |
| sfetch.c ** | Scrollable cursor example (based on xfetch.c) |
| setcolat.c | Set column attributes (using SQLSetColAttributes) |
| setcurs.c | Rewrite of getcurs.c using SQLSetCurs for positioned update |
| seteattr.c | Set environment attribute (SQL_ATTR_OUTPUT_NTS) |
| tables.c | List all tables |
| typeinfo.c | Display type information for all types for current data source |
| xfetch.c | Extended Fetch, multiple rows per fetch |
| BLOB Samples | |

*Table 9 (Page 3 of 4). Sample CLI Programs in DB2 Universal Database*

| Sample Program Name | Program Description |
|---|---|
| picin.c | Loads graphic BLOBS into the emp_photo table directly from a file using SQLBindParamToFile |
| picin2.c | Loads graphic BLOBS into the emp_photo table using SQLPutData |
| showpic.c | Extracts BLOB picture to file (using SQLBindColToFile), then displays the graphic. |
| showpic2.c | Extracts BLOB picture to file using piecewise output, then displays the graphic. |
| Stored Procedure Samples | |
| clicall.c | Defines a CLI function which is used in the embedded SQL sample mrspcli3.sqc |
| inpcli.c | Call embedded input stored procedure samples/c/inpsrv |
| inpcli2.c | Call CLI input stored procedure inpsrv2 |
| inpsrv2.c | CLI input stored procedure (rewrite of embedded sample inpsrv.sqc) |
| mrspcli.c | CLI program that calls mrspsrv.c |
| mrspcli2.c | CLI program that calls mrspsrv2.sqc |
| mrspcli3.sqc | An embedded SQL program that calls mrspsrv2.sqc using clicall.c |
| mrspsrv.c | Stored procedure that returns a multi-row result set |
| mrspsrv2.sqc | An embedded SQL stored procedure that returns a multi-row result set |
| outcli.c | Call embedded output stored procedure samples/c/inpsrv |
| outcli2.c | Call CLI output stored procedure inpsrv2 |
| outsrv2.c | CLI output stored procedure (rewrite of embedded sample inpsrv.sqc) |
| Samples using ORDER tables created by create.c (Run in the following order) | |
| create.c | Creates all tables for the order scenario |
| custin.c | Inserts customers into the customer table (array insert) |
| prodin.c | Inserts products into the products table (array insert) |
| prodpart.c | Inserts parts into the prod_parts table (array insert) |
| ordin.c | Inserts orders into the ord_line, ord_cust tables (array insert) |
| ordrep.c | Generates order report using multiple result sets |
| partrep.c | Generates exploding parts report (recursive SQL Query) |
| order.c | UDF library code (declares a 'price' UDF) |
| order.exp | Used to build order libary |
| Version 2 Samples unchanged | |
| v2sutil.c | samputil.c using old v2 functions |
| v2sutil.h | samputil.h using old v2 functions |
| v2fetch.c | fetch.c using old v2 functions |
| v2xfetch.c | xfetch.c using old v2 functions |

*Table 9 (Page 4 of 4). Sample CLI Programs in DB2 Universal Database*

| Sample Program Name | Program Description |
|---|---|

**Note:** Samples marked with a ** are new for this release.

Other files in the `samples/cli` directory include:

- README - Lists all example files.
- makefile - Makefile for all files

# Chapter 2. Setup

Before you can use the DB2 SDK to develop applications, you need to set up your programming environment for DB2. It is recommended that you ensure that your existing environment is correctly set up by first building a non-DB2 application. Then, if you encounter any problems, please see the documentation that comes with your compiler or interpreter.

To set up your programming environment for DB2, the following must be installed and working:

- The database manager on the server with the database instance for your environment. Refer to Appendix A, “About Database Manager Instances” on page 165 if you need information about database instances.
- The DB2 SDK on the client or server workstation on which you are going to develop applications.
- The connection to the remote server, if you are developing on a client workstation connected to a remote server.
- A compiler or interpreter for one of the supported programming languages on the UNIX platform you are using, listed in “Supported Software by Platform” on page 2. Consult the documentation for the compiler or interpreter you are using.

For more detailed information on installation and setup, refer to the *Quick Beginnings for UNIX* book.

When the above are installed and working, you can set up your environment by following the steps in the section “Setting Your Environment”

After you set up your environment, you may want to set up the sample database, which is used by the examples in this book. To create the database, see “Creating, Cataloging, and Binding the Sample Database” on page 22

## Setting Your Environment

You need to set environment variables so you can access the database instance that was created when the database manager was installed. The *Quick Beginnings for UNIX* book provides general information about setting environment variables. This section provides specific instructions on setting environment variables to access a database instance.

Each database manager instance has two files, `db2profile` and `db2cshrc`, which contain scripts to set the environment variable for that instance. Depending on the shell you are using, run the script by entering:

**For Korn shell:** `. $HOME/sqllib/db2profile`

**For C shell:** `source $HOME/sqllib/db2cshrc`

© Copyright IBM Corp. 1993, 1998

where $HOME is the home directory of the instance owner.

For your convenience, you may want to include this command in your .profile file, so that it runs automatically when you log on.

## Creating, Cataloging, and Binding the Sample Database

To use the examples in this book, you need to create the sample database *on a server workstation*. Refer to the *SQL Reference* for a listing of the contents of the sample database.

If you will be accessing the sample database on the server from a remote client, you need to catalog the sample database on the client workstation.

Also, if you will be accessing the sample database on the server from a remote client that is running a different version of DB2 or running on a different operating system, you need to bind the database utilities, including the DB2 CLI, to the sample database.

### Creating

To create the sample database, you must have Administrator authority. If you need more information about Administrator authority, refer to the *Quick Beginnings for UNIX* book.

To create the database , do the following on the server:

1. Ensure the location of db2sampl (the program that creates the sample database) is in your path. The db2profile or db2cshrc file will put db2sampl in your path, so it will be there unless you change it.
   - On UNIX platforms, db2sampl is located in:

     *$HOME*/sqllib/misc

     where *$HOME* is the home directory of the DB2 instance owner.
   - On OS/2, Windows NT and Windows 95, db2sampl is located in:

     *%DB2PATH%* \bin

     where *%DB2PATH%* is where DB2 is installed.
2. Set the DB2INSTANCE environment variable to the name of the instance where you want to create the sample database.
   - On UNIX platforms:

     you can do this for the Korn shell by entering:

     ```
     DB2INSTANCE=instance_name
     export DB2INSTANCE
     ```

     and for the C shell by entering:

     ```
     setenv DB2INSTANCE instance_name
     ```

     where *instance_name* is the name of the database instance.

- On OS/2, Windows NT and Windows 95, enter:

  ```
  set DB2instance=instance_name
  ```

3. Create the `sample` database by entering `db2sampl` followed by where you want to create the sample database. On UNIX platforms, this is a path, and would be entered as:

   `db2sampl` *path*

   On OS/2, Windows NT and Windows 95, this is a `drive`, and would be entered as:

   `db2sampl` *drive*

   If you do not specify the path or drive, the installation program installs the sample tables in the default path or drive specified by the DFTDBPATH parameter in the database manager configuration file. If you need information about the configuration file, refer to the *Administration Guide*.

   The authentication type for the database is the same as the instance in which it is created. If you need more information about specifying authentication when creating a database instance, refer to the *Quick Beginnings for UNIX* book.

**Creating on DRDA-Compliant Application Servers**

If you want to run the sample programs against a DRDA-compliant application server, such as DB2 for OS/390, you need to create a database that contains the sample tables described in the *SQL Reference*. You may want to refer to the sample program, `expsamp`, which uses the STAFF and ORG tables to demonstrate how APIs are used to import and export tables and table data to and from a DRDA database.

To create the database:

1. Create the `sample` database in a DB2 server instance using `db2sampl`.
2. Connect to the `sample` database.
3. Export the sample tables to a file.
4. Connect to the DRDA-compliant database.
5. Create the sample tables.
6. Import the sample tables.

If you need information about exporting and importing files, refer to the *Command Reference* and the *API Reference*. If you need information about connecting to a database and creating tables, refer to the *SQL Reference*.

## Cataloging

If you will be accessing the `sample` database on the server from a remote client, you need to catalog the `sample` database on the client workstation.

You do not need to catalog the `sample` database on the server workstation because it was cataloged when you created it.

Cataloging updates the database directory on the client workstation with the name of the database that the client application wants to access. When processing client requests, the database manager uses the cataloged name to find and connect to the database.

The *Quick Beginnings for UNIX* book provides general information on cataloging databases. This section provides specific instructions on cataloging the `sample` database.

To catalog the sample database from the remote client workstation, enter:

```
db2 catalog database sample as sample at node nodename
```

where *nodename* is the name of the server node.

The *Quick Beginnings for UNIX* book explains how to catalog nodes as part of setting up communication protocols. You must also catalog the remote node before you can connect to the database.

## Binding

If you will be accessing the `sample` database on the server from a remote client that is running a different version of DB2 or running on a different operating system, you need to bind the database utilities, including the DB2 CLI, to the `sample` database.

To bind the database utilites, do the following on the client workstation:

1. Connect to the `sample` database by entering:

   ```
   db2 connect to sample
   ```

2. Bind the utilities to the database by entering:

   ```
   db2 bind BNDPATH/@db2ubind.lst blocking all sqlerror continue messages bind.msg
   ```

   ```
   db2 bind BNDPATH/@db2cli.lst blocking all sqlerror continue messages cli.msg
   ```

   where *BNDPATH* is the path where the bind files are located, such as `$HOME/sqllib/bnd`, where `$HOME` is the home directory of the DB2 instance owner.

   **Note:** If you created the `sample` database on a DRDA-compliant application server, specify one of the following `.lst` files instead of `db2ubind.lst` :

   | | |
   |---|---|
   | **ddcsmvs.lst** | for DB2 for OS/390 |
   | **ddcsvse.lst** | for DB2 for VSE and VM |
   | **ddcs400.lst** | for DB2 for OS/400 |

3. Verify that the bind was successful by checking the bind message files `bind.msg` and `cli.msg`.

The *Quick Beginnings for UNIX* book provides general information about binding the database utilities.

## Where to Go Next

Once your environment is set up, you are ready to build your DB2 applications. The following chapters discuss the sample programs, and show you how to compile, link, and run them.

If you are developing embedded SQL applications, see Chapter 3, “Introduction to Embedded SQL Applications” on page 27, and then the embedded SQL chapter for the platform you are using. If you are developing CLI applications, see Chapter 9, “Building DB2 Call Level Interface (CLI) Applications” on page 137. If you are developing Java applications, see Chapter 10, “Building Java Applications and Applets” on page 145. If you are developing DB2 API applications see the appropriate chapter or chapters given above for the programming interface you will be using.

For further information, refer to the following books. To develop applications using embedded SQL, see the *Embedded SQL Programming Guide*. For applications using DB2 CLI or ODBC see the *CLI Guide and Reference*. For DB2 API applications, see the *API Reference*. For important new information since the release of DB2 Universal Database Version 5, see the *What's New* book.

# Chapter 3. Introduction to Embedded SQL Applications

Each DB2 SDK includes sample programs that embed SQL statements. Chapters 4 through 8 explain how to build the sample programs for the supported compilers using script files supplied with the DB2 SDK for that platform. You can also use the makefiles that are supplied. Both the makefiles and the script files show you the compiler options you can use. These options are defined for each platform's supported compilers in the appropriate chapter. You might need to modify the options for your environment.

When you run a script file to build a sample program containing embedded SQL, the script file executes the following steps:

- Connects to a database.
- Precompiles your source file.
- Binds your bind file to the database.
- Disconnects from the database.
- Compiles and links your source file.

For User-Defined Functions (UDFs), you do not need to connect to a database or precompile and bind the program.

The following are the sample programs used to demonstrate the steps for building and running your applications using the supported programming languages. The steps you follow may vary, depending on your environment:

| | |
|---|---|
| `updat` | uses static SQL to update a database. |
| `outcli` | is the client program of a client/server example; the server program is `outsrv`. |
| `outsrv` | is the server program of a client/server example; the client program is `outcli`. |
| `udf` | creates a library of user-defined functions (UDFs); `calludf` uses these functions. |
| `calludf` | uses the functions created by `udf`. |
| `thdsrver` | demonstrates thread creation and management; only available in the C language. |

For a more detailed description of these sample programs, see Table 3 on page 10.

The source files for these sample programs, where supported, are in the appropriate programming language subdirectory of `sqllib/samples`:

| | |
|---|---|
| **C** | `sqllib/samples/c` |
| **C++** | `sqllib/samples/cpp` |
| **IBM COBOL** | `sqllib/samples/cobol` |
| **Micro Focus COBOL** | `sqllib/samples/cobol_mf` |

© Copyright IBM Corp. 1993, 1998

**Fortran** `sqllib/samples/fortran`

After you build the sample programs, they can be used as templates to create your own applications. This can be done by modifying the sample programs with your own SQL statements. You can build your programs using either the makefile or the script files provided.

"Sample Programs" on page 5 lists all of the sample programs. The *Embedded SQL Programming Guide* explains how the samples containing embedded SQL work.

**Note:** It is recommended that, before you alter or build the sample programs, you copy them from `sqllib/samples` to your own working directory.

## Using the Micro Focus COBOL Compiler

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the Micro Focus COBOL compiler, keep the following points in mind:

- When you precompile your application using the command line processor command `db2 prep`, use the `target mfcob` option (the default).
- In order to use the built-in precompiler front-end, runtime interpreter or Animator debugger, add the DB2 Generic API entry points to the Micro Focus runtime module `rts32` by executing the MKRTS command provided by Micro Focus, as follows:

  1. Log in as root.
  2. Execute MKRTS with the arguments supplied in the following directories:

     For AIX:

     `/usr/lpp/db2_05_00/lib/db2mkrts.args`

     For other UNIX platforms:

     `/opt/IBMdb2/V5.0/lib/db2mkrts.args`

- You must include the DB2 COBOL COPY file directory in the Micro Focus COBOL environment variable COBCPY. The directory specifies the location of COPY files. The DB2 COPY files for Micro Focus COBOL reside in `sqllib/include/cobol_mf` under the database instance directory.

  To include the directory on AIX, enter:

  ```
  export COBCPY=$COBCPY:/usr/lpp/db2_05_00/include/cobol_mf
  ```

  To include the directory on other UNIX platforms enter:

  ```
  export COBCPY=$COBCPY:/opt/IBMdb2/V5.0/include/cobol_mf
  ```

  **Note:** You might want to set COBCPY in the `.profile` file.

## C++ Considerations for UDFs and Stored Procedures

**Note:** DB2 for Solaris does not support UDFs or stored procedures written in C++ and compiled with the IBM C Set++ compiler.

Because function names can be 'overloaded' in C++, that is, two functions with the same name can coexist if they have different arguments, as in `int foo( int i )` and `int foo( char c )`, C++ compilers 'type-decorate' or 'mangle' function names by default. This means that argument type names are appended to their function names to resolve them, as in `foo__Fi` and `foo__Fc` for the two earlier examples.

The type-decorated function name can be determined from the `.o` file using the `nm` command:

```
nm myprog.o
```

The command produces some output which includes a line similar to the following:

```
myprogen__FPlT1PsT3PcN35|     3792|unamex|          | ...
```

When registering such a UDF with CREATE FUNCTION, the EXTERNAL NAME clause must specify the mangled function name obtained from `nm` (not including the | character):

```
CREATE FUNCTION myprogo(...) RETURNS...
       ...
       EXTERNAL NAME '/whatever/path/myprog!myprogen__FPlT1PsT3PcN35'
       ...
```

Likewise, when calling a stored procedure, the function name also specifies the mangled function name:

```
CALL '/whatever/path/myprog!myprogen__FPlT1PsT3PcN35' ( ... )
```

If your stored procedure or UDF library does not contain overloaded C++ function names, you have the option of using `extern "C"` to force the compiler to not type-decorate function names. (Note that you can always overload the SQL function names given to UDFs, since DB2 resolves what library function to call based on the name and the parameters it takes.)

```
#include <string.h>
#include <stdlib.h>
#include "sqludf.h"

/*------------------------------------------------------------------------*/
/* function fold: output = input string is folded at point indicated      */
/*                         by the second argument.                        */
/*         inputs: CLOB,                 input string                     */
/*                 LONG                  position to fold on              */
/*         output: CLOB                  folded string                    */
/*------------------------------------------------------------------------*/
extern "C" void fold(
      SQLUDF_CLOB       *in1,                   /* input CLOB to fold       */
   ...
   ...
}
/* end of UDF: fold */

/*------------------------------------------------------------------------*/
/* function find_vowel:                                                   */
/*             returns the position of the first vowel.                   */
/*             returns error if no vowel.                                 */
/*             defined as NOT NULL CALL                                   */
/*         inputs: VARCHAR(500)                                           */
/*         output: INTEGER                                                */
/*------------------------------------------------------------------------*/
extern "C" void findvwl(
      SQLUDF_VARCHAR    *in,                    /* input smallint           */
   ...
   ...
}
/* end of UDF: findvwl */
```

In this example, the UDFs `fold` and `findvwl` are not type-decorated by the compiler, and should be registered in the CREATE FUNCTION statement using their plain names. Similarly, if a C++ stored procedure is coded with `extern "C"`, its undecorated function name would be used in the CALL statement.

## Error Checking

The sample programs use the following error-checking utilities:

| | |
|---|---|
| `util.c` | For C sample programs |
| `util.f` | For Fortran sample programs |
| `checkerr.cbl` | For COBOL sample programs |

The script files you use to build the sample programs create the appropriate object file:

| | |
|---|---|
| `util.o` | For C sample programs |
| `util.o` | For Fortran sample programs |
| `checkerr.o` | For COBOL sample programs |

# Chapter 4. Building AIX Embedded SQL Applications

This chapter provides detailed information for building embedded SQL applications on AIX. In the script files, commands that begin with `db2` are Command Line Processor (CLP) commands. Refer to the *Command Reference* if you need more information about CLP commands.

### General Points for Building and Running DB2 Programs

1. You must build and run DB2 applications from a window where your environment variables are set. You can do this by running `db2profile`. Refer to "Setting Your Environment" on page 21 if you need more information.
2. To build DB2 programs containing embedded SQL, or to run any DB2 programs, the database manager on the server must be started. Start the database manager, if it is not already running, by entering the following command on the server:

   ```
   db2start
   ```

### Considerations for running IBM and Micro Focus COBOL

Because of the way AIX loads stored procedures and resolves library references within them, there are requirements on how COBOL should be installed. These requirements become a factor when a COBOL program loads a shared library (stored procedure) at run time.

When a stored procedure is loaded, the chain of libraries it refers to must also be loaded. When AIX searches for a library only indirectly referenced by your program, it must use the path compiled into the library that referenced it when it was built by the language provider (IBM COBOL or Micro Focus COBOL). This path may very well not be the same path in which the compiler was installed. If the library in the chain cannot be found, the stored procedure load will fail, and you will receive SQLCODE -10013.

To ensure this does not happen, install the compiler wherever you want, then create symbolic links of all language libraries from the install directory into `/usr/lib` (a directory that is almost always searched when a library needs to be loaded). You could link the libraries into `sqllib/function` (the stored procedure directory), but this only works for one database instance; `/usr/lib` works for everyone on the machine. It is strongly recommended that you do not copy the libraries in; this especially applies to Micro Focus COBOL when multiple copies of the libraries exist.

A sample symbolic link of Micro Focus COBOL is provided below (assuming it is installed in `/usr/lpp/cobdir`):

```
[1]> su root
[2]> cd /usr/lib
[1]> ln -sf /usr/lpp/cobdir/coblib/*.a .
```

### About Stored Procedures and User-Defined Functions (UDFs)

© Copyright IBM Corp. 1993, 1998

Stored procedures are programs that access the database and return information to your client application. User-Defined Functions (UDFs) are your own scalar or table functions. Stored procedures and UDFs are compiled on the server, and stored and executed in a shared library on the server. This shared library is created when you compile the stored procedures and UDFs.

The shared library has an entry point, which is called from the server to access procedures in the shared library. The IBM XL C compiler on AIX allows you to specify any exported function name in the library as the default entry point. This is the function that is called if only the library name is specified in a stored procedure call or CREATE FUNCTION statement. This can be done with the `-e` option in the link step. For example:

```
-e funcname
```

makes `funcname` the default entry point. For information on how this relates to the CREATE FUNCTION statement, see "Relationship to Your CREATE FUNCTION Statement" on page 42.

On other UNIX platforms, no such mechanism exists, so the default entry point is assumed by DB2 to be the same name as the library itself.

AIX requires you to provide an export file which specifies which global functions in the library are callable from outside it. This file must include the names of all stored procedures and/or user-defined functions in the library. Other UNIX platforms simply export all global functions in the library. This is an example of an AIX export file:

```
#! outsrv export file
outsrv
```

The export file `outsrv.exp` lists the stored procedure `outsrv`. The linker uses `outsrv.exp` to create the shared library `outsrv` that contains the stored procedure of the same name.

**Note:** After the shared library is built, it is typically copied into a directory from which DB2 will access it. When attempting to replace either a stored procedure or a user-defined function shared library, you should either run `/usr/sbin/slibclean` to flush the AIX shared library cache, or remove the library from the target directory and then copy the library from the source directory to the target directory. Otherwise, the copy operation may fail because AIX keeps a cache of referenced libraries and does not allow the library to be overwritten.

The AIX compiler documentation has additional information on export files.

## IBM C

The script file `bldxlc`, in `sqllib/samples/c`, contains the commands to build a sample C program.

The first parameter, $1, specifies the name of your source file. The second parameter, $2, specifies the name of the database to which you want to connect. Parameter $3 specifies the user ID for the database, and $4 specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default sample database.

```
#! /bin/ksh
# bldxlc script file
# Builds a sample C program containing embedded SQL
# Usage: bldxlc <prog_name> [ <db_name> [ < userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqc bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
xlc -I/usr/lpp/db2_05_00/include -c util.c

# Compile the program.
xlc -I/usr/lpp/db2_05_00/include -c $1.c

# Link the program.
xlc -o $1 $1.o util.o -ldb2 -L/usr/lpp/db2_05_00/lib
```

| Compile and Link Options for bldxlc | |
|---|---|
| The script file contains the following compile options: | |
| `xlc` | The IBM XL C compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `xlc` | Use the compiler to link edit. |
| `-o` *filename* | Specify the name of the executable program. |
| `util.o` | Include the object file for error checking. |
| `-ldb2` | Link to the database manager library. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqc`, enter:

```
bldxlc updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build C applications that do not contain embedded SQL, you can use the script file `bldxlcapi`. It contains the same compile and link options as `bldxlc`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link the DB2 API sample programs written in C.

## Building C Stored Procedures

The script file `bldxlcsrv`, in `sqllib/samples/c`, contains the commands to build a stored procedure. The script file compiles the stored procedure into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. Parameter `$3` specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name, and for the main entry point to the shared library.

