IBM DB2 Universal Database Building Applications for Windows** and OS/2 Environments Version 5

Document Number S10J-8160-00

IBM DB2 Universal Database

Building Applications for Windows** and OS/2 Environments

Version 5

S10J-8160-00

IBM DB2 Universal Database

Building Applications for Windows** and OS/2 Environments

Version 5

S10J-8160-00

Before using this information and the product it supports, be sure to read the general information under Appendix D, “Notices” on page 151.

Contents

About This Book

This book explains how to build applications using the DB2 Software Developer's Kits (DB2 SDKs) for the following operating systems:

- Windows NT
- Windows 95
- Windows 3.1
- OS/2

Note: Whenever this book mentions Windows NT or Windows 95, both Windows NT and Windows 95 are implied, except in the case of Systems Network Architecture (SNA) support or DB2 Connect, formerly known as Distributed Database Connection Services (DDCS). These are supported on Windows NT only.

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, "Accessing DB2 Databases".

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*

  Discusses 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, refer to Appendix C, “How the DB2 Library Is Structured” on page 141.

Who Should Use This Book

You should use this book if you want to develop applications on one of the currently supported Windows or OS/2 platforms. 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 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 batch or command file for each supported compiler in these chapters is noted after the first embedded SQL batch or command 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 OS/2 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 Windows NT or OS/2 should read Chapters 1, 2, and 8.

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 Building and Running a DB2 CLI Application 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 the software of each of the Windows and OS/2 platforms currently supported by DB2. 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 Windows NT and Windows 95 Embedded SQL Applications, shows you how to build Windows NT and Windows 95 programs that use embedded SQL statements.

Chapter 5, Building Windows 3.1 Embedded SQL Applications, shows you how to build Windows 3.1 programs that use embedded SQL statements.

Chapter 6, Building OS/2 Embedded SQL Applications, shows you how to build OS/2 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 Windows and OS/2 platforms covered in this book provide the following:

- Precompilers for C, C++, COBOL, and FORTRAN (OS/2 only).
- 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 that are easily ported to ODBC and compiled with an ODBC SDK (available from Microsoft for Microsoft platforms, and from Visigenic for all other platforms). The DB2 CAE for OS/2 contains an ODBC driver for DB2 that supports applications developed with Visigenic ODBC version 2.1. For Windows platforms, the DB2 CAE contains an ODBC driver for DB2 that supports applications developed with the Microsoft ODBC Software Developer's Kit.
- DB2 Java Database Connectivity (DB2 JDBC) support to develop Java applications and applets.
- On OS/2, Windows NT, and Windows 95, support to develop database applications that use the REXX language.
- On Windows NT and Windows 95, code samples of Object Linking and Embedding (OLE) automation UDFs in Microsoft Visual Basic and Microsoft Visual C++, and a sample OLE automation controller implemented as a stored procedure which controls other OLE automation stored procedures.
- Interactive SQL through the Command Line Processor 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 OS/2, IBM PL/I uses 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 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. An ODBC driver manager must be 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 for OS/2 and for Windows; 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 (16-bit).** Any existing applications precompiled with DB2 Version 2.1.1 or earlier and compiled with Micro Focus COBOL (16-bit) 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. This support requires Micro Focus COBOL V3.2.46 or V3.2.50 plus the following patch, available on CompuServe:

Forum:	MICROFOCUS
Library:	ICD Product Updates (Library 4)
Filename:	DB2211UP.ZIP

This patch may also be available directly from Micro Focus. Obtain the patch and apply it to your V3.2.46 or V3.2.50 Compiler. Micro Focus intends to integrate the patch in their COBOL product for subsequent versions of the compiler; however, it may not be integrated in versions immediately after V3.2.50. If you are in doubt as to whether your version has the patch integrated, contact Micro Focus.

3. **VisualAge for Basic for OS/2 and for Windows.** This product includes DB2 functions for embedded 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.
4. **Microsoft Visual Basic.** The SDK for Windows NT and Windows 95 supports DB2 programming using Object Linking and Embedding (OLE) automation with Visual Basic. You can also use Visual Basic and ODBC to develop client applications that access DB2. However, no DB2 precompiler is supplied for this language. For further information, see “Object Linking and Embedding (OLE) Automation” on page 62.

Windows NT and Windows 95

The DB2 SDK for Windows NT and Windows 95 supports the following operating systems:

Microsoft Windows NT	Version 3.5.1 or later (both workstation and server versions)
Microsoft Windows 95	Version 4.00.950 or later

The DB2 SDK for Windows NT and Windows 95 supports the following programming languages:

C/C++ Microsoft Visual C++ Version 4.1 or later, and IBM VisualAge for C++ for Windows Version 3.5 or later

COBOL Micro Focus COBOL Version 4.0.20 (32-bit), and IBM VisualAge for COBOL for OS/2 Version 1.2

Note: The IBM VisualAge for COBOL for OS/2 compiler provides a COBOL development environment for OS/2, Windows 95 and Windows NT. It can be installed and run on any of these operating systems.

REXX	Object REXX for Windows NT/95 Version 1.0
Java	Java Development Kit (JDK) 1.1 for Win32 from Sun Microsystems
Basic	IBM VisualAge for Basic for OS/2 and for Windows Version 1, and Microsoft Visual Basic Version 4.0 or later (see the notes above on VisualAge for Basic and Visual Basic)

Windows 3.1

The DB2 SDK for Windows 3.1 supports the following operating systems:

Microsoft Windows	Version 3.1
Microsoft Windows for Workgroups	Version 3.11
OS/2	WIN-OS/2 session under Version 2.11, WARP 3.0, and WARP 4.0

The DB2 SDK for Windows 3.1 supports the following programming languages:

C/C++	Microsoft Visual C++ Version 1.5 or later, and Borland C++ Version 4.0 or Version 4.5
COBOL	Micro Focus COBOL Version 3.2.46 or later (16-bit)

OS/2

The DB2 SDK for OS/2 supports the following operating systems:

OS/2	Version 2.11, WARP 3.0, and WARP 4.0

The DB2 SDK for OS/2 supports the following programming languages:

C/C++	IBM VisualAge C++ for OS/2 Version 3, and Borland C++ for OS/2 Version 1.5
FORTRAN	WATCOM FORTRAN 77 32 Version 10.5
COBOL	IBM VisualAge for COBOL for OS/2 Version 1.2, Micro Focus COBOL Version 3.2.46 or later (16-bit), and Micro Focus COBOL Version 4.0.20 (32-bit)
REXX	IBM Procedures Language 2/REXX (supplied as part of OS/2)
Java	Java Development Kit (JDK) 1.1 for OS/2 from IBM
Basic	IBM VisualAge for Basic for OS/2 and for Windows Version 1 (See the note above on VisualAge for Basic)

Sample Programs

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

The sample programs providing examples of embedded SQL and DB2 API calls are shown in Table 2 on page 9. Command Line Processor (CLP) files provided by DB2 are shown in Table 3 on page 14.

Java sample programs are shown in Table 4 on page 15. Object Linking and Embedding (OLE) sample programs are shown in Table 5 on page 15. The sample programs demonstrating DB2 CLI calls are shown in Table 6 on page 16.

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 9 and Table 6 on page 16, 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 9 and Table 6 on page 16, 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 9 and Table 6 on page 16, 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 9 and Table 6 on page 16, 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 9, ‘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 a database instance for your environment. Refer to Appendix A, “About Database Manager Instances” on page 137 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 Windows or OS/2 platform you are using, listed in “Supported Software by Platform” on page 2. Consult the documentation for the compiler or interpreter you are using.

More detailed information on installation and setup can be found in the *Quick Beginnings* book for your Windows or OS/2 platform.

When the above are installed and working, you can set up your environment by following the steps in one of the following sections:

- “Setting the Windows NT and Windows 95 Environment” on page 19
- “Setting the Windows 3.1 Environment” on page 20
- “Setting the OS/2 Environment” on page 21

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 23.

Setting the Windows NT and Windows 95 Environment

When you install the DB2 SDK for Windows NT, the install program updates the Windows NT configuration registry with the environment variables `INCLUDE`, `LIB`, `PATH`, `DB2PATH`, and `DB2INSTANCE`. The default instance is DB2.

When you install the DB2 SDK for Windows 95, the install program updates the `autoexec.bat` file.

You can override these environment variables to set the values for the machine or the currently logged-on user. To override these values, use any of the following:

- The Windows 95 or Windows 3.1 command window
- The Windows NT control panel
- The Windows 95 `autoexec.bat` file

Note: Exercise caution when changing these environment variables. **Do not** change the `DB2PATH` environment variable.

These environment variables can be updated for running most Windows NT and Windows 95 programs. In addition, you must take the following specific steps for running DB2 applications:

- When building C or C++ programs, you must ensure that the `INCLUDE` environment variable contains `%DB2PATH\INCLUDE` as the first directory.
- When building Micro Focus COBOL programs, set the `COBCPY` environment variable to point to `%DB2PATH%\INCLUDE\cobol_mf`.
- When building IBM COBOL programs, set the `SYSLIB` environment variable to point to `%DB2PATH%\INCLUDE\cobol_a`.
- Ensure the LIB environment variable points to `%DB2PATH%\lib` by using:

  ```
  set LIB=%DB2PATH%\lib;%LIB%
  ```
- Ensure that the `DB2COMM` environment variable is set at the server of a remote database.
- Ensure that the security service has started at the server for SERVER authentication, and at the client, depending on the level of authentication for CLIENT authentication. To start the security service, use the `NET START DB2NTSECSERVER` command.

Notes:

1. All DB2 environment variables can be defined in the user's environment or set up as registry variables. Please see the *Command Reference* for information on registry variables and the `db2set` command.
2. DB2INSTANCE should only be defined at the user environment level. It is not required if you make use of the DB2INSTDEF registry variable which defines the default instance name to use if DB2INSTANCE is not set.

Setting the Windows 3.1 Environment

You define the Windows 3.1 environment by editing the ASCII file `db2.ini` in the `c:\windows` directory. Before you make any changes, make a backup copy of the file. Refer to the *Quick Beginnings* book for information about setting the environment using `db2.ini`.

Note: If installing the Windows 3.1 DB2 CAE under WIN-OS/2, the `db2.ini` file will normally be stored in the `d:\os2\mdos\winos2` directory. However, if Windows has been separately installed, the `db2.ini` file will normally be found in `c:\windows`.

The DB2 installation program appends the following to the variables in the AUTOEXEC.BAT file:

`\sqllib\win\bin` to the PATH variable.

`\sqllib\win\bin` to the LIB variable.

`\sqllib\win\include` to the INCLUDE variable.

You might need to modify the file to suit your environment.

Setting the OS/2 Environment

Most OS/2 compilers use environment variables to control various options. You can set these variables in your `CONFIG.SYS` file, or you can create command files to set them.

CONFIG.SYS The advantage of setting the environment variables in your `CONFIG.SYS` file is that once you get them right, they are set every time you start (boot) your computer.

Command File The advantage of setting the environment variables in a command file is that you can have a shorter path and the flexibility to use several compilers. The disadvantage is that you must remember to run the command file at the start of each programming session.

If you set environment variables by running a command file, you must build your applications in the same window in which you set the environment variables. If you build your applications in another window, you will not be using the same options you set in your first window.

When you install the DB2 SDK, the statements shown below are put into the CONFIG.SYS file. The command files shown in the rest of this book assume the statements are present in the CONFIG.SYS file. If you edit the CONFIG.SYS file after installing the DB2 SDK, make sure these statements are not removed.

The CONFIG.SYS file must have the following statement:

```
set LIB=%DB2PATH%\lib;%LIB%
```

In addition, if you are using one of the programming languages shown below, the CONFIG.SYS file must have the appropriate statement:

C++	`set INCLUDE=%DB2PATH%\include;%INCLUDE%`
FORTAN	`set FINCLUDE=%DB2PATH%\include;%FINCLUDE%`
IBM COBOL	`set SYSLIB=%SYSLIB%;%DB2PATH%\include\cobol_a`
Micro Focus COBOL	`set COBCPY=%DB2PATH%\include\cobol_mf;%COBCPY%`

Note: On OS/2, you should have no DB2 environment variables defined in CONFIG.SYS apart from DB2PATH and DB2INSTPROF. All DB2 variables should be defined in the DB2 Instance Profile Registry either at the global level, the instance level, or the instance node level (Parallel Edition). Use the `db2set.exe` command to set, modify, and list the variables.

DB2INSTANCE is not required if you make use of the DB2INSTDEF registry variable which defines the default instance name to use if DB2INSTANCE is not set.

Enabling Communications on the Server

Before you begin installing, cataloging and binding the SAMPLE database, you should ensure the server is operational and configured to support the protocol being cataloged. Do the following on the server:

1. Ensure that the `db2comm` environment variable is set. For example, enter:

   ```
   db2set DB2COMM=tcpip
   ```

2. Ensure that the protocol for TCP/IP support is configured.

   Refer to the *Quick Beginnings* book for instructions for adding the `TCP/IP` settings to the Services file.

3. Start the database instance by entering:

   ```
   db2start
   ```

Windows NT Considerations

In a DB2 for Windows NT production system, you have to start the database instance as a service. The steps are as follows:

- If using communications protocols, ensure that the `db2comm` environment variable is set in the System Environment Variables section of the Windows NT control panel.
- Start the security service. This can be done automatically (see the note below), or you can select to have this service start manually using the following command:

  ```
  NET START DB2NTSECSERVER
  ```

- Start the instance by entering:

  ```
  db2start
  ```

Note: Starting the Security Service automatically. Normally the only time you would want to set the security service to start automatically is if the workstation is acting as a DB2 client connecting to a server that is configured for Client Authentication. To have the security service start automatically, do the following:

1. Click on the "Start" button.
2. Click on "Settings".
3. Click on "Control Panel".
4. In the Control Panel, click on "Services".
5. In the Services window, highlight "DB2 Security Server".
6. If it does not have the settings "Started" and "Automatic" listed, click on "Startup".
7. Click on "Automatic".

8. Click on "OK".
9. Reboot your machine to have the settings take effect.

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.

Additionally, if you will be accessing the SAMPLE database on the server from a remote client 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 you have the location of `db2sampl` (the program that installs the SAMPLE database) in your path. The `db2install` program 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, the default instance name is `DB2`, so only if you want to use an instance other than the default, set the database instance where you want to install the SAMPLE database by entering:

     ```
     set DB2instance=instance_name
     ```

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

3. Create the SAMPLE database by entering `db2sampl` followed by where you want to create the sample database. On Windows and OS/2-based systems, this is a `drive`, and would be entered as:

   `db2sampl` *drive*

   On UNIX-based systems, this is a path, and would be entered as:

   `db2sampl` *path*

   If you do not specify the path, the installation program installs the sample tables in the default path 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. Create 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 *Administration Guide*. 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 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 about cataloging databases. This section provides specific instructions to catalog the SAMPLE database.

For OS/2, Windows 95 and Windows NT, catalog the sample database from the remote client workstation by entering:

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

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

For Windows 3.1, double-click on the Command Line Processor icon in the IBM DATABASE 2 Windows group. Then enter the following at the prompt:

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

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

You can exit the command line processor by entering either `terminate` or `quit` at the prompt.

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 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.

Binding creates the package that the database manager needs to access the database when an application is executed. Binding can be done explicitly by specifying the BIND command against the bind file created during precompilation.

The *Quick Beginnings* book provides general information about binding the database utilities. This section provides specific instructions to bind the database utilities to the SAMPLE database on each of the supported OS/2 and Windows platforms.

You bind the database utilites differently depending on the platform of the client workstation you are using.

On a Client Workstation running Windows 95 or Windows NT Version 4.0:

1. From the Start Menu, select Programs.
2. From the Programs Menu, select DB2 for Windows 95 or DB2 for Windows NT, depending on your operating system.

3. From the DB2 for Windows 95 menu, or the DB2 for Windows NT menu, select the DB2 command window.

   The command window displays.

4. Connect to the SAMPLE database. At the prompt, enter:

   ```
   db2 connect to sample
   ```

   Press Enter.

5. Bind the utilities to the database by entering:

   ```
   db2 bind %DB2PATH%\bnd\@db2ubind.lst blocking all
   sqlerror continue messages bind.msg
   ```

   where `%DB2PATH%` is the path where DB2 is installed.

   Press Enter.

6. Exit the command window, and verify that the bind was successful by checking the bind message file `bind.msg`.

Note: On a Client Workstation running Windows NT Version 3.5.1 or earlier, the steps you follow will be different due to the differences in interface with Version 4.0. See your Windows NT documentation for details.

On a Client Workstation running Windows 3.1:

1. Start Windows.
2. Double-click on the Command Line Processor icon in the IBM DATABASE 2 Windows group.
3. Connect to the SAMPLE database by entering:

   ```
   connect to sample
   ```

   The utilities will be automatically bound to the database by DB2 with this command, so the user does not have to explicitly bind them.

4. Exit the Command Line Processor, and verify that the bind was successful by checking the bind message file `bind.msg`.

On a Client Workstation running OS/2:

1. Connect to the SAMPLE database by entering:

   ```
   db2 connect to sample
   ```

   The utilities will be automatically bound to the database by DB2 with this command, so the user does not have to explicitly bind them.

2. Exit the Command Line Processor, and verify that the bind was successful by checking the bind message file `bind.msg`.

For all platforms:

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

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 29, 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 127. If you are developing Java applications, see Chapter 8, “Building Java Applications and Applets” on page 133.

