IBM DB2 Universal Database Building Applications for UNIX** Environments Version 5

Document Number S10J-8161-00

IBM DB2 Universal Database

Building Applications for UNIX** Environments

Version 5

S10J-8161-00

IBM DB2 Universal Database

Building Applications for UNIX** Environments

Version 5

S10J-8161-00

Before using this information and the product it supports, be sure to read the general information under Appendix D, “Notices” on page 127.

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.

Contents

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 on the following UNIX operating systems:

- AIX
- HP-UX
- 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 the Microsoft Corporation's Open Database Connectivity (ODBC).
DB2 Application Programming Interfaces (APIs)	Use DB2 administrative APIs in your applications to create administrative programs.

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.

- *CLI Guide and Reference*

  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*

  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.

For a complete list of the DB2 documentation library, see Appendix C, “How the DB2 Library Is Structured” on page 117.

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. You may use embedded SQL, the DB2 CLI, Java applications or Java applets 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 any of the supported UNIX platforms 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 you need to develop your applications. The first two chapters contain common information for users who will be developing either embedded SQL, DB2 CLI, Java, 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, 5, and 6 gives detailed information for developing embedded SQL applications on one of the supported platforms. In addition, the DB2 API script file for each supported compiler in these chapters 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 7 contains common information for all those developing DB2 CLI applications. Chapter 8 contains common information for all those developing Java 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 6. A user who wanted to develop DB2 CLI applications on any of the platforms should read Chapters 1, 2, and 7. A user who wanted to develop Java applications or applets for DB2 on a supporting platform should read Chapters 1, 2, and 8.

Since DB2 API calls can be made from either the embedded SQL, CLI, or Java 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.

Please note that some of the common chapters contain sections that have information specific to each platform, such as Supported Software by Platform in Chapter 1 and Coding a Script File by Platform in Chapter 7.

This book contains the following chapters and appendices:

Chapter 1, About the DB2 Software Developer's Kit, describes the DB2 SDK. It lists the supported servers and software of each of the UNIX platforms supported by DB2 Universal Database Version 5. It also describes the sample programs.

Chapter 2, Setup, explains how to set up the client/server and programming environment before you use the DB2 SDK.

Chapter 3, Introduction to Embedded SQL Applications, shows you how to build programs that use embedded SQL statements.

Chapter 4, Building AIX Embedded SQL Applications, shows you how to build AIX programs that use embedded SQL statements.

Chapter 5, Building HP-UX Embedded SQL Applications, shows you how to build HP-UX programs that use embedded SQL statements.

Chapter 6, Building Solaris Embedded SQL Applications, shows you how to build Solaris programs that use embedded SQL statements.

Chapter 7, Building DB2 Call Level Interface (CLI) Applications, shows you how to build programs that use DB2 Call Level Interface function calls.

Chapter 8, Building Java Applications and Applets, shows you how to build DB2 programs in Java.

Appendix A, About Database Manager Instances, explains database manager instances and how to use them to manage databases.

Appendix B, Problem Determination, describes build and run-time problems you can encounter, and what sources of information you can use to resolve them.

Appendix C, How the DB2 Library Is Structured, describes the components of the library, including online help, SmartGuides, and books.

Appendix D, Notices, lists notices concerning IBM publications, and trademarks of IBM and other companies.

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:

- Coding examples and code fragments
- Commands
- Examples of output, similar to what is displayed by the system
- Examples of specific data values
- Examples of system messages
- File and directory names
- Information that you are instructed to type

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 19 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.
- 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.
- 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 documented APIs can be obtained by downloading either of the following files. On Compuserve, the file is located in the IBM DB2 Family Forum on CompuServe (**GO IBMDB2**). Once in this forum, get the file called `PREPAPI.TXT` from Library 1. This file must be downloaded in ASCII format. On the Internet, go to the anonymous FTP site **ps.boulder.ibm.com**. The file is called `prepapi.txt`, and is located in the directory `/ps/products/db2/info`. This file is in ASCII format. Refer to the *DB2 Solutions Directory* for other examples of IBM and third party providers. You can get the *Directory* from CompuServe in the IBMDB2 forum, or contact your IBM representative for a copy.
- SQL 92 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.

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 OS/2
- DB2 for AIX
- DB2 for Windows NT
- DB2 for HP-UX
- DB2 for Solaris
- DB2 for SINIX
- DB2 for SCO OpenServer
- Distributed Relational Database Architecture (DRDA)-compliant application servers, such as:
  - DB2 for MVS/ESA
  - DB2 for VSE & VM (formerly SQL/DS for VM and VSE)
  - DB2 for OS/400
  - 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.

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 specific *Quick Beginnings* book for any of these platforms for information on the communication products supported by that 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. **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. **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

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

C	IBM C for AIX Version 3.1
C/C++	IBM C Set++ for AIX Version 3.1
FORTRAN	IBM XL FORTRAN for AIX Version 3.2
COBOL	IBM COBOL Set for AIX Version 1.1, and Micro Focus COBOL Version 3.2.46 or later
REXX	IBM AIX REXX/6000 AISPO Product Number: 5764-057
Java	Java Development Kit (JDK) 1.1 for AIX from IBM
Basic	IBM VisualAge for Basic Version 1

HP-UX

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

HP-UX	Version 10.10 and later, and Patch Levels: PHCO_6134, PHKL_5837, PHKL_6133, PHKL_6189, PHKL_6273, PHSS_5956

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

C	HP C/HP-UX Version A.10.13, and Patch Level PHSS_5743
C++	HP-UX C++ Version A.10.03.60, and Patch Level PHSS_5883
FORTRAN	HP FORTRAN/9000 Version 10.0
COBOL	Micro Focus COBOL Version 3.2
Java	HP-UX Developer's Kit for Java Release 1.1 from Hewlett-Packard

Solaris

The DB2 SDK for Solaris supports the following operating system:

Solaris	Version 2.5.1 or later

The DB2 SDK for Solaris supports the following programming languages:

C	SPARCompiler C Version 3.0.1 or later
C++	SPARCompiler C++ Version 4.0.1 or later, and IBM C Set++ for Solaris Version 1.1.1
FORTRAN	SPARCompiler FORTRAN Version 3.0.1
COBOL	Micro Focus COBOL Version 3.2
Java	Java Development Kit (JDK) 1.1 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 5.

The sample programs providing examples of embedded SQL and DB2 API calls are shown in Table 2 on page 8. Command Line Processor (CLP) files provided by DB2 are shown in Table 3 on page 13.

Java sample programs are shown in Table 4 on page 14. Object Linking and Embedding (OLE) sample programs are shown in Table 5 on page 14. The sample programs demonstrating DB2 CLI calls are shown in Table 6 on page 15.