```
#! /bin/ksh
# bldxlcsrv script file
# Builds a stored procedure
# Usage: bldxlcsrv <stor_proc_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqc bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
xlc -I/usr/lpp/db2_05_00/include -c util.c

# Compile the program.
xlc -I/usr/lpp/db2_05_00/include -c $1.c

# Link the program using the export file $1.exp,
# creating a shared library called $1 with the default
# entry point $1.
xlc -o $1 $1.o util.o -ldb2 -L/usr/lpp/db2_05_00/lib \
    -H512 -T512 -bE:$1.exp -e $1

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldxlcsrv | |
|---|---|
| The script file contains the following compile options: | |
| `xlc` | The IBM XL C compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `xlc` | Use the compiler to link edit. |
| `-o` *filename* | Specify the output as a shared library file. |
| `util.o` | Include the object file for error checking. |
| `-ldb2` | Link with the database manager library. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| `-H512` | Specify output file alignment. |
| `-T512` | Specify output file text segment starting address. |
| `-bE:`*filename*.exp | Specify an export file. The export file contains a list of the stored procedures. |
| `-e` *entry* | Specify the default entry point to the shared library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqc`, enter:

```
bldxlcsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldxlc` script file. Refer to "IBM C" on page 34 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
: Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid*
: Is a valid user ID.

*password*
: Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Coding and Compiling Stored Procedures

This section provides a general discussion about coding stored procedures, and the compiler options you can use.

### Relationship to Your CALL Statement

The *Embedded SQL Programming Guide* describes how to code your stored procedure. The *SQL Reference* describes how to invoke your stored procedure at the location of a database using the CALL statement. This section ties how you compile and link your stored procedure to the information you provide in the CALL statement.

When you compile and link your program, you can identify functions in two ways:

- Using the `-e` option.

  For example, you can specify the following in the link step:

  ```
  -e modify
  ```

  This indicates that the default entry point for the linked library is the function `modify`.

  If you are linking a library `mystored` in a directory `/u/mydir/procs`, and you want to use the default entry point `modify` as specified above, code your CALL statement as follows:

  ```
  CALL '/u/mydir/procs/mystored'
  ```

  The library `mystored` is loaded into memory, and the function `modify` is picked up by DB2 as the default entry point, and is executed.

- Using an export file specified using the `-bE:` option.

Generally speaking, you would use this link option when you have more than one stored procedure in your library, and you want to access additional functions as stored procedures.

To continue the example from above, suppose that the library `mystored` contains three stored procedures: `modify` as above, `remove`, and `add`. You identify `modify` as the default entry point, as above, and indicate in the link step that `remove` and `add` are additional entry points by including them in an export file.

In the link step, you specify:

```
-bE:mystored.exp
```

which identifies the export file `mystored.exp`.

The export file looks like this:

```
* additional entry points for mystored
#!
remove
add
```

Finally, your two CALL statements for the stored procedures, which invoke the `remove` and `add` functions, are coded as follows:

```
CALL '/u/mydir/procs/mystored!remove'
```

and

```
CALL '/u/mydir/procs/mystored!add'
```

## Building C User-Defined Functions (UDFs)

The script file `bldxlcudf`, in `sqllib/samples/c`, contains the commands to build a UDF. UDFs are compiled like stored procedures, but you do not need to connect to a database or precompile and bind the program.

**Note:** A UDF does not contain embedded SQL statements. Rather, the application that uses the UDF contains the statements, such as `calludf`.

The first parameter, `$1`, specifies the name of your source file.

The script file uses the source file, `$1`, for the shared library name, and for the default entry point to the shared library.

```
#! /bin/ksh
# bldxlcudf script file
# Builds a sample C UDF library.
# Usage: bldxlcudf <prog_name>

# Compile the program.
xlc -I/usr/lpp/db2_05_00/include -c $1.c

# Link the program.
xlc -o $1 $1.o -ldb2 -ldb2apie -L/usr/lpp/db2_05_00/lib -H512 -T512 -bE:$1.exp -e $1

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

**Compile and Link Options for bldxlcudf**

The script file contains the following compile options:

| | |
|---|---|
| `xlc` | The IBM XL C compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

The script file contains the following link options:

| | |
|---|---|
| `xlc` | Use the compiler to link edit. |
| `-o` *filename* | Specify the output as a shared library file. |
| `-ldb2` | Link with the database manager library. |
| `-ldb2apie` | Link with the DB2 API Engine library to allow the use of LOB locators. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| `-H512` | Specify output file alignment. |
| `-T512` | Specify output file text segment starting address. |
| `-bE:`*filename*`.exp` | Specify an export file. The export file contains a list of the UDFs. |
| `-e` *entry* | Specify the default entry point to the shared library. |

Refer to your compiler documentation for additional compiler options. Refer to “Coding and Compiling UDFs” on page 42 for a general discussion about compiler options and UDFs.

To build the user-defined function program `udf` from the source file `udf.c`, enter:

```
bldxlcudf udf
```

The script file copies the UDF to the server in the path `sqllib/function` to indicate that the UDF is fenced. If you want the UDF to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced UDF or stored procedure runs in the same address space as the database manager and results in increased performance when compared to a fenced UDF or stored procedure, which runs in an address space isolated from the database manager. With unfenced UDFs or stored procedures there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced UDFs or stored procedures when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced UDFs.

If necessary, set the file mode for the UDF so the DB2 instance can run it.

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build `calludf` using the `bldxlc` script file. Refer to "IBM C" on page 34 for details.

To call the UDF, run the sample calling application by entering:

```
calludf
```

The calling application calls functions from the `udf` library.

After you run the calling application, you can also invoke the UDF interactively using the command line processor like this:

```
db2 "SELECT name, DOLLAR(salary), SAMP_MUL(DOLLAR(salary), FACTOR(1.2)) FROM staff"
```

You do not have to type the command line processor commands in uppercase.

## Coding and Compiling UDFs

This section provides a general discussion about coding UDFs, and the compiler options you can use.

### Relationship to Your CREATE FUNCTION Statement

The *Embedded SQL Programming Guide* describes how to code your UDF. The *SQL Reference* describes how to register your UDF with DB2 using the CREATE FUNCTION statement. This section ties how you compile and link your UDF to the information you provide in the EXTERNAL NAME clause of the CREATE FUNCTION statement.

When you compile and link your program, you can identify functions in two ways:

- Using the `-e` option.

  For example, you can specify the following in the link step:

  ```
  -e modify
  ```

This indicates that the default entry point for the linked library is the function `modify`.

If you are linking a library `myudfs` in a directory `/u/mydir/procs`, and you want to use the default entry point `modify` as specified above, include the following in your CREATE FUNCTION statement:

```
EXTERNAL NAME '/u/mydir/procs/myudfs'
```

DB2 picks up the default entry point of the library `myudfs`, which is the function `modify`.

- Using an export file specified using the `-bE:` option.

  Generally speaking, you would use this link option when you have more than one UDF in your library, and you want to access additional functions as UDFs.

  To continue the example from above, suppose that the library `myudfs` contains three UDFs: `modify` as above, `remove`, and `add`. You identify `modify` as the default entry point, as above, and indicate in the link step that `remove` and `add` are additional entry points by including them in an export file.

  In the link step, you specify:

  ```
  -bE:myudfs.exp
  ```

  which identifies the export file `myudfs.exp`.

  The export file looks like this:

```
* additional entry points for myudfs
#!
remove
add
```

  Finally, your two CREATE FUNCTION statements for the UDFs, which are implemented by the `remove` and `add` functions, would contain these EXTERNAL NAME clauses:

  ```
  EXTERNAL NAME '/u/mydir/procs/myudfs!remove'
  ```

  and

  ```
  EXTERNAL NAME '/u/mydir/procs/myudfs!add'
  ```

## Multi-threaded Applications

Multi-threaded applications on AIX Version 4 need to be compiled and linked with the `xlc_r` compiler instead of the `xlc` compiler or, for C++, with the `xlC_r` compiler instead of the `xlC` compiler. The `_r` versions pass `-D_REENTRANT` to the C preprocessor and `-lpthreads` to the linker, so it is not necessary to do this explicitly. Please see the /etc/xlC.cfg file for more information.

The makefile in `sqllib/samples/c` contains the commands to build a sample C multi-threaded program, using the `xlc_r` compiler. The makefile uses variables for the

compiler and compile and link options. When the make command is run the variables are replaced by their values, as in the following example. Here, the makefile is used to compile the `thdsrver` program with the `xlc_r` compiler:

```
xlc_r -o thdsrver thdsrver.c -I/home/db2inst/sqllib/include
  -L/home/db2inst/sqllib/lib -ldb2
```

where `/home/db2inst` is the DB2 instance directory.

For definitions of these options, please see "Compile and Link options for bldxlc" in “IBM C” on page 34.

To build the sample program `thdsrver`, enter:

```
make thdsrver
```

The result is an executable file, `thdsrver`. To run the executable file against the `sample` database, enter:

```
thdsrver
```

## IBM C Set++

The script file `bldcset`, in `sqllib/samples/cpp`, contains the commands to build a sample C++ program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. Parameter `$3` specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldcset script file
# Build sample C++ program that contains embedded SQL.
# Usage: bldcset <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqC bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
xlC -I/usr/lpp/db2_05_00/include -c util.C

# Compile the program.
xlC -I/usr/lpp/db2_05_00/include -c $1.C

# Link the program.
xlC -o $1 $1.o util.o -ldb2 -L/usr/lpp/db2_05_00/lib
```

| Compile and Link Options for bldcset | |
|---|---|
| The script file contains the following compile options: | |
| `xlC` | The IBM C Set ++ compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldcset | |
|---|---|
| The script file contains the following link options: | |
| `xlC` | Use the compiler to link edit. |
| `-o` *filename* | Specify the name of the executable program. |
| `util.o` | Include the object file for error checking. |
| `-ldb2` | Link with the database manager library. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqC`, enter:

```
bldcset updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

## Building C++ Stored Procedures

The script file `bldcsetsrv`, in `sqllib/samples/cpp`, contains the commands to build a stored procedure. The script file compiles the stored procedure into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. Parameter `$3` specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name, and for the main entry point to the shared library.

```
#! /bin/ksh
# bldcsetsrv script file
# Builds a C++ stored procedure.
# Usage: bldcsetsrv <stor_proc_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqC bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
xlC -I/usr/lpp/db2_05_00/include -c util.c

# Compile the program.
xlC -I/usr/lpp/db2_05_00/include -c $1.C

# Link the program using the export file $1.exp,
# creating a shared library called $1 with the main
# entry point $1.
makeC++SharedLib -p 1024 -o $1 $1.o util.o -ldb2 -L/usr/lpp/db2_05_00/lib \
    -H512 -T512 -bE:$1.exp -e $1

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldcsetsrv | |
|---|---|
| The script file contains the following compile options: | |
| `xlC` | The IBM C Set++ compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `makeC++SharedLib` | Linker script for stored procedures with static constructors. |
| `-p 1024` | Set the priority to the arbitrary value of 1024. |
| `-o` *filename* | Specify the output as a shared library file. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| `-E` *filename*.exp | Specify an export file. The export file contains a list of the stored procedures. |
| `-ldb2` | Link with the database manager library. |
| `util.o` | Include the object file for error checking. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqc`, enter:

```
bldcsetsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldxlc` script file. Refer to “IBM C Set++” on page 44 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

### Multi-threaded Applications

C++ multi-threaded applications on AIX Version 4 need to be compiled and linked with the `xlC_r` compiler instead of the `xlC` compiler. See "Multi-threaded Applications" on page 43 for information on building C multi-threaded applications.

## IBM XL Fortran for AIX

The script file `bldxlf`, in `sqllib/samples/fortran`, contains the commands to build a sample Fortran program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. Parameter `$3` specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldxlf script file
# Build sample Fortran program containing embedded SQL.
# Usage:  bldxlf <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqf bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.f error-checking utility.
xlf -I/usr/lpp/db2_05_00/include -c util.f

# Compile the program.
xlf -I/usr/lpp/db2_05_00/include -c $1.f

# Link the program.
xlf -o $1 $1.o util.o -ldb2 -L/usr/lpp/db2_05_00/lib
```

| Compile and Link Options for bldxlf | |
|---|---|
| The script file contains the following compile options: | |
| `xlf` | The Fortran compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldxlf | |
|---|---|
| The script file contains the following link options: | |
| `xlf` | Use the compiler to link edit. |
| `-o` *filename* | Specify the name of the executable program. |
| `util.o` | Include the object file for error-checking. |
| `-ldb2` | Link with the database manager library. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqf`, enter:

```
bldxlf updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build Fortran applications that do not contain embedded SQL, you can use the script file `bldxlfapi`. It contains the same compile and link options as `bldxlf`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link the DB2 API sample programs written in Fortran.

## Building Fortran Stored Procedures

The script file `bldxlfsrv`, in `sqllib/samples/fortran`, contains the commands to build a stored procedure. The script file compiles the stored procedure into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. Parameter `$3` specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name, and for the main entry point to the shared library.

```
#! /bin/ksh
# bldxlfsrv script file
# Builds a Fortran stored procedure.
# Usage: bldxlfsrv <stor_proc_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqf bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.f error-checking utility.
xlf -I/usr/lpp/db2_05_00/include -c util.f

# Compile the program.
xlf -I/usr/lpp/db2_05_00/include -c $1.f

# Link the program using the export file $1.exp,
# creating a shared library called $1 with the main
# entry point $1.
xlf -o $1 $1.o util.o -ldb2 -L/usr/lpp/db2_05_00/lib \
    -H512 -T512 -bE:$1.exp -e $1

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldxlfsrv | |
|---|---|
| The script file contains the following compile options: | |
| `xlf` | The Fortran compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `xlf` | Use the compiler to link edit. |
| `-o` *filename* | Specify the output as a shared library file. |
| `util.o` | Include the object file for error-checking. |
| `-ldb2` | Link with the database manager library. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| `-H512` | Specify output file alignment. |
| `-T512` | Specify output file text segment starting address. |
| `-bE:`*filename*.exp | Specify an export file. The export file contains a list of the stored procedures. |
| `-e` *entry* | Specify the default entry point to the shared library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqf`, enter:

```
bldxlfsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldxlc` script file. Refer to "IBM XL Fortran for AIX" on page 49 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
: Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid*
: Is a valid user ID.

*password*
: Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Using the IBM XL Fortran for AIX Compiler

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the IBM XL Fortran for AIX compiler, keep the following points in mind:

- The Fortran compiler (xlf) treats lines with a `D` or `d` in column 1 as conditional lines. You can either compile these lines for debugging or treat them as comments.

  The precompiler always treats lines with a `D` or `d` in column one as comments.

- The compiler is case insensitive by default. You can make it case sensitive by using a compiler option.

  SQL keywords are always case insensitive. If you make the compiler case sensitive, you must enter all Fortran keywords in lowercase. Additionally, identifier references must match the case of declarations.

- A single tab introduces source lines such that the following character is positioned at column 7. The compiler treats tabs in locations other than between columns 1-6, and in character constants, as blanks.

- You cannot use the following data declaration keywords in host variable declarations: `POINTER`, `BYTE`, `STATIC`, and `AUTOMATIC`.

- Pass by-value arguments using `%VAL()` and by-reference arguments using `%REF()`. The *API Reference* uses this syntax in the Fortran DB2 API examples.

- You cannot use the XL Fortran for AIX free-format option in .sqf files.

- The DB2 precompiler is case insensitive, but XL Fortran for AIX may not be, depending on compiler options. Therefore, do not use host variables with the same spelling and expect the case of the letters in the variable to make them unique. For example, the precompiler treats `NAME`, `name`, and `Name` as equal.

  Similarly, the following keywords are recognized to differing extents by the precompiler, and always in a case insensitive manner:

| | | | |
|---|---|---|---|
| | @PROCESS | ENDIF | INTERFACE |
| | AUTOMATIC | ENDFORALL | LOGICAL |
| | BLOCKDATA | ENDINTERFACE | MODULE |
| | BYTE | ENDSELECT | PARAMETER |
| | CHARACTER | ENDTYPE | POINTER |
| | COMPLEX | ENDWHERE | PROGRAM |
| | CONTAINS | ENTRY | REAL |
| | DOUBLECOMPLEX | FORMAT | STATIC |
| | DOUBLEPRECISION | FUNCTION | SUBROUTINE |
| | END | IF | TYPE |
| | ENDDO | IMPLICIT | USE |
| | ENDFILE | INTEGER | |

- The precompiler allows only digits, blanks, and tab characters within columns 1-5 on continuation lines.
- You cannot use the \ character to include string delimiters within strings. For example, use the strings `'the''character'` or `"the""character"` instead of `'the\'character'` or `"the\"character"`.
- Fortran `.sqf` source files do not support Hollerith constants.

## IBM COBOL Set for AIX

The script file `bldcob`, in `sqllib/samples/cobol`, contains the commands to build an embedded SQL sample COBOL program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. Parameter `$3` specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldcob script file
# Builds a COBOL program containing embedded SQL
# Usage:  bldcob <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqb bindfile target ibmcob

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the checkerr.cbl error checking utility.
cob2 -qpgmname\(mixed\) -qlib -I/usr/lpp/db2_05_00/include/cobol_a \
     -c checkerr.cbl

# Compile the program.
cob2 -qpgmname\(mixed\) -qlib -I/usr/lpp/db2_05_00/include/cobol_a \
     -c $1.cbl

# Link the program.
cob2 -o $1 $1.o checkerr.o -ldb2 -L/usr/lpp/db2_05_00/lib
```

| Compile and Link Options for bldcob | |
|---|---|
| The script file contains the following compile options: | |
| `cob2` | The IBM COBOL Set compiler. |
| `-qpgmname\(mixed\)` | Instructs the compiler to permit CALLs to library entry points with mixed-case names. |
| `-qlib` | Instructs the compiler to process COPY statements. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include/cobol_a`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldcob | |
|---|---|
| The script file contains the following link options: | |
| `cob2` | Use the compiler to link edit. |
| `-o` *filename* | Specify the name of the executable program. |
| `checkerr.o` | Include the object file for error-checking. |
| `-ldb2` | Link with the database manager library. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqb`, enter:

```
bldcob updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build IBM COBOL applications that do not contain embedded SQL, you can use the script file `bldcobapi`. It contains the same compile and link options as `bldcob`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link DB2 API sample programs written in COBOL.

## Building IBM COBOL Set for AIX Stored Procedures

The script file `bldcobsrv`, in `sqllib/samples/cobol`, contains the commands to build a stored procedure. The script file compiles the stored procedure into a shared library on the server that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. Parameter `$3` specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name, and for the main entry point to the shared library.

```
#! /bin/ksh
# bldcobsrv script file
# Build a COBOL stored procedure.
# Usage:  bldcobsrv <stor_proc_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqb bindfile target ibmcob

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the checkerr.cbl error checking utility.
cob2 -qpgmname\(mixed\) -qlib -I/usr/lpp/db2_05_00/include/cobol_a \
     -c checkerr.cbl

# Compile the program.
cob2 -qpgmname\(mixed\) -qlib -c -I/usr/lpp/db2_05_00/include/cobol_a $1.cbl

# Link the program using the export file $1.exp
# creating a shared library called $1 with the main
# entry point $1.
cob2 -o $1 $1.o checkerr.o -H512 -T512 -e $1 -bE:$1.exp \
     -L/usr/lpp/db2_05_00/lib -ldb2
# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldcobsrv | |
|---|---|
| The script file contains the following compile options: | |
| `cob2` | The IBM COBOL Set compiler. |
| `-qpgmname\(mixed\)` | Instructs the compiler to permit CALLs to library entry points with mixed-case names. |
| `-qlib` | Instructs the compiler to process COPY statements. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include/cobol_a`. |
| The script file contains the following link options: | |
| `cob2` | Use the compiler to link edit. |
| `-o` *filename* | Specify the output as a shared library file. |
| `checkerr.o` | Include the object file for error-checking. |
| `-H512` | Specify output file alignment. |
| `-T512` | Specify output file text segment starting address. |
| `-e` *entry* | Specify the default entry point to the shared library. |
| `-bE:`*filename*.exp | Specify an export file. The export file contains a list of the stored procedures. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| `-ldb2` | Link with the database manager library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqb`, enter:

```
bldcobsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the

*Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldcob` script file. Refer to "IBM COBOL Set for AIX" on page 55 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
: Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid*
: Is a valid user ID.

*password*
: Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Using the IBM COBOL Set for AIX Compiler

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the IBM COBOL Set for AIX compiler, keep the following points in mind:

- When you precompile your application using the command line processor command `db2 prep`, use the `target ibmcob` option.
- Do not use tab characters in your source files.
- You can use the `PROCESS` and `CBL` keywords in the first line of your source files to set compile options.
- If your application contains only embedded SQL, but no DB2 API calls, you do not need to use the `pgmname(mixed)` compile option. If you use DB2 API calls, you must use the `pgmname(mixed)` compile option.
- If you are using the "System/390 host data type support" feature of the IBM COBOL Set for AIX compiler Version 2.0, the DB2 include files for your applications are in the following directory:

  `$HOME/sqllib/include/cobol_i`

  If you are building DB2 sample programs using the build scripts provided, the include file path specified in the build scripts must be changed to point to the `cobol_i` directory and not the `cobol_a` directory.

  If you are NOT using the "System/390 host data type support" feature of the IBM COBOL Set for AIX compiler Version 2.0, or you are using an earlier version of this

compiler, then the DB2 include files for your applications are in the following directory:

```
$HOME/sqllib/include/cobol_a
```

Specify COPY file names to include the `.cbl` extension as follows:

```
COPY "sql.cbl".
```

## Micro Focus COBOL

The script file `bldmfcob`, in `sqllib/samples/cobol_mf`, contains the commands to build a sample COBOL program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. Parameter `$3` specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldmfcob script file
# Builds a COBOL program containing embedded SQL
# Usage: bldmfcob <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqb bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=/usr/lpp/db2_05_00/include/cobol_mf:$COBCPY

# Compile the checkerr.cbl error checking utility.
cob -c -x checkerr.cbl

# Compile the program.
cob -c -x $1.cbl

# Link the program.
cob -x -o $1 $1.o checkerr.o -ldb2 -ldb2gmf -L/usr/lpp/db2_05_00/lib
```

| Compile and Link Options for bldmfcob | |
|---|---|
| The script file contains the following compile options: | |
| `cob` | The COBOL compiler. |
| `-c` | Perform compile only; no link. |
| `-x` | Produce an executable program. |

| Compile and Link Options for bldmfcob | |
|---|---|
| The script file contains the following link options: | |
| `cob` | Use the compiler to link edit. |
| `-x` | Produce an executable program. |
| `-o` *filename* | Specify the name of the executable program. |
| `-ldb2` | Link to the database manager library. |
| `-ldb2gmf` | Link to the DB2 exception-handler library for Micro Focus COBOL. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqb`, enter:

```
bldmfcob updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build Micro Focus COBOL applications that do not contain embedded SQL, you can use the script file `bldmfapi`. It contains the same compile and link options as `bldmfcob`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link DB2 API sample programs written in COBOL.

