IBM DB2 Universal Database

Application Building Guide

Version 6

SC09-2842-00

IBM DB2 Universal Database

Application Building Guide

Version 6

SC09-2842-00

Before using this information and the product it supports, be sure to read the general information under "Appendix E. Notices" on page 417.

Contents

Welcome to DB2 Application Development

This preface provides the information you need to get started in DB2 application development, specifically with the DB2 Developer’s Edition products.

The section “The DB2 Developer’s Edition” guides you to the install information for your particular development needs. This information will help you decide the best way to install DB2 from either the IBM DB2 Universal Developer’s Edition, Version 6.1, or the IBM DB2 Personal Developer’s Edition, Version 6.1.

The section “DB2 Application Development Books” on page xi describes the main books in the DB2 library for application development.

The section “DB2 Programming Interfaces” on page xii presents key programming interface concepts for DB2 application development.

The section “DB2 Features” on page xix describes the main features available to you for DB2 application development.

The DB2 Developer’s Edition

DB2 Universal Database provides two product packages for application development: DB2 Personal Developer’s Edition and DB2 Universal Developer’s Edition. The Personal Developer’s Edition provides the DB2 Universal Database and DB2 Connect personal edition products which run on OS/2, Linux, and Windows 32-bit operating systems. The DB2 Universal Developer’s Edition provides these as well as the DB2 common server family of products which run on AIX, HP-UX, Linux, OS/2, Solaris, and Windows NT.

Using the software that comes with these products, you can develop and test applications that run on one operating system and access databases on the same or a different operating system. For example, you can create an application that runs on the Windows NT operating system but accesses a database on a UNIX platform such as AIX. See your License Agreement for the terms and conditions of using the Developer’s Edition products.

The Developer’s Edition boxes contain several CD-ROMs with all the code that you need to develop and test your applications. In each box, you will find:

- The DB2 Universal Database product CD-ROMs for several operating systems. These CD-ROMs are provided to you for testing your applications only. If you need to install and use a database for your company's needs, you need to get a valid license for the Universal Database product by purchasing this product. The Personal Developer's Edition box contains the CD-ROMs for Personal Edition products including OS/2, Linux, and Windows 32-bit operating systems. The Universal Developer's Edition box contains the CD-ROMs for all the operating systems supported by DB2.
- A set of CD-ROMs containing the Administration Clients for the operating systems supported by DB2. This client contains administrative tools, and allows you to run applications on any system.
- A set of CD-ROMs containing the Run-Time Clients for the operating systems supported by DB2. An application can be run from a Run-Time Client on any system. A Run-Time Client does not have some of the features of an Administration Client, such as the DB2 Control Center and Event Analyzer, and so takes up less space.
- A set of DB2 Software Developer's Kit (DB2 SDK) CD-ROMs for DB2-supported operating systems. These CD-ROMs contain application development tools, sample programs, and header files. Each DB2 SDK includes everything you need to develop your applications. It also includes the full functionality of a DB2 client. Install the DB2 Software Developer's Kit on the operating system where you will be developing your applications.
- A CD-ROM with a Netscape browser. This allows you to view the softcopy installation manuals in case you do not have a browser installed on your machine.
- In addition, you get complimentary copies of other software that you may find useful for developing applications. This software may vary from time to time. The complimentary software is accompanied by License agreements for their use and you must ensure that you comply with these agreements.

Installation Information

Each CD-ROM contains installation information for its DB2 product components. This allows you to determine the best installation for your environment *before installing DB2*.

This information is available in both HTML and PostScript formats. Installation information for DB2 servers is found in the *Quick Beginnings* books. Installation information for the Administration Clients, Run-Time Clients, and DB2 Software Developer's Kits is found in the *Installation and Configuration Supplement*.

These books provide detailed information on all the options available to you when installing DB2. With this information, you can tailor your choices to suit your particular database development needs. Keep in mind that even if you are experienced with a previous version of DB2, there may be new features available to install with this version that you might find useful.

To access the HTML or PostScript files for these books, load the CD containing the DB2 server product for your operating system, or the DB2 SDK CD for client information.

Note: On Windows, if you have your computer set to automatically read from the CD-ROM drive when you load it, the install shield for the DB2 installation program may begin. Simply cancel the program in order to access the *Quick Beginnings* or *Installation and Configuration Supplement* book first.

With the appropriate DB2 product CD-ROM in your CD-ROM drive, change to the CD-ROM drive. Open the ″doc″ directory (or folder, on Windows or OS/2). Depending on your operating system, the directory names may be in a different case. For instance, on Windows systems, the first letter is given in upper case, as in ″Doc″.

You will be presented with several sub-directories, each with a two-character name. These are for the different languages in which this DB2 product is available. Go to the sub-directory of the language you want to work in. The books are not translated into all languages on all platforms. The following table shows the possible sub-directory names that may be in your product's ″doc″ directory, with the corresponding language each represents:

Table 1. Directory Names and Languages

Directory Name	**Language**	**Directory Name**	**Language**
BR	Brazilian Portuguese	FR	French
CN	Simplified Chinese (PRC)	IT	Italian
DE	German	JP	Japanese
DK	Danish	KR	Korean
EN	English	NO	Norwegian
ES	Spanish	SE	Swedish
FI	Finnish	TW	Traditional Chinese (Taiwan)

In the language directory you choose, there will be the following sub-directories:

html for HTML files of DB2 books

ps for PostScript files of DB2 books

Note: Some language directories may only have one of these sub-directories. That means that, for this language, only this format is available for the DB2 books shipped with this product.

Enter the sub-directory for the format you want for the *Quick Beginnings* or *Installation and Configuration Supplement* book.

HTML

In the `html` directory, there are several sub-directories with a five character name. Each contains the HTML files for a particular book. The book you want will have one of the following names:

db2i2	*DB2 for OS/2 Quick Beginnings*
db2ix	*DB2 for UNIX Quick Beginnings*
db2i6	*DB2 for Windows NT Quick Beginnings*
db2iy	*Installation and Configuration Supplement*

Enter the sub-directory available to you for the *Quick Beginnings* or *Installation and Configuration Supplement* book for your platform. To access the HTML version of this book online, your browser must be installed and available for use. Double-click on the HTML file that has the same name as the directory you are in, with an extension of `.htm`. For example, if you are in the `db2i6` directory, you would double-click on `db2i6.htm`. This is the master HTML file for the book.

The *Quick Beginnings* or *Installation and Configuration Supplement* book will appear in the browser window. You can scroll down to the table of contents. These books provide the information you need to install any of the DB2 servers or clients included in the DB2 Developer's Edition. You can use the hot-links from the table of contents to get to the particular information you are interested in before installing DB2. When you do install DB2, you can also access the other books of the DB2 library online by the links at the bottom of any page of any online DB2 book.

PostScript

In the `ps` directory there is a list of compressed PostScript files. The filename for the book you want will start with one of these sets of characters:

db2i2	*DB2 for OS/2 Quick Beginnings*
db2ix	*DB2 for UNIX Quick Beginnings*

db2i6 *DB2 for Windows NT Quick Beginnings*

db2iy *Installation and Configuration Supplement*

For instructions on printing the compressed PostScript file, see “Printing the PostScript Books” on page 414.

DB2 Application Development Books

The DB2 library is described in “Appendix D. How the DB2 Library Is Structured” on page 401. There are two general categories of DB2 books: those that provide information for administering DB2 databases, and those that provide information for DB2 application development. Some books provide both kinds of information. As a DB2 application developer, you may well find that you are referring to DB2 books in both these categories. However, your focus, and the focus of this section, will be on the main application development books of the DB2 library.

There are two main books for programming your applications. These are the *CLI Guide and Reference*, which discusses programming DB2 CLI applications, and the *Application Development Guide*, which discusses all the different kinds of DB2 programming other than DB2 CLI. Both these books also contain reference information.

You can find the information for setting up your development environment, and compiling, linking, and running your applications, in the book you are reading now, the *Application Building Guide*.

For the syntax of SQL statements and functions, you can refer to the *SQL Reference*.

Two books considered to be in the administrative category of the DB2 library are also important reference books for your application programming. The *Administrative API Reference* contains details of all administrative functions used to manage DB2 databases. You may find it convenient to use DB2 APIs in your applications along with, or instead of, SQL statements. The *Command Reference* contains details of all DB2 commands (except for SQL statements), and explains how to use the DB2 Command Line Processor (CLP).

To solve problems with environmental setup or program development, you can refer to the *Troubleshooting Guide*. It helps you to determine the source of errors, to recover from problems, and to use diagnostic tools in consultation with DB2 Customer Service. The *Message Reference* contains the full list and description of DB2 error messages, a very useful reference when debugging your applications.

These and the other books of the DB2 library, as well as the online information available in your DB2 environment, will provide you with the information you need to develop your DB2 applications. For development information not strictly pertaining to DB2, see the documentation provided by the vendor for the compiler, interpreter, or other development tools you are using.

DB2 Programming Interfaces

You can use several different programming interfaces to access DB2 databases. You can:

1. Embed static and dynamic SQL statements in your applications.
2. Code DB2 Call Level Interface (DB2 CLI) function calls in your applications to invoke dynamic SQL statements.
3. Develop Java applications and applets that call the Java Database Connectivity application programming interface (JDBC API).
4. Develop Microsoft Visual Basic and Visual C++ applications that conform to Data Access Object (DAO) and Remote Data Object (RDO) specifications, and ActiveX Data Object (ADO) applications that use the Object Linking and Embedding Database (OLE DB) Bridge.
5. Develop applications using IBM or third-party tools such as Net.Data, Excel, Perl, and Open Database Connectivity (ODBC) end-user tools such as Lotus Approach, and its programming language, LotusScript.

To perform administrative functions such as backing up and restoring databases, you can use DB2 APIs.

The way your application accesses DB2 databases will depend on the type of application you want to develop. For example, if you want a data entry application, you might choose to embed static SQL statements in your application. If you want an application that performs queries over the World Wide Web, you might choose Net.Data, Perl, or Java.

Using Embedded SQL Statements

Structured Query Language (SQL) is the database interface language used to access and manipulate data in DB2 databases. You can embed SQL statements in your applications, enabling them to perform any task supported by SQL, such as retrieving or storing data. Using DB2, you can code your embedded SQL applications in the C/C++, COBOL, FORTRAN, Java (SQLJ), and REXX programming languages.

An application in which you embed SQL statements is called a host program. The programming language you use to create a host program is called a host language. The program and language are defined this way because they host or accommodate SQL statements.

For static SQL statements, you know before compile time the SQL statement type and the table and column names. The only unknowns are specific data values the statement is searching for or updating. You can represent those values in host language variables. You precompile, compile, and bind static SQL statements before you run your application. Static SQL is best run on databases that do not change a great deal. Otherwise, the statements will soon get out of date.

In contrast, dynamic SQL statements are those that your application builds and executes at run time. An interactive application that prompts the end user for key parts of an SQL statement, such as the names of the tables and columns to be searched, is a good example of dynamic SQL. The application builds the SQL statement while it's running, and then submits the statement for processing.