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

Table 1. Sample Program File Extensions and Locations

Language		CLI Programs	Programs with Embedded SQL	Programs without Embedded SQL
C	File Ext.	`.c`	`.sqc`	`.c`
	Directory	`samples/cli`	`samples/c`	`samples/c`
C++	File Ext.	`Not Applicable`	`.sqC` (UNIX) `.sqx`	`.C` (UNIX) `.cxx` (Intel)
	Directory	`Not Applicable`	`samples/cpp`	`samples/cpp`
COBOL	File Ext.	`Not Applicable`	`.sqb`	`.cbl`
	Directory	`Not Applicable`	`samples/cobol` `samples/cobol_mf`	`samples/cobol` `samples/cobol_mf`
FORTRAN	File Ext.	`Not Applicable`	`.sqf`	`.f` (UNIX) `.for` (OS/2)
	Directory	`Not Applicable`	`samples/fortran`	`samples/fortran`
REXX	File Ext.	`Not Applicable`	`.cmd`	`.cmd`
	Directory	`Not Applicable`	`samples/rexx`	`samples/rexx`
JAVA	File Ext.	`Not Applicable`	`Not Applicable`	`.java`
	Directory	`Not Applicable`	`Not Applicable`	`samples/java`
OLE	File Ext.	`Not Applicable`	`Not Applicable`	`Not Applicable`
	Directory	`samples\ole`	`Not Applicable`	`samples\ole`

Note:

Programs without SQL Denotes programs with no SQL statements in them (primarily programs using DB2 API functions).

Directory Delimiters On UNIX are /. On OS/2 and Windows platforms, are \.

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. 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.

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

Java Samples Are stored procedures and UDFs, as well as Java Database Connectivity (JDBC) applications and applets. Java samples are available on the AIX, HP-UX, Solaris, OS/2, Windows NT and Windows 95 platforms.

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.

The above table lists the supported languages within the specified programming paradigms. Not all sample programs have been ported to all the supported programming languages.

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 2 on page 8 and Table 6 on page 15, 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 2 on page 8 and Table 6 on page 15, 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 2 on page 8 and Table 6 on page 15, 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 2 on page 8 and Table 6 on page 15, 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.

Not all of the sample programs are available in all the supported programming languages.

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 for the following countries:

- France
- Italy
- Spain
- Finland
- Norway
- People's Republic of China

In Table 2 on page 8, '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 2 (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: `ASYNCHRONOUS LOG READ`
autoloader		A UNIX Korn shell script that prepares ftp scripts for data transfer from remote hosts and generates a temporary buffer space (FIFO or named pipes). It then starts `db2split` and invokes DB2 LOAD. In a partitioned environment, partitioning keys are used to determine the partition where the data resides. Therefore, data must pass through a *splitting* phase before it can be loaded at the correct partition. The entire *split and load* process can be accomplished by the `autoLoader` utility. It uses a system-defined hashing function to partition the data into as many output files as there are partitions in the nodegroup in which the table is defined. It then loads these output files concurrently across the set of partitions in the nodegroup.

backrest		Demonstrates the use of the following APIs:
		`BACKUP DATABASE`
		`RESTORE DATABASE`
		`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:
		`SET CLIENT`
		`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: `GET DATABASE CONFIGURATION DEFAULTS`
d_dbmcon		Demonstrates the use of the following API: `GET DATABASE MANAGER CONFIGURATION DEFAULTS`
da_manip	Yes	Provides a library of routines to manipulate SQLDAs and SQLVARs.

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

Sample Program Name	Embedded SQL	Program Description
db2mon		Demonstrates how to use the Database System Monitor APIs, and how to process the output data buffer returned from the Snapshot API.
db2uext2		Provides a sample log management user exit.
dbauth	Yes	Demonstrates the use of the following API: GET AUTHORIZATIONS
dbcat		Demonstrates the use of the following APIs: CATALOG DATABASE CLOSE DATABASE DIRECTORY SCAN GET NEXT DATABASE DIRECTORY ENTRY OPEN DATABASE DIRECTORY SCAN UNCATALOG DATABASE
dbcmt		Demonstrates the use of the following APIs: CHANGE DATABASE COMMENT
dbconf		Demonstrates the use of the following APIs: CREATE DATABASE DROP DATABASE GET DATABASE CONFIGURATION RESET DATABASE CONFIGURATION UPDATE DATABASE CONFIGURATION
dbinst		Demonstrates the use of the following APIs: ATTACH TO INSTANCE DETACH FROM INSTANCE GET INSTANCE
dbmconf		Demonstrates the use of the following APIs: GET DATABASE MANAGER CONFIGURATION RESET DATABASE MANAGER CONFIGURATION UPDATE DATABASE MANAGER CONFIGURATION
dbsnap		Demonstrates the use of the following API: DATABASE SYSTEM MONITOR SNAPSHOT
dbstart		Demonstrates the use of the following API: START DATABASE MANAGER
dbstat	Yes	Demonstrates the use of the following APIs: REORGANIZE TABLE RUN STATISTICS
dbstop		Demonstrates the use of the following APIs: FORCE USERS STOP DATABASE MANAGER
db_udcs		Demonstrates the use of the following APIs in order to simulate the collating behaviour of a DB2 for MVS/ESA or OS/390 CCSID 500 (EBCDIC International) collating sequence: CREATE DATABASE DROP DATABASE

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

Sample Program Name	Embedded SQL	Program Description
dcscat		Demonstrates the use of the following APIs: ADD DCS DIRECTORY ENTRY CLOSE DCS DIRECTORY SCAN GET DCS DIRECTORY ENTRY FOR DATABASE GET DCS DIRECTORY ENTRIES OPEN DCS DIRECTORY SCAN 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: CREATE DATABASE DROP DATABASE
dynamic	Yes	Demonstrates the use of a cursor using dynamic SQL.
ebcdicdb		Demonstrates the use of the following APIs in order to simulate the collating behaviour of a DB2 for MVS/ESA or OS/390 CCSID 037 (EBCDIC US English) collating sequence: CREATE DATABASE DROP DATABASE
expsamp	Yes	Demonstrates the use of the following APIs: EXPORT IMPORT 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: EXPORT 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.

Table 2 (Page 4 of 6). Sample Programs Showing Embedded SQL and APIs

Sample Program Name	Embedded SQL	Program Description
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.
loblocud		Demonstrates the use of LOB locators in a user-defined function.
lobval	Yes	Demonstrates the use of LOBs.
makeapi	Yes	Demonstrates the use of the following APIs: BIND PRECOMPILE PROGRAM START DATABASE MANAGER STOP DATABASE MANAGER
migrate		Demonstrates the use of the following API: MIGRATE DATABASE
monreset		Demonstrates the use of the following API: RESET DATABASE SYSTEM MONITOR DATA AREAS
monsz		Demonstrates the use of the following APIs: ESTIMATE DATABASE SYSTEM MONITOR BUFFER SIZE DATABASE SYSTEM MONITOR SNAPSHOT
nodecat		Demonstrates the use of the following APIs: CATALOG NODE CLOSE NODE DIRECTORY SCAN GET NEXT NODE DIRECTORY ENTRY OPEN NODE DIRECTORY SCAN 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.