For further information, refer to the following books. To develop applications using embedded SQL or Java, 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 by running files containing compile and link commands supplied with the DB2 SDK for that platform. These files are called batch files on Windows platforms and command files on OS/2. You can also use the makefiles that are supplied. The makefiles and the batch and command 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.

The batch or command file builds a sample program by doing the following:

- 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. See "Module Definition Files for Stored Procedures and UDFs" on page 31 for more information about UDFs.

Note: The embedded SQL chapters for the supported platforms show you just some of the batch and command files. Look in the directories that contain the sample programs for all of these files, and for a README file that 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 29 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 `%DB2PATH%\samples`:

C	`%DB2PATH%\samples\c`
C++	`%DB2PATH%\samples\cpp`
IBM COBOL	`%DB2PATH%\samples\cobol`
Micro Focus COBOL (32-bit)	`%DB2PATH%\samples\cobol_mf`
Micro Focus COBOL (16-bit)	`%DB2PATH%\samples\cobol_16` (OS/2); `%DB2PATH%\samples\cobol` (Windows 3.1)
FORTRAN	`%DB2PATH%\samples\fortran`

Note: Of the samples given in Table 7 on page 29, the C++ directory, `%DB2PATH%\samples\cpp`, contains only a C++ version of the `updat` program. The stored procedure and UDF batch and command files documented for the C++ compilers use the C versions of the `outsrv`, `outcli`, `udf` and `calludf` programs found in `%DB2PATH%\samples\c`. In addition, `%DB2PATH%\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 `%DB2PATH%\samples\cpp` directory for more information.

After you build the sample programs, you can use them 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 batch or command 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 `%DB2PATH%\samples` to your own working directory.

Module Definition Files for Stored Procedures and UDFs

This section contains information for developing applications on the server, and therefore does not apply to the client-only platform, Windows 3.1.

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 stored in a dynamic link library (DLL) on the server.

When you compile stored procedures and UDFs, you create the DLL that contains your stored procedures and UDFs.

To create a stored procedure or UDF, you may need a module definition (".def") file. A module definition file can be used to define various attributes of the DLL, including which function names are exported by the DLL. DB2 requires that the names of all stored procedures and UDFs be exported by the DLL in which the stored procedures and UDFs reside. For some compiler and operating system combinations, a module definition file may not be required since all function names in a DLL are exported by default.

You can create a module definition using a text editor. Here are a couple of example module definition files for the sample program `outsrv`, one each for a C++ compiler on OS/2 and Windows NT. The module definition files for the sample stored procedures and UDFs are contained in the appropriate language sub-directory for each compiler, along with the sample programs.

This version of the file `outsrv.def` is for the VisualAge for C++ compiler on OS/2:

```
LIBRARY OUTSRV INITINSTANCE TERMINSTANCE
DESCRIPTION 'Library for DB2 Stored Procedure OUTSRV'
PROTMODE
DATA
    MULTIPLE
    NONSHARED
CODE
    LOADONCALL
    SHARED
EXPORTS
    outsrv
```

This version of `outsrv.def` is for the Microsoft Visual C++ compiler on Windows NT:

```
LIBRARY OUTSRV
EXPORTS outsrv
```

Both versions of `outsrv.def` list the stored procedure `outsrv`. The linker uses `outsrv.def` to define various attributes of the DLL `outsrv.dll`, including which function names get exported by the DLL.

On OS/2, a stored procedure or UDF must use system linkage conventions. On Windows NT and Windows 95, a stored procedure or UDF must use the _stdcall (WINAPI) linkage conventions. Since these conventions are supported by all compilers on their respective operating systems, a compiled stored procedure or UDF can be invoked by any supported compiler on that operating system.

The IBM VisualAge C++ compiler on Windows NT exports function names that are type decorated by affixing the ampersand symbol, `@`, followed by an integer representing the number of bytes of arguments. This is shown in the following sample module definition file for UDFs, `udfva.def`, where, for example, `increase` and `raise` have ten (DWORD) arguments, and `wordcount` and `findvw` have eight:

```
LIBRARY UDF
DESCRIPTION 'Library for DB2 User Defined Functions'
EXPORTS
_increase@40
_raise@40
_wordcount@32
_findvw@32
_ctr@28
_ilob@40
_leni@32
_promote@32
```

Sections in the appropriate embedded SQL chapter for your platform show you how to build stored procedures and UDFs by using, if required, a module definition (`.def`) file for the following supported programming languages:

On Windows NT and Windows 95:

Stored procedures	C/C++, COBOL
UDFs	C/C++

On OS/2:

Stored procedures	C/C++, COBOL, and FORTRAN
UDFs	C/C++

If you need more information about module definition and export files, and DLLs, refer to your compiler documentation.

Error Checking

The sample programs may use the following error checking utilities:

`util.c`	For C sample programs
`checkerr.cbl`	For COBOL sample programs.
`util.f`	For FORTRAN sample programs

The batch or command files you use to build the sample programs may require the appropriate object file:

`util.obj`	For C and FORTRAN sample programs
`checkerr.obj`	For COBOL sample programs.

The batch and command files that require an error checking utility object file create the one they use. If you want to compile an object file for the supported language on your platform outside of these batch and command files, use the compile step in the appropriate file for that language in the embedded SQL chapter for your operating system.

As an example, here are the compile steps you would follow to create the C error checking utility object files for some compilers on the Windows and OS/2 platforms. You would enter one of the following, using the appropriate compiler:

On Windows NT and Windows 95 using Microsoft Visual C++

```
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 -I%DB2PATH%\include util.c
```

On Windows 3.1 using Microsoft Visual C++

```
cl /c /Gy /ALw /W3 /Mq /DDB2WIN util.c
```

On OS/2 using IBM VisualAge C++

```
icc -C+ -O- -Ti+ util.c
```

Chapter 4. Building Windows NT and Windows 95 Embedded SQL Applications

This chapter provides detailed information for building embedded SQL applications on Windows NT and Windows 95. In the batch files, commands that begin with `db2` are Command Line Processor (CLP) commands. Refer to the *Command Reference* if you need more information about DB2 commands.

Note: All applications on Windows NT and Windows 95, both embedded SQL and non-embedded SQL, must be built in a DB2 command window, and not from an operating system command prompt.

WCHARTYPE CONVERT Precompile Option

The WCHARTYPE precompile option handles graphic data in either multi-byte format or wide-character format using the `wchar_t` data type. More information on this option can be found in the *Embedded SQL Programming Guide*.

For DB2 for Windows NT and DB2 for Windows 95, the WCHARTYPE CONVERT option is supported for applications compiled with the Microsoft Visual C++ compiler, and not supported for applications compiled with the IBM VisualAge C++ compiler.

If you are using the IBM VisualAge C++ compiler, use the default NOCONVERT option for WCHARTYPE. With the NOCONVERT option, no implicit character conversion occurs between application and the database manager. Data in a graphic host variable is sent to and received from the database manager as unaltered Double Byte Character Set (DBCS) characters.

If you need to convert your graphic data to multi-byte format from wide-character format, use the `wcstombs()` function. For example:

```
    wchar_t widechar[200];
    wchar_t mb[200];
    wcstombs((char *)mb,widechar,200);

    EXEC SQL INSERTINTO TABLENAME VALUES(:mb);
```

Similarly, you can use the `mbstowcs()` function to convert from multi-byte to wide-character format.

For the Microsoft Visual C++ compiler, do not use the CONVERT option if your application inserts data into a DB2 database in a code page that is different from the database code page. DB2 normally performs a code page conversion in this situation; however, the Microsoft C runtime environment does not handle substitution characters for certain double byte characters. This could result in run time conversion errors.

Do not issue a `setlocale()` call from your application if your application is statically bound to the C runtime libraries, as this may lead to C runtime conversion errors. Using `setlocale()` is not a problem if your application is dynamically bound to the C runtime library. This is also the case for stored procedures.

Microsoft Visual C++

Notes:

1. For information on Object Linking and Embedding (OLE) automation using Visual C++ see "Object Linking and Embedding (OLE) Automation" on page 62.
2. For Microsoft Visual C++ Version 2.1, compile errors may occur during compilation of a file generated by the DB2 precompiler, and display an incorrect file name. For example, if you precompile a source file, `func.sqc`, to produce a C file, `func.c`, in some instances the error message that the C compiler gives for an error in the file, `func.c`, is incorrect. The message may state that the error is in the file `func.c`, instead of stating that it is in the file `func.sqc`.

   This problem is only apparent if the error in the source code is after any `#include` statements at the top of the source file, and before any code that the precompiler generates for embedded SQL statements.

The batch file `bldmsemb.bat`, in `%DB2PATH%\samples\c`, contains the commands to build a sample Microsoft Visual C program.

You can also use the batch file to build a C++ program after you modify it. The comments in the batch file describe the modifications you need to make.

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.

```
@echo off
rem bldmsemb.bat file
rem Builds a sample C or C++ program containing embedded SQL
rem using the Microsoft Visual C++ compiler.
rem Usage: bldmsemb <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
rem To build a C++ program, change the source file extension to .sqx.
db2 prep %1.sqc bindfile

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem  Compile the program. To build a C++ program, change the
rem  source file extension to '.cxx'.
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 -I%DB2PATH%\include %1.c util.c

rem Link the program.
link -debug:full -debugtype:cv -out:%1.exe %1.obj util.obj db2api.lib

goto exit

:error
echo Usage: bldmsemb <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldmsemb	
The batch file contains the following compile options:	
`cl`	The Microsoft Visual C++ compiler.
`-Z7`	C7 style CodeView information generated.
`-Od`	Disable optimizations. It is easier to use a debugger with optimization off.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-W2`	Set warning level.
The batch file contains the following link options:	
`link`	Use the 32-bit linker to link edit.
`-debug:full`	Include debugging information.
`-debugtype:cv`	Indicate the debugger type.
`-out:%1.exe`	Specify a filename
`%1.obj`	Include the object file
`db2api.lib`	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

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

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

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

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

   ```
   bldmsemb updat
   ```

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

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

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

3. Run the program. If you built the `updat` sample program, from a command line, enter:

   ```
   updat
   ```

Note: To build C applications that do not contain embedded SQL, you can use the batch file `bldmsapi.bat`. It contains the same compile and link options as `bldmsemb.bat`, 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 Stored Procedures with Microsoft Visual C++

The batch file `bldmsstp.bat`, in `%DB2PATH%\samples\c`, contains the commands to build a C stored procedure for a DB2 for Windows NT server. The batch file builds the stored procedure into a DLL on the server.

You can also use the batch file to build a C++ stored procedure after you have modified the file. The comments in the batch file describe the modifications you need to make.

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 batch file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldmsstp.bat file
rem Builds a C or C++ stored procedure using the Microsoft Visual C++ compiler.
rem Usage: bldmsstp <prog_name> [ <db_name> [ < userid> <password> ]]
```

```
rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program. To build a C++ stored procedure, change the
rem source file extension to .sqx.
db2 prep %1.sqc bindfile

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem Compile the program. To build a C++ stored procedure, change
rem the source file extension to .cxx.
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c

rem Link the program.
link -debug:full -debugtype:cv -out:%1.dll %1.obj db2api.lib -def:%1.def

rem Copy the stored procedure DLL to the 'function' directory
copy %1.dll %DB2PATH%\function

goto exit

:error
echo Usage: bldmsstp <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldmsstp	
The batch file contains the following compile options:	
`cl`	The Microsoft Visual C++ compiler.
`-Z7`	C7 style CodeView information generated.
`-Od`	Disable optimization.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-W2`	Output warning, error, and severe and unrecoverable error messages.
The batch file contains the following link options:	
`link`	Use the linker to link edit.
`-debug:full`	Include debugging information.
`-debugtype:cv`	Indicates the debugger type.
`%1.dll`	Build a .DLL file.
`%1.obj`	Include the object file.
`db2api.lib`	Link with the DB2 library.
`%1.def`	Module definition file.
Refer to your compiler documentation for additional compiler options.	

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

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

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

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

   ```
   bldmsstp outsrv
   ```

   The batch file uses the module definition file `outsrv.def`, contained in the same directory as the sample programs, to build the stored procedure. The batch file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build the stored procedure, `outsrv`, you can build the client application that calls the stored procedure. You can build `outcli` using the `bldmsemb` file. See "Microsoft Visual C++" on page 36 for details.

To run the stored procedure, do the following :

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

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

3. At the command line, enter:

   `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 User-Defined Functions (UDFs) with Microsoft Visual C++

The batch file `bldmsudf`, in `%DB2PATH%\samples\c`, contains the commands to build a C UDF. UDFs are compiled like stored procedures, but you do not need to connect to a database to precompile and bind the program.

Note: A UDF does not contain embedded SQL statements. Instead, it contains C or C++ statements. See the sample UDF program `calludf`.

You can use the batch file to build a C++ UDF after you modify it. The comments in the batch file describe the modification you need to make.

The one parameter it takes, `%1`, specifies the name of your source file. The batch file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldmsudf.bat file
rem Build sample C or C++ user-defined function (UDF).
rem Usage: bldmsudf <udf_prog_name>

rem Compile the program. To build a C++ UDF, change the source
rem file extension to '.cxx'.
```

```
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c
rem Link the program.
link -debug:full -debugtype:cv -dll -out:%1.dll %1.obj db2api.lib db2apie.lib -def:%1.def

rem Copy the UDF DLL to the 'function' directory
copy %1.dll %DB2PATH%\function
@echo on
```

Compile and Link Options for bldmsudf

The batch file `bldmsudf` contains the following compile options:

`cl`	The Microsoft Visual C++ compiler.
`-Z7`	C7 style CodeView information generated.
`-Od`	Disable optimization.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-W2`	Output warning, error, and severe and unrecoverable error messages.

The batch file contains the following link options:

`link`	Use the linker to link edit.
`-debug:full`	Include debugging information.
`-debugtype:cv`	Indicates the debugger type.
`-dll`	Create a DLL.
`%1.dll`	Build a .DLL file.
`%1.obj`	Include the object file.
`db2api.lib`	Link with the DB2 library.
`db2apie.lib`	Link with the DB2 API Engine library.
`%1.def`	Module definition file.

Refer to your compiler documentation for additional compiler options.

To build the user-defined function `udf`, at the DB2 command line processor window, enter:

```
bldmsudf udf
```

The batch file uses the module definition file `udf.def`, contained in the same directory as the sample programs, to build the user-defined function. The batch file copies the user-defined function DLL, `udf.dll`, to the server in the path `%DB2PATH%\function` to indicate that the UDF is fenced. If you want the UDF to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build `udf`, you can build the application `calludf` that calls the UDF. You can build `calludf` using the `bldmsemb` batch file. Refer to “Microsoft Visual C++” on page 36 for details.

To run the UDF, do the following:

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start the Security Service for the Windows NT server by entering:
   ```
   net start DB2NTSECSERVER
   ```
3. Run the sample calling application by entering:
   ```
   calludf
   ```
   The 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. Connect to the database, then enter:
   ```
   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.

IBM VisualAge C++

The batch file `bldvaemb.bat`, in `%DB2PATH%\samples\c`, contains the commands to build a sample IBM VisualAge C program. Ensure the LIB environment variable points to `%DB2PATH%\lib` like this:

```
set LIB=%DB2PATH%\lib;%LIB%
```

You can also use the batch file to build a C++ program after you modify it. The comments in the batch file describe the modifications you need to make.

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.

```
@echo off
rem bldvaemb.bat file
rem Builds a sample C or C++ program containing embedded SQL using
rem the IBM VisualAge C++ compiler.
rem USAGE: bldvaemb <prog_name> [ <db_name> [ < userid> <password> ]]
```

```
rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
rem To build a C++ program, change the source file extension to .sqx.
db2 prep %1.sqc bindfile

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem  Compile the program. To build a C++ program, change the
rem  source file extension to '.cxx'.
icc -c -Ti -W1 %1.c util.c

rem  Link the program.
ilink /MAP /DEBUG /ST:32000 /PM:VIO %1.obj util.obj db2api.lib

goto exit

:error
echo Usage: bldvaemb <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldvaemb	
The batch file contains the following compile options:	
`icc`	The IBM VisualAge C++ compiler.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-Ti`	Generate debugger information.
`-W1`	Output warning, error, and severe and unrecoverable error messages.
The batch file contains the following link options:	
`ilink`	Use the resource linker to link edit.
`/MAP`	Generate a map file.
`/DEBUG`	Include debugging information.
`/ST:32000`	Specify a stack size of at least 32 000.
`/PM:VIO`	Enable the program to run in a window or in a full screen.
`%1.obj`	Include the object file.
`util.obj`	Include the error-checking utility object file.
`db2api.lib`	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

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

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start the Security Service for the Windows NT server by entering:
   ```
   net start DB2NTSECSERVER
   ```
3. Build the sample program, connecting to the SAMPLE database, by entering:
   ```
   bldvaemb updat
   ```

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

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start the Security Service for the Windows NT server by entering:
   ```
   net start DB2NTSECSERVER
   ```
3. Run the program. If you built the `updat` sample program, enter the following at the command line:
   ```
   updat
   ```

Note: To build VisualAge C++ applications that do not contain embedded SQL, you can use the batch file `bldvaapi.bat`. It contains the same compile and link options as `bldvaemb.bat`, 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/C++.

Building Stored Procedures with IBM VisualAge C++

The batch file `bldvastp.bat`, in `%DB2PATH%\samples\c`, contains the commands to build a C stored procedure for a DB2 for Windows NT server. The batch file compiles the stored procedure with a DLL on the server.