You can write applications that have static SQL statements, dynamic SQL statements, or a mix of both.

Generally, static SQL statements are well-suited for high-performance applications with predefined transactions. A reservation system is a good example of such an application.

Generally, dynamic SQL statements are well-suited for applications that run against a rapidly changing database where transactions need to be specified at run time. An interactive query interface is a good example of such an application.

When you embed SQL statements in your application, you must precompile and bind your application to a database with the following steps:

1. Create source files that contain programs with embedded SQL statements.
2. Connect to a database, then precompile each source file.

   The precompiler converts the SQL statements in each source file into DB2 run-time API calls to the database manager. The precompiler also produces an access package in the database and, optionally, a bind file, if you specify that you want one created.

   The access package contains access plans selected by the DB2 optimizer for the static SQL statements in your application. The access plans contain the information required by the database manager to execute the static SQL statements in the most efficient manner as determined by the optimizer. For dynamic SQL statements, the optimizer creates access plans when you run your application.

The bind file contains the SQL statements and other data required to create an access package. You can use the bind file to rebind your application later without having to precompile it first. Rebinding creates access plans that are optimized for current database conditions. You need to rebind your application if it will access a different database from the one against which it was precompiled. You are recommended to rebind your application if the database statistics have changed since the last binding.

3. Compile the modified source files (and other files without SQL statements) using the host language compiler.
4. Link the object files with the DB2 and host language libraries to produce an executable program.
5. Bind the bind file to create the access package if this was not already done at precompile time, or if a different database is going to be accessed.
6. Run the application. The application accesses the database using the access plan in the package.

Embedded SQL for Java (SQLJ)

DB2 Java embedded SQL (SQLJ) support is provided by the DB2 SDK. With DB2 SQLJ support, in addition to DB2 JDBC support, you can build and run SQLJ applets, applications, and stored procedures. These contain static SQL and use embedded SQL statements that are bound to a DB2 database.

For more information on DB2 SQLJ support, visit the Web page at:

```
http://www.software.ibm.com/data/db2/java
```

Using the DB2 Call Level Interface

DB2 CLI is a programming interface that your C and C++ applications can use to access DB2 databases. DB2 CLI is based on the Microsoft Open Database Connectivity (ODBC) specification, and the ISO CLI standard. Since DB2 CLI is based on industry standards, application programmers who are already familiar with these database interfaces may benefit from a shorter learning curve.

When you use DB2 CLI, your application passes dynamic SQL statements as function arguments to the database manager for processing. As such, DB2 CLI is an alternative to embedded dynamic SQL.

You do not need to precompile or bind DB2 CLI applications because they use common access packages provided with DB2. You simply compile and link your application.

However, before your DB2 CLI or ODBC applications can access DB2 databases, the DB2 CLI bind files that come with the DB2 SDK must be bound to each DB2 database that will be accessed. This occurs automatically on the first connection to the database, but we recommend that the database administrator bind the bind files from one client on each platform that will access a DB2 database. For the bind instructions, see “Binding” on page 41.

For example, suppose you have OS/2, AIX, and Windows 95 clients that each access two DB2 databases. The administrator should bind the bind files from one OS/2 client on each database that will be accessed. Next, the administrator should bind the bind files from one AIX client on each database that will be accessed. Finally, the administrator should do the same on one Windows 95 client.

DB2 CLI Versus Embedded Dynamic SQL

You can develop dynamic applications using either embedded dynamic SQL statements or DB2 CLI. In both cases, SQL statements are prepared and processed at run time. Each method has unique advantages listed below.

DB2 CLI Advantages

Portability
DB2 CLI applications use a standard set of functions to pass SQL statements to the database. All you need to do is compile and link DB2 CLI applications before you can run them. In contrast, you must precompile embedded SQL applications, compile them, and then bind them to the database before you can run them. This process effectively ties your application to a particular database.

No binding
You do not need to bind individual DB2 CLI applications to each database they access. You only need to bind the bind files that are shipped with DB2 CLI once for all your DB2 CLI applications. This can significantly reduce the amount of time you spend managing your applications.

Extended fetching and input
DB2 CLI functions enable you to retrieve multiple rows in the database into an array with a single call. They also let you execute an SQL statement many times using an array of input variables.

Consistent interface to catalog
Database systems contain catalog tables that have information about the database and its users. The form of these catalogs can vary among systems. DB2 CLI provides a consistent interface to query catalog information about components such as tables, columns, foreign and primary keys, and user privileges. This shields your application from

catalog changes across releases of database servers, and from differences among database servers. You don't have to write catalog queries that are specific to a particular server or product version.

Extended data conversion
DB2 CLI automatically converts data between SQL and C data types. For example, fetching any SQL data type into a C char data type converts it into a character-string representation. This makes DB2 CLI well-suited for interactive query applications.

No global data areas
DB2 CLI eliminates the need for application controlled, often complex global data areas, such as SQLDA and SQLCA, typically associated with embedded SQL applications. Instead, DB2 CLI automatically allocates and controls the necessary data structures, and provides a handle for your application to reference them.

Retrieve result sets from stored procedures
DB2 CLI applications can retrieve multiple rows and result sets generated from a stored procedure residing on the server.

Scrollable cursors
DB2 CLI supports server-side scrollable cursors that can be used in conjunction with array output. This is useful in GUI applications that display database information in scroll boxes that make use of the Page Up, Page Down, Home and End keys. You can declare a cursor as scrollable and then move forwards or backwards through the result set by one or more rows. You can also fetch rows by specifying an offset from the current row, the beginning or end of a result set, or a specific row you bookmarked previously.

Embedded Dynamic SQL Advantages

Embedded SQL supports more than just C and C++. This might be an advantage if you prefer to code your applications in another language.

Dynamic SQL is generally more consistent with static SQL. If you already know how to program static SQL, moving to dynamic SQL might not be as difficult as moving to DB2 CLI.

Using Java Database Connectivity (JDBC)

DB2's Java support includes JDBC, a vendor-neutral dynamic SQL interface that provides data access to your application through standardized Java methods. JDBC is similar to DB2 CLI in that you do not have to precompile or bind a JDBC program. As a vendor-neutral standard, JDBC applications offer increased portability. An application written using JDBC uses only dynamic SQL.

JDBC can be especially useful for accessing DB2 databases across the Internet. Using the Java programming language, you can develop JDBC applets and applications that access and manipulate data in remote DB2 databases using a network connection. You can also create JDBC stored procedures that reside on the server, access the database server, and return information to a remote client application that calls the stored procedure.

The JDBC API, which is similar to the CLI/ODBC API, provides a standard way to access databases from Java code. Your Java code passes SQL statements as function arguments to the DB2 JDBC driver. The driver handles the JDBC API calls from your client Java code.

Java's portability enables you to deliver DB2 access to clients on multiple platforms, requiring only a Java-enabled web browser.

Java applications rely on the DB2 client to connect to DB2. You 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 client connection to communicate the requests to the server and to receive the results.

Java applets do not require the DB2 client connection. Typically, you would embed the applet in a HyperText Markup Language (HTML) web page.

You need only a Java-enabled web browser or applet viewer on the client machine to run your applet. When you load your HTML page, the browser downloads the Java applet to your machine, which then downloads the Java class files and DB2's JDBC driver. When your applet calls the JDBC API to connect to DB2, the JDBC driver establishes a separate network connection with the DB2 database through the JDBC applet server residing on the web server.

For more information on DB2 JDBC support, visit the Web page at:

```
http://www.software.ibm.com/data/db2/java
```

Using DB2 APIs

Your applications may need to perform some database administration tasks, such as creating, activating, backing up, or restoring a database. DB2 provides numerous APIs so you can perform these tasks from your applications, including embedded SQL and DB2 CLI applications. This enables you to program the same administrative functions into your applications that you can perform using the DB2 server administration tools, discussed in "DB2 Universal Database Tools" on page xxv.

Additionally, you might need to perform specific tasks that can only be performed using the DB2 APIs. For example, you might want to retrieve the text of an error message so your application can display it to the end user. To retrieve the message, you must use the Get Error Message API.

Using ActiveX Data Objects (ADO) and Remote Data Objects (RDO)

You can write Microsoft Visual Basic and Microsoft Visual C++ database applications that conform to the Data Access Object (DAO) and Remote Data Object (RDO) specifications. DB2 also supports ActiveX Data Object (ADO) applications that use the OLE DB Bridge.

ActiveX Data Objects (ADO) allow you to write an application to access and manipulate data in a database server through an OLE DB provider. The primary benefits of ADO are high speed, ease of use, low memory overhead, and a small disk footprint.

Remote Data Objects (RDO) provide an information model for accessing remote data sources through ODBC. RDO offers a set of objects that make it easy to connect to a database, execute queries and stored procedures, manipulate results, and commit changes to the server. It is specifically designed to access remote ODBC relational data sources, and makes it easier to use ODBC without complex application code.

Using IBM, Third-Party, and ODBC End-User Tools

To perform a basic task, such as querying a database, you can use Net.Data or Perl.

Net.Data enables Internet and intranet access to DB2 data through your web applications. It exploits web server interfaces (APIs), providing higher performance than common gateway interface (CGI) applications. Net.Data supports client-side processing as well as server-side processing with languages such as Java, REXX, Perl and C++. Net.Data provides conditional logic and a rich macro language. The Net.Data web page is at:

```
http://www.software.ibm.com/data/net.data/
```

DB2 supports the Perl Database Interface (DBI) specification for data access through the DBD::DB2 driver. The DB2 Universal Database Perl DBI website is located at:

```
http://www.software.ibm.com/data/db2/perl/
```

and contains the latest DBD::DB2 driver, and related information.

You can also use ODBC end-user tools such as Lotus Approach, Microsoft Access, and Microsoft Visual Basic to create applications to perform these tasks. ODBC tools provide a simpler alternative to developing applications than using a high-level programming language.

Lotus Approach provides two ways to access DB2 data. You can use the graphical interface to perform queries, develop reports, and analyze data. Or you can develop applications using LotusScript, a full-featured, object-oriented programming language that comes with a wide array of objects, events, methods, and properties, along with a built-in program editor.

DB2 Features

DB2 comes with a variety of features that run on the server which you can use to supplement or extend your applications. When you use DB2 features, you do not have to write your own code to perform the same tasks. DB2 also lets you store some parts of your code at the server instead of keeping all of it in your client application. This can have performance and maintenance benefits.

There are features to protect data and to define relationships between data. As well, there are object-relational features to create flexible, advanced applications. You can use some features in more than one way. For example, constraints enable you to protect data and to define relationships between data values. Here are some key DB2 features:

- Constraints
- User-Defined Types (UDTs) and Large Objects (LOBs)
- User-Defined Functions (UDFs)
- Triggers
- Stored Procedures

To decide whether or not to use DB2 features, consider the following points:

Application independence
You can make your application independent of the data it processes. Using DB2 features that run at the database enables you to maintain and change the logic surrounding the data without affecting your application. If you need to make a change to that logic, you only need to change it in one place; at the server, and not in each application that accesses the data.