Table 2 (Page 5 of 6). Sample Programs Showing Embedded SQL and APIs

Sample Program Name	Embedded SQL	Program Description
qload	Yes	Demonstrates the use of the following API: LOAD QUERY
rebind	Yes	Demonstrates the use of the following API: REBIND PACKAGE
rechist		Demonstrates the use of the following APIs: CLOSE RECOVERY HISTORY FILE SCAN GET NEXT RECOVERY HISTORY FILE ENTRY OPEN RECOVERY HISTORY FILE SCAN PRUNE RECOVERY HISTORY FILE ENTRY UPDATE RECOVERY HISTORY FILE ENTRY
recursql	Yes	Demonstrates the use of advanced SQL recursive queries.
regder		Demonstrates the use of the following APIs: REGISTER DEREGISTER
restart		Demonstrates the use of the following API: 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: SET ACCOUNTING STRING
setrundg		Demonstrates the use of the following API: SET RUNTIME DEGREE
static	Yes	Uses static SQL to retrieve information.
sws		Demonstrates the use of the following API: DATABASE MONITOR SWITCH
system		Demonstrates most of the system-specific calls.
tabinfo	Yes	Provides a library of routines for obtaining table and column information from the system tables and for accessing the information obtained.
tabscont		Demonstrates the use of the following APIs: TABLESPACE CONTAINER QUERY OPEN TABLESPACE CONTAINER QUERY FETCH TABLESPACE CONTAINER QUERY CLOSE TABLESPACE CONTAINER QUERY SET TABLESPACE CONTAINER QUERY

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

Sample Program Name	Embedded SQL	Program Description
tabspace		Demonstrates the use of the following APIs: TABLESPACE QUERY SINGLE TABLESPACE QUERY OPEN TABLESPACE QUERY FETCH TABLESPACE QUERY GET TABLESPACE STATISTICS 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.
tload	Yes	Demonstrates the use of the following APIs: EXPORT QUIESCE TABLESPACE FOR TABLES 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: GET ERROR MESSAGE GET SQLSTATE MESSAGE INSTALL SIGNAL HANDLER INTERRUPT 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 3 (Page 1 of 2). Command Line Processor (CLP) Sample Files.

Sample File Name	File Description
const.clp	Creates a table with a CHECK CONSTRAINT clause.
cte.clp	Demonstrates a common table expression. The equivalent sample program demonstrating this advanced SQL statement is tabsql.
flt.clp	Demonstrates a recursive query. The equivalent sample program demonstrating this advanced SQL statement is recursql.
join.clp	Demonstrates an outer join of tables. The equivalent sample program demonstrating this advanced SQL statement is joinsql.

Table 3 (Page 2 of 2). Command Line Processor (CLP) Sample Files.

Sample File Name	File Description
`stock.clp`	Demonstrates the use of triggers. The equivalent sample program demonstrating this advanced SQL statement is `trigsql`.
`testdata.clp`	Uses DB2 built-in functions such as RAND() and TRANSLATE() to populate a table with randomly generated test data.

Table 4. Java Sample Programs

Sample Program Name	Program Description
`DB2Appl.java`	A Java Database Connectivity (JDBC) application that queries the sample database using the invoking user's privileges.
`DB2Applt.java`	A Java Database Connectivity (JDBC) applet that queries the sample database using a user and server specified as applet parameters.
`DB2Applt.html`	An HTML file that embeds the `DB2Applt.java` applet sample program. 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 user-defined function (UDF) that demonstrates several tasks, including integer division, manipulation of Character Large OBjects (CLOBs), and the use of Java instance variables.
`samples.zip`	A file containing compiled `.class` files for all DB2 Java samples.

Table 5. 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 6 (Page 1 of 3). 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)
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

Table 6 (Page 2 of 3). Sample CLI Programs in DB2 Universal Database

Sample Program Name	Program Description
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	
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

Table 6 (Page 3 of 3). Sample CLI Programs in DB2 Universal Database

Sample Program Name	Program Description
v2sutil.h	samputil.h using old v2 functions
v2fetch.c	fetch.c using old v2 functions
v2xfetch.c	xfetch.c using old v2 functions

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 113 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* book for your UNIX platform.

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

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 install the database, see "Installing, Cataloging, and Binding the SAMPLE Database" on page 20

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. Each database manager instance has two files, `db2profile` and `db2cshrc`, which contain scripts to set the environment variable for that instance. The *Quick Beginnings* book provides general information about setting environment variables. This section provides specific instructions to set environment variables to access a database instance.

Run the script by entering:

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

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

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 in.

Installing, Cataloging, and Binding the SAMPLE Database

To use the examples in this book, you need to install 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.

Installing

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

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

1. Ensure the location of `db2sampl` (the program that installs 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 AIX, HP_UX, Solaris, SINIX, and SCO OpenServer, `db2sampl` is located in:

     $HOME`/sqllib/misc`

     where *$HOME* is the home directory of the DB2 instance owner.
   - On OS/2, Windows 95 and Windows NT, `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 install the SAMPLE database.
   - On AIX, HP_UX, Solaris, SINIX, and SCO OpenServer, you can do this for the Korn shell by entering:

     ```
     DB2INSTANCE=instance_name
     export DB2INSTANCE
     ```

     where *instance_name* is the name of the database instance.
   - On OS/2, Windows 95 and Windows NT, 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-based systems, this is a path, and would be entered as:

   `db2sampl` *path*

   On OS/2 or Windows-based systems, 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* book.

Installing on DRDA-Compliant Application Servers

If you want to run the sample programs against a DRDA-compliant application server, such as DB2 for MVS/ESA, you need to create a database that contains the sample STAFF and ORG 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. Install the SAMPLE database in a DB2 common server instance using `db2sampl`.
2. Connect to the SAMPLE database.
3. Export the ORG and STAFF tables to a file.
4. Connect to the DRDA-compliant database.
5. Create the ORG and STAFF tables.
6. Import the ORG and STAFF 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* 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* 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 installed 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 MVS/ESA
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* 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 25, and then the embedded SQL chapter for the platform you are using. If you are developing CLI applications, see Chapter 7, “Building DB2 Call Level Interface (CLI) Applications” on page 103. If you are developing Java applications, see Chapter 8, “Building Java Applications and Applets” on page 109. 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*.

Chapter 3. Introduction to Embedded SQL Applications

Each DB2 SDK includes sample programs that embed SQL statements. Chapters 4 through 6 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.

Note: The chapters on using embedded SQL show you just some of the script files. The directories that contain the sample programs contain all the script files, as well as a README file which may contain additional information about them.

Sections in these chapters also list the steps you can follow to build and run the sample programs shown in Table 7 on page 25 using the supported programming languages. The steps you follow might vary, depending on your environment:

Table 7 (Page 1 of 2). Sample Programs Referred to in Script Files

Sample Program Name	Program Description
`updat`	Demonstrates the use of static SQL to update a database.
`outsrv`	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), and then returns the filled SQLDA and the SQLCA status to the client program. The `outsrv` program runs on the database server, and must be built there.
`outcli`	Demonstrates stored procedures using the SQLDA structure. This is the client program of a client/server example. (The server program is called `outsrv`.) The 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.