You can also use the batch file to build a C++ stored program after you modify it. The comments in the batch file describe the modifications you need to make.

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 batch file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldvastp.bat file
rem Builds a sample C or C++ stored procedure using the IBM VisualAge C++ compiler.
rem Usage: bldvastp <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program. To build a C++ stored procedure, change the
rem source file extension to '.sqx'.
db2 prep %1.sqc bindfile

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset
```

```
rem Compile the program. To build a C++ stored procedure, change the
rem source file extension to .cxx.
icc -c+ -Ti -Ge- -Gm+ -W1 %1.c

rem Import the library and create a definition file.
rem The function name in the .def file must be decorated to be consistent
rem with the function name in the .map file. Typically, this is done by
rem prepending "_" and appending "@" and the number of bytes of arguments,
rem for example, "@16". In outsrvva.def, the IBM VisualAge C++ compiler requires
rem "EXPORTS _outsrv@16" and not "EXPORTS outsrv".
ilib /GI %1va.def

rem Link the program and produce a DLL.
ilink /ST:64000 /PM:VIO /MAP /DLL %1.obj %1va.exp db2api.lib

rem Copy the Stored Procedure DLL to the 'function' directory.
copy %1.dll %DB2PATH%\function

goto exit

:error
echo Usage: bldvastp <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldvastp

The batch file contains the following compile options:

`icc`	The IBM VisualAge C++ compiler.
`-c+`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-Ti`	Generate debugger information.
`-Ge-`	Build a .DLL file. Use the version of the runtime library that is statically linked.
`-Gm+`	Link with multitasking libraries.
`-W1`	Output warning, error, and severe and unrecoverable error messages.

Compile and Link Options for bldvastp	
The batch file contains the following link options:	
`ilink`	Use the resource linker to link edit.
`/ST:64000`	Specify a stack size of least of 64 000.
`/PM:VIO`	Enable the program to run in a window or full screen.
`/MAP`	Generate a MAP file.
`/DLL`	Build a .DLL file.
`%1.obj`	Include the object file.
`%1va.exp`	VisualAge export file.
`db2api.lib`	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

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

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

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

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

   ```
   bldvastp outsrv
   ```

   The batch file uses the module definition file `outsrvva.def`, contained in the same directory as the sample programs, to build the stored procedure. The batch file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build the stored procedure `outsrv`, you can build the client application that calls the stored procedure. You can build `outcli` using the `bldvaemb` file. Refer to “IBM VisualAge C++” on page 43 for details.

To run the stored procedure, do the following :

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

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

3. Enter the following at the command line:

   `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 User-Defined Functions (UDFs) with IBM VisualAge C++

The batch file `bldvaudf`, in `%DB2PATH%\samples\c`, contains the commands to build a C UDF. UDFs are compiled like stored procedures, but you do not need to connect to a database to precompile and bind the program.

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

You can also use the batch file to build a C++ UDF after you modify it. The comments in the batch file describe the modification you need to make.

The first parameter, `%1`, specifies the name of your source file. The batch file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldvaudf.bat file
rem Builds a C or C++ user-defined function (UDF) using the
rem IBM VisualAge C++ compiler.
rem Usage: bldvaudf <udf_program_name>

rem Compile the program. To build a C++ UDF, change the source
rem file extension to '.cxx'.
icc -Ti -c+ -Ge- -Gm+ -W1 %1.c

rem Import the library and create a definition file. Note that the function
rem name in the .def file must be decorated to be consistent with the
rem decorated function name in the .map file. Typically, this involves
rem prepending an underscore, "_", and appending an ampersand along
rem with an integer representing the number of bytes of arguments, for
```

```
rem example, "@16". In udfva.def, the IBM VisualAge C++ compiler
rem requires "EXPORTS _ctr@28" and not "EXPORTS ctr".
ilib /GI %1va.def

rem Link the program to a dynamic link library
ilink /ST:64000 /PM:VIO /MAP /DLL %1.obj %1va.exp db2api.lib db2apie.lib

rem Copy the UDF DLL to the 'function' directory.
copy %1.dll %DB2PATH%\function
@echo on
```

Compile and Link Options for bldvaudf

The batch file `bldvaudf` contains the following compile options:

`icc`	The IBM VisualAge C++ compiler.
`-Ti`	Generate debugger information.
`-c+`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-Ge-`	Build a .DLL file. Use the version of the runtime library that is statically linked.
`-Gm+`	Link with multitasking libraries.
`-W1`	Output warning, error, and severe and unrecoverable error messages.

The batch file contains the following link options:

`ilink`	Use the resource linker to link edit.
`/ST:64000`	Specify a stack size of at least 64000.
`/PM:VIO`	Enable the program to run in a window or a full screen.
`/MAP`	Generate a MAP file.
`/DLL`	Build a .DLL file.
`%1.obj`	Include the object file.
`%1va.exp`	Include the VisualAge export file.
`db2api.lib`	Link with the DB2 library.
`db2apie.lib`	Link with the DB2 API Engine library.

Refer to your compiler documentation for additional compiler options.

To build the user-defined function `udf`, enter the following at a DB2 command line processor `clp.exe` window:

```
bldvaudf udf
```

The batch file uses the module definition file, `udfva.def`, contained in the same directory as the sample programs, to build the user-defined function. The batch file copies the user-defined function DLL, `udf.dll`, to the server in the path `%DB2PATH%\function` to indicate that the UDF is fenced. If you want the UDF to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build `udf`, you can build the application `calludf` that calls the UDF. You can build `calludf` using the `bldvaemb` batch file. Refer to "IBM VisualAge C++" on page 43 for details.

To run the UDF, do the following:

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

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

3. Run the sample calling application by entering:

   ```
   calludf
   ```

   The 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. Connect to the database, then enter:

   ```
   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.

Micro Focus COBOL

The batch file `bldmfcob`, in `%DB2PATH%\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.

```
@echo off
rem bldmfcob.bat file
rem Build a sample Cobol program using the Micro Focus COBOL compiler.
rem Usage: bldmfcob <prog_name> [ <db_name> [ < userid> <password> ]]
```

```
rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqb bindfile target mfcob

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem Compile the error-checking utility.
cobol checkerr.cbl constant 32-bit (1);

rem  Compile the program.
cobol %1.cbl constant 32-bit (1);

rem  Link the program.
cbllink -l %1.obj checkerr.obj db2api.lib

goto exit

:error
echo Usage: bldmfcob <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldmfcob

The batch file contains the following compile options:

`cobol`	The Micro Focus COBOL compiler.
`constant 32-bit (1)`	Set the constant 32-bit flag to true.

Compile and Link Options for bldmfcob	
The batch file contains the following link options:	
`cbllink`	Use the linker to link edit.
`-l`	Link with the lcobol library.
`checkerr.obj`	Link with the error-checking utility object file.
`db2api.lib`	Link with the DB2 API library.
Refer to your compiler documentation for additional compiler options.	

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

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start the Security Service for the Windows NT server by entering:
   ```
   net start DB2NTSECSERVER
   ```
3. Build the sample program, connecting to the SAMPLE database. From the DB2 command line processor command window, enter:
   ```
   bldmfcob updat
   ```

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

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start the Security Service for the Windows NT server by entering:
   ```
   net start DB2NTSECSERVER
   ```
3. Enter the following at the command line:
   ```
   updat
   ```

Note: To build Micro Focus COBOL applications that do not contain embedded SQL, you can use the batch file `bldapicb`. 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 the DB2 API sample programs written in COBOL.

Building Stored Procedures with Micro Focus COBOL

The batch file `bldmfcbs`, in `%DB2PATH%\samples\cobol_mf`, contains the commands to build a stored procedure. The batch file compiles the stored procedure into a DLL on the server.

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 batch file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldmfcbs.bat file
rem Build sample COBOL stored procedure using Micro Focus COBOL compiler.
rem Usage: bldmfcbs <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqb bindfile target mfcob

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem  Compile the stored procedure.
cobol %1.cbl constant 32-bit (1) /case;

rem  Link the stored procedure and create a shared library.
cbllink /d %1.obj db2api.lib

rem Copy stored procedure to the %DB2PATH%\function directory.
rem Substitute the path where DB2 is installed for %DB2PATH%.
copy %1.dll %DB2PATH%\function

goto exit

:error
echo Usage: bldmfcbs <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldmfcbs	
The batch file contains the following compile options:	
`cobol`	The Micro Focus COBOL compiler.
`constant 32-bit (1)`	Set the constant 32-bit flag to true.
`/case`	Prevent external symbols being converted to upper case.
The batch file contains the following link options:	
`cbllink`	Use the Micro Focus COBOL linker to link edit.
`/d`	Create a .dll file.
`db2api.lib`	Link with the DB2 API library.
Refer to your compiler documentation for additional compiler options.	

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

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start the Security Service for the Windows NT server by entering:
   ```
   net start DB2NTSECSERVER
   ```
3. Build the stored procedure, connecting to the SAMPLE database. Enter the following in a DB2 command line processor command window:
   ```
   bldmfcbs outsrv
   ```
   The linker uses a default entry point unspecified by the user. The `/d` option is used to create the DLL file in order to build the stored procedure. The batch file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build the stored procedure `outsrv`, you can build `outcli` that calls the stored procedure. You can build `outcli` using the `bldmfcob.bat` file. Refer to "Micro Focus COBOL" on page 51 for details.

To run the stored procedure, do the following :

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start the Security Service for the Windows NT server by entering:
   ```
   net start DB2NTSECSERVER
   ```
3. Enter the following at the command line:
   ```
   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 Micro Focus COBOL Compiler

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the Micro Focus 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.
- Ensure the LIB environment variable points to `%DB2PATH%\lib` like this:
  ```
  set LIB=%DB2PATH%\lib;%LIB%
  ```
- The DB2 COPY files for Micro Focus COBOL reside in `sqllib\include\cobol_mf`. Set the `COBCPY` environment variable to include the directory like this:
  ```
  set COBCPY=sqllib\include\cobol_mf;%COBCPY%
  ```
- DB2API.LIB provides the import library for COBOL programs and is located in the `lib` directory in the DB2 for Windows NT install directory.

Calls to all DB2 application programming interfaces and generated code must be made using calling convention 74. The DB2 COBOL precompiler automatically inserts a CALL-CONVENTION clause in a SPECIAL-NAMES paragraph. If the SPECIAL-NAMES paragraph does not exist, the DB2 COBOL precompiler creates it, as follows:

```
Identification Division
Program-ID. "static".
special-names.
    call-convention 74 is DB2API.
```

Also, the precompiler automatically places the symbol DB2API, which is used to identify the calling convention, after the "call" keyword whenever a DB2 API is called. This occurs, for instance, whenever the precompiler generates a DB2 API runtime call from an embedded SQL statement.

If calls to DB2 APIs are made in an application which is not precompiled, you should manually create a SPECIAL-NAMES paragraph in the application, similar to that given above. If you are calling a DB2 API directly, then you will need to manually add the DB2API symbol after the "call" keyword.

IBM VisualAge for COBOL

The batch file `bldvacob.bat`, in `%DB2PATH%\samples\cobol`, 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.

```
@echo off
rem bldvacob.bat file
rem Build sample Cobol program using the IBM VisualAge for COBOL compiler.
rem Usage: bldvacob <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqb bindfile target ibmcob

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem Compile the error checking facility.
cob2 -qpgmname(mixed) -c -qlib -I%DB2PATH%\include\cobol_a checkerr.cbl

rem Compile the program.
cob2 -qpgmname(mixed) -c -qlib -I%DB2PATH%\include\cobol_a %1.cbl

rem Link the program.
cob2 %1.obj checkerr.obj db2api.lib
```

```
goto exit

:error
echo Usage: bldvacob <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldvacob

The batch file contains the following compile options:

`cob2`	The IBM COBOL compiler.
`-qpgmname(mixed)`	Instructs the compiler to permit CALLs to library entry points with mixed-case names.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-qlib`	Instructs the compiler to process COPY statements.
`-I`*path*	Specify the location of the DB2 include files. For example: `-I%DB2PATH%\include\cobol_a`.
`checkerr.cbl`	Compile the error-checking utility.

The batch file contains the following link options:

`cob2`	Use the compiler to link edit.
`checkerr.obj`	Include the error-checking utility object file.
`db2api.lib`	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. Start the database manager on the server, if it is not already running, by entering:

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

3. Build the sample program, connecting to the SAMPLE database. From the DB2 command line processor command window, enter:

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

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

3. Enter the following at the command line:

   ```
   updat
   ```

Note: To build IBM VisualAge for COBOL applications that do not contain embedded SQL, you can use the batch file `bldvcapi.bat`. It contains the same compile and link options as `bldvacob.bat`, 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 COBOL.

Building IBM VisualAge for COBOL Stored Procedures

The batch file `bldvacbs.bat`, in `%DB2PATH%\samples\cobol`, contains the commands to build a stored procedure. The batch file compiles the stored procedure into a DLL on the server.

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 batch file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldvacbs.bat file
rem Build sample COBOL stored procedure using IBM VisualAge for COBOL compiler.
rem Usage: bldvacbs <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqb bindfile target ibmcob

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem  Compile the stored procedure.
cob2 -qpgmname(mixed) -c -qlib -I%DB2PATH%\include\cobol_a %1.cbl
```

```
rem  Link the stored procedure and create a shared library.
cob2 -dll %1.obj db2api.lib

rem Copy stored procedure to the %DB2PATH%\function directory.
rem Substitute the path where DB2 is installed for %DB2PATH%.
copy %1.dll %DB2PATH%\function

goto exit

:error
echo Usage: bldvacbs <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldvacbs

The batch file contains the following compile options:

`cob2`	The IBM COBOL compiler.
`-qpgmname(mixed)`	Instructs the compiler to permit CALLs to library entry points with mixed-case names.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-qlib`	Instructs the compiler to process COPY statements.
`-I`*path*	Specify the location of the DB2 include files. For example: `-I%DB2PATH%\include\cobol_a`.

The batch file contains the following link options:

`cob2`	Use the compiler to link edit.
`-dll`	Create the DLL with the source program name.
`db2api.lib`	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

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

   `db2start`

2. Start the Security Service for the Windows NT server by entering:

   `net start DB2NTSECSERVER`

3. Build the sample program, connecting to the SAMPLE database. From the DB2 command line processor command window, enter:

   `bldvacbs outsrv`

   The batch file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

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 `bldvacob` batch file. See "IBM VisualAge for COBOL" on page 57 for details.

To run the stored procedure, do the following:

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

   ```
   db2start
   ```

2. Start the Security Service for the Windows NT server by entering:

   ```
   net start DB2NTSECSERVER
   ```

3. Enter the following at the command line:

   ```
   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 VisualAge for COBOL Compiler

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the IBM VisualAge for COBOL 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 your source files to set compile options. Place the keywords in columns 8 to 72 only.
- 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 VisualAge for COBOL reside in `%DB2PATH%\include\cobol_a` under the database instance directory. Specify COPY file names to include the `.cbl` extension as follows:

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

Object REXX

Object REXX is an object-oriented version of the REXX language. Object-oriented extensions have been added to traditional REXX, but its existing functions and instructions have not changed. The Object REXX interpreter is an enhanced version of its predecessor, with additional support for:

- Classes, objects, and methods
- Messaging and polymorphism
- Single and multiple inheritance

Object REXX is fully compatible with earlier, non-object-oriented versions of REXX. In this section, whenever we refer to REXX, we are referring to all versions of REXX, including Object REXX.

You do not precompile or bind REXX programs.

On Windows NT, REXX programs are not required to start with a comment. However, for portability reasons you are recommended to start each REXX program with a comment that begins in the first column of the first line. This will allow the program to be distinguished from a batch command on other platforms:

```
/* Any comment will do. */
```

REXX sample programs can be found in the directory `%DB2PATH%\samples\rexx`. To run the sample REXX program `updat`, do the following:

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

   ```
   db2start
   ```

2. Enter:

   ```
   updat
   ```

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

Object Linking and Embedding (OLE) Automation

This section describes Object Linking and Embedding (OLE) automation UDFs in Microsoft Visual Basic and Microsoft Visual C++, as well as a sample OLE automation controller for stored procedures.

You can implement OLE automation UDFs and stored procedures in any language, as OLE is language independent, by exposing methods of OLE automation servers, and registering the methods as UDFs with DB2. Application development environments which support the development of OLE automation servers include certain versions of the following: Microsoft Visual Basic, Microsoft Visual C++, Microsoft Visual J++, Microsoft FoxPro, Borland Delphi, Powersoft PowerBuilder, and Micro Focus COBOL. Also, Java beans objects that are wrapped properly for OLE, for example with Microsoft Visual J++, can be accessed via OLE automation.

You need to refer to the documentation of the appropriate application development environment for further information on developing OLE automation servers. For more detailed information on DB2 programming using OLE automation, refer to the *Embedded SQL Programming Guide*, chapter 7. That chapter covers creating and running OLE automation UDFs.

User-Defined Functions (UDFs) with Microsoft Visual Basic

Microsoft Visual Basic supports the creation of OLE automation servers. A new kind of object is created in Visual Basic by adding a class module to the Visual Basic project. Methods are created by adding public sub-procedures to the class module. These public procedures can be registered to DB2 as OLE automation UDFs. Refer to the Microsoft Visual Basic manual, *Creating OLE Servers, Microsoft Corporation, 1995*, and to the OLE samples provided by Microsoft Visual Basic, for further documentation on creating and building OLE servers.