Performance
You can make your application perform more quickly by storing and running parts of your application on the server. This shifts some processing to generally more powerful server machines, and can reduce network traffic between your client application and the server.

Application requirements

Your application might have unique logic that other applications do not. For example, if your application processes data entry errors in a particular order that would be inappropriate for other applications, you might want to write your own code to handle this situation.

In some cases, you might decide to use DB2 features that run on the server because they can be used by several applications. In other cases, you might decide to keep logic in your application because it is used by your application only.

Constraints

To protect data and to define relationships between your data, you usually define business rules. These rules define what data values are valid for a column in a table, or how columns in one or more tables are related to each other.

DB2 provides constraints as a way to enforce those rules using the database system. By using the database system to enforce business rules, you don't have to write code in your application to enforce them. However, if a business rule applies to one application only, you should code it in the application instead of using a global database constraint.

DB2 provides the following kinds of constraints:

1. NOT NULL constraints
2. UNIQUE constraints
3. PRIMARY KEY constraints
4. FOREIGN KEY constraints
5. CHECK constraints

You define constraints using the SQL statements CREATE TABLE and ALTER TABLE.

User-Defined Types (UDTs) and Large Objects (LOBs)

Every data element in the database is stored in a column of a table, and each column is defined to have a data type. The data type places limits on the types of values you can put into the column and the operations you can perform on them. For example, a column of integers can only contain numbers within a fixed range. DB2 includes a set of built-in data types with defined characteristics and behaviors: character strings, numerics, datetime values, large objects, Nulls, graphic strings, binary strings, and datalinks.

Sometimes, however, the built-in data types might not serve the needs of your applications. DB2 provides user-defined types (UDTs) which enable you to define the distinct data types you need for your applications.

UDTs are based on the built-in data types. When you define a UDT, you also define the operations that are valid for the UDT. For example, you might define a MONEY data type that is based on the DECIMAL data type. However, for the MONEY data type, you might allow only addition and subtraction operations, but not multiplication and division operations.

Large Objects (LOBs) enable you to store and manipulate large, complex data objects in the database: objects such as audio, video, images, and large documents.

The combination of UDTs and LOBs gives you considerable power. You are no longer restricted to using the built-in data types provided by DB2 to model your business data, and to capture the semantics of that data. You can use UDTs to define large, complex data structures for advanced applications.

In addition to extending built-in data types, UDTs provide several other benefits:

Support for object-oriented programming in your applications
: You can group similar objects into related data types. These types have a name, an internal representation, and a specific behavior. By using UDTs, you can tell DB2 the name of your new type and how it is represented internally. A LOB is one of the possible internal representations for your new type, and is the most suitable representation for large, complex data structures.

Data integrity through strong typing and encapsulation
: Strong typing guarantees that only functions and operations defined on the distinct type can be applied to the type. Encapsulation ensures that the behavior of UDTs is restricted by the functions and operators that can be applied to them. In DB2, behavior for UDTs can be provided in the form of user-defined functions (UDFs), which can be written to accommodate a broad range of user requirements.

Performance through integration into the database manager
: Because UDTs are represented internally, the same way as built-in data types, they share the same efficient code as built-in data types to implement built-in functions, comparison operators, indexes, and other functions.The exception to this is UDTs that utilize LOBs, which cannot be used with comparison operators and indexes.

User-Defined Functions (UDFs)

The built-in capabilities supplied through SQL may not satisfy all of your application needs. To allow you to extend those capabilities, DB2 supports user-defined functions (UDFs). You can write your own code in Visual Basic, C/C++ or Java to perform operations within any SQL statement that returns a single scalar value or a table.

UDFs give you significant flexibility. They can return a single scalar value as part of a select list from a database, or they can return whole tables from non-database sources such as spreadsheets.

UDFs provide a way to standardize your applications. By implementing a common set of user-defined functions, many applications can process data in the same way, thus ensuring consistent results.

User-defined functions also support object-oriented programming in your applications. UDFs provide for abstraction, allowing you to define the methods that can be used to perform operations on data objects. And UDFs provide for encapsulation, allowing you to control access to the underlying data of an object, protecting it from direct manipulation and possible corruption.

OLE Automation UDFs

OLE (Object Linking and Embedding) automation is part of the OLE 2.0 architecture from Microsoft Corporation. With OLE automation, your applications, regardless of the language in which they are written, can expose their properties and methods in OLE automation objects. Other applications, such as Lotus Notes or Microsoft Exchange, can then integrate these objects by taking advantage of these properties and methods through OLE automation.

DB2 for Windows NT provides access to OLE automation objects using UDFs. To access OLE automation objects and invoke their methods, you must register the methods of the objects as UDFs. DB2 applications can then invoke the methods by calling the UDFs. The UDFs can be scalar or table functions.

For example, you can develop an application that queries data in a spreadsheet created using a product such as Microsoft Excel. To do this, you would develop an OLE automation table function that retrieves data from the worksheet, and returns it to DB2. DB2 can then process the data, perform online analytical processing (OLAP), and return the query result to your application.

OLE DB Table Functions

Microsoft OLE DB is a set of OLE/COM interfaces that provide applications with uniform access to data stored in diverse information sources. DB2 Universal Database simplifies the creation of OLE DB applications by enabling you to define table functions that access an OLE DB data source. You can perform operations including GROUP BY, JOIN, and UNION, on data sources that expose their data through OLE DB interfaces. For example, you can define an OLE DB table function to return a table from a Microsoft Access database or a Microsoft Exchange address book, then create a report that seamlessly combines data from this OLE DB table function with data in your DB2 database.

Using OLE DB table functions reduces your application development effort by providing built-in access to any OLE DB provider. For C, Java, and OLE automation table functions, the developer needs to implement the table function, whereas in the case of OLE DB table functions, a generic built-in OLE DB consumer interfaces with any OLE DB provider to retrieve data. You only need to register a table function of language type OLEDB, and refer to the OLE DB provider and the relevant rowset as a data source. You do not have to do any UDF programming to take advantage of OLE DB table functions.

Triggers

A trigger defines a set of actions executed by a delete, insert, or update operation on a specified table. When such an SQL operation is executed, the trigger is said to be activated. The trigger can be activated before the SQL operation or after it. You define a trigger using the SQL statement CREATE TRIGGER.

You can use triggers that run before an update or insert in several ways:

- To check or modify values before they are actually updated or inserted in the database. This is useful if you need to transform data from the way the user sees it to some internal database format.
- To run other non-database operations coded in user-defined functions.

Similarly, you can use triggers that run after an update or insert in several ways:

- To update data in other tables. This is useful for maintaining relationships between data or in keeping audit trail information.
- To check against other data in the table or in other tables. This is useful to ensure data integrity when referential integrity constraints aren’t appropriate, or when table check constraints limit checking to the current table only.

- To run non-database operations coded in user-defined functions. This is useful when issuing alerts or to update information outside the database.

You gain several benefits using triggers:

Faster application development
Triggers are stored in the database, and are available to all applications. This relieves you of the need to code equivalent functions for each application.

Global enforcement of business rules
Triggers are defined once, and are used by all applications that use the data governed by the triggers.

Easier maintenance
Any changes need to be made only once in the database instead of in every application that uses a trigger.

Stored Procedures

Typically, applications access the database across the network. This can result in poor performance if a lot of data is being returned. A stored procedure runs on the database server. A client application can call the stored procedure which then performs the database accessing without returning unnecessary data across the network. Only the results the client application needs are returned by the stored procedure.

You gain several benefits using stored procedures:

Reduced network traffic
Grouping SQL statements together can save on network traffic. A typical application requires two trips across the network for each SQL statement. Grouping SQL statements results in two trips across the network for each group of statements, resulting in better performance for applicatons.

Access to features that exist only on the server
Stored procedures can have access to commands that run only on the server, such as LIST DATABASE DIRECTORY and LIST NODE DIRECTORY; they might have the advantages of increased memory and disk space on server machines; and they can access any additional software installed on the server.

Enforcement of business rules
You can use stored procedures to define business rules that are common to several applications. This is another way to define business rules, in addition to using constraints and triggers.

When an application calls the stored procedure, it will process data in a consistent way according to the rules defined in the stored procedure. If you need to change the rules, you only need to make the change once in the stored procedure, not in every application that calls the stored procedure.

DB2 Universal Database Tools

You can use a variety of different tools when developing your applications. DB2 Universal Database supplies the following tools to help you write and test the SQL statements in your applications, and to help you monitor their performance:

Note: Not all tools are available on every platform.

Control Center

A graphical interface that displays database objects (such as databases, tables, and packages) and their relationship to each other. Use the Control Center to perform administrative tasks such as configuring the system, managing directories, backing up and recovering the system, scheduling jobs, and managing media.

The Control Center includes the following facilities:

Command Center
is used to enter DB2 commands and SQL statements in an interactive window, and to see the execution result in a result window. You can scroll through the results and save the output to a file.

Script Center
is used to create scripts, which you can store and invoke at a later time. These scripts can contain DB2 commands, SQL statements, or operating system commands. You can schedule scripts to run unattended. You can run these jobs once or you can set them up to run on a repeating schedule. A repeating schedule is particularly useful for tasks like backups.

Journal
is used to view the following types of information: all available information about jobs that are pending execution, executing, or that have completed execution; the recovery history log; the alerts log; and the messages log. You can also use the Journal to review the results of jobs that run unattended.

Alert Center
is used to monitor your system for early warnings of potential problems, or to automate actions to correct problems.

Tools Setting
is used to change the settings for the Control Center, Alert Center, and Replication.

Performance Monitor

An installable option for the Control Center, the Performance Monitor is a graphical interface that provides comprehensive performance data collection, viewing, reporting, analysis, and alerting capabilities for your DB2 system. Use the Performance Monitor for performance tuning.

You can choose to monitor snapshots or events. The Snapshot Monitor enables you to capture point-in-time information at specified intervals. The Event Monitor allows you to record performance information over the duration of an event, such as a connection.

Visual Explain

An installable option for the Control Center, Visual Explain is a graphical interface that enables you to analyze and tune SQL statements, including viewing access plans chosen by the optimizer for SQL statements.

Chapter 1. Introduction

This book provides the information you need to set up your environment for developing DB2 applications, and provides step-by-step instructions to compile, link, and run these applications in this environment. It explains how to build applications using the DB2 Software Developer's Kit (DB2 SDK) for DB2 Universal Database Version 6.1 on the following platforms:

- AIX
- HP-UX
- Linux
- OS/2
- Silicon Graphics IRIX
- Solaris
- Windows 32-bit operating systems

Note: Windows 32-bit operating systems refers to Windows NT, Windows 98, and Windows 95. Whenever this book mentions Windows 32-bit operating systems, all three of these operating systems are implied, except in the case of Systems Network Architecture (SNA) support, REXX suport, or DB2 Connect, formerly known as Distributed Database Connection Services (DDCS). These are supported on Windows NT only.