Table 7 (Page 2 of 2). Sample Programs Referred to in Script Files	
Sample Program Name	**Program Description**
udf	Creates a library of User-Defined Functions (UDFs) made specifically for the SAMPLE database tables, but can be used with tables with compatible column types. (The sample program calludf uses the functions created by udf.) The udf program runs on the database server, and must be built there.
calludf	Demonstrates the library of User-Defined Functions (UDFs) created by udf for the SAMPLE database tables. The calludf program uses the functions created by udf.

The source files for these sample programs 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
FORTRAN	sqllib/samples/fortran

Note: Of the samples given in Table 7 on page 25, the C++ directory, sqllib/samples/cpp, contains only a C++ version of the updat program. The stored procedure script files documented for the C++ compilers use the C versions of the outsrv and outcli programs found in sqllib/samples/c. In addition, sqllib/samples/cpp contains object-oriented sample programs specific to C++. These programs use several class source files and CLP script files to construct and manipulate a credit database system. See the README file in the sqllib/samples/cpp directory for more information.

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 the modified programs using either the makefile or the script files to see if they work correctly. You can also build your own embedded SQL programs using these files.

"Sample Programs" on page 4 lists all of the sample programs. The *Embedded SQL Programming Guide* explains how the samples containing embedded SQL work; the *CLI Guide and Reference* explains how the samples containing CLI work; and the *API Reference* explains how the samples containing DB2 APIs 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`. Refer the *Quick Beginnings* book for instructions.
- 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 HP-UX enter:

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

  To include the directory on Solaris enter:

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

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

About Stored Procedures and User-Defined Functions (UDFs)

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. Unlike compilers on other UNIX platforms, 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 40.

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.

For more information about stored procedures and UDFs, refer to your compiler documentation. The AIX compiler documentation also has additional information on export files.

C++ Considerations for UDFs and Stored Procedures

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.

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 .
```

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.

Compile and Link Options for bldxlc	
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.sqc`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. See "Setting Your Environment" on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldxlc updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Run the program. If you built the `updat` sample program, enter:

   ```
   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 `outsrv.sqc` stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.

4. 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 31 for details.

To run the stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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 40 for a general discussion about compiler options and UDFs.

To build the user-defined function `udf`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Build the UDF by entering:

   ```
   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 not fenced UDFs.
3. 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 31 for details.

To run the UDF, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. 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 on AIX Version 4

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.

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.sqC`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to "Setting Your Environment" on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldcset updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following:

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Run the program. If you built the `updat` sample program, enter:

   ```
   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 `outsrv` stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Copy the C source file `outsrv.sqc` to `outsrv.sqC` so that it has the C++ file extension `.sqC`.
4. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.

5. 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. First, copy the C source file `outcli.sqc` to `outcli.sqC` so that it has the C++ file extension `.sqC`. Then you can build `outcli` using the `bldcset` script file. Refer to "IBM C Set++" on page 41 for details.

To run the stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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 on AIX Version 4

Multi-threaded applications on AIX Version 4 need to be compiled and linked with the `xlC_r` compiler instead of the `xlC` compiler.

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.sqf`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldxlf updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Run the program. If you built the `updat` sample program, enter:

   ```
   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 `outsrv.sqf` stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.

4. 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 `bldxlf` script file. Refer to "IBM XL FORTRAN for AIX" on page 46 for details.

To run the stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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.

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 different extents by the precompiler, and always in a case insensitive manner:

  ```
  @PROCESS            END           IMPLICIT      SUBROUTINE
  AUTOMATIC           ENDDO         INTEGER
  BLOCKDATA           ENDFILE       LOGICAL
  BYTE                ENDIF         PARAMETER
  CHARACTER           ENTRY         POINTER
  COMPLEX             FORMAT        PROGRAM
  DOUBLECOMPLEX       FUNCTION      REAL
  DOUPLEPRECISION     IF            STATIC
  ```

- 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

# 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.sqb`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldcob updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Run the program. If you built the `updat` sample program, enter:

   ```
   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 `outsrv.sqb` stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.

4. 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 52 for details.

To run the stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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.

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.
- The DB2 COPY files for IBM COBOL Set for AIX reside in `sqllib/include/cobol_a` under the database instance directory. 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.
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 with the database manager library.
`-ldb2gmf`	Link with the DB2 exception-handler library for M. F. 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.sqb`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to "Setting Your Environment" on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldmfcob updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Run the program. If you built the `updat` sample program, enter:

   ```
   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

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, and for the main entry point to the shared library.

```
#! /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

# Link the program using the export file $1.exp,
# creating a shared library called $1 with the main
# entry point $1.
cob -x -o $1 $1.o -Q -bE:$1.exp -Q "-e $1" -Q -bI:/usr/lpp/db2_05_00/lib/db2g.imp \
    -B static -ldb2gmf -L/usr/lpp/db2_05_00/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 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 procedures.
`-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.
`-B static`	Produce a statically-linked 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 with the DB2 exception-handler library for M. F. COBOL.
Refer to your compiler documentation for additional compiler options.	

To build the `outsrv.sqb` stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.

4. 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 `bldmfcob` script file. Refer to "Micro Focus COBOL" on page 58 for details.

To run the stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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.

Setting Up and Running REXX Programs

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.

HP-UX C/C++

The script files in this section are coded for C programs using the C compiler. To use C++ programs you need to use the C++ compiler. To do this, make the changes to the script files given in comments at the end of the files.

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.

```
#! /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 +e -I/opt/IBMdb2/v5.0/include -c util.c

# Compile the program.
cc -Aa +e -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 -ldb2 -lhppa

# Note: To use the C++ compiler, substitute the following steps.
# Precompile the program.
# db2 prep $1.sqC bindfile
# Compile the util.c error-checking utility.
# CC +a1 -I/opt/IBMdb2/v5.0/include -c util.c
# Compile the program.
# CC +a1 -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 -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).
`+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:	
`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 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.sqc`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldcc updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Run the program. If you built the `updat` sample program, enter:

   ```
   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 +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

# Note: to use the C++ compiler, substitute the following steps.
# Precompile the program.
# db2 prep $1.sqC bindfile
# Compile the program.
# CC +Z -I/opt/IBMdb2/v5.0/include -c $1.C
# Ensure the stored procedure is coded with extern "C".
```

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).
`+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 `outsrv.sqc` stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.

4. 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 that calls the stored procedure. You can build `outcli` using the `bldcc` file. Refer to "HP-UX C/C++" on page 65 for details.

To run the stored procedure, do the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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 +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

# Note: to use the C++ compiler, substitute the following step.
# Compile the program.
# CC +Z -I/opt/IBMdb2/v5.0/include -c $1.C
# Ensure the UDF is coded with extern "C".
```