## Building Micro Focus COBOL Stored Procedures

**Note:** Some of the more recent versions of the Micro Focus COBOL compiler, used on an AIX Version 4 platform, cannot be used to create a statically-linked stored procedure. As such, the makefile and script file, `bldmfcobs`, have been adapted to allow for the creation of a dynamically-linked stored procedure.

In order for a remote client application to successfully call this dynamically-linked stored procedure, it is necessary for a Micro Focus COBOL routine, `cobinit()`, to be called on the server where the stored procedure resides just before the stored procedure is executed. A wrapper program which accomplishes this is created during the execution of the makefile, or the script file `bldmfcobs`. It is then linked with the stored procedure code to form the stored procedure shared library. Due to the use of this wrapper program, in order for a client application to call a stored procedure named `x`, it must call `x_wrap` instead of `x`.

The details of the wrapper program are explained later in this section.

The script file, `bldmfcobs`, in `sqllib/samples/cobol_mf`, contains the commands to build a stored procedure. The script file compiles the stored procedure into a shared library on the server that can be called by a client application.

The first parameter, $1, specifies the name of your source file. The second parameter, $2, specifies the name of the database to which you want to connect. Parameter $3 specifies the user ID for the database, and $4 specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default sample database.

The script file uses the source file name, $1, for the shared library name.

```
#! /bin/ksh
# bldmfcobs script file
# Build sample COBOL stored procedure
# Usage: bldmfcobs <stored_proc_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi
# Precompile the program.
db2 prep $1.sqb bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=/usr/lpp/db2_05_00/include/cobol_mf:$COBCPY
# Compile the checkerr.cbl error checking utility.
cob -c -x checkerr.cbl
# Compile the program.
cob -c -x $1.cbl
# Create the wrapper program for the stored procedure.
wrpmfcobs $1
# Link the program using the export file ${1}_wrap.exp,
# creating a shared library called $1 with the main
# entry point ${1}_wrap.
cob -x -o $1 ${1}_wrap.c $1.o -Q -bE:${1}_wrap.exp -Q "-e $1" \
-Q -bI:/usr/lpp/db2_05_00/lib/db2g.imp -ldb2gmf -L/usr/lpp/db2_05_00/lib
# Copy the shared library to the DB2 instance sqllib/function subdirectory.
# Note: this assumes the user has write permission to this directory.
eval "H= $DB2INSTANCE"
rm $H/sqllib/function/$1
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldmfcobs | |
|---|---|
| The script file contains the following compile options: | |
| `cob` | The COBOL compiler. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| `-x` | Produce an executable program. |
| The script file contains the following link options: | |
| `cob` | Use the compiler to link edit. |
| `-x` | Produce an executable program. |
| `-o` *filename* | Specify the name of the executable program. |
| `-Q -bE:`*filename*`.exp` | Specify an export file. The export file contains a list of the stored procedure entry points. If a stored procedure is called `x`, then its entry point will be `x_wrap`. |
| `-Q "-e $1"` | Specify the default entry point to the shared library. |
| `-Q -bI:/usr/lpp/db2_05_00/lib/db2g.imp` | Provides a list of entry points to the DB2 application library. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| `-ldb2gmf` | Link to the DB2 exception-handler library for Micro Focus COBOL. |
| Refer to your compiler documentation for additional compiler options. | |

The wrapper program, `wrpmfcobs`, causes the Micro Focus COBOL routine, `cobinit()`, to be called right before the stored procedure is executed. Its contents are shown below.

```
#! /bin/ksh
# wrpmfcobs script file
# Create the wrapper program for sample COBOL stored procedure
# Usage: wrpmfcobs <stored_proc_name>

# Note: With the wrapper program, the client application must call x_wrap
#       instead of x, where x is the original name of the stored procedure.

# Create the wrapper program for the stored procedure.
cat << WRAPPER_CODE > ${1}_wrap.c
#include <stdio.h>
void cobinit(void);
int $1(void *p0, void *p1, void *p2, void *p3);

int main(void)
{
  return 0;
}

int ${1}_wrap(void *p0, void *p1, void *p2, void *p3)
{
  cobinit();
  return $1(p0, p1, p2, p3);
}
WRAPPER_CODE
# Create the export file for the wrapper program
echo $1_wrap > ${1}_wrap.exp
```

To build the sample program `outsrv` from the source file `outsrv.sqb`, enter:

```
bldmfcobs outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldcob` script file. Refer to "Micro Focus COBOL" on page 61 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
: Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid*
: Is a valid user ID.

*password*
: Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

### Exiting the Stored Procedure

When you develop a stored procedure, exit the stored procedure using the following statement:

```
move SQLZ-HOLD-PROC to return-code.
```

With this statement, the stored procedure returns correctly to the client application. This is especially important when the stored procedure is called by a local COBOL client application.

## REXX

You do not precompile or bind REXX programs.

To run DB2 REXX/SQL programs on AIX, you must set the `LIBPATH` environment variable to include `sqllib/lib` under the DB2 install directory.

If `LIBPATH` has not been set yet, enter:

```
export LIBPATH=/lib:/usr/lib:/usr/lpp/db2_05_00/sqllib/lib
```

If `LIBPATH` has been set already, enter:

```
export LIBPATH=$LIBPATH:/usr/lpp/db2_05_00/sqllib/lib
```

On AIX, your application file can have any file extension. You can run your application using either of the following two methods:

1. At the shell command prompt, enter `rexx` *name* where *name* is the name of your REXX program.

2. If the first line of your REXX program contains a "magic number", (#!), and identifies the directory where the REXX/6000 interpreter resides, you can run your REXX program by entering its name at the shell command prompt. For example, if the REXX/6000 interpreter file is in the `/usr/bin` directory, include the following as the very first line of your REXX program:

   ```
   #! /usr/bin/rexx
   ```

   Then, make the program executable by entering the following command at the shell command prompt:

   `chmod +x`*name*

   Run your REXX program by entering its file name at the shell command prompt.

REXX sample programs are in the directory `sqllib/samples/rexx`. To run the sample REXX program `updat.cmd`, do one of the following:

- Run the program directly. Enter:

  ```
  updat.cmd
  ```

- Specify the REXX interpreter and the program. Enter:

  ```
  rexx updat.cmd
  ```

For further information on REXX and DB2, refer to the *Embedded SQL Programming Guide*, chapter 13, "Programming in REXX".

# Chapter 5. Building HP-UX Embedded SQL Applications

This chapter provides detailed information for building embedded SQL applications on HP-UX. In the script files, commands that begin with `db2` are Command Line Processor (CLP) commands. Refer to the *Command Reference* if you need more information about CLP commands.

**General Points for Building and Running DB2 Programs**

1. You must build and run DB2 applications from a window where your environment variables are set. You can do this by running `db2profile`. Refer to "Setting Your Environment" on page 21 if you need more information.
2. To build DB2 programs containing embedded SQL, or to run any DB2 programs, the database manager on the server must be started. Start the database manager, if it is not already running, by entering the following command on the server:

   ```
   db2start
   ```

**Note:** If you are migrating DB2 from HP-UX Version 10 or earlier to HP-UX Version 11, your DB2 programs must be re-precompiled with DB2 on HP-UX Version 11 (if they include embedded SQL), and must be re-compiled. This includes all DB2 applications, stored procedures, user-defined functions and user exit programs. As well, DB2 programs that are compiled on HP-UX Version 11 may not run on HP-UX Version 10 or earlier. DB2 programs that are compiled and run on HP-UX Version 10 may connect remotely to HP-UX Version 11 servers.

## HP-UX C

The script file, `bldcc`, in `sqllib/samples/c`, contains the commands to build a sample C program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

© Copyright IBM Corp. 1993, 1998

```
#! /bin/ksh
# bldcc script file
# Builds a sample C program containing embedded SQL
# Usage: bldcc <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqc bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
cc -Aa +DAportable +e -I/opt/IBMdb2/V5.0/include -c util.c

# Compile the program.
cc -Aa +DAportable +e -I/opt/IBMdb2/V5.0/include -c $1.c

# Link the program.
cc +DAportable -o $1 $1.o util.o -L/opt/IBMdb2/V5.0/lib -ldb2 -lhppa
```

**Compile and Link Options for bldcc**

The script file contains the following compile options:

| | |
|---|---|
| `cc` | The C compiler. |
| `-Aa` | Use ANSI standard mode (for the C compiler only). |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `+e` | Enables HP value-added features while compiling in ANSI C mode. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldcc | |
|---|---|
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `+DAportable` | Use code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `-o $1` | Specify the name of the object module. |
| `util.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| `-lhppa` | Specify the HP PA-RISC library (required). |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqc`, enter:

```
bldcc updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build C applications that do not contain embedded SQL, you can use the script file `bldccapi`. It contains the same compile and link options as `bldcc`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link the DB2 API sample programs written in C.

## Building C Stored Procedures

The script file `bldccsrv`, in `sqllib/samples/c`, contains the commands to build a C stored procedure. The script file compiles the stored procedure into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldccsrv script file
# Build C stored procedure.
# Usage: bldccsrv <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqc bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the program.
cc +u1 +Z -Aa +DAportable +e -I/opt/IBMdb2/V5.0/include -c $1.c

# Link the program to create a shared library
ld -b -o $1 $1.o -L/opt/IBMdb2/V5.0/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldccsrv | |
|---|---|
| The script file contains the following compile options: | |
| `cc` | The C compiler. |
| `+u1` | Allow unaligned data access. Use only if your application uses unaligned data. |
| `-Aa` | Use ANSI standard mode (for the C compiler only). |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `+Z` | Generate position-independent code. |
| `+e` | Enables HP value-added features while compiling in ANSI C mode. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `ld` | Use the linker to link edit. |
| `-b` | Create a shared library rather than a normal executable. |
| `-o $1` | Specify the name of the object module. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqc`, enter:

```
bldccsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldcc` script file. Refer to “HP-UX C” on page 69 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
: Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid*
: Is a valid user ID.

*password*
: Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Building C User-Defined Functions (UDFs)

The script file `bldccudf`, in `sqllib/samples/c`, contains the commands to build a UDF. UDFs are compiled like stored procedures, but you do not need to connect to a database or precompile and bind the program.

**Note:** A UDF does not contain embedded SQL statements. Rather, the application that uses the UDF contains the statements, such as `calludf`.

The first parameter, `$1`, specifies the name of your source file. The script file also uses this source file name for the shared library name.

```
#! /bin/ksh
# bldccudf script file
# Builds sample c UDF library.
# Usage: bldccudf <prog_name>

# Compile the program.
cc +u1 +Z -Aa +DAportable +e -I/opt/IBMdb2/V5.0/include -c $1.c

# Link the program and create a shared library.
ld -b -o $1 $1.o -L/opt/IBMdb2/V5.0/lib -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldccudf | |
|---|---|
| The script file contains the following compile options: | |
| `cc` | The C compiler. |
| `+u1` | Allow unaligned data access. Use only if your application uses unaligned data. |
| `-Aa` | Use ANSI standard mode (for the C compiler only). |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `+Z` | Generate position-independent code. |
| `+e` | Enables HP value-added features while compiling in ANSI C mode. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `ld` | Use the linker to link edit. |
| `-b` | Create a shared library rather than a normal executable. |
| `-o $1` | Specify the name of the object module. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| `-ldb2apie` | Link with the DB2 API Engine library to allow the use of LOB locators. |
| Refer to your compiler documentation for additional compiler options. | |

To build the user-defined function program `udf` from the source file `udf.c`, enter:

```
bldccudf udf
```

The script file copies the UDF to the server in the path `sqllib/function` to indicate that the UDF is fenced. If you want the UDF to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced UDF or stored procedure runs in the same address space as the database manager and results in increased performance when compared to a fenced UDF or stored procedure, which runs in an address space isolated from the database manager. With unfenced UDFs or stored procedures there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced UDFs or stored procedures when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced UDFs.

If necessary, set the file mode for the UDF so the DB2 instance can run it.

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build `calludf` using the `bldcc` script file. Refer to “HP-UX C” on page 69 for details.

To call the UDF, run the sample calling application by entering:

```
calludf
```

The calling application calls functions from the `udf` library.

## Multi-threaded Applications

**Note:** Multi-threaded applications are not supported by DB2 on version 10 of the HP-UX operating system. HP-UX version 11 provides a posix thread library and a DCE thread library. Multi-threaded applications using the posix thread library are supported by DB2 on HP-UX version 11.

Multi-threaded applications on HP-UX version 11 need to have _REENTRANT defined for their compilation. The HP-UX documentation recommends compiling with -D_POSIX_C_SOURCE=199506L. This will also ensure _REENTRANT is defined. Applications also need to be linked with `-lpthread`.

The makefile in `sqllib/samples/c` contains the commands to build a sample C multi-threaded program. The makefile defines variables for the compile and link options. When the make command is run the variables are replaced by their values, as in the following example where the makefile is used to compile the `thdsrver` program:

```
cc -o thdsrver thdsrver.c -Aa +DAportable +e -I/home/db2inst/sqllib/include
  -D_POSIX_C_SOURCE=199506L -L/home/db2inst/sqllib/lib -ldb2 -lhppa -lpthread
```

where `/home/db2inst` is the DB2 instance directory.

For definitions of the non-multi-threaded options, please see "Compile and Link options for bldcc" in “HP-UX C” on page 69.

To build the sample program, `thdsrver`, enter:

```
make thdsrver
```

The result is an executable file, `thdsrver`. To run the executable file against the `sample` database, enter:

```
thdsrver
```

## HP-UX C++

The script file `bldCC`, in `sqllib/samples/cpp`, contains the commands to build a sample C++ program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldCC script file
# Builds a C++ program containing embedded SQL
# Usage: bldCC <prog_name> [ <db_name> [ <userid> <password> ]]


# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqC bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.C error-checking utility.
CC +a1 +DAportable -I/opt/IBMdb2/V5.0/include -c util.C

# Compile the program.
CC +a1 +DAportable -I/opt/IBMdb2/V5.0/include -c $1.C

# Link the program.
CC +DAportable -o $1 $1.o util.o -L/opt/IBMdb2/V5.0/lib -ldb2 -lhppa
```

**Compile and Link Options for bldCC**

The script file contains the following compile options:

| | |
|---|---|
| `CC` | The C compiler. |
| `+a1` | Instruct the compiler to use ANSI C/C++. |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldCC | |
|---|---|
| The script file contains the following link options: | |
| `CC` | Use the compiler to link edit. |
| `+DAportable` | Use code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `-o $1` | Specify the name of the object module. |
| `util.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| `-lhppa` | Specify the HP PA-RISC library (required). |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqC`, enter:

```
bldCC updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

## Building C++ Stored Procedures

The script file `bldCCsrv`, in `sqllib/samples/cpp`, contains the commands to build a C++ stored procedure. The script file compiles the stored procedure into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldCCsrv script file
# Builds a C++ stored procedure.
# Usage: bldCCsrv <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqC bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the program. First ensure it is coded with extern "C".
CC +a1 +DAportable +Z -I/opt/IBMdb2/V5.0/include -c $1.C
# Link the program to create a shared library.
ld -b -o $1 $1.o -L/opt/IBMdb2/V5.0/lib -ldb2
# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H= $DB2INSTANCE"
cp $1 $H/sqllib/function
```

**Compile and Link Options for bldCCsrv**

The script file contains the following compile options:

| | |
|---|---|
| `CC` | The C++ compiler. |
| `+a1` | Instruct the compiler to use ANSI C/C++. |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `+Z` | Generate position-independent code. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldCCsrv | |
|---|---|
| The script file contains the following link options: | |
| `ld` | Use the linker to link edit. |
| `-b` | Create a shared library rather than a normal executable. |
| `-o $1` | Specify the name of the object module. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqC`, enter:

```
bldCCsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldCC` script file. Refer to “HP-UX C++” on page 76 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
: Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid*
: Is a valid user ID.

*password*
: Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## HP Fortran/9000

The script file, `bldf77`, in `sqllib/samples/fortran`, contains the commands to build a sample Fortran program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4`, specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldf77 script file
# Builds a Fortran program containing embedded SQL
# Usage: bldf77 <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqf bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.f error-checking utility.
f77 -w -c +DAportable -I/opt/IBMdb2/V5.0/include -c util.f

# Compile the program.
f77 -w -c +DAportable -I/opt/IBMdb2/V5.0/include $1.f

# Link the program.
f77 +DAportable $1.o util.o -Wl,-L/opt/IBMdb2/V5.0/lib -ldb2 -lhppa -o $1
```

| Compile and Link Options for bldf77 | |
|---|---|
| The script file contains the following compile options: | |
| `f77` | The Fortran compiler. |
| `-w` | Suppress warning messages. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| The script file contains the following link options: | |
| `f77` | Use the compiler to link edit. |
| `+DAportable` | Use code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `util.o` | Include the object file for error checking. |
| `-Wl,` | Instruct the compiler to pass the `-L`*path* directly to the linker. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| `-lhppa` | Specify the HP PA-RISC library (required). |
| `-o $1` | Specify the name of the object module. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqf`, enter:

```
bldf77 updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build Fortran applications that do not contain embedded SQL, you can use the script file `bldf77api`. It contains the same compile and link options as `bldf77`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link the DB2 API sample programs written in Fortran.

## Building Fortran Stored Procedures

The script file `bldf77sp`, in `sqllib/samples/fortran`, contains the commands to build a stored procedure. The script file compiles the stored procedure into a shared library on the server that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4`, specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, $1, for the shared library name.

```
#! /bin/ksh
# bldf77sp script file
# Builds a sample Fortran stored procedure
# Usage: bldf77sp <stored_proc_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqf bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the program.
f77 -w -n -c +DAportable +Z -I/opt/IBMdb2/V5.0/include $1.f -o $1.o

# Link the program.
ld -b -E -o $1 $1.o -L/opt/IBMdb2/V5.0/lib

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldf77sp | |
|---|---|
| The script file contains the following compile options: | |
| `f77` | The Fortran compiler. |
| `-w` | Suppress warning messages. |
| `-n` | Generate a shared object file. |
| `-c` | Perform compile only; no link. |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `+Z` | Generate position-independent code. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include`. |
| `-o $1` | Specify the name of the object module. |
| Refer to your compiler documentation for additional compiler options. | |
| The script file contains the following link options: | |
| `ld` | Use the linker to link edit. |
| `-b -E` | Specify the default export file for the stored procedure. |
| `-o $1` | Specify the name of the object module. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqf`, enter:

```
bldf77srv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldf77` script file. Refer to “HP Fortran/9000” on page 81 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Micro Focus COBOL

**Note:** This section documents the compile and link options recommended for DB2 programs using Micro Focus COBOL with HP-UX Version 10. The corresponding information for HP-UX Version 11 was not available at the time of this writing, and may vary. Please see the README file for additional information.

The script file `bldmfcc`, in `sqllib/samples/cobol_mf`, contains the commands to build a sample COBOL program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4`, specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldmfcc script file
# Builds a COBOL program containing embedded SQL
# Usage: bldmfcc <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqb bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$COBCPY:/opt/IBMdb2/V5.0/include/cobol_mf

# Compile the checkerr.cbl error checking utility.
cob -cx +DAportable checkerr.cbl

# Compile the program.
cob -cx +DAportable $1.cbl

# Link the program.
cob -x +DAportable $1.o checkerr.o -L/opt/IBMdb2/V5.0/lib -ldb2 -lhppa -ldb2gmf
```

| Compile and Link Options for bldmfcc | |
|---|---|
| The script file contains the following compile options: | |
| `cob` | The Micro Focus COBOL compiler. |
| `-cx` | Compile to object module. |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |

| Compile and Link Options for bldmfcc | |
|---|---|
| The script file contains the following link options: | |
| `cob` | Use the compiler to link edit. |
| `-x` | Specify an executable program. |
| `+DAportable` | Use code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `checkerr.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link to the DB2 library. |
| `-lhppa` | Specify the HP PA-RISC library (required). |
| `-ldb2gmf` | Link to the DB2 exception-handler library for Micro Focus COBOL. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqb`, enter:

```
bldmfcc updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build Micro Focus COBOL applications that do not contain embedded SQL, you can use the script file `bldmfapi`. It contains the same compile and link options as `bldmfcc`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link DB2 API sample programs written in COBOL.

## Building Micro Focus COBOL Stored Procedures

The script file `bldmfsp`, in `sqllib/samples/cobol_mf`, contains the commands to build a stored procedure. The script file compiles the stored procedure into a shared library on the server that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldmfsp script file
# Builds a COBOL stored procedure.
# Usage: bldmfsp <stored_proc_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqb bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$COBCPY:/opt/IBMdb2/V5.0/include/cobol_mf

# Compile the checkerr.cbl error checking utility.
cob +Z -cx +DAportable checkerr.cbl

# Compile the program.
cob +Z -cx +DAportable $1.cbl

# Link the program.
ld -b -o $1 $1.o -L/opt/IBMdb2/V5.0/lib -ldb2 -lhppa -ldb2gmf \
   -L$COBDIR/coblib -lcobol -lcrtn

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldmfsp | |
|---|---|
| The script file contains the following compile options: | |
| `cob` | The COBOL compiler. |
| `+Z` | Generate position-independent code. |
| `-cx` | Compile to object module. |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| The script file contains the following link options: | |
| `ld` | Use the linker to link edit. |
| `-b` | Create a shared library rather than a normal executable file. |
| `-o` | Produce an output object file. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link to the DB2 shared library. |
| `-lhppa` | Specify the HP PA-RISC library (required). |
| `-ldb2gmf` | Link to the DB2 exception-handler library for Micro Focus COBOL. |
| `-L`*path* | Specify the location of the COBOL runtime libraries. For example: `-L$COBDIR/coblib`. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqb`, enter:

```
bldmfsp outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldmfcc` script file. Refer to “Micro Focus COBOL” on page 85 for details.