To develop your applications, you can use the following programming interfaces:

DB2 Application Programming Interfaces (DB2 APIs)
provide administrative functions to manage DB2 databases.

DB2 Call Level Interface (DB2 CLI)
is a callable SQL interface based on the X/Open CLI specification, and is compatible with Microsoft Corporation's Open Database Connectivity (ODBC) interface.

Embedded SQL
uses SQL statements coded directly in your program which must be precompiled in order to be converted into run-time function calls.

Embedded SQL for Java (SQLJ)
uses SQL statements in a generated profile that are precompiled and customized into run-time function calls, which in turn provide an interface to the database manager.

Java Database Connectivity (JDBC)
is a dynamic SQL API for Java. The JDBC API is included in the Java Development Kits available for supported platforms.

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

- The *Application Development Guide*, which discusses how to code and design application programs that access DB2 family servers using embedded SQL, embedded SQL for Java, and Java Database Connectivity (JDBC).
- The *CLI Guide and Reference*, which explains how to code and design application programs that use the DB2 Call Level Interface and ODBC.
- The *Administrative API Reference*, which discusses how to code and design application programs that use DB2 Application Programming Interfaces.

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

- *DB2 for OS/2 Quick Beginnings*, which explains how to install the database manager, and the DB2 Software Developer's Kit (DB2 SDK) on OS/2 server and client workstations.
- *DB2 for UNIX Quick Beginnings*, which explains how to install the database manager, and the DB2 Software Developer's Kit (DB2 SDK) on UNIX server and client workstations.
- *DB2 for Windows NT Quick Beginnings*, which explains how to install the database manager, and the DB2 Software Developer's Kit (DB2 SDK) on server and client workstations for Windows 32-bit operating systems.
- The *Command Reference*, which explains how to use the DB2 Command Line Processor (CLP), and all non-SQL DB2 commands.

- The *Troubleshooting Guide*, which helps you resolve application development problems involving DB2 clients and servers, as well as problems with related tasks in database administration and connectivity.

For a complete list of the DB2 documentation library, see "Appendix D. How the DB2 Library Is Structured" on page 401.

Note: The examples in this book are provided ″as is″ without any warranty of any kind. The user, and not IBM, assumes the entire risk of quality, performance, and repair of any defects.

Who Should Use This Book

You should use this book if you want to develop programs on one of the currently supported platforms for DB2 Universal Database Version 6.1. The book describes how your programs can manage DB2 databases with DB2 APIs, and can access DB2 databases with DB2 CLI, embedded SQL, SQLJ, and JDBC.

In order to use this book, you should know one or more of the supported programming languages on the platform you will be using. These languages are listed in "Supported Software by Platform" on page 7.

How To Use This Book

The book is designed to allow easy access to the information needed to develop your applications. The first three chapters contain common information for users developing DB2 programs with any of the supported programming languages, and should, therefore, be read by all users.

Chapter 4 contains information for building DB2 Java programs: environmental set-up information for specific platforms, as well as common information for developing JDBC applets and applications, and embedded SQL for Java (SQLJ) applets and applications, for all supported platforms.

The remaining chapters in the book are platform-specific ″Building Applications″ chapters. Each of these chapters gives detailed information for building DB2 applications with the supported languages (except for Java) on one of the supported platforms.

The appendices give important additional information on various topics.

Highlighting Conventions

This book uses the following conventions:

Italics Indicates 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:
- 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 enter

Bold Emphasizes a point.

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 build and run DB2 applications with a DB2 SDK installed. You can run DB2 applications on these DB2 clients:
- DB2 Run-Time Client
- DB2 Administration Client

See “Chapter 2. Setup” on page 33 for information about setting up your programming environment.

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

- Precompilers for C/C++, Java, COBOL, and Fortran (providing the language is supported for that platform; please see “Supported Software by Platform” on page 7 for details).
- Include files and code samples to develop applications that use embedded SQL.
- Programming libraries, include files, and code samples that use the DB2 Call Level Interface (DB2 CLI) to develop applications which are easily ported to ODBC and compiled with an ODBC SDK, which is available from Microsoft for Windows 32-bit operating systems, and from various other vendors for many of the other supported platforms. For Windows 32-bit operating systems, DB2 clients contain an ODBC driver that supports applications developed with the Microsoft ODBC Software Developer’s Kit. For all other platforms, DB2 clients contain an optionally installed ODBC driver that supports applications that can be developed with an ODBC SDK for that platform, if one exists.
- DB2 Java Enablement, which includes DB2 Java Database Connectivity (DB2 JDBC) support to develop Java applications and applets, and DB2 embedded SQL for Java (DB2 SQLJ) support to develop Java embedded SQL applications and applets.
- On Windows 32-bit operating systems, Java Development Kit (JDK) 1.1.7 and Java Runtime Environment (JRE) 1.1.7 for Win32 from IBM.
- On AIX, OS/2, and Windows 32-bit operating systems, support to develop database applications that use the REXX language.
- On Windows 32-bit operating systems, code samples with ActiveX Data Objects (ADO), and Object Linking and Embedding (OLE) automation UDFs, implemented in Microsoft Visual Basic and Microsoft Visual C++. Also, code samples with Remote Data Objects (RDO) implemented in Microsoft Visual Basic.
- On Windows 32-bit operating systems, Object Linking and Embedding Database (OLE DB) table functions.
- DB2 Stored Procedure Builder (SPB), a GUI-based tool that supports the rapid development of DB2 stored procedures. It provides a single development environment for the DB2 family ranging from workstation to OS/390. It can be launched from these popular application development tools: Microsoft Visual Studio, Microsoft Visual Basic, and IBM Visual Age for Java, or launched as a separate application from the IBM DB2 Universal Database program group.
- Interactive SQL through the Command Line Processor (CLP) to prototype SQL statements or to perform ad hoc queries against the database.
- A set of documented APIs to enable other application development tools to implement precompiler support for DB2 directly within their products. For example, on AIX and OS/2, IBM COBOL and PL/I use this interface.

Information on the set of precompiler service APIs, and how to use them, is available from the anonymous FTP site, ftp://ftp.software.ibm.com. The PostScript file, called prepapi.psbin, is located in the directory /ps/products/db2/info. This file is in binary format. If you do not have access to this electronic forum and would like to get a copy of this document, you can call IBM Service as described in the Service Information Flyer.

- An SQL92 and MVS Conformance Flagger, which 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 OS/390. If you migrate applications developed on a workstation to another platform, the Flagger saves you time by showing syntax incompatibilities. Refer to the *Command Reference* for information about the SQLFLAG option in the PRECOMPILE PROGRAM command.

Supported Servers

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

- DB2 for AIX
- DB2 for HP-UX
- DB2 for Linux
- DB2 for OS/2
- DB2 for Solaris
- DB2 for Windows NT
- Distributed Relational Database Architecture (DRDA)-compliant application servers, such as:
  - DB2 for OS/390
  - DB2 for OS/400
  - DB2 for VSE & VM (formerly SQL/DS for VM and VSE)
  - DRDA-compliant application servers from database vendors other than IBM.
- DB2 CLI applications that conform to ODBC can be ported to work under ODBC, provided you recompile the application using an ODBC SDK (not included with DB2), and also provided an ODBC driver manager is available on the application platform.

Supported Software by Platform

This section lists the compilers and related software supported by DB2 for the platforms described in this book. The compiler information assumes that you are using the DB2 precompiler for that platform, and not the precompiler support that may be built into one of the listed compilers. Refer to the *Quick Beginnings* book for your operating system for information on the communication products it supports.

For the latest DB2 compiler information and related software updates, visit the Web page at:

```
http://www.software.ibm.com/data/db2/udb/ad
```

Notes:

1. The **README** file for a supported platform may contain updated information on compilers supported for that platform. The **README** file for a platform can be found in the directory in which the program files are installed.
2. **Fortran and REXX.** DB2 will not enhance features for Fortran and REXX beyond the level of support for these languages in DB2 Universal Database Version 5.2.
3. **HP-UX**. If you are migrating DB2 from HP-UX Version 10 or earlier to HP-UX Version 11, your DB2 programs must be re-precompiled with DB2 on HP-UX Version 11 (if they include embedded SQL), and must be re-compiled. This includes all DB2 applications, stored procedures, user-defined functions and user exit programs. As well, DB2 programs that are compiled on HP-UX Version 11 may not run on HP-UX Version 10 or earlier. DB2 programs that are compiled and run on HP-UX Version 10 may connect remotely to HP-UX Version 11 servers.
4. **Micro Focus COBOL**. Any existing applications precompiled with DB2 Version 2.1.1 or earlier and compiled with Micro Focus COBOL should be re-precompiled with the current version of DB2, and then recompiled with Micro Focus COBOL. If these applications built with the earlier versions of the IBM precompiler are not re-precompiled, there is a possibility of database corruption if abnormal termination occurs.

AIX

The DB2 SDK for AIX supports the following operating system:

AIX/6000
Version 4.2 and later

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

C IBM C for AIX Version 3.1.4 and 3.6.4

C++ IBM C Set++ for AIX Version 3.1.4 and 3.6.4

IBM VisualAge C++ 4.0 for AIX

COBOL
IBM COBOL Set for AIX Version 1.1

Micro Focus COBOL Version 3.2.46

Micro Focus COBOL Version 4.0.20 (PRN 12.03 or higher)

Micro Focus COBOL Version 4.1.10 (PRN 13.04 or higher)

Fortran
IBM XL Fortran for AIX Versions 4.1 and 5.1

Java Java Development Kit (JDK) Version 1.1.2 or later for AIX from IBM

REXX IBM AIX REXX/6000 AISPO Product Number: 5764-057

HP-UX

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

HP-UX
Versions 10.20 and 11.0

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

C HP C/HP-UX Version A.10.13, with Patch Level PHSS_5743

HP C Compiler version A.11.00.00 (for HP-UX Version 11)

C++ HP-UX C++ Version A.10.03.60, with Patch Level PHSS_5883

HP C++ Version B.11.00, with Patch Level PHKL_14174 (for HP-UX 11.0 only)

COBOL
Micro Focus COBOL Version 3.2

Fortran
HP Fortran/9000 Version 10.0

HP-UX F77 B.11.00.01 (for HP-UX Version 11)

Java HP-UX Developer’s Kit for Java Release 1.1.3 or later from Hewlett-Packard

Linux

The DB2 SDK for Linux supports the following operating system:

Linux Linux kernel Version 2.0.35 or later, glibc Version 2.0.7, libstdc++ version 2.8.0, rpm (required to install), and the pdksh package (required to run the DB2 command line processor)

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

C GNU/Linux gcc Version 2.7.2.3

C++ GNU/Linux g++ Version egcs-2.90.27 980315 (egcs-1.0.2 release)

Java Java Development Kit (JDK) 1.1.7, Version 1a for Linux from the Blackdown Organization

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 4.0

COBOL
IBM VisualAge COBOL for OS/2 Versions 1.2 and 2.2

Micro Focus COBOL Version 4.0.20

FORTRAN
WATCOM FORTRAN 77 32 Version 10.5

Java Java Development Kit (JDK) 1.1.4 or later for OS/2 from IBM

REXX IBM Procedures Language 2/REXX (supplied as part of OS/2)

Silicon Graphics IRIX

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

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

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

C MIPSpro C Compiler 7.2

C++ MIPSpro C++ 7.2

Fortran
MIPSpro Fortran-77 7.2

Java Java Development Environment 3.1 (Sun JDK 1.1.5) and Java Runtime Environment 3.1 (Sun JRE 1.1.5) from Silicon Graphics, Inc.