DB2 provides self-containing samples of OLE automation UDFs in Microsoft Visual Basic, located in the directory `%DB2PATH%\samples\ole\msvb`.

User-Defined Functions (UDFs) with Microsoft Visual C++

Microsoft Visual C++ supports the creation of OLE automation servers. Servers can be implemented using Microsoft Foundation Classes and the Microsoft Foundation Class application wizard, or as Win32 applications. Servers can be DLLs or EXEs. Refer to the Microsoft Visual C++ documentation and to the OLE samples provided by Microsoft Visual C++ for further information. For information on building Visual C++ UDFs for DB2, see “Building User-Defined Functions (UDFs) with Microsoft Visual C++” on page 41.

DB2 provides self-containing samples of OLE automation UDFs in Microsoft Visual C++, located in the directory `%DB2PATH%\samples\ole\msvc`.

Sample OLE Automation Controller for Stored Procedures

Directory `%DB2PATH%\samples\ole\stpcntr` contains a sample OLE automation controller implemented in Microsoft Visual C++ as a stored procedure. The automation controller can be used to invoke stored procedures through OLE automation. The first SQLVAR in the SQLDA provides the OLE programmable identifier, `progID`, and the name of the method which should be invoked. OLE automation stored procedures must be implemented as in-process OLE automation servers.

The directory `%DB2PATH%\samples\ole\msvb` contains a Visual Basic project, `salarysvr`, with a "median" stored procedure which calculates the median salary in the STAFF table of the DB2 samples database. The stored procedure is implemented in Microsoft Visual Basic and DB2 CLI. The Visual Basic project, `salaryclt`, shows a DB2 client implemented in Visual Basic, which invokes the "median" stored procedure. The directory `%DB2PATH%\samples\ole\msvc` contains a DB2 client program, `salaryclt`, implemented in Microsoft Visual C++, which invokes the "median" stored procedure.

Chapter 5. Building Windows 3.1 Embedded SQL Applications

This chapter provides detailed information for building embedded SQL applications on Windows 3.1. The samples and batch files used in the chapter are in the appropriate language subdirectory of `%DB2PATH%\samples`. 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.

Programming notes for WIN 3.1 applications

The Windows 3.1 operating environment has certain limitations with regard to issuing a NotifyRegister and unloading the DLL in the _WEP procedure. These limitations affect DB2 for Windows 3.1 applications in the following ways.

If the application calls the Windows API NotifyRegister, DB2 for Windows 3.1 will not perform cleanup when that application exits. This may cause a memory leak problem when DB2 for Windows 3.1 is used concurrently by multiple applications.

If the application uses a DLL to load the main DB2 for Windows 3.1 DLL (DB2W.DLL), it is recommended that a connect reset be issued before the DLL is unloaded. The connect reset must not be issued in the _WEP of that DLL.

The Microsoft Windows and WIN-OS/2 Environments

There are two environments for running DB2 applications in Windows 3.1, the Microsoft Windows environment, and the WIN-OS/2 environment which is accessed through OS/2. An important difference between these environments is that WIN-OS/2 does not support the WXServer tool, due to the fact that OS/2 does not support .386 device drivers. The steps you need to run batch files in these environments vary, and are explained in the following sections.

Running Batch Files in a Microsoft Windows Environment

To run the batch files we supply in a Microsoft Windows environment, you must install a tool that lets you run Windows programs from a DOS session. From the DOS session, the batch file invokes the DB2 command line processor, a Windows program, to connect to a database, precompile your source file, and then bind the bind file to the database.

One example of a tool that lets you run Windows programs from a DOS session is the WXServer tool that comes with Microsoft Visual C++. If you do not have such a tool, you can study the batch files to learn what steps you need to perform manually to build your applications, and what compiler options to use.

Using WXServer

Microsoft Visual C++ comes with WXServer, which lets you run Microsoft Windows programs from a DOS session inside Windows 3.1. However, WXServer is not installed when you install the compiler.

To install WXServer, do the following:

1. Get the Microsoft Visual C++ CD-ROM.
2. Go to the `\vc152\bin` directory on the product CD-ROM.

   Note: Depending on the version and release of the Microsoft Visual C++ compiler you are using, the `vc152` directory may be called by a different name. If in doubt, contact Microsoft.
3. Copy the following files from the `\vc152\bin` directory on the product CD-ROM to the `\msvc\bin` directory on your workstation:

   ```
   wx.exe

   wxsrvr.exe

   vmb.386
   ```
4. Edit the `system.ini` file on your workstation. In the `386Enh` section of the file, add the following line:

   ```
   device=c:\msvc\bin\vmb.386
   ```

   If the drive and/or path are different, substitute the correct drive and path.
5. If you are in Windows, exit and restart Windows for the changes to take effect.

Before using the batch files, run the `\msvc\bin\wxsrvr.exe` program to start the server. Its also possible to create an icon for it and place it in the startup folder.

Preparing Your Source Files Without WXServer

If you do not have a tool like WXServer that lets you run Windows programs from a DOS session, do the following to precompile your source file, and to bind the bind file to the DB2 database:

1. Start Windows.
2. Double-click on the Command Line Processor icon in the IBM Database 2 Windows group.

   Alternatively, you can click on **File** and then **Run...** in Windows File Manager. In the Command Line entry field, enter:

   ```
   %DB2PATH%\bin\db2clpw.exe
   ```
3. Execute the following commands from the command line processor prompt:

   ```
   connect to sample

   prep updat.sqx bindfile nolinemacro
   ```

```
bind updat.bnd

connect reset

terminate
```

To execute the above commands, first modify the Command Line Processor Windows Program Item to specify the samples source file directory as the working directory. If you do not do this, you will have to enter the full path for the source file name, as follows:

```
prep %DB2PATH%\samples\updat.sqx bindfile nolinemacro

bind %DB2PATH%\samples\updat.bnd
```

Running Batch Files in a WIN-OS/2 Environment

In this environment, you must run a batch file in a DOS Full Screen session. You cannot run it in a DOS Window.

Additionally, to use the Microsoft Visual C++ compiler in a DOS session, you might need to change the DOS settings shown below. Refer to the OS/2 documentation for information about changing DOS settings.

DPMI_DOS_API	Must be set to ENABLED (default is AUTO).
DPMI_MEMORY_LIMIT	Must be set to 10 or higher.

Microsoft Visual C++

This section presents batch files for building sample C++ programs for both the WIN-OS/2 environment and the Microsoft Windows environment. The batch file `winos2bd.bat` contains the commands to build a sample C++ program using WIN-OS/2. The batch file `winbld.bat` contains the commands to build a sample C++ program using Microsoft Windows.

You can modify either batch file to build a C program. The comments in the batch file describe the modifications you need to make. The batch files can be found in `%DB2PATH%\samples\c`.

The first parameter, `%1`, in each batch file specifies the name of your source file. The second parameter, `%2`, specifies the name of the database to which you want to connect.

The batch files put DB2 command line processor commands to connect to a database and precompile your program into the temporary file `preptmp`. You cannot execute Command Line Processor (CLP) commands directly from a batch file. Refer to the *Command Reference* if you need more information about CLP commands.

Using Winos2bd.bat

You must run `winos2bd.bat` in a DOS Full Screen session. You cannot run `winos2bd.bat` in a DOS Window. Refer to “Running Batch Files in a WIN-OS/2 Environment” on page 67 for details.

```
@echo off
rem  winos2bd.bat file
rem  Build sample C or C++ program containing embedded SQL.

rem  Prepare the source file: connect to a database,
rem  precompile the program, and bind the program to the
rem  database. To build a C program, change the source file
rem  extension to .sqc.
echo connect to %2 > preptmp
echo prep %1.sqx bindfile nolinemacro >> preptmp
echo bind %1.bnd >> preptmp
echo connect reset >> preptmp
echo quit >> preptmp
rem  Invoke command line processor with input file preptmp.
win db2clpw -f preptmp

rem NOTE: Update the following assumed drive and directory to your current
rem       working drive and directory (where your application source files
rem       reside).  This way, the compiler will find your source files after
rem       exiting the Windows environment.

d:
cd \sqllib\win\samples\c

erase preptmp
rem  Compile the program. To build a C program, change the
rem  source file extension to .c.
cl /c /Gy /ALw /W3 /Mq /DDB2WIN %1.cxx
rem  Link the program.
link /ST:32000 /SE:400 /NOD %1.obj util.obj,,,llibcewq+libw+oldnames+db2w;
@echo on
```

Compile and Link Options for winos2bd

The batch file contains the following compile options:

`cl`	The Microsoft Visual C++ compiler.
`/c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`/Gy`	Generate separate functions for the linker.
`/ALw`	Use large memory model.
`/W3`	Set warning level; 1 is most severe, 4 is least severe.
`/Mq`	Use QuickWin compile and include library.
`/DDB2WIN`	Identifies the Windows platform.

Compile and Link Options for winos2bd	
The batch file contains the following link options:	
`link`	Use the Microsoft Visual C++ linker to link edit.
`/ST:32000`	Specify a stack size of 32000.
`/SE:400`	Specify the maximum number of segments.
`/NOD`	Do not use default libraries
`util.obj`	Include error checking object file.
`,,,`	Use the default executable and map filenames.
`llibcewq+libw+oldnames`	Microsoft Visual C++ LIB files.
`db2w`	DB2 SDK for Windows import LIB file.
Refer to your compiler documentation for additional compiler options.	

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

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

   ```
   db2start
   ```

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

   ```
   winos2bd updat sample
   ```

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

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

   ```
   db2start
   ```

2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. If you built the `updat` sample program, enter:

   path`\updat`

   where *path* specifies the location of the executable.

Using Winbld.bat

To use `winbld.bat`, you must have a tool that lets you run Windows programs from a DOS session, such as WXServer, installed and running. Refer to “Running Batch Files in a Microsoft Windows Environment” on page 65 for details.

Run `winbld.bat` from a DOS prompt in Microsoft Windows.

```
@echo off
rem  winbld.bat file
rem  Builds a C or C++ program containing embedded SQL.
rem  Prepare the source file: connect to a database,
rem  precompile the program, and bind the program to the
```

```
rem  database. To build a C program, change the source file
rem  extension to .sqc.
echo connect to %2 > preptmp
echo prep %1.sqx bindfile nolinemacro >> preptmp
echo bind %1.bnd >> preptmp
echo connect reset >> preptmp
echo quit >> preptmp
rem  Invoke command line processor with input file preptmp.
wx db2clpw -f preptmp
erase preptmp
rem  Compile the program. To build a C program, change the
rem  source file extension to .c.
cl /c /Gy /ALw /W3 /Mq /DDB2WIN %1.cxx
rem  Link the program.
link /ST:32000 /SE:400 /NOD %1.obj util.obj,,,llibcewq+libw+oldnames+db2w;
@echo on
```

Compile and Link Options for winbld	
The batch file contains the following compile options:	
`cl`	The Microsoft Visual C++ compiler.
`/c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`/Gy`	Generate separate functions for the linker.
`/ALw`	Use large memory model.
`/W3`	Set warning level; 1 is most severe, 4 is least severe.
`/Mq`	Use QuickWin compile and include library.
`/DDB2WIN`	Identifies the Windows platform.
The batch file contains the following link options:	
`link`	Use the Microsoft Visual C ++ linker to link edit.
`/ST:32000`	Specify a stack size of 32000.
`/SE:400`	Specify the maximum number of segments.
`/NOD`	Do not use default libraries
`util.obj`	Include error checking object file.
`,,,`	Use the default executable and map filenames.
`llibcewq+libw+oldnames`	Microsoft Visual C ++ LIB files.
`db2w`	DB2 SDK for Windows import LIB file.
Refer to your compiler documentation for additional compiler options.	

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

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Build the sample program, connecting to the SAMPLE database, by entering:

```
winbld updat sample
```

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

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

   ```
   db2start
   ```

2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. If you built the `updat` sample program, enter:

   path`\updat`

   where *path* specifies the location of the executable.

Note: To build C applications that do not contain embedded SQL, you can use the batch file `makeapi.bat`. It contains the same compile and link options as `winos2bd.bat` and `winbld.bat`, 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 the Microsoft Visual C++ Client Application for Stored Procedures

Stored procedures are programs that access the database and return information to your Windows client application. You build and store stored procedures on the server. As DB2 for Windows 3.1 is client-only, the server runs on another operating system platform.

To build the stored procedure, `outsrv`, on Windows NT, see Chapter 4, “Building Windows NT and Windows 95 Embedded SQL Applications” on page 35. To build the stored procedure, `outsrv`, on OS/2, see Chapter 6, “Building OS/2 Embedded SQL Applications” on page 85. If you are using a UNIX server, refer to the embedded SQL chapter for that platform in *Building Applications for UNIX Environments*.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using either the `winos2bd.bat` or the `winbld.bat` file. Refer to “Microsoft Visual C++” on page 67 for details.

To run the stored procedure, do the following:

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

   ```
   db2start
   ```

2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. Enter the sample program name:

   path`\outcli` *remote_database userid password*

   where

path	Specifies the location of the executable.
remote_database	Is the name of the database to which you want to connect. The name could be SAMPLE, or its remote alias, or some other name.
userid	Is a valid user ID.
password	Is a valid password.

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

Building the Microsoft Visual C++ Client Application for UDFs

UDFs are your own scalar functions that you build and store on the server. As DB2 for Windows 3.1 is client-only, the server runs on another operating system platform.

To build the user-defined function, `udf`, on Windows NT, see Chapter 4, "Building Windows NT and Windows 95 Embedded SQL Applications" on page 35. To build the user-defined function, `udf`, on OS/2, see Chapter 6, "Building OS/2 Embedded SQL Applications" on page 85. If you are using a UNIX server, refer to the embedded SQL chapter for that platform in *Building Applications for UNIX Environments*.

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build `calludf` using either the `winos2bd.bat` or the `winbld.bat` file. Refer to "Microsoft Visual C++" on page 67 for details.

To run the UDF, do the following:

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. Enter the sample program name:

   path`\calludf`

   where *path* specifies the location of the executable.

   The application calls functions from the `udf` program.

After you run the calling application, you can also invoke the UDF interactively in a DOS session from the DB2 command line processor. To start the processor, double-click on the Command Line Processor icon in the IBM Database 2 Windows group. Connect to the database, then enter:

```
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.

Borland C++

For the Borland C++ compiler, you must run the batch files in a Microsoft Windows environment. You can find the batch files in `%DB2PATH%\samples\c`.

The batch file `bldbprep.bat` puts the DB2 command line processor commands to connect to a database and precompile your program into the temporary file, `preptmp`. You cannot execute command line processor commands directly from a batch file. Refer to the *Command Reference* if you need more information about the commands.

To use `bldbprep.bat`, you must either have a tool installed and running that lets you run Windows programs from a DOS session, such as WXServer, or you must prepare your source file to be used without such a tool. Refer to “Running Batch Files in a Microsoft Windows Environment” on page 65 for details.

```
@echo off
rem  bldbprep.bat file
rem  Prepare the source file: connect to a database,
rem  precompile the program, and bind the program to the
rem  database.

echo connect to %2 > preptmp
echo prep %1.sqx bindfile nolinemacro >> preptmp
echo bind %1.bnd >> preptmp
echo quit >> preptmp
rem  Invoke command line processor with input file preptmp using
rem  the WXServer tool.
wx db2clpw -f preptmp
erase preptmp
@echo on
```

Note: The output file from `bldbprep.bat` has a file extension `.cxx`. To build a C++ program using `bldbor.bat`, change the extension to `.cpp`. To build a C program using `bldbor.bat`, change the extension to `.c`. You may also need to change the drive letter for the path.

After using `bldbprep.bat`, you can use `bldbor.bat` to compile and link the C or C++ file, or you can use the Integrated Development Environment with the compiler options shown in `bldbor.bat`. To use `bldbor.bat`, we recommend that you increase the available XMS and EMS memory for the Windows DOS prompt.

To increase the memory, edit the `dosprmpt.pif` file using the Windows PIF editor. For both XMS and EMS memory, increase the required memory to 4096 KB, and increase the KB Limit to -1. The KB Limit setting of -1 allocates as much memory as the application requests, up to the limit of the system memory. For more information, refer to the online Help for the Windows PIF editor.

The batch file `bldbor.bat` 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.