To call the stored procedure, run the sample client application by entering:

```
outcli
```

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Exiting the Stored Procedure

When you develop your stored procedures, exit your stored procedure using the following statement:

```
move SQLZ-HOLD-PROC to return-code.
```

With this statement, the stored procedure returns correctly to the client application.

# Chapter 6. Building SCO UnixWare 7 Embedded SQL Applications

This chapter provides detailed information for building embedded SQL applications on SCO UnixWare 7. In the script files, commands that begin with `db2` are Command Line Processor (CLP) commands. Refer to the *Command Reference* if you need more information about CLP commands.

**General Points for Building and Running DB2 Programs**

1. You must build and run DB2 applications from a window where your environment variables are set. You can do this by running `db2profile`. Refer to "Setting Your Environment" on page 21 if you need more information.
2. To build DB2 programs containing embedded SQL, or to run any DB2 programs, the database manager on the server must be started. Start the database manager, if it is not already running, by entering the following command on the server:

   ```
   db2start
   ```

## SCO UnixWare 7 C

The script file `bldcc`, in `sqllib/samples/c`, contains the commands to build a sample C program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

© Copyright IBM Corp. 1993, 1998

```
#! /bin/ksh
# bldcc script file
# Builds a sample c program.
# Usage: bldcc <prog_name> [ <db_name> [ < userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi
# Precompile the program.
db2 prep $1.sqc bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
cc -Kthread -I/opt/IBMdb2/V5.0/include -c util.c
# Compile the program.
cc -Kthread -I/opt/IBMdb2/V5.0/include -c $1.c
# Link the program.
cc -o $1 $1.o util.o -Kthread -L/opt/IBMdb2/V5.0/lib -ldb2
```

**Compile and Link Options for bldcc**

The script file contains the following compile options:

| | |
|---|---|
| `cc` | The C compiler. |
| `-Kthread` | Use the compiler's multi-threaded facilities and arrange for the appropriate preprocessor flags to be turned on. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldcc | |
|---|---|
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `-o $1` | Specify the name of the object module. |
| `util.o` | Include the object file for error checking. |
| `-Kthread` | Link the threading library in the correct library order. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqc`, enter:

```
bldcc updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build C applications that do not contain embedded SQL, you can use the script file `bldccapi`. It contains the same compile and link options as `bldcc`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link the DB2 API sample programs written in C.

## Building C Stored Procedures

The script file `bldccsrv`, in `sqllib/samples/c`, contains the commands to build a C stored procedure. The script file compiles the stored procedure into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldccsrv script file
# Build sample c stored procedure.
# Usage: bldccsrv <prog_name> [ <db_name> [ < userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi
# Precompile the program.
db2 prep $1.sqc bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
cc -Kthread -I/opt/IBMdb2/V5.0/include -c util.c
# Compile the program.
cc -Kthread -I/opt/IBMdb2/V5.0/include -c $1.c
# Link the program and create a shared library
cc -G -o $1 $1.o -Kthread -L/opt/IBMdb2/V5.0/lib -ldb2
# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H= $DB2INSTANCE"
cp $1 $H/sqllib/function
```

**Compile and Link Options for bldccsrv**

The script file contains the following compile options:

| | |
|---|---|
| `cc` | The C compiler. |
| `-Kthread` | Use the compiler's multi-threaded facilities and arrange for the appropriate preprocessor flags to be turned on. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldccsrv | |
|---|---|
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `-G` | Generate a shared library. |
| `-o $1` | Specify the name of the object module. |
| `-Kthread` | Link the threading library in the correct library order. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqc`, enter:

```
bldccsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldcc` script file. Refer to "SCO UnixWare 7 C" on page 91 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
: Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Building C User-Defined Functions (UDFs)

The script file `bldccudf`, in `sqllib/samples/c`, contains the commands to build a UDF. UDFs are compiled like stored procedures, but you do not need to connect to a database or precompile and bind the program.

**Note:** A UDF does not contain embedded SQL statements. Rather, the application that uses the UDF contains the statements, such as `calludf`.

The first parameter, `$1`, specifies the name of your source file. The script file also uses this source file name for the shared library name.

```
#! /bin/ksh

# bldccudf script file
# Builds a C user-defined function library.
# Usage: bldccudf <prog_name>

# Compile the program.
cc -Kthread -I/opt/IBMdb2/V5.0/include -c $1.c
# Link the program and create a shared library.
cc -o $1 $1.o -G -Kthread -L/opt/IBMdb2/V5.0/lib -ldb2 -ldb2apie
# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H= $DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldccudf | |
|---|---|
| The script file contains the following compile options: | |
| `cc` | The C compiler. |
| `-Kthread` | Use the compiler's multi-threaded facilities and arrange for the appropriate preprocessor flags to be turned on. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldccudf | |
|---|---|
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `-o $1` | Specify the name of the object module. |
| `-G` | Generate a shared library. |
| `-Kthread` | Link the threading library in the correct library order. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| `-ldb2apie` | Link with the DB2 API Engine library to allow the use of LOB locators. |
| Refer to your compiler documentation for additional compiler options. | |

To build the user-defined function program `udf` from the source file `udf.c`, enter:

```
bldccudf udf
```

The script file copies the UDF to the server in the path `sqllib/function` to indicate that the UDF is fenced. If you want the UDF to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced UDF or stored procedure runs in the same address space as the database manager and results in increased performance when compared to a fenced UDF or stored procedure, which runs in an address space isolated from the database manager. With unfenced UDFs or stored procedures there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced UDFs or stored procedures when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced UDFs.

If necessary, set the file mode for the UDF so the DB2 instance can run it.

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build `calludf` using the `bldcc` script file. Refer to "SCO UnixWare 7 C" on page 91 for details.

To call the UDF, run the sample calling application by entering:

```
calludf
```

The calling application calls functions from the `udf` library.

## Multi-threaded Applications

DB2 C Applications for SCO UnixWare 7 should have the source module containing the `main()` function compiled and linked with `-Kthread` or the equivalent compiler and linker options. This will ensure that the thread library `libthread.so` will be linked and loaded in the correct order with respect to other libraries.

Multi-threaded DB2 applications created with SCO UnixWare 7 C must use the `-Kthread` option for compiling and linking all modules in addition to the one containing the `main()` function.

Posix thread support is not available on SCO UnixWare 7 at the time of this writing. SCO UnixWare 7 provides an implementation of the Unix International multi-thread APIs (as found on Solaris systems).

The makefile in `sqllib/samples/c` contains the commands to build a sample C multi-threaded program. The makefile compiles a multi-threaded program with the option `-DUSE_UI_THREADS`. This defines `USE_UI_THREADS`, used in the multi-threaded sample program `thdsrver` to include the `thread.h` and `synch.h` header files, instead of the header files for Posix threads.

The makefile also defines variables for the other compile and link options. When the makefile is run, the variables are replaced by their values, as in the following example where the makefile is used to compile the `thdsrver` program:

```
cc -o thdsrver thdsrver.c -I/home/db2inst/sqllib/include -Kthread
  -DUSE_UI_THREADS -L/home/db2inst/sqllib/lib -ldb2
```

where `/home/db2inst` is the DB2 instance directory.

For definitions of the non-multi-threaded options please see "Compile and Link options for bldcc" in “SCO UnixWare 7 C” on page 91.

To build the sample program, `thdsrver`, enter:

```
make thdsrver
```

The result is an executable file, `thdsrver`. To run the executable file against the `sample` database, enter:

```
thdsrver
```

## SCO UnixWare 7 C++

The script file `bldCC`, in `sqllib/samples/cpp`, contains the commands to build a sample C++ program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password.

Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldCC script file
# Builds a sample C++ program.
# Usage: bldCC <prog_name> [ <db_name> [ < userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi
# Precompile the program.
db2 prep $1.sqC bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
CC -Kthread -I/opt/IBMdb2/V5.0/include -c util.C
# Compile the program.
CC -Kthread -I/opt/IBMdb2/V5.0/include -c $1.C
# Link the program.
CC -o $1 $1.o util.o -Kthread -L/opt/IBMdb2/V5.0/lib -ldb2
```

**Compile and Link Options for bldCC**

The script file contains the following compile options:

| | |
|---|---|
| `CC` | The C++ compiler. |
| `-Kthread` | Use the compiler's multi-threaded facilities and arrange for the appropriate preprocessor flags to be turned on. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldCC | |
|---|---|
| The script file contains the following link options: | |
| `CC` | Use the compiler to link edit. |
| `-o $1` | Specify the name of the object module. |
| `util.o` | Include the object file for error checking. |
| `-Kthread` | Link the threading library in the correct library order. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqC`, enter:

```
bldCC updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

## Building C++ Stored Procedures

The script file, `bldCCsrv`, in `sqllib/samples/cpp`, contains the commands to build a C++ stored procedure. The script file compiles the stored procedure into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldCCsrv script file
# Builds a C++ stored procedure.
# Usage: bldCCsrv <prog_name> [ <db_name> [ < userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi
# Precompile the program.
db2 prep $1.sqC bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.C error-checking utility.
CC -Kthread -I/opt/IBMdb2/V5.0/include -c util.C
# Compile the program.
CC -Kthread -I/opt/IBMdb2/V5.0/include -c $1.C
# Link the program and create a shared library
CC -G -o $1 $1.o -Kthread -L/opt/IBMdb2/V5.0/lib -ldb2
# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H= $DB2INSTANCE"
cp $1 $H/sqllib/function
```

**Compile and Link Options for bldCCsrv**

The script file contains the following compile options:

| | |
|---|---|
| `CC` | The C++ compiler. |
| `-Kthread` | Use the compiler's multi-threaded facilities and arrange for the appropriate preprocessor flags to be turned on. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldCCsrv | |
|---|---|
| The script file contains the following link options: | |
| `CC` | Use the compiler to link edit. |
| `-G` | Generate a shared library. |
| `-o $1` | Specify the name of the object module. |
| `-Kthread` | Link the threading library in the correct library order. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqC`, enter:

```
bldCCsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldCC` script file. Refer to “SCO UnixWare 7 C++” on page 98 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Micro Focus COBOL

The script file `bldmfcc`, in `sqllib/samples/cobol_mf`, contains the commands to build a sample COBOL program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4`, specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldmfcc script file.
# Usage: bldmfcc <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqb bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=/opt/IBMdb2/V5.0/include/cobol_mf:$COBCPY

# Compile the checkerr.cbl error checking utility.
cob -cx checkerr.cbl

# Compile the program.
cob -cx $1.cbl

# Link the program.
cob -x $1.o checkerr.o -L/opt/IBMdb2/V5.0/lib -ldb2 -ldb2gmf -lthread
```

| Compile and Link Options for bldmfcc | |
|---|---|
| The script file contains the following compile options: | |
| `cob` | The Micro Focus COBOL compiler. |
| `-cx` | Compile to object module. |

| Compile and Link Options for bldmfcc | |
|---|---|
| The script file contains the following link options: | |
| `cob` | Use the compiler to link edit. |
| `-x` | Specify an executable program. |
| `checkerr.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| `-ldb2gmf` | Link with the DB2 exception-handler library for Micro Focus COBOL. |
| `-lthread` | Link with the thread library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqb`, enter:

```
bldmfcc updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build Micro Focus COBOL applications that do not contain embedded SQL, you can use the script file `bldmfapi`. It contains the same compile and link options as `bldmfcc`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link DB2 API sample programs written in COBOL.

# Chapter 7. Building Silicon Graphics IRIX Embedded SQL Applications

This chapter provides detailed information for building embedded SQL applications on Silicon Graphics IRIX. In the script files, commands that begin with `db2` are Command Line Processor (CLP) commands. Refer to the *Command Reference* if you need more information about CLP commands.

DB2 for Silicon Graphics IRIX is client-only. To run DB2 applications, and to build DB2 embedded SQL applications, you need to access a DB2 database on a server machine from your client machine. The server machine will be running a different operating system. See *Quick Beginnings for UNIX* for information on configuring client-to-server communication.

Also, since you will be accessing a database on the server from a remote client that is running on a different operating system, you need to bind the database utilities, including the DB2 CLI, to the database. See “Binding” on page 24 for more information.

**General Points for Building and Running DB2 Programs**

1. You must build and run DB2 applications on your client machine from a window where your environment variables are set. You can do this by running `db2profile`. Refer to “Setting Your Environment” on page 21 if you need more information.
2. To build DB2 programs containing embedded SQL, or to run any DB2 programs, the database manager on the server machine must be started, and be accessible to your client machine. Start the database manager, if it is not already running, by entering the following command on the server:

   ```
   db2start
   ```

## MIPSpro C

The script file `bldcc`, in `sqllib/samples/c`, contains the commands to build a sample C program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4`, specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

© Copyright IBM Corp. 1993, 1998

```
#! /bin/ksh
# bldcc script file
# Builds a sample C program containing embedded SQL
# Usage: bldcc <prog_name> [ <db_name> [ < userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi
# Precompile the program.
db2 prep $1.sqc bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
cc -n32 -I/opt/IBMdb2/V5.0/include -c util.c

# Compile the program.
cc -n32 -I/opt/IBMdb2/V5.0/include -c $1.c

# Link the program.
cc -n32 -o $1 $1.o util.o -L/opt/IBMdb2/V5.0/lib -rpath /opt/IBMdb2/V5.0/lib -lm -ldb2
```

**Compile and Link Options for bldcc**

The script file contains the following compile options:

| | |
|---|---|
| `cc` | Use the C compiler. |
| `-n32` | Generate an n32 object. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldcc | |
|---|---|
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `-n32` | Use the n32 object. |
| `-o $1` | Specify the name of the object module. |
| `util.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-rpath` *path* | Specify the location of the DB2 shared libraries at run-time. For example: `-rpath /opt/IBMdb2/V5.0/lib`. |
| `-lm` | Link with the math library. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqc`, connecting to the `sample` database on the server, enter:

`bldcc updat` *remote_database userid password*

where

*remote_database* Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The result is an executable file, `updat`. To run the executable file against the `sample` database, enter:

`updat` *userid password*

where

*userid* Is a valid user ID.

*password* Is a valid password.

**Note:** To build C applications that do not contain embedded SQL, you can use the script file `bldccapi`. It contains the same compile and link options as `bldcc`, but does not connect to, prep, bind, or disconnect from, the `sample` database. It is used to compile and link the DB2 API sample programs written in C.

## Building the C Client Application for Stored Procedures

Stored procedures are programs that access the database and return information to the client application. You compile and store stored procedures on the server. The server runs on another platform.

To build the stored procedure `outsrv` on a UNIX server, refer to the embedded SQL chapter for that platform in this book. If you are using an OS/2 or Windows server, refer to the embedded SQL chapter for that platform in *Building Applications for Windows and OS/2 Environments*.

Once you build the stored procedure `outsrv`, you can build the client application that calls the stored procedure, `outcli`, by using the `bldcc` script file. Refer to "MIPSpro C" on page 107 for details.

To call the stored procedure, run the client application by entering:

`outcli` *remote_database userid password*

where

*remote_database* Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value, and then returns the variable to the client application.

## Building the C Client Application for User-defined Functions (UDFs)

User-defined functions (UDFs) are your own scalar functions that you compile and store on the server. The server runs on another platform.

To build the user-defined function program, `udf`, on a UNIX server, refer to the embedded SQL chapter for that platform in this book. If you are using an OS/2 or Windows server, refer to the embedded SQL chapter for that platform in *Building Applications for Windows and OS/2 Environments*.

Once you build `udf`, you can build the client application `calludf` that calls it by using the `bldcc` script file. Refer to "MIPSpro C" on page 107 for details.

To call the UDF program, run the calling application by entering:

`calludf` *userid password*

where

*userid* Is a valid user ID.

*password* Is a valid password.

The calling application calls functions from the `udf` library.

## MIPSpro C++

The script file `bldCC`, in `sqllib/samples/cpp`, contains the commands to build a sample C++ program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4`, specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldCC script file
# Builds a sample C++ program containing embedded SQL
# Usage: bldCC <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi
# Precompile the program.
db2 prep $1.sqC bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
CC -n32 -I/opt/IBMdb2/V5.0/include -c util.C

# Compile the program.
CC -n32 -I/opt/IBMdb2/V5.0/include -c $1.C

# Link the program.
CC -n32 -o $1 $1.o util.o -L/opt/IBMdb2/V5.0/lib -rpath /opt/IBMdb2/V5.0/lib -lm -ldb2
```

| Compile and Link Options for bldCC | |
|---|---|
| The script file contains the following compile options: | |
| `CC` | Use the C++ compiler. |
| `-n32` | Generate an n32 object. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `CC` | Use the compiler to link edit. |
| `-n32` | Use the n32 object. |
| `-o $1` | Specify the name of the object module. |
| `util.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-rpath` *path* | Specify the location of the DB2 shared libraries at run-time. For example: `-rpath /opt/IBMdb2/V5.0/lib`. |
| `-lm` | Link with the math library. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqC`, connecting to the `sample` database on the server, enter:

`bldCC updat` *remote_database userid password*

where

*remote_database* Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The result is an executable file, `updat`. To run the executable file against the `sample` database, enter:

`updat` *userid password*

where

*userid* Is a valid user ID.

*password* Is a valid password.

## Building the C++ Client Application for Stored Procedures

Stored procedures are programs that access the database and return information to the client application. You compile and store stored procedures on the server. The server runs on another platform.

To build the stored procedure `outsrv` on a UNIX server, refer to the embedded SQL chapter for that platform in this book. If you are using an OS/2 or Windows server, refer to the embedded SQL chapter for that platform in *Building Applications for Windows and OS/2 Environments*.

Once you build the stored procedure `outsrv`, you can build the client application that calls the stored procedure, `outcli`, by using the `bldCC` script file. Refer to "MIPSpro C++" on page 111 for details.

To call the stored procedure, run the client application by entering: `outcli` *remote_database userid password*

where

*remote_database* Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value, and then returns the variable to the client application.

## MIPSpro Fortran-77

The script file, `bldf77`, in `sqllib/samples/fortran`, contains the commands to build a sample Fortran program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4`, specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldf77 script file
# Builds a Fortran program that contains embedded SQL
# Usage: bldf77 <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi
# Precompile the program.
db2 prep $1.sqf bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.f error-checking utility.
f77 -n32 -I/opt/IBMdb2/V5.0/include -w -c util.f
# Compile the program.
f77 -n32 -I/opt/IBMdb2/V5.0/include -w -c $1.f
# Link the program.
f77 -n32 -o $1 $1.o util.o -L/opt/IBMdb2/V5.0/lib -rpath /opt/IBMdb2/V5.0/lib -ldb2
```

**Compile and Link Options for bldf77**

The script file contains the following compile options:

| | |
|---|---|
| `f77` | The Fortran compiler. |
| `-n32` | Generate an n32 object. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include`. |
| `-w` | Suppress warning messages. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldf77 | |
|---|---|
| The script file contains the following link options: | |
| `f77` | Use the compiler to link edit. |
| `-n32` | Use the n32 object. |
| `-o $1` | Specify the name of the object module. |
| `util.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. |
| `-rpath` *path* | Specify the location of the DB2 shared libraries at run-time. For example: `-rpath /opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqf`, connecting to the `sample` database on the server, enter:

```
bldf77 updat remote_database userid password
```

where

*remote_database* Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The result is an executable file, `updat`. To run the executable file against the `sample` database, enter:

```
updat
```

## Building the Fortran Client Application for Stored Procedures

Stored procedures are programs that access the database and return information to the client application. You compile and store stored procedures on the server. The server runs on another platform.

To build the stored procedure `outsrv` on a UNIX server, refer to the embedded SQL chapter for that platform in this book. If you are using an OS/2 or Windows server, refer to the embedded SQL chapter for that platform in *Building Applications for Windows and OS/2 Environments*.

Once you build the stored procedure `outsrv`, you can build `outcli` that calls the stored procedure. You can build `outcli` using the `bldf77` script file. Refer to “MIPSpro Fortran-77” on page 113 for details.

To call the stored procedure, run the client application by entering:

| `outcli`

| The client application passes a variable to the server program, `outsrv`, which gives it a
| value and then returns the variable to the client application.

# Chapter 8. Building Solaris Embedded SQL Applications

This chapter provides detailed information for building embedded SQL applications on Solaris. In the script files, commands that begin with `db2` are Command Line Processor (CLP) commands. Refer to the *Command Reference* if you need more information about CLP commands.

**General Points for Building and Running DB2 Programs**

1. You must build and run DB2 applications from a window where your environment variables are set. You can do this by running `db2profile`. Refer to "Setting Your Environment" on page 21 if you need more information.
2. To build DB2 programs containing embedded SQL, or to run any DB2 programs, the database manager on the server must be started. Start the database manager, if it is not already running, by entering the following command on the server:

   ```
   db2start
   ```

## SPARCompiler C

**Note:** The compile and link steps in the script files in this section are for SPARCompiler C. They also contain, commented out, the compile and link steps for the IBM C compiler. To use the scripts with the IBM C compiler, just comment out the SPARCompiler compile and link steps and uncomment those for IBM C.

The script file, `bldcc`, in `sqllib/samples/c`, contains the commands to build a sample C program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

© Copyright IBM Corp. 1993, 1998

```
#! /bin/ksh
# bldcc script file
# Builds a sample c program.
# Usage:  bldcc <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqc bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
cc -I/opt/IBMdb2/V5.0/include -c util.c

# Compile the program.  (Using the SPARCompiler C compiler)
cc -I/opt/IBMdb2/V5.0/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2

# Using the IBM C compiler.
# Compile the util.c error-checking utility.
# xlc -I/opt/IBMdb2/V5.0/include -c util.c
# Compile the program.
# xlc -I/opt/IBMdb2/V5.0/include -c $1.c
# Link the program.
# xlc -o $1 $1.o util.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2
```

| Compile and Link Options for bldcc | |
|---|---|
| The script file contains the following compile options: | |
| `cc` | The C compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `-o $1` | Specify the name of the object module. |
| `util.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-R`*path* | Specify the location of the DB2 shared libraries at run-time. For example: `-R/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqc`, enter:

```
bldcc updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build C applications that do not contain embedded SQL, you can use the script file `bldccapi`. It contains the same compile and link options as `bldcc`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link the DB2 API sample programs written in C.

## Building C Stored Procedures

The script file, `bldccsrv`, in `sqllib/samples/c`, contains the commands to build a C stored procedure. The script file compiles the stored procedure into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldccsrv script file
# Build sample c stored procedure.
# Usage: bldccsrv <prog_name> [ <db_name> [ <userid> <password> ]]
# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi
# Precompile the program.
db2 prep $1.sqc bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
cc -Xa -misalign -Kpic -I/opt/IBMdb2/V5.0/include -c util.c

# Compile the program. (Using the SPARCompiler C compiler)
cc -Xa -misalign -Kpic -I/opt/IBMdb2/V5.0/include -c $1.c

# Link the program and create a shared library
cc -G -o $1 $1.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2

# Using the IBM C compiler.
# Compile the util.c error-checking utility.
# xlc -qmisalign -qpic=small -I/opt/IBMdb2/V5.0/include -c util.c
# Compile the program.
# xlc -qmisalign -qpic=small -I/opt/IBMdb2/V5.0/include -c $1.c
# Link the program and create a shared library
# xlc -G -o $1 $1.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

| Compile and Link Options for bldccsrv | |
|---|---|
| The script file contains the following compile options: | |
| `cc` | The C compiler. |
| `-Xa` | Compile assuming ANSI conformance. |
| `-misalign` | Allow loading and storage of misaligned data. Use only if your application uses misaligned data. |
| `-Kpic` | Generate position-independent code for shared libraries. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `-G` | Generate a shared library. |
| `-o $1` | Specify the name of the object module. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-R`*path* | Specify the location of the DB2 shared libraries at run-time. For example: `-R/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqc`, enter:

```
bldccsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldcc` script file. Refer to "SPARCompiler C" on page 117 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
: Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid*
: Is a valid user ID.