Solaris

The DB2 SDK for Solaris supports the following operating system:

Solaris
Version 2.5.1, Version 2.6, and Version 2.7 (Solaris 7)

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

C SPARCompiler C Version 4.2

SPARCompiler C Version 5.0

C++ SPARCompiler C++ Version 4.2

SPARCompiler C++ Version 5.0

COBOL
Micro Focus COBOL Version 3.2

Fortran
SPARCompiler Fortran Version 4.2

Java Java Development Kit (JDK) Version 1.1.4 for Solaris from Sun Microsystems

Windows 32-bit Operating Systems

The DB2 SDK for Windows 32-bit operating systems supports the following:

Microsoft Windows NT
Version 4.0 or later (both workstation and server versions)

Microsoft Windows 98

Microsoft Windows 95
Version 4.00.950 or later

The DB2 SDK for Windows 32-bit operating systems supports the following programming languages:

Basic Microsoft Visual Basic Version 4.0 or later (no DB2 precompiler is supplied for this language)

C/C++ Microsoft Visual C++ Version 5.0 and 6.0

IBM VisualAge C++ for Windows Version 3.5 and 4.0

COBOL
Micro Focus COBOL Version 4.0.20

	IBM VisualAge COBOL Version 1.2 and 2.2
REXX	Object REXX for Windows NT/95 Version 1.0.2
Java	Java Development Kit (JDK) 1.1.7 and Java Runtime Environment (JRE) 1.1.7 for Win32 from IBM (shipped with DB2)
	Java Development Kit (JDK) 1.1 or later for Win32 from Sun Microsystems
	Microsoft Software Developer's Kit for Java, Version 3.1

Sample Programs

Notes:

1. This section describes sample programs for the programming languages for all platforms supported by DB2. Not all sample programs have been ported to all supported programming languages.
2. DB2 sample programs are provided ″as is″ without any warranty of any kind. The user, and not IBM, assumes the entire risk of quality, performance, and repair of any defects.

The sample programs come with the DB2 SDK. You can use the sample programs as templates to create your own applications. The file extensions for each supported language, as well as for programs categorized by group, are given in the following tables:

Sample File Extensions by Language
Table 2 on page 12.

Sample File Extensions by Program Group
Table 3 on page 13.

The following tables document the sample programs by type:

DB2 API Sample Programs with No Embedded SQL
Table 4 on page 15.

DB2 API Embedded SQL Sample Programs
Table 5 on page 18.

Embedded SQL Sample Programs with No DB2 APIs
Table 6 on page 20.

User-Defined Function Sample Programs
Table 7 on page 22

DB2 CLI Sample Programs
Table 8 on page 22.

Java JDBC Sample Programs
Table 9 on page 25.

Java SQLJ Sample Programs
Table 10 on page 27.

ActiveX Data Objects, Remote Data Objects, and Microsoft Transaction Server Sample Programs
Table 11 on page 28.

Object Linking and Embedding (OLE) Automation Sample Programs
Table 12 on page 29.

Object Linking and Embedding Database (OLE DB) Table Functions
Table 13 on page 29.

Command Line Processor (CLP) Sample Programs
Table 14 on page 30.

Log Management User Exit Programs
Table 15 on page 30.

Notes:

1. Table 5 on page 18 contains programs that have both DB2 APIs and embedded SQL statements. For all DB2 API sample programs, please see both Table 4 on page 15 and Table 5 on page 18. For all embedded SQL sample programs (except for Java SQLJ), please see both Table 5 on page 18 and Table 6 on page 20.
2. Table 7 on page 22 of UDF sample programs does not contain DB2 CLI UDF programs. For these, please see Table 8 on page 22.

Table 2. Sample File Extensions by Language

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

Table 3. Sample File Extensions by Program Group

Sample Group	Directory	File Extension
CLP	`samples/clp`	`.db2`
OLE	`samples\ole\msvb` (Visual Basic) `samples\ole\msvc` (Visual C++)	`.bas .vbp` (Visual Basic) `.cpp` (Visual C++)
OLE DB	`samples\oledb`	`.db2`

ADO, RDO, MTS	`samples\ADO\VB` (Visual Basic)
	`samples\ADO\VC` (Visual C++)
	`samples\RDO`
	`samples\MTS`
		`.bas .frm .vbp` (Visual Basic)
		`.cpp .dsp .dsw` (Visual C++)

User Exit	`samples/c`	`.cad` (OS/2) `.cadsm` (UNIX & Windows) `.cdisk` (UNIX & Windows) `.ctape` (UNIX)

Note:

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

File Extensions
Are provided for the samples in the tables where only one extension exists.

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

IBM COBOL samples
Are only supplied for the OS/2, AIX, and Windows 32-bit operating systems in the `cobol` subdirectory.

Micro Focus Cobol Samples
Are supplied on all platforms except Linux and Silicon Graphics IRIX. On all other platforms, the Micro Focus COBOL samples are in the `cobol_mf` subdirectory.

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

Java Samples
Are Java Database Connectivity (JDBC) applets, applications, and stored procedures, embedded SQL for Java (SQLJ) applets,

applications, and stored procedures, as well as Java UDFs. Java samples are available on all supported DB2 platforms.

REXX Samples
Are only supplied for the AIX, OS/2, and Windows NT operating systems.

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

OLE Samples
Are for Object Linking and Embedding (OLE) in Microsoft Visual Basic and Microsoft Visual C++, supplied for Windows 32-bit operating systems only.

ADO, RDO, and MTS Samples
Are ActiveX Data Objects samples in Microsoft Visual Basic and Microsoft Visual C++, and Remote Data Objects and Microsoft Transaction Server samples in Microsoft Visual Basic, supplied for Windows 32-bit operating systems only.

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

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

- On UNIX platforms.

  You can find the C source code for embedded SQL and DB2 API programs in `sqllib/samples/c` under your database instance directory; the C source code for DB2 CLI programs is in `sqllib/samples/cli`. For additional information about the programs in the samples tables, refer to the `README` file in the appropriate `samples` subdirectory under your DB2 instance. The `README` file will contain any additional samples that are not listed in this book.
- On OS/2 and Windows 32-bit operating systems.

  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 the drive where DB2 is installed, `%DB2PATH%` will point to *drive*:`\sqllib`. For additional information about the sample programs in the samples tables, 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.

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.

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

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

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

DB2 API Non-Embedded SQL Samples

Table 4. DB2 API Sample Programs with No Embedded SQL

Sample Program	Included APIs
backrest	• sqlbftcq - Fetch Tablespace Container Query • sqlbstsc - Set Tablespace Containers • sqlfudb - Update Database Configuration • sqlubkp - Backup Database • sqluroll - Rollforward Database • sqlurst - Restore Database
checkerr	• sqlaintp - Get Error Message • sqlogstt - Get SQLSTATE Message
cli_info	• sqleqryi - Query Client Information • sqleseti - Set Client Information
client	• sqleqryc - Query Client • sqlesetc - Set Client

Table 4. DB2 API Sample Programs with No Embedded SQL (continued)

Sample Program	Included APIs
d_dbconf	• sqleatin - Attach • sqledtin - Detach • sqlfddb - Get Database Configuration Defaults
d_dbmcon	• sqleatin - Attach • sqledtin - Detach • sqlfdsys - Get Database Manager Configuration Defaults
db_udcs	• sqleatin - Attach • sqlecrea - Create Database • sqledrpd - Drop Database
db2mon	• sqleatin - Attach • sqlmon - Get/Update Monitor Switches • sqlmonss - Get Snapshot • sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer • sqlmrset - Reset Monitor
dbcat	• sqlecadb - Catalog Database • sqledcls - Close Database Directory Scan • sqledgne - Get Next Database Directory Entry • sqledosd - Open Database Directory Scan • sqleuncd - Uncatalog Database
dbcmt	• sqledcgd - Change Database Comment • sqledcls - Close Database Directory Scan • sqledgne - Get Next Database Directory Entry • sqledosd - Open Database Directory Scan • sqleisig - Install Signal Handler
dbconf	• sqleatin - Attach • sqlecrea - Create Database • sqledrpd - Drop Database • sqlfrdb - Reset Database Configuration • sqlfudb - Update Database Configuration • sqlfxdb - Get Database Configuration

Table 4. DB2 API Sample Programs with No Embedded SQL (continued)

Sample Program	Included APIs
dbinst	• sqleatcp - Attach and Change Password • sqleatin - Attach • sqledtin - Detach • sqlegins - Get Instance
dbmconf	• sqleatin - Attach • sqledtin - Detach • sqlfrsys - Reset Database Manager Configuration • sqlfusys - Update Database Manager Configuration • sqlfxsys - Get Database Manager Configuration
dbsnap	• sqleatin - Attach • sqlmonss - Get Snapshot
dbstart	• sqlepstart - Start Database Manager
dbstop	• sqlefrce - Force Application • sqlepstp - Stop Database Manager
dcscat	• sqlegdad - Catalog DCS Database • sqlegdcl - Close DCS Directory Scan • sqlegdel - Uncatalog DCS Database • sqlegdge - Get DCS Directory Entry for Database • sqlegdgt - Get DCS Directory Entries • sqlegdsc - Open DCS Directory Scan
dmscont	• sqleatin - Attach • sqlecrea - Create Database • sqledrpd - Drop Database
ebcdicdb	• sqleatin - Attach • sqlecrea - Create Database • sqledrpd - Drop Database
migrate	• sqlemgdb - Migrate Database
monreset	• sqleatin - Attach • sqlmrset - Reset Monitor
monsz	• sqleatin - Attach • sqlmonss - Get Snapshot • sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer

Table 4. DB2 API Sample Programs with No Embedded SQL (continued)

Sample Program	Included APIs
nodecat	• sqlectnd - Catalog Node • sqlencls - Close Node Directory Scan • sqlengne - Get Next Node Directory Entry • sqlenops - Open Node Directory Scan • sqleuncn - Uncatalog Node
regder	• sqledreg - Deregister • sqleregs - Register
restart	• sqlerstd - Restart Database
setact	• sqlesact - Set Accounting String
setrundg	• sqlesdeg - Set Runtime Degree
sws	• sqleatin - Attach • sqlmon - Get/Update Monitor Switches
util	• sqlaintp - Get Error Message • sqlogstt - Get SQLSTATE Message