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).
`+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 `udf`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Build the UDF by entering:

   ```
   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 not fenced UDFs.
3. 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` file. Refer to “HP-UX C/C++” on page 65 for details.

To run the UDF, do the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Run the sample calling application by entering:

   ```
   calludf
   ```

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

Multi-threaded Applications

Multi-threaded applications on HP-UX need to be linked with `libcma.sl`. Add `-lcma` to the end of the link command when building a multi-threaded 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 -I/opt/IBMdb2/v5.0/include -c util.f

# Compile the program.
f77 -w -c -I/opt/IBMdb2/v5.0/include $1.f

# Link the program.
f77 $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.
-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.
util.o	Include the object file for error checking.
-Wl,	The linker can use the path in -L to find the shared library.
-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.sqf, do the following:

1. Go to the window in which you set your environment variables by running db2profile. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldf77 updat
   ```

The result is an executable file updat. You can run the executable file against the SAMPLE database to see how it works by doing the following :

1. Go to the window in which you set your environment variables by running db2profile.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Run the program. If you built the updat sample program, enter:

   ```
   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 +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.
`+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 stored procedure `outsrv.sqf` do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.

4. 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 `outcli` that calls the stored procedure. You can build `outcli` using the `bldf77` script file. Refer to "HP FORTRAN/9000" on page 73 for details.

To run the stored procedure, do the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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
# 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 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 -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.

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.
`-lhppa`	Specify the HP PA-RISC library (required).
`-ldb2gmf`	Link to the DB2 library.
Refer to your compiler documentation for additional compiler options.	

To build the sample program `updat.sqb`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldmfcc updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Run the program. If you built the `updat` sample program, enter:

   ```
   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 checkerr.cbl

# Compile the program.
cob +Z -cx $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.

Compile and Link Options for bldmfsp	
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 with the DB2 shared library.
`-lhppa`	Specify the HP PA-RISC library (required).
`-ldb2gmf`	Link to the DB2 library.
`-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 `outsrv.sqb` stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.
4. 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 that calls the stored procedure. You can build `outcli` using the `bldmfcc` file. Refer to “Micro Focus COBOL” on page 78 for details.

To run the stored procedure, do the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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 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.

SPARCompiler C/C++

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 this compiler, just comment out the Sparcompiler compile and link steps and uncomment those for C Set++.

The script files are coded for C programs using a C compiler. To use C++ programs you need to use a C++ compiler. To do this, make the changes to the script files given in comments at the end of the files.

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.

```
#! /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 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

# To compile C++ Programs.
# Change 'cc' to 'CC' in the compile and link steps or 'xlc' to 'xlC' for IBM
# C Set++.
# Change '.sqc' to '.sqC' in the precompile step and '.c' to '.C' in the
# compile step.
```

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.sqc`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldcc updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Run the program. If you built the `updat` sample program, enter:

   ```
   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 -e$1

# Using the IBM C Set++ 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 -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

# To compile C++ Programs, change 'cc' to 'CC' in the compile and link steps or
# 'xlc' to 'xlC' for IBM C Set++.  Change '.sqc' to '.sqC' in the precompile step.
# In the compile step change '.c' to '.C' and do not use '-Xa'. In the link step
# do not use '-e$1'. Ensure the stored procedure is coded with extern "C".
```

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:	
`ld`	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 `outsrv.sqc` stored procedure, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.

4. 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` file. Refer to "SPARCompiler C/C++" on page 85 for details.

To run the stored procedure, do the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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 Set++ 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

# To compile C++ Programs, change 'cc' to 'CC' in the compile and link steps
# or 'xlc' to 'xlC' for IBM C Set++. In the precompile step change '.sqc' to '.sqC'.
# In the compile step change '.c' to '.C' and do not use '-Xa'.
# Ensure the UDF is coded with extern "C".
```

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 udf, do the following:

1. Go to the window in which you set your environment variables by running db2profile. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Build the UDF by entering:

   ```
   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 not fenced UDFs.
3. 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 file. Refer to “SPARCompiler C/C++” on page 85 for details.

To run the UDF, do the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Run the sample calling application by entering:

   ```
   calludf
   ```

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

Multi-threaded Applications

Multi-threaded applications on Solaris need to be compiled with the `-D_REENTRANT` flag, and linked with `libthread.so`. Add `-D_REENTRANT` following the `cc` or `xlc` compile command, and add `-lthread` to the end of the link command, when building a multi-threaded application.

SPARCompiler FORTRAN

Note: Before using the SPARCompiler FORTRAN compiler, make sure `db2ln` was run when DB2 was installed. This command links the DB2 header files into `/usr/include`. If this has not been done, SPARCompiler FORTRAN may not be able to find the header files because it does not have an environment variable or a switch such as `-l` to locate them.

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 -w -c util.f

# Compile the program.
f77 -w -c $1.f

# Link the program.
f77 $1.o util.o -L/opt/IBMdb2/v5.0/lib -R/opt/IBMdb2/v5.0/lib \
    -R/opt/SUNWspro/lib -ldb2 -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.

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`.
`-R`*path*	Specify the compiler-specific library. For example: `-R/opt/SUNWspro/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.sqf`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldf77 updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Run the program. If you built the `updat` sample program, enter:

   ```
   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 -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 and Link Options for bldf77sp	
The script file contains the following compile options:	
`f77`	The FORTRAN compiler.
`-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 stored procedure `outsrv.sqf` do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```
3. Build the stored procedure, connecting to the SAMPLE database, by entering:

   ```
   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 not fenced stored procedures.
4. 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 `outcli` that calls the stored procedure. You can build `outcli` using the `bldf77` script file. Refer to “SPARCompiler FORTRAN” on page 94 for details.

To run the stored procedure, do the following :

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. 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 library.
Refer to your compiler documentation for additional compiler options.	

To build the sample program `updat.sqb`, do the following:

1. Go to the window in which you set your environment variables by running `db2profile`. Refer to "Setting Your Environment" on page 19 if you need more information.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldmfcc updat
   ```

The result is an executable file `updat`. You can run the executable file against the SAMPLE database to see how it works by doing the following:

1. Go to the window in which you set your environment variables by running `db2profile`.
2. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

3. Run the program. If you built the `updat` sample program, enter:

   ```
   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 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 in your applications using DB2 CLI function calls. You can also use stored procedures with 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 by 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 6 on page 15.

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.

Coding a Script File 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 and the compiler options for the platform you are using. Then go to "Building and Running a CLI Program" on page 106 for the steps to follow in order to build and run the program.

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

# Link the program.
xlc -o clisampl clisampl.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.
`-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.	

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.

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 +e -I/opt/IBMdb2/v5.0/include -c clisampl.c

# Link the program.
cc -o clisampl clisampl.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.
`+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.
`-o` *filename*	Specify the name of the executable program.
`-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 need to be linked with `libcma.sl`. Add `-lcma` to the end of the link command when building a multi-threaded application.