```
@echo off
rem  bldbor.bat file
rem  Compile and link sample C++ program.

rem  Create a temporary compile response file.
echo /DDB2WIN /DBORLAND /X- /u /A- /k /N /y /v /3 /a2 /p- /ml /dc
     /Vf /Ff /tWS /Od /c /W > bcc.rsp
rem  Compile the program. To build a C program, change the source
rem  file extension to .c. You might need to change the drive letter
rem  for the path.
bcc /ID:\BC4\INCLUDE;C:\SQLLIB\WIN\INCLUDE @bcc.rsp %1.cpp
erase bcc.rsp
rem  Create a temporary link response file. You might need to change
rem  the drive letters for the path.
echo /LD:\BC4\LIB;C:\SQLLIB\WIN\LIB /c /v /n /P /s /l
     /e c0wl+ > tlink.rsp
echo %1.obj+util.obj,%1.exe >> tlink.rsp
echo %1.map >> tlink.rsp
echo db2w+mathwl+cwl+import >> tlink.rsp
echo %default.def >> tlink.rsp
rem  Link the program.
tlink @tlink.rsp
erase tlink.rsp
@echo on
```

Compile and Link Options for bldbor

The batch file contains the following compile options:

`bcc`	The Borland C++ compiler.
`I`*path*	Include path for Borland C++ and DB2 include files.
`/DDB2WIN`	Identifies the Windows platform.
`/DBORLAND`	Compiler define to isolate Borland-only code.
`/X-`	Generate autodependency information.
`/u`	Generate underscores.
`/A-`	Use Borland C++ keywords, not ANSI.
`/k`	Generate standard stack frame.
`/N`	Test for stack overflow (recommended for debugging).
`/y`	Generate line numbers for use with debugger.
`/v`	Generate debug information.
`/3`	Use the instruction set for the 80386 processor.
`/a2`	Align on word boundary.
`/p-`	Use C calling conventions.
`/ml`	Use the large memory model.
`/dc`	Put constant strings in code segments.
`/Vf`	Use far virtual tables.
`/Ff`	Use automatic far data.
`/tWS`	Make windows `.exe` with smart callbacks, and all functions exportable.
`/Od`	Disable all optimizations during testing.
`/c`	Create object file only.
`/W`	Create an EasyWin application.

Compile and Link Options for bldbor	
The batch file contains the following link options:	
`tlink`	Use the Borland C++ linker to link edit.
`L`*path*	Library path for Borland C++ and DB2 library files.
`/v`	Include debug information.
`/n`	Ignore default libraries.
`/P`	Pack code segments.
`/s`	Create detailed map file.
`/l`	Create a section in the map file for source code line numbers.
`/e`	Process extended dictionaries.
`c0wl+`	The command line linker requires that the startup module `cowl.obj` be the first object file in the tlink statement.
`db2w`	DB2 import LIB file.
`mathwl+cwl+import`	Borland C++ LIB files.
`%default.def`	The default definition file.
Refer to your compiler documentation for additional compiler options.	

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

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

   `db2start`

2. Ensure that you have either a tool installed and running that lets you run Windows programs from a DOS session, such as WXServer, or that you have prepared your source file to be used without such a tool. Refer to "Running Batch Files in a Microsoft Windows Environment" on page 65 for details.
3. Precompile the sample program, connecting to the SAMPLE database, by entering:

   `bldbprep updat sample`

4. Build the sample program by entering:

   `bldbor updat`

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

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

   `db2start`

2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. If you built the `updat` sample program, enter:

   path`\updat`

where *path* specifies the location of the executable.

Building the Borland C++ Client Application for Stored Procedures

Stored procedures are programs that access the database and return information to your Windows 3.1 client application. Stored procedures are built and stored on the server. As DB2 for Windows 3.1 is client-only, the server runs on another operating system platform.

To build the stored procedure, `outsrv`, on Windows NT, see Chapter 4, “Building Windows NT and Windows 95 Embedded SQL Applications” on page 35. To build the stored procedure, `outsrv`, on OS/2, see Chapter 6, “Building OS/2 Embedded SQL Applications” on page 85. If you are using a UNIX server, refer to the embedded SQL chapter for that platform in *Building Applications for UNIX Environments*.

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 `bldbprep.bat` and `bldbor.bat` files. Refer to “Borland C++” on page 73 for details.

To run the stored procedure, do the following:

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. Enter the sample program name:

   path\`outcli` *remote_database userid password*

   where

path	Specifies the location of the executable.
remote_database	Is the name of the database to which you want to connect. The name could be SAMPLE, or its remote alias, or some other name.
userid	Is a valid user ID.
password	Is a valid password.

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

Building the Borland C++ Client Application for UDFs

User-Defined Functions (UDFs) are your own scalar functions that you build and store on the server. As DB2 for Windows 3.1 is client-only, the server runs on another operating system platform.

To build the user-defined function, `udf`, on Windows NT, see Chapter 4, “Building Windows NT and Windows 95 Embedded SQL Applications” on page 35. To build the user-defined function, `udf`, on OS/2, see Chapter 6, “Building OS/2 Embedded SQL

Applications" on page 85. If you are using a UNIX server, refer to the embedded SQL chapter for that platform in *Building Applications for UNIX Environments*.

Once you build the user-defined function, `udf`, you can build the client application, `calludf`, that calls it. You can build `calludf` using the `bldbprep.bat` and `bldbor.bat` files. Refer to "Borland C++" on page 73 for details.

To run the stored procedure, do the following:

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

   ```
   db2start
   ```

2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. Enter the sample program name:

   path`\calludf`

   where *path* specifies the location of the executable.

   The application calls functions from the `udf` program.

After you run the calling application, you can also invoke the UDF interactively in a DOS session from the DB2 command line processor. To start the processor, double-click on the Command Line Processor icon in the IBM Database 2 Windows group. Connect to the database, then enter:

```
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.

Micro Focus COBOL

This section presents batch files for building sample COBOL programs for both the Microsoft Windows environment and the WIN-OS/2 environment. The batch file `bldos2cb.bat` contains the commands to build a sample COBOL program using WIN-OS/2. The batch file `bldwincb.bat` contains the commands to build a sample COBOL program using Microsoft Windows.

The first parameter, `%1`, in each batch file specifies the name of your source file. The second parameter, `%2`, specifies the name of the database to which you want to connect.

The batch files put DB2 command line processor commands to connect to a database and precompile your program into the temporary file `preptmp`. You cannot execute Command Line Processor (CLP) commands directly from a batch file. Refer to the *Command Reference* if you need more information about CLP commands.

Using Bldos2cb.bat

You must run `bldos2cb.bat` in a DOS Full Screen session. You cannot run `bldos2cb.bat` in a DOS Window. See “Running Batch Files in a WIN-OS/2 Environment” on page 67 for details on running batch files in a WIN-OS/2 environment.

```
@echo off
rem  bldos2cb.bat file
rem  Build a sample COBOL program.

rem  Prepare the source file: connect to a database,
rem  precompile the program, and bind the program to the
rem  database.
echo connect to %2 > preptmp
echo prep %1.sqb bindfile >> preptmp
echo bind %1.bnd >> preptmp
echo connect reset >> preptmp
echo quit >> preptmp
rem  Invoke command line processor with input file preptmp.
win db2clpw -f preptmp

rem NOTE: Update the following assumed drive and directory to your current
rem       working drive and directory (where your application source files
rem       reside).  This way, the compiler will find your source files after
rem       exiting the Windows environment.

d:
cd \sqllib\win\samples\cobol

erase preptmp
rem  Compile the program.
cobol %1.cbl /notrunc;
rem  Link the program.
link /ST:16000 /se:400 /nod /noe /nopackc
  %1.obj+checkerr.obj+cblwina.obj,,,lcoboldw+lcobol+cobw+db2w+db2gmfw, %1.def;
@echo on
```

Compile and Link Options for bldos2cb

The batch file contains the following compile options:

`cobol`	The Micro Focus COBOL compiler.
`/notrunc`	Do not truncate in decimal to the number of digits given by the PICTURE clause on all stores into COMP items.

Compile and Link Options for bldos2cb	
The batch file contains the following link options:	
`link`	Use the Micro Focus COBOL linker to link edit.
`/ST:16000`	Specify a stack size of at least 16000.
`/se:400`	Specify the maximum number of segments.
`/nod`	Do not use default libraries.
`/noe`	Turn off extended dictionary searches.
`/nopackc`	Turn off code-segment packing.
`checkerr.obj`	Include the error-checking utility object file.
`cblwina.obj`	Micro Focus COBOL Windows object file.
`lcoboldw+lcobol+cobw`	Micro Focus COBOL LIB files.
`db2w`	DB2 SDK for Windows import LIB file.
`db2gmfw`	Link with the DB2 exception-handler library for Windows.
`%1.def`	Module definition file. You can copy and modify the sample Micro Focus COBOL `winhello.def` file. You may need to increase the values for HEAPSIZE and STACKSIZE for larger applications. Also, remove the `EXPORTS` line. The stack size specified in `%1.def` should match the stack size specified using the `/ST:` compiler option.
Refer to your compiler documentation for additional compiler options.	

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

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

   ```
   db2start
   ```

2. Copy the `winhello.def` file from the Micro Focus COBOL install area to `updat.def`.
3. Edit `updat.def` to remove the `EXPORTS` line.
4. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldos2cb updat sample
   ```

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

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

   ```
   db2start
   ```

2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. If you built the `updat` sample program, enter:

   path`\updat`

where *path* specifies the location of the executable.

Using Bldwincb.bat

`Bldwincb.bat` assumes a tool such as WXServer, that lets you run Windows programs from a DOS session, is installed and running. Refer to "The Microsoft Windows and WIN-OS/2 Environments" on page 65 for details.

```
@echo off
rem  bldwincb.bat file
rem  Build a sample COBOL program.

rem  Prepare the source file: connect to a database,
rem  precompile the program, and bind the program to the
rem  database.
echo connect to %2 > preptmp
echo prep %1.sqb bindfile >> preptmp
echo bind %1.bnd >> preptmp
echo connect reset >> preptmp
echo quit >> preptmp

rem  Invoke command line processor with input file preptmp using
rem  a tool such as WXServer. If you are using another tool,
rem  replace wx with the appropriate command.
wx db2clpw -f preptmp
erase preptmp

rem  Compile the program.
cobol %1.cbl /notrunc;

rem  Link the program.
link /ST:16000 /se:400 /nod /noe /nopackc %1.obj+checkerr.obj+cblwina.obj,,,lcob
oldw+lcobol+cobw+db2w+db2gmfw, sample.def;
@echo on
```

Compile and Link Options for bldwincb

The batch file contains the following compile options:

`cobol`	The Micro Focus COBOL compiler.
`/notrunc`	Do not truncate in decimal to the number of digits given by the PICTURE clause on all stores into COMP items.

Compile and Link Options for bldwincb	
The batch file contains the following link options:	
`link`	Use the Micro Focus COBOL linker to link edit.
`/ST:16000`	Specify a stack size of at least 16000.
`/se:400`	Specify the maximum number of segments.
`/nod`	Do not use default libraries.
`/noe`	Turn off extended dictionary searches.
`/nopackc`	Turn off code-segment packing.
`checkerr.obj`	Include the error-checking utility object file.
`cblwina.obj`	Micro Focus COBOL Windows object file.
`lcoboldw+lcobol+cobw`	Micro Focus COBOL LIB files.
`db2w`	DB2 SDK for Windows import LIB file.
`db2gmfw`	Link with the DB2 exception-handler library for Windows.
`sample.def`	
Refer to your compiler documentation for additional compiler options.	

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

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

   ```
   db2start
   ```

2. Copy the `winhello.def` file from the Micro Focus COBOL install area to `updat.def`.
3. Edit `updat.def` to remove the `EXPORTS` line.
4. Build the sample program, connecting to the SAMPLE database, by entering:

   ```
   bldwincb updat sample
   ```

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

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

   ```
   db2start
   ```

2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. If you built the `updat` sample program, enter:

   path`\updat`

   where *path* specifies the location of the executable.

Note: To build Micro Focus COBOL applications that do not contain embedded SQL, you can use the batch file `bldwnapi.bat`. It contains the same compile and link options as `bldos2cb.bat` and `bldwincb.bat`, 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 COBOL.

Building the Micro Focus COBOL Client Application for Stored Procedures

Stored procedures are programs that access the database and return information to your Windows client application. Stored procedures are built and stored on the server. As DB2 for Windows 3.1 is client-only, the server runs on another operating system platform.

To build the stored procedure, `outsrv`, on Windows NT, see Chapter 4, "Building Windows NT and Windows 95 Embedded SQL Applications" on page 35. To build the stored procedure, `outsrv`, on OS/2, see Chapter 6, "Building OS/2 Embedded SQL Applications" on page 85. If you are using a UNIX server, refer to the embedded SQL chapter for that platform in *Building Applications for UNIX Environments*.

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 `bldos2cb.bat` file if you are in the WIN-OS/2 environment, or `bldwincb.bat` if you are in the Microsoft Windows environment. Refer to "Micro Focus COBOL" on page 78 for details.

To run the stored procedure, do the following:

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

   ```
   db2start
   ```

2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. Enter the sample program name:

   path\\`outcli` *remote_database userid password*

   where

path	Specifies the location of the executable.
remote_database	Is the name of the database to which you want to connect. The name could be SAMPLE, or its remote alias, or some other name.
userid	Is a valid user ID.
password	Is a valid password.

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

Building the Micro Focus COBOL Client Application for UDFs

User-Defined Functions (UDFs) are your own scalar functions that you store on the server. As DB2 for Windows 3.1 is client-only, the server runs on another operating system platform.

To build the user-defined function, `udf`, on Windows NT, see Chapter 4, “Building Windows NT and Windows 95 Embedded SQL Applications” on page 35. To build the user-defined function, `udf`, on OS/2, see Chapter 6, “Building OS/2 Embedded SQL Applications” on page 85. If you are using a UNIX server, refer to the embedded SQL chapter for that platform in *Building Applications for UNIX Environments*.

Once you build `udf`, you can build the Micro Focus COBOL client application, `calludf`, that calls it. You can build `calludf` using the `bldos2cb.bat` file if you are in the WIN-OS/2 environment, or `bldwincb.bat` if you are in the Microsoft Windows environment. Refer to “Micro Focus COBOL” on page 78 for details.

To run the UDF, do the following:

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

   ```
   db2start
   ```

2. Start Windows if it is not already started.
3. In Windows File Manager, click on **File** and then **Run...**.
4. Enter the sample program name:

   path`\calludf`

   where *path* specifies the location of the executable.

   The application calls functions from the `udf` program.

After you run the calling application, you can also invoke the UDF interactively in a DOS session from the DB2 command line processor. To start the processor, double-click on the Command Line Processor icon in the IBM Database 2 Windows group. Connect to the database, then enter:

```
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.

Chapter 6. Building OS/2 Embedded SQL Applications

This chapter provides detailed information for building embedded SQL applications on OS/2. In the command 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.

Compound SQL Statements

Compound SQL statements containing user-defined SQLDAs are not permitted in a 16-bit application on OS/2.

C++ Type Decoration Consideration

When writing a stored procedure or a UDF using a C++ compiler on OS/2, you may want to consider declaring the stored procedure or UDF as:

`extern "C"` *procedure or function declaration*

The `extern "C"` prevents type decoration or mangling of the function name by the C++ compiler. Without this declaration, you have to include all the type decoration for the function name when you call the stored procedure, or issue the CREATE FUNCTION statement.

Data Access Builder

VisualAge C++ Version 3 provides a database application development tool called the Data Access Builder. VisualAge C++ corrective service contains Data Access Builder enhancements to access DB2 for OS/2 Version 2.1 databases. This service is included in a CTV30x.ZIP file (where x is a numeric identifier of the particular CSD). To find out if the enhancement is in a particular CTV30x.ZIP file, look in the CTV30x.LST file that can be found in the same place as the CTV30x.ZIP file.

These files are available in the following places:

COMPUSERVE

- GO OS2DF1
- Look in library 4

INTERNET

- Use anonymous ftp to get to the `software.watson.ibm.com` site.
- The files will be in the `pub/os2/cset++` directory.

IBM VisualAge C++ for OS/2

The command file `bldvaemb.cmd`, in `%DB2PATH%\samples\c`, contains the commands to build a sample C program.

You can also use the command file to build a C++ program after you modify it. The comments in the command file describe the modifications you need to make.

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.

```
@echo off
rem bldvaemb command file
rem Builds a C or C++ program that contains embedded SQL
rem Usage: bldvaemb <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program. To build a C++ program, change
rem the source file extension to .sqx.
db2 prep %1.sqc bindfile

rem Compile the util.c error checking utility. For C++,
rem change the source file extension to .cxx
icc -c util.c

rem Compile the program. To build a C++ program, change
rem the source file extension to .cxx.
icc -C+ -O- -Ti+ %1.c

rem Link the program.
ilink /NOFREE /NOI /DEBUG /ST:32000 /PM:VIO %1.obj util.obj,,,db2api;
```

```
rem To use the LINK386 linker, uncomment the following link386 command
rem and comment out the above ilink command.
rem link386 /NOI /DEBUG /ST:32000 /PM:VIO %1.obj util.obj,,,db2api;

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

goto exit

:error
echo Usage: bldvaemb <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldvaemb	
The command file contains the following compile options:	
`icc`	The IBM VisualAge C++ compiler.
`-C+`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-O-`	No optimization. It is easier to use a debugger with optimization off.
`-Ti+`	Generate debugger information
The command file contains the following link options:	
`ilink`	Use the ilink linker to link edit.
`/NOFREE`	No free format.
`/NOI`	No Ignore Case. Force case sensitive identifiers.
`/DEBUG`	Include debugging information.
`/ST:32000`	Specify a stack size of at least 32000.
`/PM:VIO`	Enable the program to run in an OS/2 window.
`util.obj`	Include the object file for error checking.
`db2api`	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. If necessary, go to the window in which you set your environment variables. See "Setting the OS/2 Environment" on page 21 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:

   ```
   bldvaemb 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. If necessary, go to the window in which you set your environment variables.
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 VisualAge C++ applications that do not contain embedded SQL, you can use the command file `bldvaapi.cmd`. It contains the same compile and link options as `bldvaemb.cmd`, 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/C++.

Building IBM VisualAge C++ Stored Procedures

The command file `bldvastp`, in `%DB2PATH%\samples\c`, contains the commands to build a C stored procedure for a DB2 for OS/2 server. The command file compiles the stored procedure into a DLL on the server.

You can also use the command file to build a C++ stored procedure after you modify it. The comments in the command file describe the modifications you need to make.