DB2 API Embedded SQL Samples

Table 5. DB2 API Embedded SQL Sample Programs

Sample Program	Included APIs
asynrlog	• sqlurlog - Asynchronous Read Log
bindfile	• sqlabndx - Bind
dbauth	• sqluadau - Get Authorizations
dbstat	• sqlureot - Reorganize Table • sqlustat - Runstats
expsamp	• sqluexpr - Export • sqluimpr - Import
impexp	• sqluexpr - Export • sqluimpr - Import
loadqry	• db2LoadQuery - Load Query
makeapi	• sqlabndx - Bind • sqlaprep - Precompile Program • sqlepstp - Stop Database Manager • sqlepstr - Start Database Manager

Table 5. DB2 API Embedded SQL Sample Programs (continued)

Sample Program	Included APIs
qload	• sqluqry - Load Query
rebind	• sqlarbnd - Rebind
rechist	• sqlubkp - Backup Database • sqluhcls - Close Recovery History File Scan • sqluhgne - Get Next Recovery History File Entry • sqluhops - Open Recovery History File Scan • sqluhprn - Prune Recovery History File • sqluhupd - Update Recovery History File
tabscont	• sqlbctcq - Close Tablespace Container Query • sqlbftcq - Fetch Tablespace Container Query • sqlbotcq - Open Tablespace Container Query • sqlbtcq - Tablespace Container Query • sqlefmem - Free Memory
tabspace	• sqlbctsq - Close Tablespace Query • sqlbftpq - Fetch Tablespace Query • sqlbgtss - Get Tablespace Statistics • sqlbmtsq - Tablespace Query • sqlbotsq - Open Tablespace Query • sqlbstpq - Single Tablespace Query • sqlefmem - Free Memory
tload	• sqluexpr - Export • sqluload - Load • sqluvqdp - Quiesce Tablespaces for Table

Table 5. DB2 API Embedded SQL Sample Programs (continued)

Sample Program	Included APIs
tspace	• sqlbctcq - Close Tablespace Container Query • sqlbctsq - Close Tablespace Query • sqlbftcq - Fetch Tablespace Container Query • sqlbftpq - Fetch Tablespace Query • sqlbgtss - Get Tablespace Statistics • sqlbmtsq - Tablespace Query • sqlbotcq - Open Tablespace Container Query • sqlbotsq - Open Tablespace Query • sqlbstpq - Single Tablespace Query • sqlbstsc - Set Tablespace Containers • sqlbtcq - Tablespace Container Query • sqlefmem - Free Memory

Embedded SQL Samples With No DB2 APIs

Table 6. Embedded SQL Sample programs with No DB2 APIs

Sample Program Name	Program Description
`adhoc`	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`	Demonstrates the use of advanced SQL expressions like CASE, CAST, and scalar full selects.
`blobfile`	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 this file can be displayed using an external viewer.
`calludf`	Demonstrates calling user-defined functions (UDFs) created by the `udf` program, and stored on the server to accesss tables in the `sample` database.
`columns`	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`	Demonstrates the use of a cursor using static SQL.
`delet`	Demonstrates static SQL to delete items from a database.
`dynamic`	Demonstrates the use of a cursor using dynamic SQL.
`fillcli`	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.

Table 6. Embedded SQL Sample programs with No DB2 APIs (continued)

Sample Program Name	Program Description
fillsrv	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.
inpcli	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	Demonstrates stored procedures using the SQLDA structure. This is the server program of a client/server example. (The client program is called inpcli.) The program creates a table (PRESIDENTS) in the sample database with the information received in the SQLDA. The server program does all the database processing and returns the SQLCA status to the client program.
joinsql	Demonstrates using advanced SQL join expressions.
largevol	Demonstrates parallel query processing in a partitioned environment, and the use of an NFS file system to automate the merging of the result sets. Only available on AIX.
lobeval	Demonstrates the use of LOB locators and defers the evaluation of the actual LOB data.
lobfile	Demonstrates the use of LOB file handles.
lobloc	Demonstrates the use of LOB locators.
lobval	Demonstrates the use of LOBs.
openftch	Demonstrates fetching, updating, and deleting of rows using static SQL.
outcli	Demonstrates stored procedures using a host variable. This is the client program of a client/server example. (The server program is called outsrv.) This program declares a single OUT variable to hold the value returned from the server program. This program demonstrates the use of the CREATE PROCEDURE statement to register a stored procedure and shows the invocation of stored procedures using an embedded SQL CALL statement.
outsrv	Demonstrates stored procedures using host variables. This is the server program of a client/server example. (The client program is called outcli.) The program sets the value of the function parameter to the median SALARY of the employees in the STAFF table of the sample database. The server does all the database processing (finding the median). The server program returns the host variable and the SQLCA status to the client program.
recursql	Demonstrates the use of advanced SQL recursive queries.
sampudf	Demonstrates User-Defined Types (UDTs) and User-Defined Functions (UDFs) implemented to modify table entries. All UDFs declared in this program are sourced UDFs.

Table 6. Embedded SQL Sample programs with No DB2 APIs (continued)

Sample Program Name	Program Description
`static`	Demonstrates static SQL to retrieve information.
`tabsql`	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.
`thdsrver`	Demonstrates the use of POSIX threads APIs for thread creation and management. The program maintains a pool of contexts. A generate_work function is executed from main, and creates dynamic SQL statements that are executed by worker threads. When a context becomes available, a thread is created and dispatched to do the specified work. The work generated consists of statements to delete entries from either the STAFF or EMPLOYEE tables of the `sample` database. This program is only available on UNIX platforms.
`trigsql`	Demonstrates using advanced SQL triggers and constraints.
`updat`	Demonstrates static SQL to update a database.
`varinp`	Demonstrates variable input to Embedded Dynamic SQL statement calls using parameter markers.

User-Defined Function Samples

Table 7. User-Defined Function Sample programs

Sample Program Name	Program Description
`DB2Udf.java`	A Java UDF that demonstrates several tasks, including integer division, manipulation of Character Large Objects (CLOBs), and the use of Java instance variables.
`tblsrv.c`	Demonstrates a table function (server-side) that processes weather information for a number of cities.
`udf.c`	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.
`UDFsrv.java`	Demonstrates the use of Java User-Defined Functions (UDFs).

DB2 Call Level Interface Samples

Table 8. 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

Table 8. Sample CLI Programs in DB2 Universal Database (continued)

Sample Program Name	Program Description
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
calludf.c	Register and call a UDF
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 the resume clob field 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)

Table 8. Sample CLI Programs in DB2 Universal Database (continued)

Sample Program Name	**Program Description**
`setcurs.c`	Rewrite of `getcurs.c` using SQLSetCurs for positioned update
`seteattr.c`	Set environment attribute (SQL_ATTR_OUTPUT_NTS)
`tables.c`	List all tables
`typeinfo.c`	Display type information for all types for current data source
`xfetch.c`	Extended Fetch, multiple rows per fetch
BLOB Samples	
`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

Table 8. Sample CLI Programs in DB2 Universal Database (continued)

Sample Program Name	Program Description
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
Samples unchanged from DB2 Version 2	
v2sutil.c	samputil.c using old v2 functions
v2sutil.h	samputil.h using old v2 functions
v2fetch.c	fetch.c using old v2 functions
v2xfetch.c	xfetch.c using old v2 functions

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

- README - Lists all example files.
- makefile - Makefile for all files
- build files for applications and stored procedures

Java Samples

Table 9. Java Database Connectivity (JDBC) Sample Programs

Sample Program Name	Program Description
BasicCon.java	Basic connection.
Browser.java	List columns, foreign keys, index columns or stats for a table
ColPriv.java	List column Privileges
Columns.java	List all columns for table search string.
DB2Appl.java	A JDBC application that queries the sample database using the invoking user's privileges.
DB2Applt.java	A JDBC applet that queries the database using the JDBC applet driver. It uses the user name, password, server, and port number parameters specified in DB2Applt.html.
DB2Applt.html	An HTML file that embeds the applet sample program, DB2Applt. It needs to be customized with server and user information.
DB2SpCli.java	A Java client application that calls the JDBC stored procedure, DB2Stp.
DB2Stp.java	A Java stored procedure that updates the EMPLOYEE table on the server, and returns new salary and payroll information to the client.
DB2UdCli.java	A Java client application that calls the Java user-defined function, DB2Udf.
Dynamic.java	Demonstrates a cursor using dynamic SQL.
Embedded.java	Create, populate, list data from and drop a table.

Table 9. Java Database Connectivity (JDBC) Sample Programs (continued)

Sample Program Name	Program Description
GetAttrs.java	List some common environment, connection and statement options/attributes.
GetData.java	Simple querying and displaying of data from database.
Inpcli.java	A Java client application that calls the JDBC stored procedure, Inpsrv.
Inpsrv.java	A Java stored procedure demonstrating the SQLDA structure. It creates a PRESIDENTS table in the sample database with information received from the SQLDA, and returns the SQLCA status to the client application, Inpcli.
JavaSample.java	Creates the table, JAVA_SAMPLE, adds data, displays data, and deletes the table.
JobChange.java	Uses a statement and a prepared statement to query the database and update information at the same time (does the same as getcurs.c and setcurs.c but cannot do both ways therefore above method is used).
LookRes.java	Extract string from the resume clob field by converting the entire clob into a string, and searching for the desired substring.
MRSPcli.java	This is the client program that calls the server program MRSPsrv. The program demonstrates multiple result sets being returned from a Java stored procedure.
MRSPsrv.java	This is the server program that is called by the client program, MRSPcli. The program demonstrates multiple result sets being returned from a Java stored procedure.
MultiCon.java	Multiple connections.
Outcli.java	A Java client application that calls the SQLJ stored procedure, Outsrv.
PicIn.java	Loads graphic BLOBS into the emp_photo using the setBinaryStream method.
PluginEx.java	A Java program that adds new menu items and toolbar buttons to the DB2 Web Control Center.
Prepare.java	Rewrite of Simple.java, using a prepared statement.
ProcCols.java	List procedure parameters using getProcedureColumns under DatabaseMetaData.
Procs.java	List procedures using DatabaseMetaData.getProcedures.
ShowPic.java	Extracts BLOB picture to file using the getBinaryStream method. The file is read using the InputStream.read method.
Simple.java	Shows basic connecting querying and displaying the result set. (JDBC version of CLI sample fetch.c).
StpCli.java	A Java client application that calls the SQLJ stored procedure, Stp.
Tables.java	List all tables that match user defined search pattern.
Tools.java	Demonstrates a toolkit for Java programs written as DB2 samples.
TypeInfo.java	Display type information for all types for current data source.
UDFcli.java	A JDBC client application that calls functions in the Java user-defined function library, UDFsrv.