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

# Link the program.
cc -o clisampl clisampl.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.
`-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 on Solaris need to be compiled with the `-D_REENTRANT` flag, and linked with `libthread.so`. Add `-D_REENTRANT` following the `cc` or `xlc` compile command, and add `-lthread` to the end of the link command, when building a multi-threaded application.

Building and Running a CLI Program

To build the sample program `clisampl`:

1. Go to the window in which you set your environment variables. In order to do this, run `db2profile`. Refer to “Setting Your Environment” on page 19 if you need more information.
2. Build the sample program by entering:

   ```
   clibld
   ```

The result is an executable file `clisampl`. You can run the executable file to see how it works. 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* book for your platform.

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 8. Building Java Applications and Applets

You can access DB2 databases through the appropriate port of the Java Development Kit (JDK) Version 1.1 on AIX, Solaris, or HP-UX. The JDK includes Java Database Connectivity (JDBC) support to build the following types of Java programs:

- JDBC applications, which rely on the DB2 Client Application Enabler (CAE) to connect to DB2.
- JDBC applets, that do not require any other DB2 component code on the client.

See the Web Page at `http://www.software.ibm.com/data/db2/java` for more information.

DB2 also provides support for user-defined functions (UDFs) and stored procedures created in Java.

For more detailed 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 and applets, and creating Java UDFs and stored procedures.

This chapter presents information to set up your environment for running Java applications on AIX, HP-UX and Solaris. This is followed by sections explaining how to build and run a DB2 JDBC application and a DB2 JDBC applet.

Setting Up the AIX Environment

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 2.1.0 or later.

To run JDBC programs on AIX, the following environment variables must be set correctly. You must 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`

Setting Up the HP-UX Environment

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 2.1.0 or later.

To run JDBC programs on HP-UX, the following environment variables must be set correctly. You must 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`

Setting Up the Solaris Environment

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 for Solaris 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 2.1.0 or later.

To run JDBC programs on Solaris, the following environment variables must be set correctly. You must 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`

Building and Running a JDBC Application

You do not precompile or bind Java programs.

Start your application from the desktop or command line, like any other application. The DB2 JDBC driver handles the JDBC API calls from your application and uses the CAE to communicate the requests to the server and receive the results.

A sample application, `DB2Appl.java`, is provided in the `sqllib/samples/java` directory. If you installed the DB2 SAMPLE database, you can run the sample by changing to the `sqllib/samples/java` directory, and doing the following:

1. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

2. Enter:

   ```
   javac DB2Appl.java
   java DB2Appl
   ```

As an alternative to step 2 above, you can use the pre-compiled version of `DB2Appl.java` in `samples.zip`. To do this, ensure CLASSPATH also includes the file `sqllib/samples/java/samples.zip`. Then, run the java interpreter on the application by entering:

```
java DB2Appl
```

Building and Running a JDBC Applet

Like other Java applets, JDBC applets are distributed over the Web. Typically, you would embed the applet in an HTML page, as the following steps demonstate. These steps assume that the appropriate port for your platform of the Java Development Kit (JDK) Version 1.1, and at least the client package of DB2, are installed and working.

1. Run the Java compiler ("javac") on your applet's Java source. For the basic JDBC applet sample, `DB2Applt.java`, DB2 provides a compiled version in `sqllib/samples/java/samples.zip` so you may omit this step.
2. Construct an HTML file that will embed the applet. Unless you hard-code this into the applet source, you may opt to include applet parameters to identify the JDBC applet server, user ID and password information. For `DB2Applt.java`, DB2 provides the file, `DB2Applt.html`.
3. For a larger JDBC applet that consists of several Java classes, you may choose to package all its classes into a single ZIP file. In this case, add your ZIP file into the `archive` parameter in the "applet" tag. For details, see the JDK Version 1.1 documentation.
4. Along with the DB2 client package, you must install JDBC applets on a Web server. If necessary, configure the DB2 client package by cataloging remote nodes and/or databases.
5. Pick an unused TCP/IP port number for use by the JDBC applet server. This is **not** the TCP/IP port used by the `svcename` of a DB2 server. Start the server by the `db2jstrt` program. For example, if you designate port 6789 for JDBC access to your DB2 instance, enter `db2jstrt 6789` to start the JDBC applet server.
6. Copy the embedding HTML file, the JDBC applet's `.class` or ZIP file, and the `sqllib/java/db2java.zip` file into a directory under the Web browser's document root. For `DB2Applt.java`, copy `sqllib/samples/java/samples.zip`, `sqllib/samples/java/DB2Applt.html`, and `sqllib/java/db2java.zip`. You will need to customize this copy of the `DB2Applt.html` file to identify your Web server, JDBC applet server port number, user ID and password.
7. You may wish to place the ZIP files 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.
8. To run JDBC applets you must install a Web browser, or other compatible applet viewer, capable of running programs compiled with the JDK Version 1.1.

9. In the Web browser, open the URL identifying the HTML file at the Web server. The JDBC applet and the JDBC applet driver will be downloaded and executed inside the browser.

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.

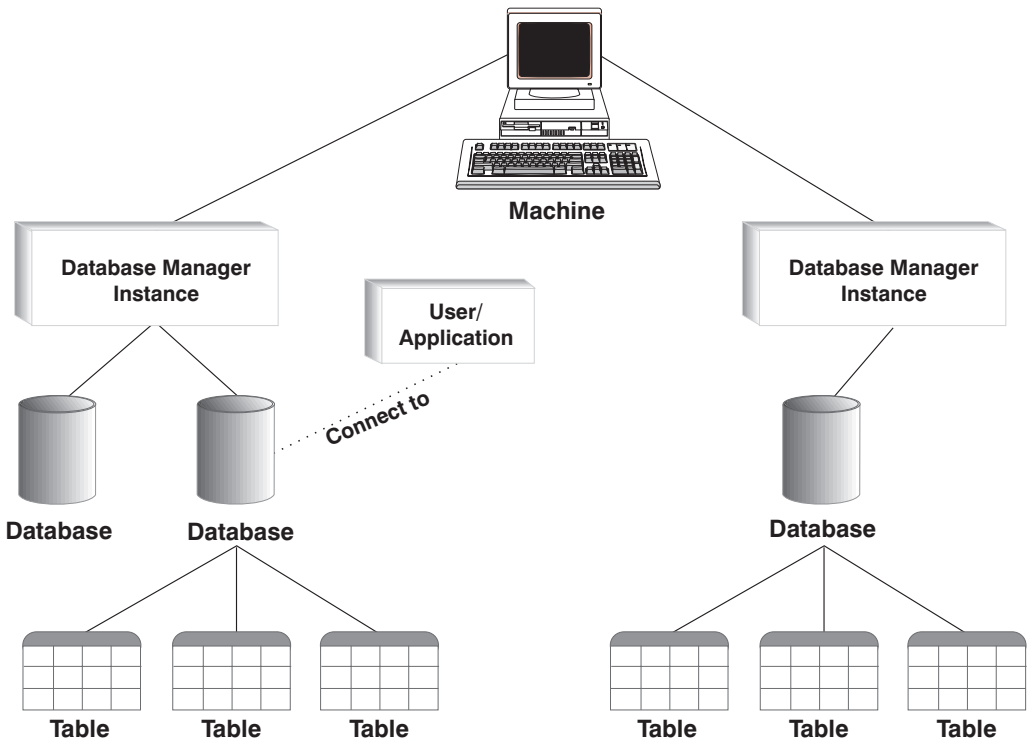

Figure 1. Database Manager Instances

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.