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 command file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldvastp command file
rem Builds a C or C++ stored procedure
rem Usage: bldvastp <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
```

```
   goto continue
:continue

rem Precompile the program. To build a C++ stored procedure, change
rem the source file extension to .sqx.
db2 prep %1.sqc bindfile

rem Compile the program. To build a C++ stored procedure, change
rem the source file extension to .cxx
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c

rem Link the program.
ilink /NOFREE /NOI /DEBUG /ST:32000 %1.obj,%1.dll,,db2api,%1.def;

rem To use LINK386 linker, comment out the above ilink command
rem and uncomment the following link386 command.
rem link386 /NOI /DEBUG /ST:32000 %1.obj,%1.dll,,db2api,%1.def;

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem Copy stored procedure to the %DB2PATH%\function directory.
rem Substitute the path where DB2 is installed for %DB2PATH%.
copy %1.dll %DB2PATH%\function

goto exit

:error
echo Usage: bldvastp <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldvastp

The command file contains the following compile options:

`icc`	The IBM VisualAge C++ compiler.
`-C+`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-Ti+`	Generate debugger information.
`-Ge-`	Build a .DLL file. Use the version of the runtime library that is statically linked.
`-Gm+`	Link with multitasking libraries.
`-W2`	Output warning, error, and severe and unrecoverable error messages.

Compile and Link Options for bldvastp

The command file contains the following link options:

`ilink`	Use the ilink linker to link edit.
`/NOFREE`	No free format.
`/NOI`	No Ignore Case. Force case sensitive identifiers.
`/DEBUG`	Include debugging information.
`/ST:32000`	Specify a stack size of at least 32000.
`%1.dll`	Include the dynamic link library.
`db2api`	Link with the DB2 library.
`%1.def`	Module definition file.

Refer to your compiler documentation for additional compiler options.

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

1. If necessary, go to the window in which you set your environment variables. Refer to "Setting the OS/2 Environment" on page 21 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:
   ```
   bldvastp outsrv
   ```
   The command file uses the module definition file `outsrv.def`, contained in the same directory as the sample programs, to build the stored procedure. The command file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` by using the `bldvaemb` command file. Refer to "IBM VisualAge C++ for OS/2" on page 86 for details.

To run the stored procedure, do the following:

1. If necessary, go to the window in which you set your environment variables.

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 IBM VisualAge C++ User-Defined Functions (UDFs)

The command file `bldvaudf`, in `%DB2PATH%\samples\c`, contains the commands to build a C 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`.

You can also use the command file to build a C++ UDF after you modify it. The comments in the command file describe the modifications you need to make.

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

The command file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem  bldvaudf command file
rem  Builds a C or C++ user-defined function (UDF)
rem  Usage: bldvaudf <UDF_name>

rem Compile the program. To build a C++ UDF, change
rem the source file extension to .cxx.
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c

rem Link the program.
ilink /NOFREE /MAP /NOI /DEBUG /ST:32000 %1.obj,%1.dll,,db2api db2apie,%1.def;

rem To use the LINK386 linker, uncomment the following link386 command
rem and comment out the above ilink command.
rem link386 /MAP /NOI /DEBUG /ST:32000 %1.obj,%1.dll,,db2api db2apie,%1.def;

rem Copy the UDF to the %DB2PATH%\function directory
copy %1.dll %DB2PATH%\function
@echo on
```

Compile and Link Options for bldvaudf	
The command file contains the following compile options:	
`icc`	The IBM VisualAge C++ compiler.
`-C+`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-Ti+`	Generate debugger information.
`-Ge-`	Build a .DLL file. Use the version of the runtime library that is statically linked.
`-Gm+`	Link with multitasking libraries.
`-W2`	Output warning, error, and severe and unrecoverable error messages.
The command file contains the following link options:	
`ilink`	Use the ilink linker to link edit.
`/NOFREE`	No free format.
`/MAP`	Generate a map file.
`/NOI`	No Ignore Case. Force case sensitive identifiers.
`/DEBUG`	Include debugging information.
`/ST:32000`	Specify a stack size of at least 32000.
`%1.dll`	Include the dynamic link library.
`db2api`	Link with the DB2 library.
`db2apie`	Link with the DB2 API Engine library.
`%1.def`	Module definition file.
Refer to your compiler documentation for additional compiler options.	

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

1. If necessary, go to the window in which you set your environment variables. Refer to “Setting the OS/2 Environment” on page 21 if you need more information.
2. Build the user-defined function by entering:

   ```
   bldvaudf udf
   ```

   The command file uses the module definition file, `udf.def`, contained in the same directory as the sample programs, to build the user-defined function. The command file copies the user-defined function DLL, `udf.dll`, to the server in the path `%DB2PATH%\function` to indicate that the UDF is fenced. If you want the UDF to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build `calludf` by using the `bldvaemb` command file. Refer to "IBM VisualAge C++ for OS/2" on page 86 for details.

To run the UDF, do the following:

1. If necessary, go to the window in which you set your environment variables.
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 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. Connect to the database, then enter:

   ```
   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.

Borland C++ for OS/2

The command file `bldbremb.cmd`, in `%DB2PATH%\samples\c`, contains the commands to build a sample Borland C program.

You can also use the text of the command file to build a Borland C++ program after you modify it. The comments in the command file describe the modifications you need to make.

The first parameter, `%1`, specifies the name of the 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.

```
@echo off
rem bldbremb command file
rem Builds a Borland C or C++ program that contains embedded SQL
rem Usage: bldbremb <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
```

```
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program. For a C++ program, change the target BORLAND_C
rem to BORLAND_CPLUSPLUS, and use either the .sqc or .sqx extension.
db2 prep %1.sqc bindfile target BORLAND_C

rem Compile the program. For a C++ program, change .c to .cpp, and add the -P
rem option to bcc. Note: All C/C++ files that include DB2 header files, and have not
rem been precompiled using one of the targets BORLAND_C or BORLAND_CPLUSPLUS,
rem must be compiled using the -DSQL_BORLAND_C_OR_CPLUSPLUS define.
rem For example, the file util.c must be compiled with this define.
bcc -I%db2path%\include -c -w- -a4 -DSQL_BORLAND_C_OR_CPLUSPLUS %1.c util.c

rem Link the program (using the Borland linker).
tlink /v /Toe /Le:\bcos2\lib e:\bcos2\lib\c02.obj %1.obj
    util.obj,%1.exe,,%db2path%\lib\db2api c2 os2

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

goto exit

:error
echo Usage: bldbremb <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldbremb	
The command file contains the following compile options:	
`bcc`	The Borland C++ compiler.
`-I%db2path%\include`	Search path for the DB2 header files.
`-c`	Perform compile only; no link. This section assumes that compile and link are separate steps.
`-P`	For C++ only. Perform a C++ compile regardless of source file extension.
`-w-`	Do not display compiler warnings.
`-a4`	Align to double word boundaries.
`-DSQL_BORLAND_C_OR_CPLUSPLUS`	Compile a C or C++ file using the Borland C++ compiler. All C/C++ files that include DB2 header files and have not been precompiled using one of the targets BORLAND_C or BORLAND_CPLUSPLUS must be compiled using the -DSQL_BORLAND_C_OR_CPLUSPLUS define. The file util.c is an example of a file that must be compiled with this define.
The command file contains the following link options:	
`tlink`	The Borland C++ linker.
`/v`	Includes full symbolic debugging information.
`/Toe`	Build an .exe file.
`/Le:\bcos2\lib`	Search path for the Borland C++ libraries (assuming that the Borland C++ libraries are installed in e:\bcos2\lib).
`e:\bcos2\lib\c02.obj`	Borland initialization module for programs (assuming that the Borland C++ libraries are installed in e:\bcos2\lib). This must be the first object file in the list.
`util.obj`	Include this object file for error checking.
`%db2path%\lib\db2api`	Link with the DB2 library.
`c2`	Link with the Borland run-time library.
`os2`	Link with the OS/2 import library.
Refer to your compiler documentation for additional compiler options.	

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

1. If necessary, go to the window in which you set your environment variables. Refer to “Setting the OS/2 Environment” on page 21 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:

   ```
   bldbremb updat
   ```

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

1. If necessary, go to the window in which you set your environment variables.
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 Borland C++ applications that do not contain embedded SQL, you can use the command file `bldbrapi.cmd`. It contains the same compile and link options as `bldbremb.cmd`, 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/C++.

Building Borland C++ Stored Procedures

The command file `bldbrstp.cmd`, in `%DB2PATH%\samples\c`, contains the commands to build a Borland C stored procedure for a DB2 for OS/2 server. The command file compiles the stored procedure into a DLL on the server.

You can also use the command file to build a Borland C++ stored procedure after you modify it. The comments in the command file describe the modifications you need to make.

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 command file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldbrstp command file
rem Builds a Borland C or C++ stored procedure
rem Usage: bldbrstp <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue
```

```
rem Precompile the program. For a C++ program, change the target BORLAND_C
rem to BORLAND_CPLUSPLUS, and use either the .sqc or the .sqx extension.
db2 prep %1.sqc bindfile target BORLAND_C

rem Compile the program. For a C++ stored procedure, change .c to .cpp and add the
rem -P option to bcc. Note: All C/C++ files that include DB2 header files and have
rem not been precompiled using one of the targets BORLAND_C or BORLAND_CPLUSPLUS,
rem must be compiled using the -DSQL_BORLAND_C_OR_CPLUSPLUS define.
rem For example, the file util.c must be compiled with this define.
bcc -I%db2path%\include -c -w- -a4 -DSQL_BORLAND_C_OR_CPLUSPLUS %1.c

rem Link the program (using the Borland linker).
tlink /v /Tod /Le:\bcos2\lib e:\bcos2\lib\c02d.obj
    %1.obj,%1.dll,,%db2path%\lib\db2api c2 os2,%1.def

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem Copy the dynamic link library to the function subdirectory.
rem Note: Substitute the DB2 instance directory for %db2path%
copy %1.dll %db2path%\function

goto exit

:error
echo Usage: bldbrstp <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldbrstp

The command file contains the following compile options:

`bcc`	The Borland C++ compiler.
`-I%db2path%\include`	Search path for the DB2 header files.
`-c`	Perform compile only; no link. This section assumes that compile and link are separate steps.
`-P`	For C++ only. Perform a C++ compile regardless of source file extension.
`-w-`	Do not display compiler warnings.
`-a4`	Align to double word boundaries.
`-DSQL_BORLAND_C_OR_CPLUSPLUS`	Compile a C or C++ file using the Borland C++ compiler. All C/C++ files that include DB2 header files and have not been precompiled using one of the targets BORLAND_C or BORLAND_CPLUSPLUS must be compiled using the -DSQL_BORLAND_C_OR_CPLUSPLUS define. The file util.c is an example of a file that must be compiled with this define.

Compile and Link Options for bldbrstp

The command file contains the following link options:

`tlink`	The Borland C++ linker.
`/v`	Includes full symbolic debugging information.
`/Tod`	Build a DLL.
`/Le:\bcos2\lib`	Search path for the Borland C++ libraries (assuming that the Borland C++ libraries are installed in e:\bcos2\lib).
`e:\bcos2\lib\c02d.obj`	Borland initialization module for programs (assuming that the Borland C++ libraries are installed in e:\bcos2\lib). This must be the first object file in the list.
`util.obj`	Include this object file for error checking.
`%db2path%\lib\db2api`	Link with the DB2 library.
`c2`	Link with the Borland run-time library.
`os2`	Link with the OS/2 import library.
`%1.def`	Module definition file.

Refer to your compiler documentation for additional compiler options.

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

1. If necessary, go to the window in which you set your environment variables. Refer to "Setting the OS/2 Environment" on page 21 if you need more information.
2. The command file uses the module definition file, `outsrv.def`, contained in the same directory as the sample programs, to build the stored procedure. Edit the outsrv.def file to make the following changes:
   a. Remove `TERMINSTANCE` from the line:
   ```
   LIBRARY OUTSRV INITINSTANCE TERMINSTANCE
   ```
   b. Remove the line:
   ```
   SHARED
   ```
3. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
4. Build the stored procedure, connecting to the SAMPLE database, by entering:
   ```
   bldbrstp outsrv
   ```
   The command file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

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 `bldbremb` command file. Refer to "Borland C++ for OS/2" on page 93 for details.

To run the stored procedure, do the following:

1. If necessary, go to the window in which you set your environment variables.
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 Borland C++ User-Defined Functions (UDFs)

The command file `bldbrudf.cmd`, in `%DB2PATH%\samples\c`, contains the commands to build a Borland C UDF. UDFs are compiled like stored procedures, but you do not need to connect to a database or precompile and bind the program.

You can also use the text of the command file to build a Borland C++ UDF after you modify it. The comments in the command file describe the modifications you need to make.

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 command file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem  bldbrudf command file
rem  Builds a C or C++ user-defined function (UDF)
rem  Usage: bldbrudf <UDF_name>
rem Compile the program. To build a C++ user-defined function (UDF),
rem change the source file extension from .c to .cpp, and add
rem the -P option to bcc. Note: All C/C++ files that include DB2
rem header files, and have not been precompiled using one of the
rem targets BORLAND_C or BORLAND_CPLUSPLUS, must be compiled
rem using the -DSQL_BORLAND_C_OR_CPLUSPLUS define.
rem For example, the file util.c must be compiled with this define.
bcc -I%db2path%\include -c -w- -a4 -DSQL_BORLAND_C_OR_CPLUSPLUS %1.c

rem Link the program (using the Borland linker).
tlink /v /Tod /Le:\bcos2\lib e:\bcos2\lib\c02d.obj
    %1.obj,%1.dll,,%db2path%\lib\db2api %db2path%\lib\db2apie c2 os2,%1.def

rem Copy the dynamic link library to the function subdirectory.
rem Note: Substitute the DB2 instance directory for %db2path%
copy %1.dll %db2path%\function

@echo on
```

Compile and Link Options for bldbrudf

The command file contains the following compile options:

`bcc`	The Borland C++ compiler.
`-I%db2path%\include`	Search path for the DB2 header files.
`-c`	Perform compile only; no link. This section assumes that compile and link are separate steps.
`-P`	For C++ only. Perform a C++ compile regardless of source file extension.
`-w-`	Do not display compiler warnings.
`-a4`	Align to double word boundaries.
`-DSQL_BORLAND_C_OR_CPLUSPLUS`	Compile a C or C++ file using the Borland C++ compiler. All C/C++ files that include DB2 header files and have not been precompiled using one of the targets BORLAND_C or BORLAND_CPLUSPLUS must be compiled using the -DSQL_BORLAND_C_OR_CPLUSPLUS define. The file util.c is an example of a file that must be compiled with this define.

Compile and Link Options for bldbrudf

The command file contains the following link options:

`tlink`	The Borland C++ linker.
`/v`	Includes full symbolic debugging information.
`/Tod`	Build a DLL.
`/Le:\bcos2\lib`	Search path for the Borland C++ libraries (assuming that the Borland C++ libraries are installed in e:\bcos2\lib).
`e:\bcos2\lib\c02d.obj`	Borland initialization module for programs (assuming that the Borland C++ libraries are installed in e:\bcos2\lib). This must be the first object file in the list.
`%db2path%\lib\db2api`	Link with the DB2 library.
`%db2path%\lib\db2apie`	Link with the DB2 API Engine library.
`c2`	Link with the Borland run-time library.
`os2`	Link with the OS/2 import library.
`%1.def`	Module definition file.

Refer to your compiler documentation for additional compiler options.

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

1. If necessary, go to the window in which you set your environment variables. Refer to "Setting the OS/2 Environment" on page 21 if you need more information.
2. The command file uses the module definition file, `udf.def`, contained in the same directory as the sample programs, to build the user-defined function. Edit the `udf.def` file to make the following changes:
   a. Remove `TERMINSTANCE` from the line:
      ```
      LIBRARY OUTSRV INITINSTANCE TERMINSTANCE
      ```
   b. Remove the line:
      ```
      SHARED
      ```
3. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
4. Build the UDF by entering:
   ```
   bldbrudf udf
   ```
   The command file copies the user-defined function DLL, `udf.dll`, to the server in the path `%DB2PATH%\function` to indicate that the UDF is fenced. If you want the UDF to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build `udf`, you can build the application `calludf` that calls it. You can build `calludf` using the `bldbremb` command file. Refer to “Borland C++ for OS/2” on page 93 for details.

To run the UDF, do the following:

1. If necessary, go to the window in which you set your environment variables.
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 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. Connect to the database, then enter:

   ```
   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.

IBM VisualAge for COBOL for OS/2

The command file `bldibmcb.cmd`, in `%DB2PATH%\samples\cobol`, 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.

```
@echo off
rem bldibmcb command file
rem Builds a COBOL program that contains embedded SQL
rem Usage: bldibmcb <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
```

```
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqb bindfile target ibmcob

rem Compile the checkerr error checking utility.
cob2 -c -g -qpgmname(mixed) -qlib -I%DB2PATH%\include\cobol_a checkerr.cbl

rem Compile the program.
cob2 -c -g -qpgmname(mixed) -qlib -I%DB2PATH%\include\cobol_a %1.cbl

rem Link the program.
ilink %1.obj checkerr.obj db2api.lib /ST:32000 /PM:VIO /NOI /DEBUG

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

goto exit

:error
echo Usage: bldibmcb <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldibmcb

The command file contains the following compile options:

`cob2`	The IBM COBOL compiler.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-g`	Include debug information.
`-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%DB2PATH%\include\cobol_a`.