Table 9. Java Database Connectivity (JDBC) Sample Programs (continued)

Sample Program Name	Program Description
UsingThreads.java	Shows how to use threads to run an SQL statement asynchronously (JDBC version of CLI sample async.c).
Varinp.java	Demonstrates variable input to Embedded Dynamic SQL statement calls using parameter markers.
Samples using ORDER tables created by Create.java (Run in the following order).	
Create.java	Creates all tables for the order scenario.
CustIn.java	Inserts customers into the customer table (array insert).
ProdIn.java	Inserts products into the products table (array insert).
ProdPart.java	Inserts parts into the prod_parts table (array insert).
OrdIn.java	Inserts orders into the ord_line, ord_cust tables (array insert).
OrdRep.java	Generates a customer order report and uses the user defined function ’price’ defined in order.c.
PartRep.java	Generates exploding parts report (recursive SQL Query).
DropJava.java	Removes the new tables created for the ORDER scenario.

Table 10. Embedded SQL for Java (SQLJ) Sample Programs

Sample Program Name	Program Description
App.sqlj	Uses static SQL to retrieve and update data from the EMPLOYEE table of the sample database.
Applt.sqlj	An applet that queries the database using the JDBC applet driver. It uses the user name, password, server, and port number parameters specified in Applt.html.
Applt.html	An HTML file that embeds the applet sample program, Applt. It needs to be customized with server and user information.
Cursor.sqlj	Demonstrates an iterator using static SQL.
OpF_Curs.sqlj	Class file for the Openftch program.
Openftch.sqlj	Demonstrates fetching, updating, and deleting rows using static SQL.
Outsrv.sqlj	Demonstrates a stored procedure using the SQLDA structure. It fills the SQLDA with the median salary of the employees in the STAFF table of the sample database. After the database processing (finding the median), the stored procedure returns the filled SQLDA and the SQLCA status to the JDBC client application, Outcli.
Static.sqlj	Uses static SQL to retrieve information.
Stp.sqlj	A stored procedure that updates the EMPLOYEE table on the server, and returns new salary and payroll information to the JDBC client program, StpCli.
UDFclie.sqlj	A client application that calls functions from the Java user-defined function library, UDFsrv.

Table 10. Embedded SQL for Java (SQLJ) Sample Programs (continued)

Sample Program Name	Program Description
`Updat.sqlj`	Uses static SQL to update a database.

ADO, RDO, and MTS Samples

Table 11. ADO, RDO, and MTS Sample Programs

Sample Program Name	Program Description
`Bank.vbp`	An RDO program to create and maintain data for bank branches, with the ability to perform transactions on customer accounts. The program can use any database specified by the user as it contains the DDL to create the necessary tables for the application to store data.
`Blob.vbp`	This ADO program demonstrates retrieving BLOB data. It retrieves and displays pictures from the emp_photo table of the `sample` database. The program can also replace an image in the emp_photo table with one from a local file.
`BLOBAccess.dsw`	This sample demonstrates highlighting ADO/Blob access using Microsoft Visual C++. It is similar to the Visual Basic sample, `Blob.vbp`. The BLOB sample has two main functions: 1. Read a BLOB from the Sample database and display it to the screen. 2. Read a BLOB from a file and insert it into the database. (Import)
`Connect.vbp`	This ADO program will create a connection object, and establish a connection, to the `sample` database. Once completed, the program will disconnect and exit.
`Commit.vbp`	This application demonstrates the use of autocommit/manual-commit features of ADO. The program queries the EMPLOYEE table of the `sample` database for employee number and name. The user has an option of connecting to the database in either autocommit or manual-commit mode. In the autocommit mode, all of the changes that a user makes on a record are updated automatically in the database. In the manual-commit mode, the user needs to begin a transaction before he/she can make any changes. The changes made since the beginning of a transaction can be undone by performing a rollback. The changes can be saved permanently by committing the transaction. Exiting the program automatically rolls back the changes.
`db2com.vbp`	This Visual Basic project demonstrates updating a database using the Microsoft Transaction Server. It creates a server DLL used by the client program, `db2mts.vbp`, and has four class modules: • UpdateNumberColumn.cls • UpdateRow.cls • UpdateStringColumn.cls • VerifyUpdate.cls For this program a temporary table, `DB2MTS`, is created in the `sample` database.

Table 11. ADO, RDO, and MTS Sample Programs (continued)

Sample Program Name	Program Description
db2mts.vbp	This is a Visual Basic project for a client program that uses the Microsoft Transaction Server to call the server DLL created from db2com.vbp.
Select-Update.vbp	This ADO program performs the same functions as Connect.vbp, but also provides a GUI interface. With this interface, the user can view, update, and delete data stored in the ORG table of the sample database.
Sample.vbp	This Visual Basic project uses Keyset cursors via ADO to provide a graphical user interface to all data in the sample database.
VarCHAR.dsp	A Visual C++ program that uses ADO to access VarChar data as textfields. It provides a graphical user interface to allow users to view and update data in the ORG table of the sample database.

Object Linking and Embedding Samples

Table 12. Object Linking and Embedding (OLE) Sample Programs

Sample Program Name	Program Description
sales.vbp	Demonstrates rollup queries on a Microsoft Excel sales spreadsheet (implemented in Visual Basic).
names.vbp	Queries a Lotus Notes address book (implemented in Visual Basic).
inbox.vbp	Queries Microsoft Exchange inbox e-mail messages through OLE/Messaging (implemented in Visual Basic).
invoice.vbp	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.vbp	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 13. Object Linking and Embedding Database (OLE DB) Table Functions

Sample Program Name	Program Description
jet.db2	Microsoft.Jet.OLEDB.3.51 Provider
mapi.db2	INTERSOLV Connect OLE DB for MAPI
msdaora.db2	Microsoft OLE DB Provider for Oracle
msdasql.db2	Microsoft OLE DB Provider for ODBC Drivers
msidxs.db2	Microsoft OLE DB Index Server Provider
notes.db2	INTERSOLV Connect OLE DB for Notes

Table 13. Object Linking and Embedding Database (OLE DB) Table Functions (continued)

Sample Program Name	Program Description
sampprov.db2	Microsoft OLE DB Sample Provider
sqloledb.db2	Microsoft OLE DB Provider for SQL Server

Command Line Processor Samples

Table 14. Command Line Processor (CLP) Sample Programs.

Sample File Name	File Description
const.db2	Creates a table with a CHECK CONSTRAINT clause.
cte.db2	Demonstrates a common table expression. The equivalent sample program demonstrating this advanced SQL statement is tabsql.
flt.db2	Demonstrates a recursive query. The equivalent sample program demonstrating this advanced SQL statement is recursql.
join.db2	Demonstrates an outer join of tables. The equivalent sample program demonstrating this advanced SQL statement is joinsql.
stock.db2	Demonstrates the use of triggers. The equivalent sample program demonstrating this advanced SQL statement is trigsql.
testdata.db2	Uses DB2 built-in functions such as RAND() and TRANSLATE() to populate a table with randomly generated test data.
thaisort.db2	This script is particularly for Thai users. Thai sorting is by phonetic order requiring pre-sorting/swapping of the leading vowel and its consonant, as well as post-sorting in order to view the data in the correct sort order. The file implements Thai sorting by creating UDF functions presort and postsort, and creating a table; then it calls the functions against the table to sort the table data. To run this program, you first have to build the user-defined function program, udf, from the C source file, udf.c.

Log Management User Exit Samples

Table 15. Log Management User Exit Sample Programs.

Sample File Name	File Description
db2uext2.cadsm	This is a sample User Exit utilizing ADSTAR DSM (ADSM) APIs to archive and retrieve database log files. The sample provides an audit trail of calls (stored in a separate file for each option) including a timestamp and parameters received. It also provides an error trail of calls in error including a timestamp and an error isolation string for problem determination. These options can be disabled. The file must be renamed db2uext2.c and compiled as a C program. Available on UNIX and Windows 32-bit operating systems. The OS/2 version is db2uexit.cad.
db2uexit.cad	This is the OS/2 version of db2uext2.cadsm. The file must be renamed db2uexit.c and compiled as a C program.

Table 15. Log Management User Exit Sample Programs. (continued)

Sample File Name	File Description
db2uext2.cdisk	This is a sample User Exit utilizing the system copy command for the particular platform on which it ships. The program archives and retrieves database log files, and provides an audit trail of calls (stored in a separate file for each option) including a timestamp and parameters received. It also provides an error trail of calls in error including a timestamp and an error isolation string for problem determination. These options can be disabled. The file must be renamed db2uext2.c and compiled as a C program. Available on UNIX and Windows 32-bit operating systems.
db2uext2.ctape	This is a sample User Exit utilizing system tape commands for the particular UNIX platform on which it ships. The program archives and retrieves database log files. All limitations of the system tape commands are limitations of this user exit. The sample provides an audit trail of calls (stored in a separate file for each option) including a timestamp and parameters received. It also provides an error trail of calls in error including a timestamp and an error isolation string for problem determination. These options can be disabled. The file must be renamed db2uext2.c and compiled as a C program. Available on UNIX platforms only.

Chapter 2. Setup

Creating a proper environment for building and running DB2 applications requires properly setting up the following:

1. Compiler or interpreter
2. DB2 (database manager, DB2 SDK, and client connection)
3. Operating system environment
4. DB2 Sample database (optional)

Checking your Compiler or Interpreter Environment

To develop DB2 programs, you must use a compiler or interpreter for one of the supported programming languages for your operating system, listed in “Supported Software by Platform” on page 7. It is recommended that you ensure your existing compiler or interpreter 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.

Setting Up the DB2 Environment

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

- The database manager on the server with the database instance for your environment. Refer to “Appendix A. About Database Manager Instances” on page 391 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.

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

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

- "Setting the OS/2 Environment" on page 34
- "Setting the UNIX Environment" on page 35
- "Setting the Windows 32-bit Operating Systems Environment" on page 36

After you set up your operating system environment, you may want to create the sample database, which is used by the examples in this book. To create the database, see "Creating, Cataloging, and Binding the Sample Database" on page 38.

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 first 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, this statement is put into the CONFIG.SYS file:

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

The command files in this book assume that this statement is present. If you edit the CONFIG.SYS file after installing the DB2 SDK, make sure this statement is not removed.

As well, these environment variables are automatically updated by DB2:

- PATH, to include the directory `%DB2PATH%\bin`
- LIBPATH, to include the directory `%DB2PATH%\dll`

For the Java environment variables updated by DB2, see "OS/2" on page 61.