*password*
: Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Building C User-Defined Functions (UDFs)

The script file, `bldccudf`, in `sqllib/samples/c`, contains the commands to build a UDF. UDFs are compiled like stored procedures, but you do not need to connect to a database or precompile and bind the program.

**Note:** A UDF does not contain embedded SQL statements. Rather, the application that uses the UDF contains the statements, such as `calludf`.

The first parameter, `$1`, specifies the name of your source file. The script file also uses this source file name for the shared library name.

```
#! /bin/ksh
# bldccudf script file
# Builds a C user-defined function library.
# Usage: bldccudf <prog_name>

# Compile the program. (Using the SPARCompiler C compiler)
cc -Xa -misalign -Kpic -I/opt/IBMdb2/V5.0/include -c $1.c

# Link the program and create a shared library.
cc -o $1 $1.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2 -ldb2apie -G

# Using the IBM C compiler.
# Compile the program.
# xlc -qmisalign -qpic=small -I/opt/IBMdb2/V5.0/include -c $1.c
# Link the program and create a shared library.
# xlc -o $1 $1.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2 -ldb2apie -G

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

**Compile and Link Options for bldccudf**

The script file contains the following compile options:

| | |
|---|---|
| `cc` | The C compiler. |
| `-Xa` | Compile assuming ANSI conformance. |
| `-misalign` | Allow loading and storage of misaligned data. Use only if your application uses misaligned data. |
| `-Kpic` | Generate position-independent code for shared libraries. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include`. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldccudf | |
|---|---|
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `-o $1` | Specify the name of the object module. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-R`*path* | Specify the location of the DB2 shared libraries at run-time. For example: `-R/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| `-ldb2apie` | Link with the DB2 API Engine library to allow the use of LOB locators. |
| `-G` | Generate a shared library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the user-defined function program `udf` from the source file `udf.c`, enter:

```
bldccudf udf
```

The script file copies the UDF to the server in the path `sqllib/function` to indicate that the UDF is fenced. If you want the UDF to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced UDF or stored procedure runs in the same address space as the database manager and results in increased performance when compared to a fenced UDF or stored procedure, which runs in an address space isolated from the database manager. With unfenced UDFs or stored procedures there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced UDFs or stored procedures when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced UDFs.

If necessary, set the file mode for the UDF so the DB2 instance can run it.

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build `calludf` using the `bldcc` script file. Refer to "SPARCompiler C" on page 117 for details.

To call the UDF, run the sample calling application by entering:

```
calludf
```

The calling application calls functions from the `udf` library.

## Multi-threaded Applications

Multi-threaded applications using SPARCompiler C on Solaris need to be compiled with `-mt`. This will pass `-D_REENTRANT` to the preprocessor, and `-lthread` to the linker. Posix threads also require `-lpthread` to be passed to the linker. In addition, using the compiler option `-D_POSIX_PTHREAD_SEMANTICS` allows posix variants of functions such as `getpwnam_r()`.

The makefile in `sqllib/samples/c` contains the commands to build a sample C multi-threaded program. The makefile defines variables for the compile and link options. When the makefile is run, the variables are replaced by their values, as in the following example where the makefile is used to compile the `thdsrver` program:

```
cc -o thdsrver thdsrver.c -I/home/db2inst/sqllib/include -mt
  -D_POSIX_PTHREAD_SEMANTICS -L/home/db2inst/sqllib/lib
  -R/home/db2inst/sqllib/lib -ldb2 -lpthread
```

where `/home/db2inst` is the DB2 instance directory.

For definitions of the non-multi-threaded options please see "Compile and Link options for bldcc" in “SPARCompiler C” on page 117.

To build the sample program, `thdsrver`, enter:

```
make thdsrver
```

The result is an executable file, `thdsrver`. To run the executable file against the `sample` database, enter:

```
thdsrver
```

## SPARCompiler C++

**Notes:**

1. The compile and link steps in the script files in this section are for SPARCompiler C++. They also contain, commented out, the compile and link steps for the IBM C Set++ compiler. To use the scripts with C Set++, just comment out the SPARCompiler compile and link steps and uncomment those for C Set++.
2. DB2 for Solaris does not support stored procedures or UDFs written in C++ and compiled with the IBM C Set++ compiler. For best results, stored procedures and UDFs written in C++ should be compiled with the SPARCompiler C++ version 4.2 compiler.

The script file, `bldCC`, in `sqllib/samples/cpp`, contains the commands to build a sample C++ program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password.

Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldCC script file
# Builds a C++ program.
# Usage:  bldCC <prog_name> [ <db_name> [ <userid> <password> ]]
# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqC bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.C error-checking utility.
CC -I/opt/IBMdb2/V5.0/include -c util.C
# Compile the program.  (Using the SPARCompiler C++ compiler)
CC -I/opt/IBMdb2/V5.0/include -c $1.C
# Link the program.
CC -o $1 $1.o util.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2

# Using the IBM C Set++ Compiler.
# Compile the util.C error-checking utility.
# xlC -I/opt/IBMdb2/V5.0/include -c util.C
# Compile the program.
# xlC -I/opt/IBMdb2/V5.0/include -c $1.C
# Link the program.
# xlC -o $1 $1.o util.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2
```

**Compile and Link Options for bldCC**

The script file contains the following compile options:

| | |
|---|---|
| `CC` | The C++ compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldCC | |
|---|---|
| The script file contains the following link options: | |
| `CC` | Use the compiler to link edit. |
| `-o $1` | Specify the name of the object module. |
| `util.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-R`*path* | Specify the location of the DB2 shared libraries at run-time. For example: `-R/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqC`, enter:

```
bldCC updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

## Building C++ Stored Procedures

The script file `bldCCsrv`, in `sqllib/samples/cpp`, contains the commands to build a C++ stored procedure. The script file compiles the stored procedure into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldCCsrv script file
# Builds a C++ stored procedure.
# Usage: bldCCsrv <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqC bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
db2 connect reset

# Compile the util.C error-checking utility.
CC -misalign -Kpic -I/opt/IBMdb2/V5.0/include -c util.C
# Compile the program. (Using the SPARCompiler C++ compiler)
# Ensure the program is coded with extern "C".
CC -misalign -Kpic -I/opt/IBMdb2/V5.0/include -c $1.C
# Link the program and create a shared library
CC -G -o $1 $1.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H= $DB2INSTANCE"
cp $1 $H/sqllib/function
```

**Compile and Link Options for bldCCsrv**

The script file contains the following compile options:

| | |
|---|---|
| `CC` | The C++ compiler. |
| `-misalign` | Allow loading and storage of misaligned data. Use only if your application uses misaligned data. |
| `-Kpic` | Generate position-independent code for shared libraries. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldCCsrv | |
|---|---|
| The script file contains the following link options: | |
| CC | Use the compiler to link edit. |
| -G | Generate a shared library. |
| -o $1 | Specify the name of the object module. |
| -L*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: -L/opt/IBMdb2/V5.0/lib. If you do not specify the -L option, /usr/lib:/lib is assumed. |
| -R*path* | Specify the location of the DB2 shared libraries at run-time. For example: -R/opt/IBMdb2/V5.0/lib. |
| -ldb2 | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `outsrv` from the source file `outsrv.sqC`, enter:

```
bldCCsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldCC` script file. Refer to "SPARCompiler C++" on page 125 for details.

To call the stored procedure, run the sample client application by entering:

`outcli` *remote_database userid password*

where

*remote_database*
: Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

*userid* Is a valid user ID.

*password* Is a valid password.

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## SPARCompiler Fortran

The script file `bldf77`, in `sqllib/samples/fortran`, contains the commands to build a sample Fortran program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldf77 script file
# Builds a Fortran program that contains embedded SQL
# Usage: bldf77 <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqf bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Compile the util.f error-checking utility.
f77 -I/opt/IBMdb2/V5.0/include -w -c util.f

# Compile the program.
f77 -I/opt/IBMdb2/V5.0/include -w -c $1.f

# Link the program.
f77 $1.o util.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2 -o $1
```

| Compile and Link Options for bldf77 | |
|---|---|
| The script file contains the following compile options: | |
| `f77` | The Fortran compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include`. |
| `-w` | Suppress warning messages. |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |

| Compile and Link Options for bldf77 | |
|---|---|
| The script file contains the following link options: | |
| `f77` | Use the compiler to link edit. |
| `util.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. |
| `-R`*path* | Specify the library search path for the dynamic library. For example: `-R/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| `-o $1` | Specify the name of the object module. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqf`, enter:

```
bldf77 updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build Fortran applications that do not contain embedded SQL, you can use the script file, `bldf77api`. It contains the same compile and link options as `bldf77`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link the DB2 API sample programs written in Fortran.

## Building Fortran Stored Procedures

The script file `bldf77sp`, in `sqllib/samples/fortran`, contains the commands to build a stored procedure. The script file compiles the stored procedure into a shared library on the server that can be called by the client application.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldf77sp script file
# Builds a Fortran stored procedure
# Usage: bldf77 <stored_proc_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqf bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Build the stored procedure.
f77 -I/opt/IBMdb2/V5.0/include -w -G $1.f -o $1

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
eval "H=~$DB2INSTANCE"
cp $1 $H/sqllib/function
```

**Compile Options for bldf77sp**

The script file contains the following compile options:

| | |
|---|---|
| `f77` | The Fortran compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include`. |
| `-w` | Suppress warning messages. |
| `-G` | Generate a shared library. |
| `-o $1` | Specify the name of the object module. |

Refer to your compiler documentation for additional compiler options.

To build the sample program `outsrv` from the source file `outsrv.sqf`, enter:

```
bldf77sp outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. These paths are in the home directory of the DB2 instance.

**Note:** An unfenced stored procedure or UDF runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure or UDF, which runs in an address space isolated from the database manager. With unfenced stored procedures or UDFs there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures or UDFs when you need to maximize the performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Embedded SQL Programming Guide* for more information about fenced and unfenced stored procedures.

If necessary, set the file mode for the stored procedure so the DB2 instance can run it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the `bldf77` script file. Refer to "SPARCompiler Fortran" on page 130 for details.

To call the stored procedure, run the sample client application by entering:

```
outcli
```

The client application passes a variable to the server program `outsrv`, which gives it a value and then returns the variable to the client application.

## Micro Focus COBOL

The script file `bldmfcc`, in `sqllib/samples/cobol_mf`, contains the commands to build a sample COBOL program.

The first parameter, `$1`, specifies the name of your source file. The second parameter, `$2`, specifies the name of the database to which you want to connect. The third parameter, `$3`, specifies the user ID for the database, and `$4`, specifies the password. Only the first parameter, the source file name, is required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldmfcc script file.
# Usage: bldmfcc <prog_name> [ <db_name> [ <userid> <password> ]]

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

# Precompile the program.
db2 prep $1.sqb bindfile

# Bind the program to the database.
db2 bind $1.bnd

# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=/opt/IBMdb2/V5.0/include/cobol_mf:$COBCPY

# Compile the checkerr.cbl error checking utility.
cob -cx checkerr.cbl

# Compile the program.
cob -cx $1.cbl

# Link the program.
cob -x $1.o checkerr.o -L/opt/IBMdb2/V5.0/lib -ldb2 -ldb2gmf
```

| Compile and Link Options for bldmfcc | |
|---|---|
| The script file contains the following compile options: | |
| `cob` | The Micro Focus COBOL compiler. |
| `-cx` | Compile to object module. |

| Compile and Link Options for bldmfcc | |
|---|---|
| The script file contains the following link options: | |
| `cob` | Use the compiler to link edit. |
| `-x` | Specify an executable program. |
| `checkerr.o` | Include the object file for error checking. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| `-ldb2gmf` | Link with the DB2 exception-handler library for Micro Focus COBOL. |
| Refer to your compiler documentation for additional compiler options. | |

To build the sample program `updat` from the source file `updat.sqb`, enter:

```
bldmfcc updat
```

The result is an executable file `updat`. You can run the executable file against the `sample` database by entering:

```
updat
```

**Note:** To build Micro Focus COBOL applications that do not contain embedded SQL, you can use the script file `bldmfapi`. It contains the same compile and link options as `bldmfcc`, but does not connect, prep, bind, or disconnect from the `sample` database. It is used to compile and link DB2 API sample programs written in COBOL.

# Chapter 9. Building DB2 Call Level Interface (CLI) Applications

The DB2 SDK comes with sample programs that use DB2 Call Level Interface (DB2 CLI) function calls. You can study the samples to learn how to access DB2 databases using these function calls in your applications. You can also create programs that call stored procedures using DB2 CLI. For information on DB2 CLI stored procedures refer to the *CLI Guide and Reference*.

This chapter shows you how to build and run a sample program using a script file we supply. The script file shows you the compiler options you can use. It builds the sample program by compiling and linking the source file.

The sample programs and a makefile are contained in the `sqllib/samples/cli` directory. You can build the sample programs using the `make` facility. See the README file in `sqllib/samples/cli` for details about using the `makefile`, and for more information about the sample programs. You may need to modify the compiler options in the script file and the `makefile` for your environment.

Once you have compiled and run the supplied sample programs you can modify the source files, and the makefile, for your own needs. You can then build the modified sample programs using the `makefile` to see if they work correctly. You can also build your own programs using the `makefile`. All the sample programs are listed in Table 9 on page 17.

**Note:** It is recommended that, before you alter or build the sample programs, you copy them from `sqllib/samples/cli` to your own working directory.

**General Points for Building and Running DB2 CLI Programs**

1. You must build and run DB2 CLI applications from a window where your environment variables are set. You can do this by running `db2profile`. Refer to "Setting Your Environment" on page 21 if you need more information.
2. To run DB2 CLI programs, the database manager on the server must be started. Start the database manager, if it is not already running, by entering the following command on the server:

   ```
   db2start
   ```

## Compiling and Linking by Platform

The script file `clibld` contains the commands to build the sample DB2 CLI program `clisampl.c`. You can find both the script file and `clisampl.c` in `sqllib/samples/cli`.

Study the script file's compile and link options for the platform you are using. Then go to "Building and Running a CLI Program" on page 143 for the steps to follow in order to build and run the program.

© Copyright IBM Corp. 1993, 1998

### AIX

IBM XL C is used in the following version of the `clibld` script file:

```
#! /bin/ksh
# clibld script file -- AIX
# Build clisampl

# Compile the program.
xlc -I/usr/lpp/db2_05_00/include -c clisampl.c

# Compile the common utility functions used by most CLI sample programs
xlc -I/usr/lpp/db2_05_00/include -c samputil.c

# Link the program.
xlc -o clisampl clisampl.o samputil.o -L/usr/lpp/db2_05_00/lib -ldb2
```

| Compile and Link Options for clibld | |
|---|---|
| The script file contains the following compile options: | |
| `xlc` | The IBM XL C compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/lpp/db2_05_00/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `xlc` | Use the compiler to link edit. |
| `-o` *filename* | Specify the name of the executable program. |
| `samputil.o` | Include the utility object file for error checking. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. For example: `-L/usr/lpp/db2_05_00/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. |
| `-ldb2` | Link with the database manager library. |
| Refer to your compiler documentation for additional compiler options. | |

**Note:** Multi-threaded applications on AIX Version 4 need to be compiled and linked with the `xlc_r` compiler instead of the `xlc` compiler, or with the `xlC_r` compiler instead of the `xlC` compiler. See “Multi-threaded Applications” on page 43 for more information on building C multi-threaded applications.

### HP-UX

HP-UX C is used in the following version of the `clibld` script file:

```
#! /bin/ksh
# clibld script file -- HP-UX
# Build clisampl

# Compile the program.
cc -Aa +DAportable +e -I/opt/IBMdb2/V5.0/include -c clisampl.c

# Compile the common utility functions used by most CLI sample programs
cc -Aa +DAportable +e -I/opt/IBMdb2/V5.0/include -c samputil.c

# Link the program.
cc +DAportable -o clisampl clisampl.o samputil.o -L/opt/IBMdb2/V5.0/lib -ldb2 -lhppa
```

| Compile and Link Options for clibld | |
|---|---|
| The script file contains the following compile options: | |
| `cc` | Use the C compiler. |
| `-Aa` | Use ANSI standard mode. |
| `+DAportable` | Generate code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `+e` | Enables HP value-added features while compiling in ANSI C mode. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `+DAportable` | Use code compatible across PA_RISC 1.1 and 2.0 workstations and servers. |
| `-o` *filename* | Specify the name of the executable program. |
| `samputil.o` | Include the utility object file for error checking. |
| `-L`*path* | Specify the location of the DB2 runtime shared libraries. |
| `-ldb2` | Link with the database manager library. |
| `-lhppa` | Specify the HP PA-RISC library (required). |
| Refer to your compiler documentation for additional compiler options. | |

**Note:** Multi-threaded applications on HP-UX version 11 need to have _REENTRANT defined for their compilation. The HP-UX documentation recommends compiling with -D_POSIX_C_SOURCE=199506L. This will also ensure _REENTRANT is defined. Applications also need to be linked with `-lpthread`. Multi-threaded applications are not supported by DB2 on HP-UX version 10. See "Multi-threaded Applications" on page 76 for more information on building C multi-threaded applications.

## SCO UnixWare 7

SCO UnixWare 7 Optimizing C Compilation System is used in the following version of the `clibld` script file:

```
#! /bin/ksh
# clibld script file
# Build clisampl

# Compile the program.
cc -Kthread -I/opt/IBMdb2/V5.0/include -c clisampl.c

# Compile the common utility functions used by most CLI sample programs
cc -Kthread -I/opt/IBMdb2/V5.0/include -c samputil.c

# Link the program.
cc -o clisampl clisampl.o samputil.o -Kthread -L/opt/IBMdb2/V5.0/lib -ldb2
```

| Compile and Link Options for clibld | |
|---|---|
| The script file contains the following compile options: | |
| `cc` | The C compiler. |
| `-Kthread` | Use the compiler's multi-threaded facilities and arrange for the appropriate preprocessor flags to be turned on. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `-o $1` | Specify the name of the object module. |
| `samputil.o` | Include the utility object file for error checking. |
| `-Kthread` | Link the threading library in the correct library order. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

**Note:** SCO UnixWare 7 C multi-threaded applications use Unix International threads APIs, which require that the `thread.h` and `synch.h` header files be included in the program compilation. See “Multi-threaded Applications” on page 98 for more information on building C multi-threaded applications.

## Silicon Graphics IRIX

DB2 for Silicon Graphics IRIX is client-only. To run DB2 applications, you need to access a DB2 database on a server machine from your client machine. The server machine will be running a different operating system. See *Quick Beginnings for UNIX* for information on configuring client-to-server communication.

Also, since you will be accessing a database on the server from a remote client that is running on a different operating system, you need to bind the database utilities, including the DB2 CLI, to the database. See "Binding" on page 24 for more information.

The MIPSpro C compiler is used in the following version of the `clibld` script file:

```
#! /bin/ksh
# clibld script file
# Build clisampl

# Compile the program.
cc -I/opt/IBMdb2/V5.0/include -c clisampl.c

# Compile the common utility functions used by most CLI sample programs
cc -I/opt/IBMdb2/V5.0/include -c samputil.c

# Link the program.
ld -o clisampl clisampl.o samputil.o -L/opt/IBMdb2/V5.0/lib
  -rpath /opt/IBMdb2/V5.0/lib -ldb2
```

| Compile and Link Options for clibld | |
|---|---|
| The script file contains the following compile options: | |
| `cc` | Use the C compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/opt/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `ld` | Use the linker to link edit. |
| `-o $1` | Specify the name of the object module. |
| `samputil.o` | Include the utility object file for error checking. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. For example: `-L/opt/IBMdb2/V5.0/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. |
| `-rpath` *path* | Specify the location of the DB2 shared libraries at run-time. For example: `-rpath /opt/IBMdb2/V5.0/lib`. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

## Solaris

SPARCompiler C is used in the following version of the `clibld` script file:

```
#! /bin/ksh
# clibld script file -- Solaris
# Build clisampl

# Compile the program.
cc -I/opt/IBMdb2/V5.0/include -c clisampl.c

# Compile the common utility functions used by most CLI sample programs
cc -I/opt/IBMdb2/V5.0/include -c samputil.c

# Link the program.
cc -o clisampl clisampl.o samputil.o -L/opt/IBMdb2/V5.0/lib -R/opt/IBMdb2/V5.0/lib -ldb2
```

| Compile and Link Options for clibld | |
|---|---|
| The script file contains the following compile options: | |
| `cc` | Use the C compiler. |
| `-I`*path* | Specify the location of the DB2 include files. For example: `-I/usr/IBMdb2/V5.0/include` |
| `-c` | Perform compile only; no link. This book assumes that compile and link are separate steps. |
| The script file contains the following link options: | |
| `cc` | Use the compiler to link edit. |
| `-o` *filename* | Specify the name of the executable program. |
| `samputil.o` | Include the utility object file for error checking. |
| `-L`*path* | Specify the location of the DB2 static and shared libraries at link-time. |
| `-R`*path* | Specify the location of the DB2 shared libraries at run-time. |
| `-ldb2` | Link with the DB2 library. |
| Refer to your compiler documentation for additional compiler options. | |

**Note:** Multi-threaded applications using SPARCompiler C on Solaris need to be compiled with `-mt`. This will pass `-D_REENTRANT` to the preprocessor, and `-lthread` to the linker. Posix threads also require `-lpthread` to be passed to the linker. In addition, using the compiler option `-D_POSIX_PTHREAD_SEMANTICS` allows posix variants of functions such as `getpwnam_r()`. See “Multi-threaded Applications” on page 125 for more information on building C multi-threaded applications.