Compile and Link Options for bldibmcb	
The command file contains the following link options:	
`ilink`	Use the ilink linker to link edit.
`checkerr.obj`	Include the error-checking utility object file.
`db2api.lib`	Link with the DB2 library.
`/ST:32000`	Specify a stack size of at least 32000.
`/PM:VIO`	Enable the program to run in an OS/2 window.
`/NOI`	Do not ignore case when linking.
`/DEBUG`	Include debugging information.
Refer to your compiler documentation for additional compiler options.	

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

1. If necessary, go to the window in which you set your environment variables. Refer to "Setting the OS/2 Environment" on page 21 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:
   ```
   bldibmcb 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. If necessary, go to the window in which you set your environment variables.
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 VisualAge for COBOL applications that do not contain embedded SQL, you can use the command file `bldapicb.cmd`. It contains the same compile and link options as `bldibmcb.cmd`, 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 COBOL.

Building IBM VisualAge for COBOL Stored Procedures

The command file `bldicobs`, in `%DB2PATH%\samples\cobol`, contains the commands to build a stored procedure. The command file compiles the stored procedure into a DLL on the server.

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 command file uses the source file name, %1, for the DLL name.

```
@echo off
rem bldicobs command file
rem Builds a COBOL stored procedure
rem Usage: bldicobs <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqb bindfile target ibmcob

rem Compile the program.
cob2 -c -g -qpgmname(mixed) -qlib -I%DB2PATH%\include\cobol_a %1.cbl

rem Link the program.
ilink %1.obj checkerr.obj %1.def db2api.lib /ST:32000 /PM:VIO /NOI /DEBUG

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem Copy stored procedure to the %DB2PATH%\function directory.
rem Substitute the path where DB2 is installed for %DB2PATH%.
copy %1.dll %DB2PATH%\function

goto exit

:error
echo Usage: bldicobs <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldicobs

The command file contains the following compile options:

`cob2`	The IBM COBOL compiler.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-g`	Include debug information.
`-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%DB2PATH%\include\cobol_a`.

The command file contains the following link options:

`ilink`	Use the ilink linker to link edit.
`checkerr.obj`	Include the error-checking utility object file.
`%1.def`	Module definition file.
`db2api.lib`	Link with the DB2 library.
`/ST:32000`	Specify a stack size of at least 32000.
`/PM:VIO`	Enable the program to run in an OS/2 window.
`/NOI`	Do not ignore case when linking.
`/DEBUG`	Include debugging information.

Refer to your compiler documentation for additional compiler options.

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

1. If necessary, go to the window in which you set your environment variables. Refer to “Setting the OS/2 Environment” on page 21 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:

   ```
   bldicobs outsrv
   ```

   The command file uses the module definition file `outsrv.def`, contained in the same directory as the sample programs, to build the stored procedure. The command file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

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

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 `bldibmcb` command file. Refer to “IBM VisualAge for COBOL for OS/2” on page 102 for details.

To run the stored procedure, do the following:

1. If necessary, go to the window in which you set your environment variables.
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 VisualAge for COBOL Compiler

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the IBM VisualAge for COBOL 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 your source files to set compile options. Place the keywords in columns 8 to 72 only.
- 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 VisualAge for COBOL reside in `%DB2PATH%\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 (16-bit)

The command file `bldmfcob.cmd`, in `%DB2PATH%\samples\cobol_16`, 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.

```
@echo off
rem bldmfcob command file (for building examples containing embedded SQL)
rem Usage: bldmfcob <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqb bindfile target mfcob16

rem Compile the checkerr error checking utility
cobol checkerr.cbl /NOTRUNC;

rem Compile the program.
cobol %1.cbl /NOTRUNC;

rem Link the program.
link /ST:32000 /PM:VIO /NOI
    %1.obj+checkerr.obj,,,coblib+os2286.lib+sqldyn16+db2gmf16;

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

goto exit

:error
echo Usage: bldmfcob <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldmfcob

The command file contains the following compile options:

`cobol`	The Micro Focus COBOL compiler.
`/NOTRUNC`	Do not truncate in decimal to the number of digits given by the PICTURE clause on all stores into COMP items.

The command file contains the following link options:

`link`	Use the 16-bit linker to link edit.
`/ST:32000`	Specify a stack size of at least 32000.
`/PM:VIO`	Enable the program to run in an OS/2 window.
`/NOI`	Do not ignore case when linking.
`checkerr.obj`	Include the error-checking utility object file.
`coblib`	Micro Focus COBOL library.
`os2286.lib`	Link with the 16-bit OS/2 library.
`sqldyn16`	Link with the 16-bit DB2 library.
`db2gmf16`	Link with the DB2 exception-handler library for 16-bit M. F. COBOL.

Refer to your compiler documentation for additional compiler options.

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

1. If necessary, go to the window in which you set your environment variables. Refer to "Setting the OS/2 Environment" on page 21 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. If necessary, go to the window in which you set your environment variables.
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 command file `bldmfapi.cmd`. It contains the same compile and link options as `bldmfcob.cmd`, 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 COBOL.

Building 16-bit Micro Focus COBOL Stored Procedures

The command file `bldmfcbs`, in `%DB2PATH%\samples\cobol_16`, contains the commands to build a stored procedure. The command file compiles the stored procedure into a DLL on the server.

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 command file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldmfcbs command file (for building a stored procedure)
rem Usage: bldmfcbs <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqb bindfile target mfcob16

rem Compile the program.
cobol %1.cbl /NOTRUNC;

rem Link the program.
link /ST:32000 /PM:VIO /NOI /NOD
    %1.obj,%1.dll,,lcobol+os2286+sqldyn16+db2gmf16,%1.def;

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem Copy stored procedure to the %DB2PATH%\function directory.
rem Substitute the path where DB2 is installed for %DB2PATH%.
copy %1.dll %DB2PATH%\function

goto exit
```

```
:error
echo Usage: bldmfcbs <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldmfcbs

The command file contains the following compile options:

`cobol`	The Micro Focus COBOL compiler.
`/NOTRUNC`	Do not truncate in decimal to the number of digits given by the PICTURE clause on all stores into COMP items.

The command file contains the following link options:

`link`	Use the 16-bit linker to link edit.
`/ST:32000`	Specify a stack size of at least 32000.
`/PM:VIO`	Enable the program to run in an OS/2 window.
`/NOI`	Do not ignore case when linking.
`/NOD`	Ignore library files referenced in object files, and accept libraries input on the command line.
`%1.dll`	Include the dynamic link library.
`lcoblib`	Micro Focus COBOL library.
`os2286`	Link with the 16-bit OS/2 library.
`sqldyn16`	Link with the 16-bit DB2 library.
`db2gmf16`	Link with the DB2 exception-handler library for 16-bit M. F. COBOL.
`%1.def`	Module definition file.

Refer to your compiler documentation for additional compiler options.

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

1. If necessary, go to the window in which you set your environment variables. Refer to “Setting the OS/2 Environment” on page 21 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:

   ```
   bldmfcbs outsrv
   ```

   The command file uses the module definition file `outsrv.def`, contained in the same directory as the sample programs, to build the stored procedure. The command file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build the stored procedure, `outsrv`, you can build the client application, `outcli`, that calls it. You can build `outcli` using the `bldmfcob` command file. See "Micro Focus COBOL (16-bit)" on page 107 for details.

To run the stored procedure, do the following:

1. If necessary, go to the window in which you set your environment variables.
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 16-bit Micro Focus COBOL Compiler

Compound SQL statements containing user-defined SQLDAs are not permitted in a 16-bit application on OS/2.

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 mfcob16` option.
- The DB2 COPY files for Micro Focus COBOL reside in `%DB2PATH%\include\cobol_mf`. Set the `COBCPY` environment variable to include the directory like this:

  ```
  set COBCPY=%DB2PATH%\include\cobol_mf;%COBCPY%
  ```

- If you develop a 16-bit Micro Focus COBOL application that calls DB2 APIs, you must make the following changes from what appears in the *API Reference*:

  1. Add two underscore characters before the API name.
  2. Reverse the order of the parameters.

  For example:

```
call "__sqlgback" using
by reference db-name
by value 0
by reference sqlca
by value drive
         type
         db-name-l
         0.
```

- The following table lists the APIs available to 16-bit applications. These are all DB2 Version 1 APIs. These are the only APIs on OS/2 for which conversion between 16 and 32 bit is provided. The *DATABASE 2 OS/2 Programming Reference* (S62G-3666) describes them. That book is not part of the DB2 library, but you can order it from IBM. See “Contacting IBM” on page 155 for details.

  DB2 Version 2 and DB2 Universal Database Version 5 APIs cannot be called from 16-bit COBOL applications. To use these APIs, and especially to take advantage of the enhanced APIs in Version 5, you are recommended to move to 32-bit Micro Focus COBOL. See “Micro Focus COBOL (32-bit)” on page 114 for details.

API	Name	Purpose
SQLGBACK	Backup Database	Creates a backup copy of a database.
SQLGBNDR	Bind	Invokes the bind utility.
SQLGCATD	Catalog Database	Stores database location information in the system database directory.
SQLGCRDB	Create Database	Initializes a new database with an optional user-defined collating sequence, creates the system tables, and allocates the recovery log.
SQLGCHG	Change Database Comment	Allows changing of a database comment.
SQLGDOPS	Open Database Directory Scan	Stores a copy in memory of the system or a volume database directory, and returns the number of entries.
SQLGDRES	Restore a Database	Rebuild a damaged or corrupted database to the state it was in at the time it was backed up.
SQLGEUDB	Update Database Configuration File	Allows individual entries in a specific database configuration file to be modified.
SQLGEXP	Export From	Exports data from a database to one of several external file formats.
SQLGFREE	Terminate Database Status	Releases all resources obtained by the COLLECT DATABASE STATUS API.

API	Name	Purpose
SQLGFROL	Roll Forward a Database	Rolls forward a database by applying transactions recorded and retained in the archive log files.
SQLGIMP	Import to	Inserts data from an external file with a supported file format into a table or view.
SQLGNEXT	Get Next Database Status Block	Obtains information about the next database or group of databases following a call to the COLLECT DATABASE STATUS API.
SQLGOPST	Collect Database Status	Collects a brief summary of database activity at the time of the request.
SQLGRDBC	Reset Database Configuration File	Resets the configuration of a specific database to the system defaults.
SQLGREOR	Reorganize Table	Reorganizes the physical storage of a table.
SQLGREST	Restart Database	Restarts a database that has been abnormally terminated and left in an uncommitted state.
SQLGSTAT	Run Statistics	Updates statistics about the physical characteristics of a table and the associated indexes.
SQLGSTPD	Stop Using Database	Disconnects an application from a database.
SQLGUSER	Get User Status	Returns the status of all users connected to a specific database.
SQLGXDBC	Get Database Configuration File	Returns values of individual entries in a specific database configuration.

Micro Focus COBOL (32-bit)

The command file `bldmfcob`, in `%DB2PATH%\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 Parameter `%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.

```
@echo off
rem bldmfcob.cmd file
rem Build sample Cobol program using the Micro Focus COBOL compiler.
rem Usage: bldmfcob <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program. If target mfcob is
rem not specified target mfcob16 is assumed.
db2 prep %1.sqb bindfile target mfcob

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem Compile the error-checking utility.
cobol checkerr.cbl;

rem  Compile the program.
cobol %1.cbl;

rem  Link the program.
cbllink %1.obj checkerr.obj db2api.lib db2gmf32.lib

goto exit

:error
echo Usage: bldmfcob <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldmfcob	
The command file contains the following compile option:	
`cobol`	The Micro Focus COBOL compiler.
The command file contains the following link options:	
`cbllink`	Use the linker to link edit.
`checkerr.obj`	Include the error-checking utility object file.
`db2api.lib`	Link with the DB2 API library.
`db2gmf32.lib`	Link with the DB2 exception-handler library for 32-bit M. F. COBOL.
Refer to your compiler documentation for additional compiler options.	

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

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Build the sample program, connecting to the SAMPLE database. From a DB2 Command Line Processor command window, enter:
   ```
   bldmfcob updat
   ```

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

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. From a command line, enter:
   ```
   updat
   ```

Note: To build 32-bit Micro Focus COBOL applications that do not contain embedded SQL, you can use the command file `bldmfapi.cmd`. It contains the same compile and link options as `bldmfcob.cmd`, 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 COBOL.

Building 32-bit Stored Procedures with Micro Focus COBOL

The command file `bldmfcbs`, in `%DB2PATH%\samples\cobol_mf`, contains the commands to build a stored procedure. The command file compiles the stored procedure into a DLL on the server.

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 parameter `%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 command file uses the source file name, `%1`, for the DLL name.

```
@echo off
rem bldmfcbs.cmd file
rem Build sample COBOL stored procedure using Micro Focus COBOL compiler.
rem Usage: bldmfcbs <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program. If target mfcob is
rem not specified target mfcob16 is assumed.
db2 prep %1.sqb bindfile target mfcob

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem  Compile the stored procedure.
cobol %1.cbl;

rem  Link the stored procedure and create a shared library.
cbllink /d %1.obj db2api.lib db2gmf32.lib

rem Copy stored procedure to the %DB2PATH%\function directory.
rem Substitute the path where DB2 is installed for %DB2PATH%.
copy %1.dll %DB2PATH%\function

goto exit

:error
echo Usage: bldmfcbs <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for blmfcbs	
The command file contains the following compile option:	
`cobol`	The Micro Focus COBOL compiler.
The command file contains the following link options:	
`cbllink`	Use the Micro Focus COBOL linker to link edit.
`/d`	Create a .dll file.
`db2api.lib`	Include the DB2 API library.
`db2gmf32.lib`	Link with the DB2 exception-handler library for 32-bit M. F. COBOL.
Refer to your compiler documentation for additional compiler options.	

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

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

   ```
   db2start
   ```

2. Build the stored procedure, connecting to the SAMPLE database. In a DB2 Command Line Processor command window, enter:

   ```
   bldmfcbs outsrv
   ```

   The linker uses a default entry point unspecified by the user. The `/d` option is used to create the `.dll` file in order to build the stored procedure. The command file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

Once you build the stored procedure `outsrv`, you can build `outcli` that calls the stored procedure. You can build `outcli` using the `bldmfcob.cmd` file. Refer to “Micro Focus COBOL (16-bit)” on page 107 for details.

To run the stored procedure, do the following:

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

```
db2start
```

2. From the command line, enter:

   ```
   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 32-bit Micro Focus COBOL Compiler

When building applications with the Micro Focus COBOL compiler that contain embedded SQL and DB2 API calls, keep the following points in mind:

- When you precompile your application using the command line processor command `db2 prep`, use the `target mfcob` option.
- Ensure the LIB environment variable points to `%DB2PATH%\lib` like this:

  ```
  set LIB=%DB2PATH%\lib;%LIB%
  ```

- The DB2 COPY files for Micro Focus COBOL reside in `%DB2PATH%\include\cobol_mf`. Set the `COBCPY` environment variable to include the directory like this:

  ```
  set COBCPY=%DB2PATH%\include\cobol_mf;%COBCPY%
  ```

- The DB2API.LIB provides the import library for COBOL programs and is located in the `lib` directory in the DB2 for OS/2 install directory.

Calls to all DB2 application programming interfaces and generated code must be made using calling convention 8. The DB2 COBOL precompiler automatically inserts a CALL-CONVENTION clause in a SPECIAL-NAMES paragraph. If the SPECIAL-NAMES paragraph does not exist, the DB2 COBOL precompiler creates it, as follows:

```
Identification Division
Program-ID. "static".
special-names.
    call-convention 8 is DB2API.
```

Also, the precompiler automatically places the symbol DB2API, which is used to identify the calling convention, after the "call" keyword whenever a DB2 API is called. This occurs, for instance, whenever the precompiler generates a DB2 API runtime call from an embedded SQL statement.

If calls to DB2 APIs are made in an application which is not precompiled, you should manually create a SPECIAL-NAMES paragraph in the application, similar to that given above. If you are calling a DB2 API directly, then you will need to manually add the DB2API symbol after the "call" keyword.

FORTRAN 77

The command file `bldfor`, in `%DB2PATH%\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.

```
@echo off
rem bldfor command file
rem Builds a FORTRAN program that contains embedded SQL
rem Usage: bldfor <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqf bindfile

rem Compile the util.for error-checking utility.
wfc386 /debug /d2 /noref util.for

rem Compile the program.
wfc386 /debug /d2 /noref %1.for

rem Link the program.
wlink debug all sys os2v2 file %1.obj file util.obj library db2api.lib
    option stack=32000

rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

goto exit

:error
echo Usage: bldfor <prog_name> [ <db_name> [ < userid> <password> ]]
```

```
:exit
@echo on
```

Compile and Link Options for bldfor

The command file contains the following compile options:

`wfc386`	The FORTRAN compiler.
`debug`	Perform runtime checking.
`d2`	Include full debugging information.
`noref`	Do not issue warnings about unreferenced symbols. This will avoid extraneous warnings.

The command file contains the following link options:

`wlink`	Use the WATCOM linker to link edit.
`debug all`	Include debugging information.
`sys os2v2`	Produce OS/2 Version 2.0 executables.
`file` *%1.obj*	Specify the input object file.
`util.obj`	Include the error-checking utility object file.
`library db2api.lib`	Link with the DB2 library.
`db2api.lib`	Include the DB2 application programming interface library.
`option stack=32000`	Specify a stack size of at least 32000.

Refer to your compiler documentation for additional compiler options.