You need to know the following for each instance you use:

instance name

For AIX, HP-UX, Solaris, SINIX, and SCO OpenServer, 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 AIX, HP-UX, Solaris, SINIX, and SCO OpenServer, 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. Problem Determination

You can 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.

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
The database manager, database administration utility, installation and configuration process, and the command line processor generate online error messages. Each of these messages has a unique prefix as follows:

Prefix	Source
SQL	Database manager
DBA	Database Director
DBI	Installation and configuration

DB2 Command line processor

A four or five digit message number follows the prefix. A single letter follows the message number indicating the severity of the error.

You can use the command line processor to see the help for the message. Type:

```
db2 "? xxxnnnn"
```

where `xxx` is the message prefix, and `nnnn` is the message number. Include the quotes.

Refer to the *Message Reference* for more information about online error messages.

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 C. How the DB2 Library Is Structured

The DB2 Universal Database library consists of SmartGuides, online help, and books. This section describes the information that is provided, and how to access it.

To help you access product information online, DB2 provides the Information Center on OS/2, Windows 95, and the Windows NT operating systems. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web. “About the Information Center” on page 124 has more details.

SmartGuides

SmartGuides help you complete some administration tasks by taking you through each task one step at a time. SmartGuides are available on OS/2, Windows 95, and the Windows NT operating systems. The following table lists the SmartGuides.

SmartGuide	Helps you to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click on **Add**.
Create Database	Create a database, and to perform some basic configuration tasks.	From the Control Center, click with the right mouse button on the **Databases** icon and select **Create**->**New**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, click with the right mouse button on the database you want to tune and select **Configure performance**.
Backup Database	Determine, create, and schedule a backup plan.	From the Control Center, click with the right mouse button on the database you want to backup and select **Backup**->**Database using SmartGuide**.
Restore Database	Recover a database after a failure. It helps you understand which backup to use, and which logs to replay.	From the Control Center, click with the right mouse button on the database you want to restore and select **Restore**->**Database using SmartGuide**.
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, click with the right mouse button on the **Tables** icon and select **Create**->**Table using SmartGuide**.
Create Table Space	Create a new table space.	From the Control Center, click with the right mouse button on the **Table spaces** icon and select **Create**->**Table space using SmartGuide**.

Online Help

Online help is available with all DB2 components. The following table describes the various types of help.

Type of Help	Contents	How to Access...
Command Help	Explains the syntax of commands in the command line processor.	From the command line processor in interactive mode, enter: **?** *command* where *command* is a keyword or the entire command. For example, **?** *catalog* displays help for all the CATALOG commands, whereas **?** *catalog database* displays help for the CATALOG DATABASE command.
Control Center Help	Explains the tasks you can perform in a window or notebook. The help includes prerequisite information you need to know, and describes how to use the window or notebook controls.	From a window or notebook, click on the **Help** push button or press the F1 key.
Message Help	Describes the cause of a message number, and any action you should take.	From the command line processor in interactive mode, enter: **?** *message number* where *message number* is a valid message number. For example, **?** *SQL30081* displays help about the SQL30081 message. To view message help one screen at a time, enter: **?** *XXXnnnnn* **| more** where *XXX* is the message prefix, such as SQL, and *nnnnn* is the message number, such as 30081. To save message help in a file, enter: **?** *XXXnnnnn* > *filename.ext* where *filename.ext* is the file where you want to save the message help. **Note:** On UNIX-based systems, enter: **\?** *XXXnnnnn* **| more** or **\?** *XXXnnnnn* > *filename.ext*

Type of Help	Contents	How to Access...
SQL Help	Explains the syntax of SQL statements.	From the command line processor in interactive mode, enter: **help** *statement* where *statement* is an SQL statement. For example, **help** *SELECT* displays help about the SELECT statement.
SQLSTATE Help	Explains SQL states and class codes.	From the command line processor in interactive mode, enter: **?** *sqlstate* or **?** *class-code* where *sqlstate* is a valid five digit SQL state and *class-code* is a valid two digit class code. For example, **?** *08003* displays help for the 08003 SQL state, whereas **?** *08* displays help for the 08 class code.

DB2 Books

The table in this section lists the DB2 books. They are divided into two groups:

- Cross-platform books: These books are for DB2 on any of the supported platforms.
- Platform-specific books: These books are for DB2 on a specific platform. For example, there is a separate *Quick Beginnings* book for DB2 on OS/2, Windows NT, and UNIX-based operating systems.

Most books are available in HTML and PostScript format, and in hardcopy that you can order from IBM. The exceptions are noted in the table.

You can obtain DB2 books and access information in a variety of different ways:

View To view an HTML book, you can do the following:

- If you are running DB2 administration tools on OS/2, Windows 95, or the Windows NT operating systems, you can use the Information Center. “About the Information Center” on page 124 has more details.
- Use the open file function of the Web browser supplied by DB2 (or one of your own) to open the following page:

  sqllib/doc/html/index.htm

  The page contains descriptions of and links to the DB2 books. The path is located on the drive where DB2 is installed.

  You can also open the page by double-clicking on the **DB2 Online Books** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

Search To search for information in the HTML books, you can do the following:

- Click on **Search the DB2 Books** at the bottom of any page in the HTML books. Use the search form to find a specific topic.
- Click on **Index** at the bottom of any page in an HTML book. Use the Index to find a specific topic in the book.
- Display the Table of Contents or Index of the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. “About the Information Center” on page 124 has more details.

Print To print a book on a PostScript printer, look for the file name shown in the table.

Order To order a hardcopy book from IBM, use the form number.