## Building and Running a CLI Program

To build the sample program `clisampl`, enter:

```
clibld
```

The result is an executable file, `clisampl`. The sample program accepts command line arguments for a database, user ID, and password so you can connect to any database to which you have access.

To run the sample program, enter:

`clisampl` *database userid password*

where

| | |
|---|---|
| *database* | Is the name of a cataloged database. |
| *userid* | Is a user ID that has SYSADM authority. |
| *password* | Is a valid password. |

If you need information about cataloging databases, or about SYSADM authority and passwords, refer to the *Quick Beginnings for UNIX* book.

The `clisampl` program performs the following SQL operations using DB2 CLI function calls:

1. Connects to a database.
2. Creates a table.
3. Inserts data into the table using a parameter marker.
4. Selects the data.
5. Drops the table.
6. Disconnects from the database.

You should see the following output:

```
Connecting
Create table - CREATE TABLE CLISAMPL (COL1 VARCHAR(50))
Insert - INSERT INTO CLISAMPL VALUES (?)
Select - SELECT * FROM CLISAMPL
Number of columns - 1
Column name - COL1
Column type - 12
Column precision - 50
Column scale - 0
Column nullable - TRUE
Column value - Row 1
Column value - Row 2
Disconnecting
Exiting program
```

# Chapter 10. Building Java Applications and Applets

You can develop Java programs to access DB2 databases with the appropriate Java Development Kit (JDK) on AIX, HP-UX, SCO UnixWare 7, Silicon Graphics IRIX, or Solaris. The JDK includes Java Database Connectivity (JDBC), a dynamic SQL API for Java.

DB2 JDBC support is provided by the DB2 Client Application Enabler (DB2 CAE). With this support you can build and run JDBC applications and applets. These contain dynamic SQL only, and use a Java call interface to pass SQL statements to DB2.

The DB2 Software Developer's Kit (DB2 SDK) provides support for Java embedded SQL (SQLJ). With DB2 SQLJ support and DB2 JDBC support you can build and run SQLJ applications and applets. These contain static SQL and use embedded SQL statements that are bound to a DB2 database.

The SQLJ support provided by the DB2 SDK includes:

- The DB2 SQLJ translator, `sqlj`, which replaces embedded SQL statements in the SQLJ program with Java source statements, and generates a serialized profile which contains information about the SQL operations found in the SQLJ program. The SQLJ translator uses the `sqllib/java/sqlj.zip` file.
- The DB2 SQLJ profile customizer, `db2profc`, which precompiles the SQL statements stored in the generated profile, customizing them into calls to the SQLJ runtime function, and generates a package in the DB2 database. For more information on the `db2profc` command, see the *Command Reference*.
- The DB2 SQLJ runtime function, which provides a runtime interface to the DB2 database manager. It uses the `sqllib/java/runtime.zip` file.

Building and running different types of Java programs requires support from different components of DB2:

- To build JDBC applications requires the DB2 Client Application Enabler (DB2 CAE). To run JDBC applications requires the DB2 CAE in order to connect to DB2.
- To build SQLJ applications requires the DB2 CAE and the DB2 SDK. To run SQLJ applications requires the DB2 CAE in order to connect to DB2.
- To build JDBC applets requires the DB2 CAE. No DB2 component is required to run JDBC applets on a client machine.
- To build SQLJ applets requires the DB2 CAE and the DB2 SDK. No DB2 component is required to run SQLJ applets on a client machine.

For more information on DB2 programming in Java, refer to the *Embedded SQL Programming Guide*, chapter 15, "Programming in Java". This covers creating and running JDBC applications, applets, stored procedures and UDFs. Information on SQLJ applications, applets, stored procedures and UDFs can be found in the *What's New* book.

© Copyright IBM Corp. 1993, 1998

For the latest, updated DB2 Java information, visit the Web Page at http://www.software.ibm.com/data/db2/java.

## Setting the Environment

### AIX

To build Java applications on AIX with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The Java Development Kit (JDK) Version 1.1 for AIX from IBM (refer to http://www.software.ibm.com/data/db2/java).
2. The DB2 Client Application Enabler for AIX from the DB2 Client Pack. It must be Version 2.1.2 or later.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where /home/db2inst/jdk11 is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the JDK11_PATH field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to pipe the output to a file for easier viewing. The JDK11_PATH field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run Java programs on AIX with DB2 JDBC support, the following environment variables are automatically updated during DB2 instance creation to ensure that:

- CLASSPATH includes "." and the file sqllib/java/db2java.zip
- PATH includes the directory sqllib/bin
- LIBPATH includes the directory sqllib/lib

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

### HP-UX

To build Java applications on HP-UX with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The HP-UX Developer's Kit for Java Release 1.1 from Hewlett-Packard (refer to http://www.software.ibm.com/data/db2/java).
2. The DB2 Client Application Enabler for HP-UX from the DB2 Client Pack. It must be Version 2.1.2 or later.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where /home/db2inst/jdk11 is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the JDK11_PATH field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to pipe the output to a file for easier viewing. The JDK11_PATH field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run Java programs on HP-UX with DB2 JDBC support, the following environment variables are automatically updated during DB2 instance creation to ensure that:

- CLASSPATH includes "." and the file sqllib/java/db2java.zip
- PATH includes the directory sqllib/bin
- SHLIB_PATH includes the directory sqllib/lib

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

## SCO UnixWare 7

To build Java applications on SCO UnixWare 7 with DB2 JDBC support, you need to install and configure the DB2 Client Application Enabler for SCO UnixWare 7 from the DB2 Client Pack. It must be Version 5.2 or later.

SCO UnixWare 7 ships with Java Development Kit Version 1.1.3 installed in the /usr/java directory. You may have to configure it for your environment.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where /home/db2inst/jdk11 is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the JDK11_PATH field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to pipe the output to a file for easier viewing. The JDK11_PATH field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run Java programs on SCO UnixWare 7 with DB2 JDBC support, the following environment variables are automatically updated during DB2 instance creation to ensure that:

- CLASSPATH includes "." and the file sqllib/java/db2java.zip
- PATH includes the directory sqllib/bin
- LD_LIBRARY_PATH includes the directory sqllib/lib

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

## Silicon Graphics IRIX

To build Java applications on Silicon Graphics IRIX with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The Java Development Environment 3.1 (Sun JDK 1.1.5) and the Java Execution Environment 3.1 (Sun JRE 1.1.5) from Silicon Graphics, Inc. (refer to http://www.software.ibm.com/data/db2/java).
2. The DB2 Client Application Enabler for Silicon Graphics IRIX from the DB2 Client Pack. It must be Version 2.1.2 or later.

DB2 for Silicon Graphics IRIX is client-only. To run DB2 applications and applets, and to build DB2 embedded SQL applications and applets, you need to access a DB2 database on a server machine from your client machine. The server machine will be running a different operating system. See *Quick Beginnings for UNIX* for information on configuring client-to-server communication.

Also, since you will be accessing a database on the server from a remote client that is running on a different operating system, you need to bind the database utilities, including the DB2 CLI, to the database. See “Binding” on page 24 for more information.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where /home/db2inst/jdk11 is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the JDK11_PATH field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to pipe the output to a file for easier viewing. The JDK11_PATH field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run Java programs on Silicon Graphics IRIX with DB2 JDBC support, the following environment variables are automatically updated during DB2 instance creation to ensure that:

- CLASSPATH includes "." and the file sqllib/java/db2java.zip
- PATH includes the directory sqllib/bin
- LD_LIBRARY_PATH includes the directory sqllib/lib

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

## Solaris

To build Java applications on Solaris with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The Java Development Kit (JDK) Version 1.1.4 for Solaris, and the Solaris Native Thread pack, from Sun Microsystems (refer to http://www.software.ibm.com/data/db2/java).
2. The DB2 Client Application Enabler for Solaris from the DB2 Client Pack. It must be Version 2.1.2 or later.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where /home/db2inst/jdk11 is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the JDK11_PATH field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to pipe the output to a file for easier viewing. The JDK11_PATH field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

**Note:** On Solaris, some Java Virtual Machine implementations do not work well in programs that run in a "setuid" environment. The shared library that contains the

Java interpreter, `libjava.so`, may fail to load. As a workaround, you can create symbolic links for all needed JVM shared libraries in `/usr/lib`, with a command similar to the following (and depending on where Java is installed on your machine):

```
ln -s /opt/jdk1.1.3/lib/sparc/native_threads/*.so /usr/lib
```

For more information on this and other workarounds available, please visit:

```
http://www.software.ibm.com/data/db2/java/v5/faq.html
```

To run Java programs on Solaris with DB2 JDBC support, the following environment variables are automatically updated during DB2 instance creation to ensure that:

- CLASSPATH includes "." and the file `sqllib/java/db2java.zip`
- PATH includes the directory `sqllib/bin`
- LD_LIBRARY_PATH includes the directory `sqllib/lib`
- THREADS_FLAG is set to "native"

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

## Java Sample Programs

DB2 provides sample programs, used in the following sections, to demonstrate building and running JDBC programs that exclusively use dynamic SQL, and SQLJ programs that use static SQL. The Java samples are located in the `sqllib/samples/java` directory. The directory also contains a `README` and a `makefile`. Please see the section “The Java makefile” on page 151.

Before modifying or building the sample programs, it is recommended that you copy them from the `sqllib/samples/java` directory to a separate working directory.

To run these sample programs, you must first create and populate the `sample` database by entering:

```
db2sampl
```

### General Points for Building and Running DB2 Java Programs

1. You must build and run DB2 Java applications and applets from a window where your environment variables are set. You can do this by running `db2profile`. Refer to “Setting Your Environment” on page 21 if you need more information.
2. To build DB2 SQLJ programs, or to run any DB2 Java programs, the database manager on the server must be started. Start the database manager, if it is not already running, by entering the following command on the server:

```
db2start
```

## The Java makefile

The `makefile` provided for the Java sample programs is presented below. The `makefile` will only work if a compatible make executable program is resident on your system in a directory included in your PATH variable. A suitable make utility may be provided by another language compiler. Please read the comment at the beginning of the text of the `makefile` for more information.

The make commands used to build specific Java sample programs are cited in the sections that follow. There is one change from makefiles normally used for other languages. The `make clean` command removes all files produced by the java compiler: `.java` files and `core` dumps; the `make cleanall` command removes these files as well as any files produced by the SQLJ translator.

```
# IBM DB2 Universal Database Version 5
# For AIX, HP-UX, SCO UnixWare 7, Silicon Graphics IRIX, and Solaris
# Makefile for DB2 java samples

# This makefile will only work if a compatible make executable program is
# resident on your system in a directory included in your PATH variable.
# Such a make utility may be provided by another language compiler. If you
# do not have a compatible make utility you cannot use this makefile, but
# you can still compile and run these programs by entering the commands
# at the command line, as explained in the README.

# To build your applications using this makefile, you can use one of
# the following commands:

#   make all         - builds all the programs in this directory
#   make <prog_name> - builds a program designated by <prog_name>
#   make clean       - removes all files produced by the java compiler
#                      from your working directory (such as .class files)
#   make cleanall    - removes all files from your working directory produced
#                            by both the sqlj translator and java compiler.

# This file assumes the DB2 instance path is defined by the variable HOME.
# It also assumes that DB2 is installed under the DB2 instance.
# If these conditions are not true, redefine DB2INSTANCEPATH
DB2INSTANCEPATH = $(HOME)

# Use the java compiler
CC= javac

# To connect to another database update the DATASOURCE variable.
# User ID and password are optional. If you want to use them,
# update TESTUID with your user ID, and TESTPWD with your password.

DATASOURCE=sample
TESTUID=
TESTPWD=
```

```
COPY = cp
ERASE = rm -f

# Note: 'all' contains RunCatUdf, which executes the 'java CatUdf' command,
# as this must be run before the 'make Udf' command.
all : DB2Appl DB2Applt DB2Stp DB2Udf App Applt Stp CatUdf RunCatUdf Udf DropUdf

RunCatUdf :
        java CatUdf

clean :
        $(ERASE) *.class
        $(ERASE) core

cleanall : clean
        $(ERASE) App.java
        $(ERASE) Applt.java
        $(ERASE) Stp.java
        $(ERASE) CatUdf.java
        $(ERASE) Udf.java
        $(ERASE) DropUdf.java
        $(ERASE) *.ser

# Build and run the following JDBC application with these commands:
#
#     make DB2Appl
#     java DB2Appl
#
DB2Appl.class : DB2Appl.java
DB2Appl : DB2Appl.class
        $(CC) DB2Appl.java

# After following the setup instructions in the README, you can
# build and run the following JDBC applet with these commands:
#
#     make DB2Applt
#     appletviewer DB2Applt.html
#
DB2Applt.class : DB2Applt.java
DB2Applt : DB2Applt.class
        $(CC) DB2Applt.java

# Build and run the following JDBC stored procedure with these commands:
#
#     make DB2Stp
#     java DB2Stp
#
DB2Stp.class : DB2Stp.java
DB2Stp : DB2Stp.class
        $(CC) DB2Stp.java
        $(ERASE) $(DB2INSTANCEPATH)/sqllib/function/DB2StpSample.class
        $(COPY) DB2StpSample.class $(DB2INSTANCEPATH)/sqllib/function/DB2StpSample.class
```

```
# Build and run the following JDBC UDF with these commands:
#
#     make DB2Udf
#     java DB2Udf
#
DB2Udf.class : DB2Udf.java
DB2Udf : DB2Udf.class
        $(CC) DB2Udf.java
        $(ERASE) $(DB2INSTANCEPATH)/sqllib/function/DB2UdfSample.class
        $(COPY) DB2UdfSample.class $(DB2INSTANCEPATH)/sqllib/function/DB2UdfSample.class

# Build and run the following SQLJ application with these commands:
#
#     make  App
#     java  App
#
App.java : App.sqlj
        sqlj App.sqlj
App.class : App.java App_SJProfile0.ser
App : App.class
        $(CC) App.java
        db2profc -url=jdbc:db2:sample -prepoptions="package using App" App_SJProfile0


# After following the setup instructions in the README, you can
# build and run the following SQLJ applet with these commands:
#
#     make Applt
#     appletviewer Applt.html
#
Applt.java : Applt.sqlj
        sqlj Applt.sqlj
Applt.class : Applt.java Applt_SJProfile0.ser
Applt : Applt.class
        $(CC) Applt.java
        db2profc -url=jdbc:db2:sample -prepoptions="package using Applt"
            Applt_SJProfile0


# Build and run the following SQLJ stored procedure with these commands:
#
#     make  Stp
#     java  Stp
#
Stp.java : Stp.sqlj
        sqlj Stp.sqlj
Stp.class : Stp.java Stp_SJProfile0.ser
Stp : Stp.class
        $(CC) Stp.java
        db2profc -url=jdbc:db2:sample -prepoptions="package using Stp" Stp_SJProfile0
        $(ERASE) $(DB2INSTANCEPATH)/sqllib/function/Stpsrv.class
```

```
        $(ERASE) $(DB2INSTANCEPATH)/sqllib/function/Stp_Cursor1.class
        $(ERASE) $(DB2INSTANCEPATH)/sqllib/function/Stp_Cursor2.class
        $(ERASE) $(DB2INSTANCEPATH)/sqllib/function/Stp_SJProfileKeys.class
        $(ERASE) $(DB2INSTANCEPATH)/sqllib/function/Stp_SJProfile0.ser
        $(COPY) Stpsrv.class $(DB2INSTANCEPATH)/sqllib/function/Stpsrv.class
        $(COPY) Stp_Cursor1.class $(DB2INSTANCEPATH)/sqllib/function/Stp_Cursor1.class
        $(COPY) Stp_Cursor2.class $(DB2INSTANCEPATH)/sqllib/function/Stp_Cursor2.class
        $(COPY) Stp_SJProfileKeys.class
                        $(DB2INSTANCEPATH)/sqllib/function/Stp_SJProfileKeys.class
        $(COPY) Stp_SJProfile0.ser $(DB2INSTANCEPATH)/sqllib/function/Stp_SJProfile0.ser

# The following SQLJ User-Defined Functions application requires three programs.
# Build and run the UDF programs with these commands:
#
#     make CatUdf
#     java CatUdf
#     make Udf
#     java Udf
#     make DropUdf
#     java DropUdf
#
# Note: 'java CatUdf' must be executed before 'make Udf'. The 'make all'
# command calls RunCatUdf, which executes 'java CatUdf', before calling Udf.

CatUdf.java : CatUdf.sqlj
        sqlj CatUdf.sqlj
CatUdf.class : CatUdf.java CatUdf_SJProfile0.ser
CatUdf : CatUdf.class
        $(CC) CatUdf.java
        db2profc -url=jdbc:db2:sample -prepoptions="package using CatUdf"
                        CatUdf_SJProfile0


Udf.java : Udf.sqlj
        sqlj Udf.sqlj
Udf.class : Udf.java Udf_SJProfile0.ser
Udf : Udf.class
        $(CC) Udf.java
        db2profc -url=jdbc:db2:sample -prepoptions="package using Udf" Udf_SJProfile0
        $(ERASE) $(DB2INSTANCEPATH)/sqllib/function/Udfsrv.class
        $(COPY) Udfsrv.class $(DB2INSTANCEPATH)/sqllib/function/Udfsrv.class

DropUdf.java : DropUdf.sqlj
        sqlj DropUdf.sqlj
DropUdf.class : DropUdf.java DropUdf_SJProfile0.ser
DropUdf : DropUdf.class
        $(CC) DropUdf.java
        db2profc -url=jdbc:db2:sample -prepoptions="package using DropUdf"
            DropUdf_SJProfile0
```

## JDBC Programs

### Applications

DB2Appl demonstrates a dynamic SQL Java application using the JDBC Application driver to access a DB2 database.

**Command Line**. To build and run this application by commands entered at the command line:

1. Compile DB2Appl.java with this command:

   ```
   javac DB2Appl.java
   ```

   to produce the file: DB2Appl.class.

2. Run the java interpreter on the application with this command:

   ```
   java DB2Appl
   ```

**makefile**. To build this application with the makefile, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section "The Java makefile" on page 151.
2. Build the application with this command:

   ```
   make DB2Appl
   ```

3. Run the java interpreter on the application with this command:

   ```
   java DB2Appl
   ```

### Applets

DB2Applt demonstrates a dynamic SQL Java applet using the JDBC applet driver to access a DB2 database.

**Command Line**. To build and run this applet by commands entered at the command line:

1. Ensure that a web server is installed on your DB2 machine (server or client).
2. Modify the DB2Applt.html file according to the instructions there.
3. Start the JDBC applet server on the TCP/IP port specified in DB2Applt.html; for example, if in DB2Applt.html, you specified:

   ```
   param name=port value='6789'
   ```

   then you would enter:

   ```
   db2jstrt 6789
   ```

4. Compile DB2Applt.java with this command:

   ```
   javac DB2Applt.java
   ```

   to produce the file DB2Applt.class.

5. Ensure that your working directory is accessible by your web browser. If it is not, copy `DB2Applt.class` and `DB2Applt.html` into a directory that is accessible.
6. Copy the file `sqllib/java/db2java.zip` into the same directory as `DB2Applt.class` and `DB2Applt.html`.
7. On your client machine, start your web browser (which supports JDK 1.1) and load `DB2Applt.html`.

As an alternative to steps (1), (5) and (7), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer DB2Applt.html
```

**makefile**. To build this applet with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section "The Java makefile" on page 151.
2. Ensure that a web server is installed on your DB2 machine (server or client).
3. Modify the `DB2Applt.html` file according to the instructions there.
4. Start the JDBC applet server on the TCP/IP port specified in `DB2Applt.html`; for example, if in `DB2Applt.html`, you specified:

   ```
   param name=port value='6789'
   ```

   then you would enter:

   ```
   db2jstrt 6789
   ```
5. Build the applet with this command:

   ```
   make DB2Applt
   ```
6. Ensure that your working directory is accessible by your web browser. If it is not, copy `DB2Applt.class` and `DB2Applt.html` into a directory that is accessible.
7. Copy the file `sqllib/java/db2java.zip` into the same directory as `DB2Applt.class` and `DB2Applt.html`.
8. On your client machine, start your web browser (which supports JDK 1.1) and load `DB2Applt.html`.

As an alternative to steps (2), (6) and (8), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer DB2Applt.html
```

## Stored Procedures

`DB2Stp` demonstrates how to write a dynamic SQL Java stored procedure using the JDBC Application driver to access a DB2 database.

**Command Line**. To build and run this stored procedure by commands entered at the command line:

1. Compile `DB2Stp.java` with this command:

   ```
   javac DB2Stp.java
   ```

   This will produce the files `DB2Stp.class` and `DB2StpSample.class`.
2. Copy `DB2StpSample.class` to the `sqllib/function` directory.
3. Run the java interpreter on the stored procedure with this command:

   ```
   java DB2Stp
   ```

**makefile**. To build this stored procedure with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section "The Java makefile" on page 151.
2. Build the stored procedure with this command:

   ```
   make DB2Stp
   ```
3. Run the java interpreter on the stored procedure with this command:

   ```
   java DB2Stp
   ```

## User-Defined Functions

`DB2Udf` demonstrates implementing dynamic SQL user-defined functions using the JDBC Application driver to access a DB2 database.

**Command Line**. To build and run this UDF program by commands entered at the command line:

1. Compile `DB2Udf.java` with this command:

   ```
   javac DB2Udf.java
   ```

   This will produce the files `DB2Udf.class` and `DB2UdfSample.class`.
2. Copy `DB2UdfSample.class` to the `sqllib/function` directory.
3. Run the java interpreter on the UDF program with this command:

   ```
   java DB2Udf
   ```

**makefile**. To build this UDF program with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section "The Java makefile" on page 151.
2. Build the UDF program with this command:

   ```
   make DB2Udf
   ```
3. Run the java interpreter on the UDF program with this command:

   ```
   java DB2Udf
   ```

## SQLJ Programs

### Applications

`App` demonstrates an SQLJ application that accesses a DB2 database.

**Command Line**. To build and run this application by commands entered at the command line:

1. Translate `App.sqlj` with this command:

   ```
   sqlj App.sqlj
   ```

   This will produce the files `App.java` and `App_SJProfile0.ser`.

2. Compile `App.java` with this command:

   ```
   javac App.java
   ```

   This will produce the files: `App.class`, `App_Cursor1.class`, `App_Cursor2.class` and `App_SJProfileKeys.class`.