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

1. If necessary, go to the window in which you set your environment variables. Refer to “Setting the OS/2 Environment” on page 21 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:

   ```
   bldfor 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. If necessary, go to the window in which you set your environment variables.
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 command file `bldapi.cmd`. It contains the same compile and link options as `bldfor.cmd`, 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 77 Stored Procedures

When building FORTRAN stored procedures on OS/2, you require the following statement in your stored procedure:

```
c$pragma aux (_syscall) <sp_nm> parm ( data_reference, data_reference,
c                                   data_reference, data_reference )
```

The command file `bldforsr`, in `%DB2PATH%\samples\fortran`, contains the commands to build a stored procedure. The command file compiles the stored procedure into a DLL on the server.

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.

```
@echo off
rem bldforsr command file
rem Builds a FORTRAN stored procedure program
rem Usage: bldforsr <prog_name> [ <db_name> [ < userid> <password> ]]

rem Connect to a database.
if "%1" == "" goto error
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2 connect to sample
   goto continue
:case2
   db2 connect to %2
   goto continue
:case3
   db2 connect to %2 user %3 using %4
   goto continue
:continue

rem Precompile the program.
db2 prep %1.sqf bindfile

rem Compile the program.
wfc386 /noref %1.for

rem Link the program.
wlink sys os2v2 dll export %1 file %1.obj library db2api.lib
    library os2386.lib option stack=32000
```

```
rem Bind the program to the database.
db2 bind %1.bnd

rem Disconnect from the database.
db2 connect reset

rem Copy the dynamic link library to the function subdirectory.
rem Note: Substitute the DB2 instance directory for %db2path%
copy %1.dll %db2path%\function

goto exit

:error
echo Usage: bldforsr <prog_name> [ <db_name> [ < userid> <password> ]]

:exit
@echo on
```

Compile and Link Options for bldforsr

The command file contains the following compile options:

`wfc386`	The FORTRAN compiler.
`noref`	Do not issue warnings about unreferenced symbols. This will avoid extraneous warnings.

The command file contains the following link options:

`wlink`	Use the WATCOM linker to link edit.
`sys os2v2`	Produce OS/2 Version 2.0 executables.
`dll`	Link with the dynamic link library.
`export %1`	Export the entry point for the stored procedure.
`file` *%1.obj*	Specify the input object file.
`library db2api.lib`	Link with the DB2 library.
`library os2386.lib`	Link to the OS/2 32-bit library.
`option stack=32000`	Specify a stack size of at least 32000.

Refer to your compiler documentation for additional compiler options.

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

1. If necessary, go to the window in which you set your environment variables. Refer to “Setting the OS/2 Environment” on page 21 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:

   ```
   bldforsr outsrv
   ```

Note: The command file does not use a module definition file. Instead, the linker accepts an entry point as an argument for the `export` option. In this case, the entry point for the stored procedure is the same name as the source file. This may not be the case for other stored procedures you build. If it is different, modify the command file to accept another argument for the entry point, and modify the link step to have the `export` option accept the entry point argument (instead of `%1`, as it does now). For example, if the entry point was the fifth argument, you would write the link step as:

```
wlink sys os2v2 dll export %5 file %1.obj library db2api.lib
    library os2386.lib option stack=32000
```

The command file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function` to indicate that the stored procedure is fenced. If you want the stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. These paths are in the home directory of the DB2 instance.

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

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 `bldfor` command file. Refer to "FORTRAN 77" on page 119 for details.

To run the stored procedure, do the following:

1. If necessary, go to the window in which you set your environment variables.
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 WATCOM FORTRAN 77 Compiler

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the WATCOM FORTRAN 77 compiler, keep the following points in mind:

- The WATCOM compiler 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.

- In the Fortran examples that call DB2 APIs, the *API Reference* uses `%val()` to show parameters passed by value, and `%ref()` to show parameters passed by reference. Remove these modifiers when you code applications for WATCOM FORTRAN 77. They are not required.

  The DB2 include files for WATCOM FORTRAN 77 use the compiler's pragma mechanism to indicate which parameters should be passed by reference and which by value. As long as your application includes the appropriate file, parameters are passed correctly.

- The precompiler allows only digits, blanks, and tab characters within columns 1-5 on continuation lines.

- `.sqf` source files do not support Hollerith constants.

REXX

You do not compile or bind REXX programs.

On OS/2, your application file must have a `.cmd` extension. After creation, you can run your application directly from the operating system command prompt.

An OS/2 REXX program must contain a comment that begins in the first column of the first line, to distinguish it from a batch command:

```
/* Any comment will do. */
```

REXX sample programs can be found in the directory `%DB2PATH%\samples\rexx`. To run the sample REXX program `updat`, do the following:

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

   ```
   db2start
   ```

2. Enter:

   ```
   updat
   ```

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

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.

This chapter shows you how to build and run a sample program using a file containing compile and link commands supplied with the DB2 SDK for that platform. On Windows platforms it is called a batch file. On OS/2, it is called a command file. The batch or command file shows you the compiler options you can use. It builds the sample program by compiling and linking the source file.

The sample programs are contained in `%DB2PATH%\samples\cli`. The value of `%DB2PATH%` determines where DB2 is installed. It is an environmental variable on Windows NT, Windows 95 and OS/2, which by default points to *drive*:`\sqllib`. On Windows 3.1, 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.

You can build the sample programs by using the `makefile` with the `make` facility found in the same directory. Read the README file in the directory for details about using the `makefile`, and for more information about the sample programs. You may need to modify the compiler options in the command or batch 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 16.

Note: It is recommended that, before you alter or build the sample programs, you copy them from `%DB2PATH%\samples\cli` to your own working directory.

Building and Running a DB2 CLI Application

The CLI samples directory contains a batch or command file to build a sample program on each of the supported Windows or OS/2 platforms. On Windows NT and Windows 95, the batch file `clibld.bat` builds the program `clisampl`; on Windows 3.1, the batch file `clibldw.bat` builds the program `clisampw`; and on OS/2, the command file `clibld` builds the sample program `clisampl`. For each platform, you can find both the batch or command file, and the sample program, in the appropriate language sub-directory.

In the following sections you can study how to construct the batch or command file with the compiler options for the platform you are using. You can then see how the program is run, once it is built with the batch or command file. Also, you can see the output produced from running the sample program. All the programs described in this chapter produce the same output.

Windows NT and Windows 95

Note: All applications on Windows NT and Windows 95, both embedded SQL and non-embedded SQL, must be built in a DB2 command window, and not from an operating system command prompt.

Microsoft Visual C++ is used in the following batch file, `clibld.bat`.

```
rem  clibld batch file - Windows 95 and Windows NT - Microsoft Visual C++
rem  Build a CLI sample C program.
rem  Compile the program.
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 -I%DB2PATH%\include clisampl.c
rem  Link the program.
link -debug:full -debugtype:cv -out:clisampl.exe clisampl.obj db2cli.lib
```

Compile and Link Options for clibld

The batch file contains the following compile options:	
`cl`	The Microsoft Visual C++ compiler.
`-Z7`	C7 style CodeView information generated.
`-Od`	Disable optimizations. It is easier to use a debugger with optimization off.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-W2`	Set warning level.
The batch file contains the following link options:	
`link`	Use the 32-bit linker to link edit.
`-debug:full`	Include debugging information.
`-debugtype:cv`	Indicate the debugger type.
`-out:clisampl.exe`	Specify the executable.
`clisampl.obj`	Include the object file.
`db2cli.lib`	Link with the DB2 CLI library.
Refer to your compiler documentation for additional compiler options.	

To build the sample program `clisampl`, do the following:

1. Start the database manager on the server, if it is not already running, by entering:
   ```
   db2start
   ```
2. Start the Security Service for the Windows NT server by entering:
   ```
   net start DB2NTSECSERVER
   ```
3. 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 catalogued database.
userid	Is a user ID that has Administrator authority.
password	Is a valid password.

If you need information about cataloged databases, or about Administrator authority and passwords, refer to the *Quick Beginnings* book.

The `clisampl` program performs the following SQL operations using DB2 CLI function calls:

1. Connects to a database.
2. Creates a table.
3. Inserts data into the table using a parameter marker.
4. Selects the data.
5. Drops the table.
6. Disconnects from the database.

You should see the following output:

```
Connecting
Create table - CREATE TABLE CLISAMPL (COL1 VARCHAR(50))
Insert - INSERT INTO CLISAMPL VALUES (?)
Select - SELECT * FROM CLISAMPL
Number of columns - 1
Column name - COL1
Column type - 12
Column precision - 50
Column scale - 0
Column nullable - TRUE
Column value - Row 1
Column value - Row 2
Disconnecting
Exiting program
```

Windows 3.1

Microsoft Visual C++ is used in the following batch file, `clibldw.bat`.

```
rem  clibldw batch file - Windows 3.1 - Microsoft Visual C++
rem  Build a CLI sample C program.
rem  Compile the program.
cl /c /Gy /ALw /W3 /Mq /DDB2WIN clisampw.c
rem  Link the program.
link /ST:32000 /SE:512 /NOD clisampw.obj,,,llibcewq+libw+oldnames+db2cliw;
```

Compile and Link Options for clibldw	
The batch file contains the following compile options:	
`cl`	The Microsoft Visual C++ compiler.
`/c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`/Gy`	Generate separate functions for the linker.
`/ALw`	Use large memory model.
`/W3`	Set warning level; 1 is most severe, 4 is least severe.
`/Mq`	Use QuickWin compile and include library.
`/DDB2WIN`	Identifies the Windows platform.
The batch file contains the following link options:	
`link`	Use the Microsoft Visual C ++ linker to link edit.
`/ST:32000`	Specify a stack size of 32 000.
`/SE:512`	Specify the maximum number of segments.
`/NOD`	Do not use default libraries
`,,,`	Use the default executable and map filenames.
`llibcewq+libw+oldnames`	Microsoft Visual C ++ LIB files.
`db2cliw`	The DB2 CLI library.
Refer to your compiler documentation for additional compiler options.	

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

```
clibldw
```

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

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

   ```
   db2start
   ```

2. To run the program, start Windows if it is not started already.
3. In Windows File Manager, click on **File** and then **Run...**.
4. Enter the name of the sample program:

   path\\`clisampw`

   where *path* specifies the location of the executable.

The `clisampw` program writes its output to the file `clisampl.log` in the directory where you invoked `clisampw`. You should see the following output in `clisampl.log`:

```
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
```

OS/2

IBM VisualAge C++ is used in the following command file, `clibld.cmd`.

```
rem clibld command file - OS/2 - IBM VisualAge C++ compiler
rem Build a CLI sample C program.
rem Compile the program.
icc -C+ -O- -Ti+ clisampl.c
rem Link the program.
ilink /NOFREE /NOI /DEBUG /ST:128000 /PM:VIO clisampl.obj,,,db2cli;
```

Compile and Link Options for clibld

The command file contains the following compile options:

`icc`	The IBM VisualAge C++ compiler.
`-C+`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-O-`	No optimization. It is easier to use a debugger with optimization off.
`-Ti+`	Generate debugger information

The command file contains the following link options:

`ilink`	Use the ilink linker to link edit.
`/NOFREE`	No free format.
`/NOI`	No Ignore Case. Force case sensitive identifiers.
`/DEBUG`	Include debugging information.
`/ST:128000`	Specify a stack size of at least 128 000.
`/PM:VIO`	Enable the program to run in an OS/2 window.
`db2cli`	Link with the DB2 CLI library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `clisampl`, do the following:

1. If necessary, go to the window in which you set your environment variables. Refer to Chapter 2, “Setup” 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 cataloged databases, or about SYSADM authority and passwords, refer to the *Quick Beginnings* book.

The `clisampl` program performs the following SQL operations using DB2 CLI function calls:

1. Connects to a database.
2. Creates a table.
3. Inserts data into the table using a parameter marker.
4. Selects the data.
5. Drops the table.
6. Disconnects from the database.

You should see the following output:

```
Connecting
Create table - CREATE TABLE CLISAMPL (COL1 VARCHAR(50))
Insert - INSERT INTO CLISAMPL VALUES (?)
Select - SELECT * FROM CLISAMPL
Number of columns - 1
Column name - COL1
Column type - 12
Column precision - 50
Column scale - 0
Column nullable - TRUE
Column value - Row 1
Column value - Row 2
Disconnecting
Exiting program
```

Chapter 8. Building Java Applications and Applets

You can access DB2 databases through the Java Development Kit (JDK) Version 1.1, which 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". That chapter 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 Windows NT and OS/2. This is followed by sections explaining how to build a DB2 JDBC application and a DB2 JDBC applet.

Setting the Windows NT Environment

To build Java applications on Windows NT 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 Win32 from Sun Microsystems (refer to `http://www.software.ibm.com/data/db2/java`).
2. The DB2 Client Application Enabler for Windows NT from the DB2 Client Pack. It must be Version 2.1.0 or later.

To run JDBC programs on Windows NT, the following environment variables must be set correctly. You must ensure that:

- CLASSPATH includes "." and the file `%DB2PATH%\java\db2java.zip`
- PATH includes the directory `%DB2PATH%\bin`

Setting the OS/2 Environment

To build Java applications on OS/2 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 OS/2 from IBM (refer to `http://www.software.ibm.com/data/db2/java`).

2. The DB2 Client Application Enabler for OS/2 from the DB2 Client Pack. It must be Version 2.1.0 or later.

To run JDBC programs on OS/2, the following environment variables must be set correctly. You must ensure that:

- CLASSPATH includes "." and the file `%DB2PATH%\java\db2java.zip`
- PATH includes the directory `%DB2PATH%\bin`
- LIBPATH includes the directory `%DB2PATH%\dll`

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 `%DB2PATH%\samples\java` directory. If you installed the DB2 SAMPLE database, you can run the sample by first changing to the `%DB2PATH%\samples\java` directory. For Windows NT, you would start the Security Service for the Windows NT server by entering:

```
net start DB2NTSECSERVER
```

For both Windows NT and OS/2, you would then do 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 precompiled version of `DB2Appl.java` in `samples.zip`. To do this, ensure CLASSPATH also includes the file `%DB2PATH%\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 imbed the applet in an HTML page, as the following steps demonstrate. These steps assume 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 `%DB2PATH\samples\java\samples.zip` so you can omit this step.

2. Construct an HTML file that will imbed the applet. Unless you hard-code this into the applet source, you can 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. See the JDK Version 1.1 documentation for details.
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 imbedding HTML file, the JDBC applet's `.class` or ZIP file, and the `%DB2PATH%\java\db2java.zip` file into a directory under the Web browser's document root. For `DB2Applt.java`, copy `%DB2PATH%\samples\java\samples.zip`, `%DB2PATH%\samples\java\DB2Applt.html`, and `%DB2PATH%\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 want to place the ZIP files into a directory that is shared by several applets that can 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. See the JDK Version 1.1 documentation for details.
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.

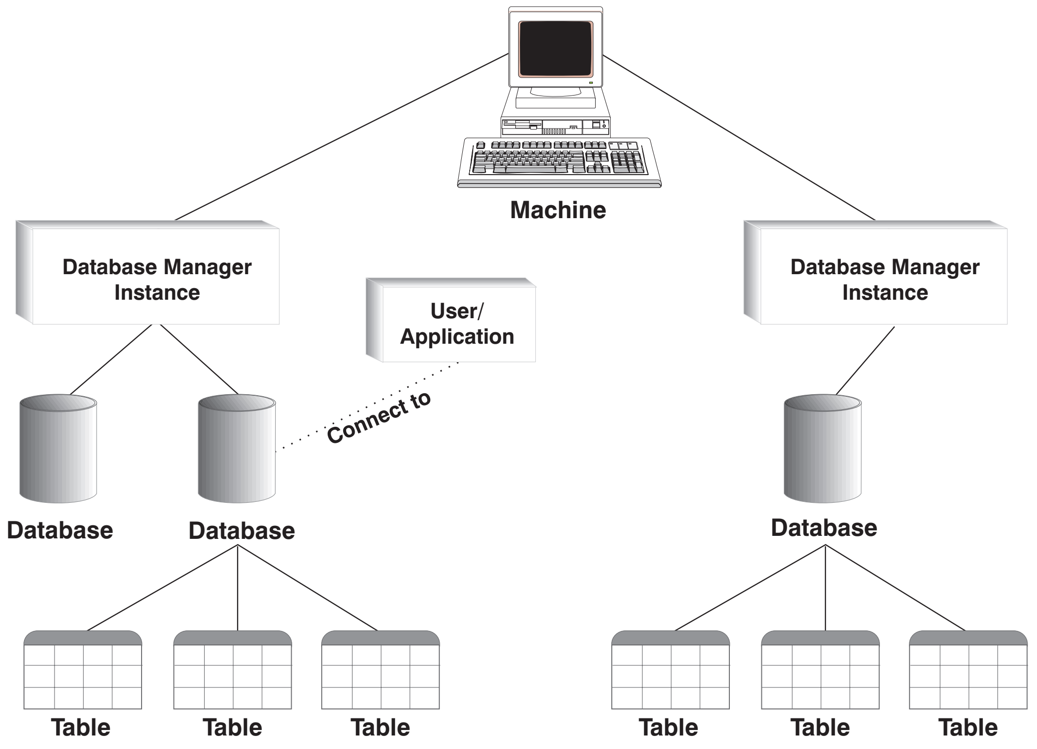

Figure 1. Database Manager Instances

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 user name 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 "? xxxnnnnn"
```

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
Use these for build or runtime problems that 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 148 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 148 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 148 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

Trademarks

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

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: 10J8160

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

10J8160

S10J-8160-00