Book Name	Book Description	Form Number File Name
	Cross-Platform Books	
Administration Getting Started	Introduces basic DB2 database administration concepts and tasks, and walks you through the primary administrative tasks.	S10J-8154 db2k0x50
Administration Guide	Contains information required to design, implement, and maintain a database to be accessed either locally or in a client/server environment.	S10J-8157 db2d0x50
API Reference	Describes the DB2 application programming interfaces (APIs) and data structures you can use to manage your databases. Explains how to call APIs from your applications.	S10J-8167 db2b0x50
CLI Guide and Reference	Explains how to develop applications that access DB2 databases using the DB2 Call Level Interface, a callable SQL interface that is compatible with the Microsoft ODBC specification.	S10J-8159 db2l0x50
Command Reference	Explains how to use the command line processor, and describes the DB2 commands you can use to manage your database.	S10J-8166 db2n0x50
DB2 Connect Enterprise Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Connect Enterprise Edition. Also contains installation and setup information for all supported clients.	S10J-7888 db2cyx50
DB2 Connect Personal Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Connect Personal Edition.	S10J-8162 db2c1x50
DB2 Connect User's Guide	Provides concepts, programming and general using information about the DB2 Connect products.	S10J-8163 db2c0x50
DB2 Connectivity Supplement	Provides setup and reference information for customers who want to use DB2 for AS/400, DB2 for OS/390, DB2 for MVS, or DB2 for VM as DRDA Application Requesters with DB2 Universal Database servers, and customers who want to use DRDA Application Servers with DB2 Connect (formerly DDCS) application requesters. **Note:** Available in HTML and PostScript formats only.	No form number db2h1x50
Embedded SQL Programming Guide	Explains how to develop applications that access DB2 databases using embedded SQL, and includes discussions about programming techniques and performance considerations.	S10J-8158 db2a0x50
Glossary	Provides a comprehensive list of all DB2 terms and definitions. **Note:** Available in HTML format only.	No form number db2t0x50

Book Name	Book Description	Form Number File Name
Installing and Configuring DB2 Clients	Provides installation and setup information for all DB2 Client Application Enablers and DB2 Software Developer's Kits. **Note:** Available in HTML and PostScript formats only.	No form number db2iyx50
Master Index	Contains a cross reference to the major topics covered in the DB2 library. **Note:** Available in PostScript format and hardcopy only.	S10J-8170 db2w0x50
Message Reference	Lists messages and codes issued by DB2, and describes the actions you should take.	S10J-8168 db2m0x50
Replication Guide and Reference	Provides planning, configuring, administering, and using information for the IBM Replication tools supplied with DB2.	S95H-0999 db2e0x50
Road Map to DB2 Programming	Introduces the different ways your applications can access DB2, describes key DB2 features you can use in your applications, and points to detailed sources of information for DB2 programming.	S10J-8155 db2u0x50
SQL Getting Started	Introduces SQL concepts, and provides examples for many constructs and tasks.	S10J-8156 db2y0x50
SQL Reference	Describes SQL syntax, semantics, and the rules of the language. Also includes information about release-to-release incompatibilities, product limits, and catalog views.	S10J-8165 db2s0x50
System Monitor Guide and Reference	Describes how to collect different kinds of information about your database and the database manager. Explains how you can use the information to understand database activity, improve performance, and determine the cause of problems.	S10J-8164 db2f0x50
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	S10J-8169 db2p0x50
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database. **Note:** Available in HTML and PostScript formats only.	No form number db2q0x50
	Platform-Specific Books	
Building Applications for UNIX Environments	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a UNIX system.	S10J-8161 db2axx50
Building Applications for Windows and OS/2 Environments	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Windows or OS/2 system.	S10J-8160 db2a1x50

Book Name	Book Description	Form Number File Name
DB2 Extended Enterprise Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Universal Database Extended Enterprise Edition for AIX.	S72H-9620 db2v3x50
DB2 Personal Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Universal Database Personal Edition on OS/2, Windows 95, and the Windows NT operating systems.	S10J-8150 db2i1x50
DB2 SDK for Macintosh Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Macintosh system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S50H-0528 sqla7x02
DB2 SDK for SCO OpenServer Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SCO OpenServer system. **Note:** Available for DB2 Version 2.1.2 only.	S89H-3242 sqla9x02
DB2 SDK for Silicon Graphics IRIX Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Silicon Graphics system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S89H-4032 sqlaax02
DB2 SDK for SINIX Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SINIX system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S50H-0530 sqla8x00
Quick Beginnings for OS/2	Provides planning, installing, configuring, and using information for DB2 Universal Database on OS/2. Also contains installing and setup information for all supported clients.	S10J-8147 db2i2x50
Quick Beginnings for UNIX	Provides planning, installing, configuring, and using information for DB2 Universal Database on UNIX-based platforms. Also contains installing and setup information for all supported clients.	S10J-8148 db2ixx50
Quick Beginnings for Windows NT	Provides planning, installing, configuring, and using information for DB2 Universal Database on the Windows NT operating system. Also contains installing and setup information for all supported clients.	S10J-8149 db2i6x50

Notes:

1. The character in the sixth position of the file name indicates the language of a book. For example, the file name `db2d0e50` indicates that the *Administration Guide* is in English. The following letters are used in the file names to indicate the language of a book:

Language	Identifier	Language	Identifier
Brazilian Portuguese	B	Hungarian	H
Bulgarian	U	Italian	I
Czech	X	Norwegian	N
Danish	D	Polish	P
English	E	Russian	R
Finnish	Y	Slovenian	L
French	F	Spanish	Z
German	G	Swedish	S

2. For late breaking information that could not be included in the DB2 books, see the README file. Each DB2 product includes a README file which you can find in the directory where the product is installed.

About the Information Center

The Information Center provides quick access to DB2 product information. The Information Center is available on OS/2, Windows 95, and the Windows NT operating systems. You must install the DB2 administration tools to see the Information Center.

Depending on your system, you can access the Information Center from the:

- Main product folder
- Toolbar in the Control Center
- Windows Start menu.

The Information Center provides the following kinds of information. Click on the appropriate tab to look at the information:

Tasks Lists tasks you can perform using DB2.

Reference Lists DB2 reference information, such as keywords, commands, and APIs.

Books Lists DB2 books.

Troubleshooting Lists categories of error messages and their recovery actions.

Sample Programs Lists sample programs that come with the DB2 Software Developer's Kit. If the Software Developer's Kit is not installed, this tab is not displayed.

Web Lists DB2 information on the World Wide Web. To access this information, you must have a connection to the Web from your system.

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the kind of information you select.

The Information Center provides search capabilities so you can look for specific topics, and filter capabilities to limit the scope of your searches.

Appendix D. Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the

IBM Director of Licensing,
IBM Corporation,
500 Columbus Avenue,
Thornwood, NY, 10594
USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Canada Limited
Department 071
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries:

ACF/VTAM
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 and HotJava are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Solaris is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Index

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 selecting the "Roadmap to IBM Support" item at: http://www.ibm.com/support/.

Note that in some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

World Wide Web
http://www.software.ibm.com/data/
http://www.software.ibm.com/data/db2/library/

The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more. The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information. (Note that this information may be in English only.)

Anonymous FTP Sites
ftp.software.ibm.com

Log on as anonymous. In the directory /ps/products/db2, you can find demos, fixes, information, and tools concerning DB2 and many related products.

Internet Newsgroups
comp.databases.ibm-db2, bit.listserv.db2-l

These newsgroups are available for users to discuss their experiences with DB2 products.

CompuServe
GO IBMDB2 to access the IBM DB2 Family forums

All DB2 products are supported through these forums.

To find out about the IBM Professional Certification Program for DB2 Universal Database, go to http://www.software.ibm.com/data/db2/db2tech/db2cert.html

Part Number: 10J8161

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.