3. Customize the generated profile and create the package `App` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample prepoptions="package using App" App_SJProfile0
   ```

4. Run the application with this command:

   ```
   java App
   ```

**makefile**. To build this application with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java makefile” on page 151.
2. Build the application with this command:

   ```
   make App
   ```

3. Run the application with this command:

   ```
   java App
   ```

### Applets

`Applt` demonstrates an SQLJ applet that accesses a DB2 database.

**Command Line**. To build and run this applet by commands entered at the command line:

1. Ensure that a web server is installed on your DB2 machine (server or client).
2. Modify the `Applt.html` file according to the instructions there.
3. Start the JDBC applet server on the TCP/IP port specified in `Applt.html`; for example, if in `Applt.html`, you specified:

   ```
   param name=port value='6789'
   ```

   then you would enter:

```
db2jstrt 6789
```

4. Translate `Applt.sqlj` with this command:

   ```
   sqlj Applt.sqlj
   ```

   This will produce the files: `Applt.java` and `Applt_SJProfile0.ser`.

5. Compile `Applt.java` with this command:

   ```
   javac Applt.java
   ```

   This will produce the files: `Applt.class`, `Applt_Cursor1.class`, `Applt_Cursor2.class` and `Applt_SJProfileKeys.class`.

6. Customize the generated profile and create the package `Applt` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample -prepoptions="package using Applt" Applt_SJProfile0
   ```

7. Ensure that your working directory is accessible by your web browser. If it is not, copy the following files into a directory that is accessible:

   ```
   Applt.html                          Applt.class,
   Applt_Cursor1.class,                Applt_Cursor2.class,
   Applt_SJProfileKeys.class,          Applt_SJProfile0.ser
   ```

8. Copy the files `sqllib/java/db2java.zip` and `sqllib/java/runtime.zip` into the same directory as your other `Applt` files.

9. On your client machine, start your web browser (which must support JDK 1.1) and load `Applt.html`.

As an Alternative to steps (1), (7) and (9), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer Applt.html
```

**makefile**. To build this applet with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java makefile” on page 151.
2. Ensure that a web server is installed on your DB2 machine (server or client).
3. Modify the `Applt.html` file according to the instructions there.
4. Start the JDBC applet server on the TCP/IP port specified in `Applt.html`. For example, if in `Applt.html`, you specified:

   ```
   param name=port value='6789'
   ```

   then you would enter:

   ```
   db2jstrt 6789
   ```

5. Build the applet with this command:

   ```
   make Applt
   ```

6. Ensure that your working directory is accessible by your web browser. If it is not, copy the following files into a directory that is accessible:

   ```
   Applt.html,                      Applt.class,
   Applt_Cursor1.class,             Applt_Cursor2.class,
   Applt_SJProfileKeys.class,       Applt_SJProfile0.ser
   ```

7. Copy the files `sqllib/java/db2java.zip` and `sqllib/java/runtime.zip` into the same directory as your other `Applt` files.
8. On your client machine, start your web browser (which must support JDK 1.1) and load `Applt.html`.

As an Alternative to steps (2), (6) and (8), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer Applt.html
```

## Stored Procedures

`Stp` demonstrates an SQLJ stored procedure that accesses a DB2 database.

**Command Line**. To build and run this stored procedure by commands entered at the command line:

1. Translate `Stp.sqlj` with this command:

   ```
   sqlj Stp.sqlj
   ```

   This will produce the files `Stp.java` and `Stp_SJProfile0.ser`.

2. Compile `Stp.java` with this command:

   ```
   javac Stp.java
   ```

   This will produce the files: `Stp.class`, `Stpsrv.class`, `Stp_Cursor1.class`, `Stp_Cursor2.class` and `Stp_SJProfileKeys.class`.

3. Customize the generated profile and create the package `Stp` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample -prepoptions="package using Stp" Stp_SJProfile0
   ```

4. Copy these files to the `sqllib/function` directory: `Stpsrv.class`, `Stp_Cursor1.class`, `Stp_Cursor2.class`, `Stp_SJProfileKeys.class` and `Stp_SJProfile0.ser`.
5. Run the stored procedure with this command:

   ```
   java Stp
   ```

**makefile**. To build this stored procedure with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java makefile” on page 151.
2. Build the stored procedure with this command:

```
make Stp
```

3. Run the stored procedure with this command:

   ```
   java Stp
   ```

## User-Defined Functions

The `sqllib/samples/java` directory includes a UDF application consisting of three SQLJ programs:

- `CatUdf` demonstrates cataloging Java user-defined functions (UDFs) and creating a sample table, `udftest`, for testing them.
- `Udf` demonstrates calling Java UDFs against the sample table, `udftest`.
- `DropUdf` demonstrates dropping Java UDFs and the sample table, `udftest`.

**CatUdf**.

**Command Line**. To build and run this SQLJ program by commands entered at the command line:

1. Translate `CatUdf.sqlj` with this command:

   ```
   sqlj CatUdf.sqlj
   ```

   This will produce the files `CatUdf.java` and `CatUdf_SJProfile0.ser`.

2. Compile `CatUdf.java` with this command:

   ```
   javac CatUdf.java
   ```

   This will produce the files `CatUdf.class` and `CatUdf_SJProfileKeys.class`.

3. Customize the generated profile and create the package `CatUdf` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample -prepoptions="package using CatUdf" CatUdf_SJProfile0
   ```

4. Run `CatUdf` with this command:

   ```
   java CatUdf
   ```

5. Next, run the `Udf` program.

**makefile**. To build and run this SQLJ program with the `makefile`:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java makefile” on page 151.
2. Build and run CatUdf with this command:

   ```
   make CatUdf
   ```

3. Next, run the `Udf` program.

**Udf**.

**Command Line**. To build and run this SQLJ program by commands entered at the command line:

1. Translate `Udf.sqlj` with this command:

   ```
   sqlj Udf.sqlj
   ```

   This will produce the files `Udf.java` and `Udf_SJProfile0.ser`.
2. Compile `Udf.java` with this command:

   ```
   javac Udf.java
   ```

   This will produce the files: `Udf.class`, `Udf_Cursor1.class`, `Udf_Cursor2.class`, `Udf_Cursor4.class`, `Udf_Cursor5.class`, `Udf_SJProfileKeys.class` and `Udfsrv.class`.

   **Note:** There is no file `Udf_Cursor3.class`.
3. Customize the generated profile and create the package `Udf` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample -prepoptions="package using Udf" Udf_SJProfile0
   ```
4. Copy the `Udfsrv.class` file into `sqllib/function`.
5. Run `Udf` with this command:

   ```
   java Udf
   ```
6. Next, run the `DropUdf` program.

**makefile**. To build this SQLJ program with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java makefile” on page 151.
2. Build `Udf` with this command:

   ```
   make Udf
   ```
3. Run `Udf` with this command:

   ```
   java Udf
   ```
4. Next, run the `DropUdf` program.

**DropUdf**.

**Command Line**. To build and run this SQLJ program by commands entered at the command line:

1. Translate `DropUdf.sqlj` with this command:

   ```
   sqlj DropUdf.sqlj
   ```

   This will produce the files `DropUdf.java` and `DropUdf_SJProfile0.ser`.
2. Compile `DropUdf.java` with this command:

   ```
   javac DropUdf.java
   ```

   This will produce the files: `DropUdf.class` and `DropUdf_SJProfileKeys.class`.
3. Customize the generated profile and create the package `DropUdf` in the `sample` database with this command:

```
db2profc -url=jdbc:db2:sample -prepoptions="package using DropUdf"
                DropUdf_SJProfile0
```

4. Run `DropUdf` with this command:

   ```
   java DropUdf
   ```

**makefile**. To build this SQLJ program with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java makefile” on page 151.
2. Build `DropUdf` with this command:

   ```
   make DropUdf
   ```

3. Run `DropUdf` with this command:

   ```
   java DropUdf
   ```

## General Points for DB2 Java Applets

1. For a larger JDBC or SQLJ applet that consists of several Java classes, you may choose to package all its classes in a single Jar file. For an SQLJ applet, you would also have to package its serialized profiles along with its classes. If you choose to do this, add your Jar file into the `archive` parameter in the "applet" tag. For details, see the JDK Version 1.1 documentation.
2. You may wish to place the file `sqllib/java/db2java.zip` (and for SQLJ applets, also the file `sqllib/java/runtime.zip`) into a directory that is shared by several applets that may be loaded from your Web site. In this case, you may need to add a `codebase` parameter into the "applet" tag in the HTML file to identify that directory. For details, see the JDK Version 1.1 documentation.
3. For information on running DB2 Java applets on a webserver, specifically the Domino Go Webserver, see:

   ```
   http://www.software.ibm.com/data/db2/db2lotus/gojava.htm
   ```

# Appendix A. About Database Manager Instances

DB2 supports multiple database manager instances on the same machine. A database manager instance has its own configuration files, directories, and databases.

Each database manager instance can manage several databases. However, a given database belongs to only one instance. Figure 1 shows this relationship.

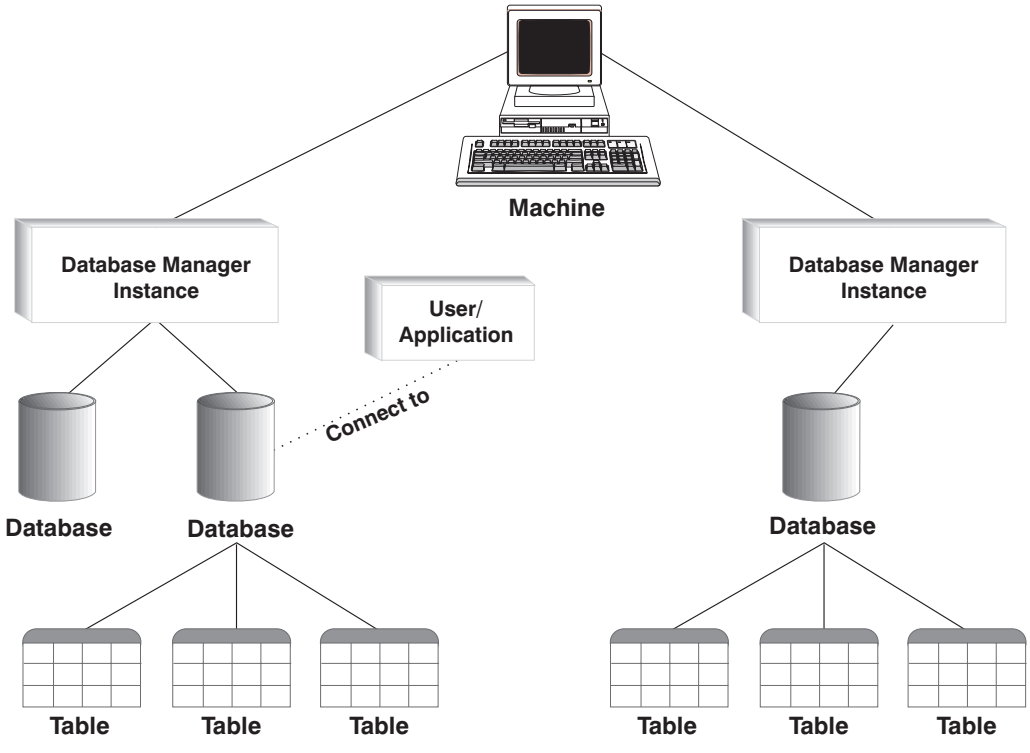


*Figure 1. Database Manager Instances*

Database manager instances give you the flexibility to have multiple database environments on the same machine. For example, you can have one database manager instance for development, and another instance for production.

With UNIX servers you can have different DB2 versions on different database manager instances. For example, you can have one database manager instance running DB2 Version 2, and another running DB2 Universal Database Version 5.

With OS/2 and NT servers you must have the same DB2 version, release, and modification level on each database manager instance. You cannot have one database manager instance running DB2 Version 2, and another instance running DB2 Universal Database Version 5.

© Copyright IBM Corp. 1993, 1998

You need to know the following for each instance you use:

**instance name**

For UNIX platforms, this is a valid username that you specify when you create the database manager instance.

For OS/2 and Windows NT, this is an alphanumeric string of up to eight characters. The DB2 instance is created for you during install.

**instance directory**

The home directory where the instance is located.

For UNIX platforms, the home directory is `$HOME/sqllib`, where $HOME is the home directory of the instance owner.

For OS/2 and Windows NT, the directory is `%DB2PATH%\`*instance_name*. The variable `%DB2PATH%` determines where DB2 is installed. Depending on which drive DB2 is installed, `%DB2PATH%` will point to *drive*`:\sqllib`.

The instance path on OS/2 and Windows NT is created based on either:

`%DB2PATH%\%DB2INSTANCE%` (for example, `C:\SQLLIB\DB2`)

or, if DB2INSTPROF is defined:

`%DB2INSTPROF%\%DB2INSTANCE%` (for example, `C:\PROFILES\DB2`)

The DB2INSTPROF environment is used on OS/2 and Windows NT to support running DB2 on a network drive in which the client machine has only read access. In this case, DB2 will be set to point to *drive*`:\sqllib`, and DB2INSTPROF will be set to point to a local path, for example, `C:\PROFILES`, which will contain all instance specific information such as catalogs and configurations, since DB2 requires update access to these files.

For information about creating and managing database manager instances, refer to the *Quick Beginnings* book.

# Appendix B. Migrating Your Applications

When you upgrade to DB2 Universal Database Version 5 from a previous installation of DB2, DB2 Client Application Enabler, or DB2 Software Developer's Kit, your database and node directories are migrated automatically. To migrate your existing databases, use the tools described in the *Administration Guide*.

**Notes:**

1. **HP-UX**. If you are migrating DB2 from HP-UX Version 10 or earlier to HP-UX Version 11, your DB2 programs must be re-precompiled with DB2 on HP-UX Version 11 (if they include embedded SQL), and must be re-compiled. This includes all DB2 applications, stored procedures, user-defined functions and user exit programs. As well, DB2 programs that are compiled on HP-UX Version 11 may not run on HP-UX Version 10 or earlier. DB2 programs that are compiled and run on HP-UX Version 10 may connect remotely to HP-UX Version 11 servers.
2. **Micro Focus COBOL**. Any existing applications precompiled with DB2 Version 2.1.1 or earlier and compiled with Micro Focus COBOL should be re-precompiled with the current version of DB2, and then recompiled with Micro Focus COBOL. If these applications built with the earlier versions of the IBM precompiler are not re-precompiled, there is a possibility of database corruption if abnormal termination occurs.
3. **SCO OpenServer**. If you are migrating from DB2 for SCO OpenServer to DB2 for SCO UnixWare 7, any existing DB2 applications precompiled on SCO OpenServer should be re-precompiled on SCO UnixWare 7, and all DB2 applications compiled on SCO OpenServer should be recompiled on SCO UnixWare 7.

If you have applications from DB2 Version 1 or DB2 Version 2, and you want them to run in both a database instance of the previous version as well as a DB2 Version 5 instance on the same machine, you may need to make some changes to your environment. To determine what changes to make, answer the following questions, and then review the "Conditions" section to see if any of the conditions apply to your situation.

An AIX system is used to explain the points raised. The same concepts apply to other UNIX platforms, but the details may differ, such as environment variables and specific commands. If you are unfamiliar with these details for your operating system, please see the *Administration Guide* or the "Migrated from Previous Versions" chapter in the *Quick Beginnings for UNIX* book.

## Questions

Question 1: How was the previous version application linked to the DB2 client runtime library (libdb2.a)?

To determine the embedded shared library search path for an executable, use one of the following system commands:

© Copyright IBM Corp. 1993, 1998

| | |
|---|---|
| **AIX** | /usr/bin/dump -H executable |
| **HP-UX** | /usr/bin/chatr executable |
| **SCO UnixWare 7** | /usr/bin/dump -Lv executable |
| **Silicon Graphics IRIX** | /bin/odump -D executable |
| **Solaris** | /usr/bin/dump -Lv executable |

where executable is the name of the application executable.

The following is a sample dump listing from a DB2 Version 1 for AIX application:

```
dbcat:

  ***Loader Section***
                      Loader Header Information
VERSION#         #SYMtableENT     #RELOCent        LENidSTR
0x00000001       0x00000012       0x00000029       0x00000064

#IMPfilID        OFFidSTR         LENstrTBL        OFFstrTBL
0x00000004       0x000003bc       0x00000077       0x00000420


  ***Import File Strings***
INDEX  PATH                          BASE               MEMBER
0      /usr/lpp/db2_01_01_0000/lib:/usr/lpp/xlC/lib:/usr/lib:/lib



1                                    libc.a             shr.o
2                                    libC.a             shr.o
3                                    libdb2.a           shr.o
```

Line 0 (zero) shows the directory paths that the executable searches to find the shared libraries to which it is linked. Lines 1, 2, and 3 show the shared libraries to which the application is linked.

Depending on how the application was built, you may see the following paths: /usr/lpp/db2_01_01_0000/lib, INSTHOME/sqllib/lib (where INSTHOME is the home directory of the database instance owner), or just the /usr/lib:/lib combination.

Question 2: How are the DB2 runtime libraries configured on your system?

When either of DB2 Versions 1, 2, or 5 is installed, there is an optional step which creates symbolic links from the system default shared library path /usr/lib to the DB2 install path which contains the DB2 client runtime libraries.

For Version 1, the install path is /usr/lpp/db2_01_01_0000/lib. For Version 2, the install path is /usr/lpp/db2_02_01/lib. For Version 5, the install path is /usr/lpp/db2_05_00/lib. In all cases, the runtime shared libraries are named libdb2.a.

Only one version of these libraries can be the default at any one time. DB2 provides this default so that when you build an application, it does not depend on a particular version of DB2.

Question 3: Do you specify different search paths in your environment?

You can override the shared library search path coded in your application using the LIBPATH environment variable on AIX, SHLIB_PATH on HP-UX, and LD_LIBRARY_PATH on SCO UnixWare 7, Silicon Graphics IRIX, and Solaris. You can see the library search path using the appropriate system command for your platform given in the answer to Question 1.

If you use this environment variable, you should include the /usr/lib:/lib path because general-purpose shared libraries, such as the C runtime libraries, are located in that path.

## Conditions

Once you have the answers to the questions above, you may need to make changes to your environment. Read the conditions listed below. If one of the conditions applies to your situation, make the necessary changes.

Condition 1: If a Version 2 application loads a shared library out of the AIX default shared library path /usr/lib/libdb2.a, and

- If there is a symbolic link from /usr/lib/libdb2.a to /usr/lpp/db2_02_01/lib/libdb2.a, and the database server is DB2 Universal Database Version 5 for AIX, do one of the following:
  - Change the symbolic link to point to /usr/lpp/db2_05_00/lib/libdb2.a. Refer to *Quick Beginnings for UNIX* for information about setting links between libraries. As root, you can change links using the "db2ln" command as follows:

    ```
    /usr/lpp/db2_05_00/cfg/db2ln
    ```

  - Set the LIBPATH environment variable to point to /usr/lpp/db2_05_00/lib or INSTHOME/sqllib/lib, where INSTHOME is the home directory of the Version 5 DB2 instance owner.
  - Configure a TCP/IP connection from the application (client) instance to the server instance. Refer to *Installing and Configuring DB2 Clients* for information about configuring TCP/IP.
- If there is a symbolic link from /usr/lib/libdb2.a to /usr/lpp/db2_05_00/lib/libdb2.a, and the database server is DB2 Version 2, configure a TCP/IP connection from the application (client) instance to the server instance. Refer to *Installing and Configuring DB2 Clients* for information about configuring TCP/IP.

Condition 2: If a Version 2 application loads a shared library out of the $HOME path of a DB2 Version 2 instance owner ($HOME/sqllib/lib/libdb2.a), and the database server is DB2 Universal Database Version 5 for AIX, do one of the following:

- Migrate the application instance to the same version as the database server instance.
- Set the LIBPATH environment variable to point to /usr/lpp/db2_05_00/lib or INSTHOME/sqllib/lib, where INSTHOME is the home directory of the Version 5 instance owner.
- Configure a TCP/IP connection from the application (client) instance to the server instance. Refer to *Installing and Configuring DB2 Clients* for information about configuring TCP/IP.

Condition 3: If a Version 2 application loads a shared library out of the DB2 Version 2 install path (/usr/lpp/db2_02_01/lib/libdb2.a), and the database server is DB2 Universal Database Version 5 for AIX, do one of the following:

- Set the LIBPATH environment variable to point to /usr/lpp/db2_05_00/lib or INSTHOME/sqllib/lib, where INSTHOME is the home directory of the database instance owner.
- Configure a TCP/IP connection from the application (client) instance to the server instance. Refer to *Installing and Configuring DB2 Clients* for information about configuring TCP/IP.

Condition 4: If a Version 2 application loads a shared library out of the DB2 Universal Database Version 5 for AIX install path (/usr/lpp/db2_05_00/lib/libdb2.a), and the database server is DB2 Version 2, configure a TCP/IP connection from the application (client) instance to the server instance. Refer to *Installing and Configuring DB2 Clients* for information about configuring TCP/IP.

## Other Migration Considerations

Consider the following points when you develop your applications. They might help make your applications more portable:

- Use only the default path /usr/lib:/lib in your applications, and create symbolic links between the default path and the version of DB2 you are using. Ensure that the link is to the minimum level of DB2 required by your applications. Refer to *Quick Beginnings for UNIX* for information about setting links.
- If your application requires a particular version of DB2, code the path that specifies the DB2 version in your application. For example, if your AIX application requires DB2 Version 2, code /usr/lpp/db2_02_01/lib. Ordinarily, you do not need to do this.
- Generally, the path in your application should not point to the instance owner's copy of the sqllib/lib directory. This makes applications highly dependent on specific user names and environments.
- Generally, do not use the LIBPATH environment variable to alter search paths in a particular environment. The variable overrides the search paths specified in the

applications running in that environment. Applications might not be able to find the libraries or the files that they need.

- In DB2 Universal Database Version 5, all character array items with string semantics have type char, instead of other variations, such as unsigned char. Any applications you code with DB2 Universal Database Version 5 should follow this practice.

  If you have DB2 Version 1 applications which use unsigned char, your compiler might produce warnings or errors because of type clashes between unsigned char in Version 1 applications and char in Version 5 function prototypes. If this occurs, use the compiler option -DSQLOLDCHAR to eliminate the problem.

- Refer to the *SQL Reference* for a list of incompatibilities between DB2 Universal Database Version 5 and previous versions of DB2. Refer to the *API Reference* for a list of API incompatibilities between DB2 Universal Database Version 5 and previous versions of DB2.

# Appendix C. Problem Determination

You may encounter the following kinds of problems when building or running your applications:

- Client or server problems, such as failing to connect to the database during a build or when running your application.
- Operating system problems, such as not being able to find files during a build.
- Compiler option problems during a build.
- Syntax and coding problems during a build or when running your application.

You can use the following sources of information to resolve these problems:

**Build script files**
: For build problems, such as connecting to a database, precompiling, compiling, linking, and binding, you can use the script files shown in this book to see command line processor commands and compiler options that work.

**Compiler documentation**
: For compiler option problems not covered by the build script files.

**Embedded SQL Programming Guide**
: Refer to the *Embedded SQL Programming Guide* for syntax and other coding problems.

**CLI Guide and Reference**
: Refer to the *CLI Guide and Reference* for syntax and other coding problems related to CLI programs.

**SQL Reference**
: Refer to the *SQL Reference* for syntax of SQL statements and functions.

**SQLCA data structure**
: If your application issues SQL statements or calls database manager APIs, it must check for error conditions by examining the SQLCA data structure.

  The SQLCA data structure returns error information in the SQLCODE and SQLSTATE fields. The database manager updates the structure after every SQL statement is executed, and after most database manager API calls.

  Your application can retrieve and print the error information or display it on the screen. Refer to the *Embedded SQL Programming Guide* for more information.

**Online error messages**
: Different components of DB2, including the database manager, database administration utility, installation and configuration process, and command line processor, generate online error messages. Each of these messages has a unique prefix and a four or five digit message number following the prefix. A single letter is displayed after the message number indicating the severity of the error.

© Copyright IBM Corp. 1993, 1998

You can use the command line processor to see the help for the message by entering:

```
db2 "? xxxnnnn"
```

where `xxx` is the message prefix, and `nnnn` is the message number. Include the quotes.

For the full list and description of DB2 error messages, see the *Messages Reference*.

**Diagnostic tools and error log**

For build or runtime problems you cannot resolve using the other sources of information. The diagnostic tools include a trace facility, system log, and message log, among others. DB2 puts error and warning conditions in an error log based on priority and origin. Refer to the *Troubleshooting Guide* for more information. There is also a CLI trace facility specifically for debugging CLI programs. For more information, refer to the *CLI Guide and Reference*.

# Appendix D. How the DB2 Library Is Structured

The DB2 Universal Database library consists of SmartGuides, online help, and books. This section describes the information that is provided, and how to access it.

To access product information online, you can use the Information Center. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web. See “Information Center” on page 183 for details.

## SmartGuides

SmartGuides help you complete some administration tasks by taking you through each task one step at a time. SmartGuides are available through the Control Center. The following table lists the SmartGuides.

**Note:** Not all SmartGuides are available for the partitioned database environment.

| SmartGuide | Helps you to... | How to Access... |
|---|---|---|
| Add Database | Catalog a database on a client workstation. | From the Client Configuration Assistant, click on **Add**. |
| Create Database | Create a database, and perform some basic configuration tasks. | From the Control Center, click with the right mouse button on the **Databases** icon and select **Create**->**New**. |
| Performance Configuration | Tune the performance of a database by updating configuration parameters to match your business requirements. | From the Control Center, click with the right mouse button on the database you want to tune and select **Configure performance**. |
| Backup Database | Determine, create, and schedule a backup plan. | From the Control Center, click with the right mouse button on the database you want to backup and select **Backup**->**Database using SmartGuide**. |
| Restore Database | Recover a database after a failure. It helps you understand which backup to use, and which logs to replay. | From the Control Center, click with the right mouse button on the database you want to restore and select **Restore**->**Database using SmartGuide**. |
| Create Table | Select basic data types, and create a primary key for the table. | From the Control Center, click with the right mouse button on the **Tables** icon and select **Create**->**Table using SmartGuide**. |
| Create Table Space | Create a new table space. | From the Control Center, click with the right mouse button on the **Table spaces** icon and select **Create**->**Table space using SmartGuide**. |

© Copyright IBM Corp. 1993, 1998

## Online Help

Online help is available with all DB2 components. The following table describes the various types of help. You can also access DB2 information through the Information Center. For information see “Information Center” on page 183.

| Type of Help | Contents | How to Access... |
|---|---|---|
| Command Help | Explains the syntax of commands in the command line processor. | From the command line processor in interactive mode, enter:<br>**?** *command*<br>where *command* is a keyword or the entire command.<br>For example, **? catalog** displays help for all the CATALOG commands, while **? catalog database** displays help for the CATALOG DATABASE command. |
| Control Center Help | Explains the tasks you can perform in a window or notebook. The help includes prerequisite information you need to know, and describes how to use the window or notebook controls. | From a window or notebook, click on the **Help** push button or press the F1 key. |
| Message Help | Describes the cause of a message, and any action you should take. | From the command line processor in interactive mode, enter:<br>**?** *XXXnnnnn*<br>where *XXXnnnnn* is a valid message identifier.<br>For example, **? SQL30081** displays help about the SQL30081 message.<br>To view message help one screen at a time, enter:<br>**?** *XXXnnnnn* **\| more**<br>To save message help in a file, enter:<br>**?** *XXXnnnnn* > *filename.ext*<br>where *filename.ext* is the file where you want to save the message help. |
| SQL Help | Explains the syntax of SQL statements. | From the command line processor in interactive mode, enter:<br>**help** *statement*<br>where *statement* is an SQL statement.<br>For example, **help** *SELECT* displays help about the SELECT statement. |

| Type of Help | Contents | How to Access... |
|---|---|---|
| SQLSTATE Help | Explains SQL states and class codes. | From the command line processor in interactive mode, enter:<br>**?** *sqlstate* or **?** *class-code*<br>where *sqlstate* is a valid five-digit SQL state and the *class-code* is first two digits of the SQL state.<br>For example, **? 08003** displays help for the 08003 SQL state, while **? 08** displays help for the 08 class code. |

## DB2 Books

The table in this section lists the DB2 books. They are divided into two groups:

**Cross-platform books** These books contain the common DB2 information for UNIX-based and Intel-based platforms.

**Platform-specific books** These books are for DB2 on a specific platform. For example, for DB2 on OS/2, on Windows NT, and on the UNIX-based platforms, there are separate *Quick Beginnings* books.

Most books are available in HTML and PostScript format, and in hardcopy that you can order from IBM. The exceptions are noted in the table.

If you want to read the English version of the books, they are always provided in the directory that contains the English documentation.

You can obtain DB2 books and access information in a variety of different ways:

**View** See “Viewing Online Books” on page 181.
**Search** See “Searching Online Books” on page 182.
**Print** See “Printing the PostScript Books” on page 182.
**Order** See “Ordering the Printed DB2 Books” on page 183.

| Book Name | Book Description | Form Number<br>File Name |
|---|---|---|
| **Cross-Platform Books** | | |
| *Administration Getting Started* | Introduces basic DB2 database administration concepts and tasks, and walks you through the primary administrative tasks. | S10J-8154<br>db2k0x50 |
| *Administration Guide* | Contains information required to design, implement, and maintain a database to be accessed either locally or in a client/server environment. | S10J-8157<br>db2d0x51 |
| *API Reference* | Describes the DB2 application programming interfaces (APIs) and data structures you can use to manage your databases. Explains how to call APIs from your applications. | S10J-8167<br>db2b0x51 |

| Book Name | Book Description | Form Number<br>File Name |
|---|---|---|
| *CLI Guide and Reference* | Explains how to develop applications that access DB2 databases using the DB2 Call Level Interface, a callable SQL interface that is compatible with the Microsoft ODBC specification. | S10J-8159<br>db2l0x50 |
| *Command Reference* | Explains how to use the command line processor, and describes the DB2 commands you can use to manage your database. | S10J-8166<br>db2n0x51 |
| *DB2 Connect Enterprise Edition Quick Beginnings* | Provides planning, migrating, installing, configuring, and using information for DB2 Connect Enterprise Edition. Also contains installation and setup information for all supported clients. | S10J-7888<br>db2cyx51 |
| *DB2 Connect Personal Edition Quick Beginnings* | Provides planning, installing, configuring, and using information for DB2 Connect Personal Edition. | S10J-8162<br>db2c1x51 |
| *DB2 Connect User's Guide* | Provides concepts, programming and general using information about the DB2 Connect products. | S10J-8163<br>db2c0x51 |
| *DB2 Connectivity Supplement* | Provides setup and reference information for customers who want to use DB2 for AS/400, DB2 for OS/390, DB2 for MVS, or DB2 for VM as DRDA Application Requesters with DB2 Universal Database servers, and customers who want to use DRDA Application Servers with DB2 Connect (formerly DDCS) application requesters.<br>**Note:** Available in HTML and PostScript formats only. | No form number<br>db2h1x51 |
| *Embedded SQL Programming Guide* | Explains how to develop applications that access DB2 databases using embedded SQL, and includes discussions about programming techniques and performance considerations. | S10J-8158<br>db2a0x50 |
| *Glossary* | Provides a comprehensive list of all DB2 terms and definitions.<br>**Note:** Available in HTML format only. | No form number<br>db2t0x50 |
| *Installing and Configuring DB2 Clients* | Provides installation and setup information for all DB2 Client Application Enablers and DB2 Software Developer's Kits.<br>**Note:** Available in HTML and PostScript formats only. | No form number<br>db2iyx51 |
| *Master Index* | Contains a cross reference to the major topics covered in the DB2 library.<br>**Note:** Available in PostScript format and hardcopy only. | S10J-8170<br>db2w0x50 |
| *Messages Reference* | Lists messages and codes issued by DB2, and describes the actions you should take. | S10J-8168<br>db2m0x51 |

| Book Name | Book Description | Form Number<br>File Name |
|---|---|---|
| *DB2 Replication Guide and Reference* | Provides planning, configuring, administering, and using information for the IBM Replication tools supplied with DB2. | S95H-0999<br>db2e0x52 |
| *Road Map to DB2 Programming* | Introduces the different ways your applications can access DB2, describes key DB2 features you can use in your applications, and points to detailed sources of information for DB2 programming. | S10J-8155<br>db2u0x50 |
| *SQL Getting Started* | Introduces SQL concepts, and provides examples for many constructs and tasks. | S10J-8156<br>db2y0x50 |
| *SQL Reference* | Describes SQL syntax, semantics, and the rules of the language. Also includes information about release-to-release incompatibilities, product limits, and catalog views. | S10J-8165<br>db2s0x51 |
| *System Monitor Guide and Reference* | Describes how to collect different kinds of information about your database and the database manager. Explains how you can use the information to understand database activity, improve performance, and determine the cause of problems. | S10J-8164<br>db2f0x50 |
| *Troubleshooting Guide* | Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service. | S10J-8169<br>db2p0x50 |
| *What's New* | Describes the new features, functions, and enhancements in DB2 Universal Database, Version 5.2, including information about Java-based tools. | S04L-6230<br>db2q0x51 |
| **Platform-Specific Books** | | |
| *Building Applications for UNIX Environments* | Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a UNIX system. | S10J-8161<br>db2axx51 |
| *Building Applications for Windows and OS/2 Environments* | Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Windows or OS/2 system. | S10J-8160<br>db2a1x50 |
| *DB2 Personal Edition Quick Beginnings* | Provides planning, installing, migrating, configuring, and using information for DB2 Universal Database Personal Edition on OS/2, Windows 95, and the Windows NT operating systems. | S10J-8150<br>db2i1x50 |
| *DB2 SDK for Macintosh Building Your Applications* | Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Macintosh system.<br>**Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only. | S50H-0528<br>sqla7x02 |
| *DB2 SDK for SCO OpenServer Building Your Applications* | Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SCO OpenServer system.<br>**Note:** Available for DB2 Version 2.1.2 only. | S89H-3242<br>sqla9x02 |

| Book Name | Book Description | Form Number<br>File Name |
|---|---|---|
| *DB2 SDK for SINIX Building Your Applications* | Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SINIX system.<br>**Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only. | S50H-0530<br>sqla8x00 |
| *Quick Beginnings for OS/2* | Provides planning, installing, migrating, configuring, and using information for DB2 Universal Database on OS/2. Also contains installing and setup information for all supported clients. | S10J-8147<br>db2i2x50 |
| *Quick Beginnings for UNIX* | Provides planning, installing, configuring, migrating, and using information for DB2 Universal Database on UNIX-based platforms. Also contains installing and setup information for all supported clients. | S10J-8148<br>db2ixx51 |
| *Quick Beginnings for Windows NT* | Provides planning, installing, configuring, migrating, and using information for DB2 Universal Database on the Windows NT operating system. Also contains installing and setup information for all supported clients. | S10J-8149<br>db2i6x50 |
| *DB2 Extended Enterprise Edition for UNIX Quick Beginnings* | Provides planning, installing, configuring, and using information for DB2 Universal Database Extended Enterprise Edition for UNIX.<br>This book supercedes the DB2 Extended Enterprise Edition Quick Beginnings for AIX book, and is suitable for use with all versions of DB2 Extended Enterprise Edition that run on UNIX-based platforms. | S99H-8314<br>db2v3x51 |
| *DB2 Extended Enterprise Edition for Windows NT Quick Beginnings* | Provides planning, installing, configuring, and using information for DB2 Universal Database Extended Enterprise Edition for Windows NT. | S09L-6713<br>db2v6x51 |

**Notes:**

1. The character in the sixth position of the file name indicates the language of a book. For example, the file name `db2d0e50` indicates that the *Administration Guide* is in English. The following letters are used in the file names to indicate the language of a book:

| Language | Identifier | Language | Identifier |
|---|---|---|---|
| Brazilian Portuguese | B | Japanese | J |
| Bulgarian | U | Korean | K |
| Czech | X | Norwegian | N |
| Danish | D | Polish | P |
| English | E | Russian | R |
| Finnish | Y | Simp. Chinese | C |
| French | F | Slovenia | L |
| German | G | Spanish | Z |
| Greek | A | Swedish | S |
| Hungarian | H | Trad. Chinese | T |

| | | | |
|---|---|---|---|
| Italian | I | Turkish | M |

2. For late breaking information that could not be included in the DB2 books:
   - On UNIX-based platforms, see the Release.Notes file. This file is located in the DB2DIR/Readme/%L directory, where %L is the locale name and DB2DIR is:
     - /usr/lpp/db2_05_00 on AIX
     - /opt/IBMdb2/V5.0 on HP-UX, Solaris, SCO UnixWare 7, and SGI.
   - On other platforms, see the RELEASE.TXT file. This file is located in the directory where the product is installed.

## Viewing Online Books

The manuals included with this product are in Hypertext Markup Language (HTML) softcopy format. Softcopy format enables you to search or browse the information, and provides hypertext links to related information. It also makes it easier to share the library across your site.

You can use any HTML Version 3.2-compliant browser to view the online books.

To view online books:

- If you are running DB2 administration tools, use the Information Center. See “Information Center” on page 183 for details.
- Use the open file function of your Web browser. The page you open contains descriptions of and links to DB2 books:
  - On UNIX-based platforms, open the following page:

    ```
    file:/INSTHOME/sqllib/doc/%L/html/index.htm
    ```

    where *%L* is the locale name.
  - On other platforms, open the following page:

    ```
    sqllib\doc\html\index.htm
    ```

    The path is located on the drive where DB2 is installed.

    You can also open the page by double-clicking on the **DB2 Online Books** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

    **Note:** The **DB2 Online Books** icon is only available if you do not install the Information Center.

## Setting up a Document Server

By default the DB2 information is installed on your local system. This means that each person who needs access to the DB2 information must install the same files. To have the DB2 information stored in a single location, use the following instructions:

1. Copy all files and sub-directories from \sqllib\doc\html on your local system to a web server. Each book has its own sub-directory containing all the necessary

HTML and GIF files that make up the book. Ensure that the directory structure remains the same.

2. Configure the web server to look for the files in the new location. For information, see Setting up DB2 Online Documentation on a Web Server at:

   `http://www.software.ibm.com/data/pubs/papers/db2html.html`

3. If you are using the Java version of the Information Center, you can specify a base URL for all HTML files. You should use the URL for the list of books.
4. Once you are able to view the book files, you should bookmark commonly viewed topics such as:
   - List of books
   - Tables of contents of frequently used books
   - Frequently referenced articles like the ALTER TABLE topic
   - Search form.

For information about setting up a search, see the *What's New* book.

## Searching Online Books

To search for information in the HTML books, you can do the following:

- Click on **Search the DB2 Books** at the bottom of any page in the HTML books. Use the search form to find a specific topic.
- Click on **Index** at the bottom of any page in an HTML book. Use the Index to find a specific topic in the book.
- Display the Table of Contents or Index of the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. See “Information Center” on page 183 for details.

## Printing the PostScript Books

If you prefer to have printed copies of the manuals, you can decompress and print PostScript versions. For the file name of each book in the library, see the table in “DB2 Books” on page 177.

**Note:** Specify the full path name for the file you intend to print.

On OS/2 and Windows platforms:

1. Copy the compressed PostScript files to a hard drive on your system. The files have a file extension of .exe and are located in the x:\doc*language*\books\ps directory, where x: is the letter representing the CD-ROM drive and *language* is the two-character country code that represents your language (for example, EN for English).
2. Decompress the file that corresponds to the book that you want. The result from this step is a printable PostScript file with a file extension of .psz.

3. Ensure that your default printer is a PostScript printer capable of printing Level 1 (or equivalent) files.
4. Enter the following command from a command line:

   ```
   print filename.psz
   ```

On UNIX-based platforms:

1. Mount the CD-ROM. Refer to your *Quick Beginnings* manual for the procedures to mount the CD-ROM.
2. Change to /cdrom/doc/%L/ps directory on the CD-ROM, where */cdrom* is the mount point of the CD-ROM and *%L* is the name of the desired locale. The manuals will be installed in the previously-mentioned directory with file names ending with .ps.Z.
3. Decompress and print the manual you require using the following command:
   - For AIX:

     ```
     zcat filename | qprt -P PSPrinter_queue
     ```

   - For HP-UX, Solaris, or SCO UnixWare 7:

     ```
     zcat filename | lp -d PSPrinter_queue
     ```

   - For Silicon Graphics IRIX and SINIX:

     ```
     zcat < filename | lp -d PSPrinter_queue
     ```

   where *filename* is the name of the full path name and extension of the compressed PostScript file and *PSprinter_queue* is the name of the PostScript printer queue.

   For example, to print the English version of *Quick Beginnings for UNIX* on AIX, you can use the following command:

   ```
   zcat /cdrom/doc/en/ps/db2ixe50.ps.Z | qprt -P ps1
   ```

## Ordering the Printed DB2 Books

You can order the printed DB2 manuals either as a set, or individually. There are two sets of books available. The form number for the entire set of DB2 books is SBOF-8915-00. The form number for the books listed under the heading "Cross-Platform Books" is SBOF-8914-00.

**Note:** These form numbers only apply if you are ordering books that are printed in the English language.

You can also order books individually by the form number listed in “DB2 Books” on page 177. To order printed versions, contact your IBM authorized dealer or marketing representative, or phone 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.

## Information Center

The Information Center provides quick access to DB2 product information. You must install the DB2 administration tools to obtain the Information Center.

Depending on your system, you can access the Information Center from the:

- Main product folder
- Toolbar in the Control Center
- Windows Start menu
- Help menu of the Control Center
- **db2ic** command.

The Information Center provides the following kinds of information. Click on the appropriate tab to look at the information:

| | |
|---|---|
| **Tasks** | Lists tasks you can perform using DB2. |
| **Reference** | Lists DB2 reference information, such as keywords, commands, and APIs. |
| **Books** | Lists DB2 books. |
| **Troubleshooting** | Lists categories of error messages and their recovery actions. |
| **Sample Programs** | Lists sample programs that come with the DB2 Software Developer's Kit. If the Software Developer's Kit is not installed, this tab is not displayed. |
| **Web** | Lists DB2 information on the World Wide Web. To access this information, you must have a connection to the Web from your system. |

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the kind of information you select.

The Information Center provides some search capabilities so you can look for specific topics, and filter capabilities to limit the scope of your searches.

For a full text search, follow the Search DB2 Books link in each HTML file, or use the search feature of the help viewer.

The HTML search server is usually started automatically. If a search in the HTML information does not work, you may have to start the search server via its icon on the Windows or OS/2 desktop.

Refer to the release notes if you experience any other problems when searching the HTML information.

# Appendix E. Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the

IBM Director of Licensing,
IBM Corporation,
500 Columbus Avenue,
Thornwood, NY, 10594
USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Canada Limited
Department 071
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries:

© Copyright IBM Corp. 1993, 1998

ACF/VTAM
ADSTAR
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
CICS
C Set++
C/370
DATABASE 2
DatagLANce
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
Distributed Relational Database Architecture
DRDA
Extended Services
FFST
First Failure Support Technology
IBM
IMS
Lan Distance
MVS/ESA
MVS/XA
NetView
OS/400
OS/390
OS/2
PowerPC
QMF
RACF
RISC System/6000
SAA
SP
SQL/DS
SQL/400
S/370
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WIN-OS/2

## Trademarks of Other Companies

The following terms are trademarks or registered trademarks of the companies listed:

C-bus is a trademark of Corollary, Inc.

HP-UX is a trademark of Hewlett-Packard.

Java, HotJava, Solaris, Solstice, and Sun are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, Visual Basic, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

SCO is a trademark of The Santa Cruz Operation.

SINIX is a trademark of Siemens Nixdorf.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

# Index



© Copyright IBM Corp. 1993, 1998

# D



# E



# F



# H



# I



# J

# L



# M



# O



# P



# R



# S

# T



# U



# V



# W



# X

# Contacting IBM

This section lists ways you can get more information from IBM.

If you have a technical problem, please take the time to review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. Depending on the nature of your problem or concern, this guide will suggest information you can gather to help us to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

**Telephone**

If you live in the U.S.A., call one of the following numbers:

- 1-800-237-5511 to learn about available service options.
- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, see Appendix A of the IBM Software Support Handbook. You can access this document by accessing the following page:

`http://www.ibm.com/support/`

then performing a search using the keyword “handbook.”

Note that in some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

**World Wide Web**
http://www.software.ibm.com/data/
http://www.software.ibm.com/data/db2/library/

The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more. The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information. (Note that this information may be in English only.)

**Anonymous FTP Sites**
ftp.software.ibm.com

Log on as anonymous. In the directory /ps/products/db2, you can find demos, fixes, information, and tools concerning DB2 and many related products.

**Internet Newsgroups**
comp.databases.ibm-db2, bit.listserv.db2-l

These newsgroups are available for users to discuss their experiences with DB2 products.

**CompuServe**
**GO IBMDB2** to access the IBM DB2 Family forums

All DB2 products are supported through these forums.

To find out about the IBM Professional Certification Program for DB2 Universal Database, go to http://www.software.ibm.com/data/db2/db2tech/db2cert.html

© Copyright IBM Corp. 1993, 1998

Part Number: 04L9264

Printed in U.S.A.