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

C/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%
```

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.

Note: 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.

Setting the UNIX Environment

You need to set environment variables so you can access the database instance that was created when the database manager was installed. The *DB2 for UNIX Quick Beginnings* book provides general information about setting environment variables. This section provides specific instructions on setting environment variables to access a database instance.

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

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

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

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

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

Depending on the UNIX platform you are on, the following environment variables are automatically updated during DB2 instance creation:

AIX:

- PATH, to include several DB2 directories including `sqllib/bin`
- LIBPATH, to include the directory `sqllib/lib`

HP-UX:

- PATH, to include several DB2 directories including `sqllib/bin`
- SHLIB_PATH, to include the directory `sqllib/lib`

Linux, Silicon Graphics IRIX, and Solaris:

- PATH, to include several DB2 directories including `sqllib/bin`
- LD_LIBRARY_PATH, to include the directory `sqllib/lib`

For the Java environment variables updated by DB2, see “Setting the Environment” on page 58.

Setting the Windows 32-bit Operating Systems Environment

When you install the DB2 SDK on 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.

For the Java environment variables updated by DB2, see “Windows 32-bit Operating Systems” on page 65.

When you install the DB2 SDK on Windows 98 or 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 NT control panel
- The Windows 95 or Windows 98 command window
- The Windows 95 or Windows 98 `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 programs on Windows 32-bit operating systems. 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 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.
3. The database manager on a Windows NT environment is implemented as an NT service, and hence does not return errors or warnings if the service is started successfully, though other problems may have occurred. This means that when you run the `db2start` or the `NET START` command, no warnings will be returned if any communication subsystem failed to start. Therefore, the user should always examine the NT Event Log or the `DB2DIAG.LOG` for any errors that may have occurred during the running of these commands.

Enabling Communications on the Server

This section explains how to connect to DB2 Universal Database servers.

Before you begin installing, cataloging, and binding the `sample` database, you should ensure that 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 your platform's *Quick Beginnings* book for instructions on adding the `TCP/IP` settings to the Services file.

3. Start the database instance by entering:

   ```
   db2start
   ```

Windows NT

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 start this service 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.

Creating, Cataloging, and Binding the Sample Database

To use the sample programs shipped with DB2, you need to create the `sample` database on a server workstation. Refer to the *SQL Reference* for a listing of the contents of the `sample` database.

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

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

Creating

To create the `sample` database, you must have SYSADM authority. If you need more information about SYSADM authority, refer to the *Quick Beginnings* book for your operating system.

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

1. Ensure that the location of `db2sampl` (the program that creates the `sample` database) is in your path. The `db2profile` or `db2cshrc` file will put `db2sampl` in your path, so it will remain there unless you change it.
   - On UNIX servers, `db2sampl` is located in:
     ```
     $HOME/sqllib/bin
     ```
     where `$HOME` is the home directory of the DB2 instance owner.
   - On OS/2, Windows NT, Windows 98, and Windows 95 servers, `db2sampl` is located in:
     ```
     %DB2PATH%\bin
     ```
     where `%DB2PATH%` is where DB2 is installed.
2. Ensure that the DB2INSTANCE environment variable is set to the name of the instance where you want to create the `sample` database. If it is not set, you can set it with the following commands:
   - On UNIX platforms:

     you can do this for the bash or Korn shell by entering:
     ```
     DB2INSTANCE=instance_name
     export DB2INSTANCE
     ```
     and for the C shell by entering:
     ```
     setenv DB2INSTANCE instance_name
     ```
   - On OS/2, Windows NT, Windows 98, and Windows 95, enter:
     ```
     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 UNIX platforms, this is a *path*, and would be entered as:
   ```
   db2sampl path
   ```

On OS/2, Windows NT, Windows 98, and Windows 95, this is a *drive*, and would be entered as:

```
db2sampl drive
```

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

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

Creating on Host or AS/400 Servers

If you want to run the sample programs against a Host server, such as DB2 for OS/390, or an AS/400 server, you need to create a database that contains the sample tables described in the *SQL Reference*. You may want to refer to the sample program, `expsamp`, which uses the STAFF and ORG tables to demonstrate how APIs are used to import and export tables and table data to and from a DB2 Connect database.

To create the database:

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

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

Cataloging

To access the `sample` database on the server from a remote client, you need to catalog the `sample` database on the client workstation.

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

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

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

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

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

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

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

Binding

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

Binding creates the package that the database manager needs in order 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.

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

On an OS/2 Client Workstation:

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`.

On a UNIX Client Workstation:

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

   ```
   db2 connect to sample user <userid> using <password>
   ```

2. Bind the utilities to the database by entering:

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

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

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

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

On a Client Workstation running a Windows 32-bit operating system:

1. From the Start Menu, select Programs.
2. From the Programs Menu, select DB2 for Windows 32-bit operating systems.
3. From the DB2 for Windows 32-bit Operating Systems menu, select the DB2 command window.

   The command window displays.

4. Connect to the `sample` database. At the prompt, enter:

   ```
   db2 connect to sample
   ```

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.

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

For all Platforms

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

ddcsmvs.lst
: for DB2 for OS/390

ddcsvm.lst
: for DB2 for VM

ddcsvse.lst
for DB2 for VSE

ddcs400.lst
for DB2 for OS/400

The *Quick Beginnings* book for your operating system provides general information about binding the database utilities.

Where to Go Next

Once your environment is set up, you are ready to build your DB2 applications. The following chapters discuss the sample programs, and show you how to compile, link, and run them. First read "Chapter 3. General Information for Building DB2 Applications" on page 45, then the specific chapter that follows for the applications you are building.

For programming with DB2 APIs, DB2 CLI, and Embedded SQL, see the 'Building Applications' chapter for your platform. If you are programming with Java, see "Chapter 4. Building Java Applets and Applications" on page 57.

For further information, refer to the following books. To develop applications using embedded SQL, JDBC, and SQLJ, see the *Application Development Guide*. For applications using DB2 CLI or ODBC, see the *CLI Guide and Reference*. For DB2 API applications, see the *Administrative API Reference*.

Chapter 3. General Information for Building DB2 Applications

The information in this chapter applies to more than one operating system. The majority of the topics covered apply to most DB2-supported platforms, and are provided to help you improve your application development.

For the latest DB2 application development updates, visit the Web page at:

```
http://www.software.ibm.com/data/db2/udb/ad
```

General Points for Building and Running DB2 Programs

1. On OS/2, if you set your environment variables by a command file rather than in the CONFIG.SYS file, you must build your applications in the same window. On UNIX, you must build and run DB2 applications from a shell where your environment variables are set. Depending on the shell you are using, you can do this by running `db2profile` or `db2cshrc`. On Windows 32-bit operating systems, you must build your applications in a DB2 command window. Refer to “Chapter 2. Setup” on page 33 for more information.
2. To build DB2 programs containing embedded SQL, or to run any DB2 programs, the database manager on the server must be started. To start the database manager, you need SYSADM (system administration) authority. Refer to *Quick Beginnings* for information on SYSADM authority.

   Start the database manager (if it is not already running) by entering the following command on the server:

   ```
   db2start
   ```
3. It is recommended that, before altering or building the sample programs, you copy the samples of the language you will be using from `sqllib/samples` on UNIX, or from `%DB2PATH%\samples` on OS/2 or Windows 32-bit operating systems, to your own working directory. This allows you to preserve the original samples in case you need to refer to them in the future.

Build Files, Makefiles, and Error-checking Utilities

DB2 provides an array of building tools for your program development needs. These tools make it easy to build the supplied sample programs, which demonstrate the broad range of DB2 functionality. The tools also allow you to build your own database programs. For each supported compiler, DB2 provides build files, makefiles, and error-checking utilities, which are made available in the samples directory along with the sample programs. This section explains how these tools can be used.

Build Files

Each of the following chapters uses files that contain compile and link commands for building programs with the supported platform compilers. These files are known as command files on OS/2, script files on UNIX, and batch files on Windows 32-bit operating systems. We refer to them, generally, as build files.

The build files are documented in this book because they demonstrate very clearly the compile and link options that DB2 recommends for building various kinds of programs with the supported compilers. There are generally many other compile and link options available, and users are free to experiment with them. See your compiler documentation for all the compile and link options provided. Besides building the sample programs, developers can also build their own programs with the build files. The sample programs can be used as templates that can be modified by users to assist their programming development.

Conveniently, the build files are designed to build a source file with any file name allowed by the compiler. This is unlike the makefiles, where the program names are hardcoded into the file. The build files use the $1 variable on UNIX, or the %1 variable on OS/2 and Windows 32-bit operating systems, to substitute internally for the program name. Other similarly named variables substitute for other arguments that may be required. The build files allow for quick and easy experimentation, as each one is suited to a specific kind of program-building, such as DB2 APIs, DB2 CLI, embedded SQL, stored procedures, or user-defined functions. Each type of build file is provided wherever the specific kind of program it is designed for is supported by the compiler.

The object and executable files produced by a build file are automatically over-written each time a program with the same name is built. This is unlike a `makefile`. It means a developer can quickly modify an existing program and rerun it without having to delete previous object and executable files.

The build files contain a default setting for the sample database. If the user is accessing another database, he or she can simply supply another parameter to over-write the default. If they are using the other database consistently, they may wish to hardcode this database name, replacing `sample`, within the build file itself.

The build files for the embedded SQL samples also contain optional parameters (user ID and password) as they are needed to connect to the database in the building process (precompiling and binding). Making these parameters optional means that if a developer is building a program on a server instance where the database is located, then the user ID and password will be common to both, and therefore need not be provided as input parameters to the build file. On the other hand, if a developer is in a different instance, such as on a client machine accessing a server database remotely, then these parameters would have to be specified when running the build file.

Finally, the build files can be modified by the developer for his or her convenience. Besides changing the database name in the build file (explained above) the developer can easily hardcode other parameters within the file, change compile and link options, or change the default DB2 instance path. The simple, straightforward, and specific nature of the build files makes tailoring them to your needs an easy task.

Makefiles

Each samples directory for a supported compiler includes a makefile for building the supplied sample programs. The makefile hardcodes all the DB2 sample programs shipped for the compiler. It uses variables for many of the common elements used for each sample program compilation. The syntax for the makefiles and the output from their commands differ in some important respects from that of the build files supplied. The make commands are simple and powerful to use:

`make <program_name>`
Compiles and links the program specified.

`make all`
Compiles and links all programs listed in the makefile.

`make clean`
Deletes all intermediate files, such as object files, for all programs listed in the makefile.

`make cleanall`
Deletes all intermediate and executable files for all programs listed in the makefile.

Unlike the build files, the makefiles will not over-write existing intermediate and executable files for programs listed within it. This makes it faster, using the `make all` command, to create executables for some of the files if other files already have executables, as `make all` will just ignore these files. But it also assumes the need for the `make clean` and `make cleanall` commands, to get rid of existing object and executable files when they are not needed.

The makefiles can be used for program development. They are less convenient to use then the build files, but if you want the power and convenience of the make commands, this is a route to consider. To include a new program in an existing makefile, code in the syntax for the program entry, using as a template a similar kind of existing sample program. Note that the syntax differs for DB2 API, DB2 CLI, and embedded SQL programs, as well as for stored procedures and UDFs.

Here is an example of how you can use the supplied makefiles. This makefile, from the `samples/cli` directory on AIX, builds all the supplied DB2 CLI samples on AIX with the IBM C compiler. This example demonstrates how to build the stored procedure `outsrv2` into a shared library that can be called by the client application, `outcli2`.

Before using the `makefile`, you may need to edit the following variables contained in the file:

DB The database being used. As the default, set to `sample`.

UID The user ID being used. As the default, no value is set.

PWD The password for the `UID` user ID. As the default, no value is set.

On AIX, the DB2 CLI `makefile` defines the variables `DB2PATH`, `CC`, `COPY`, `ERASE`, `CFLAGS`, and `LIBS` as follows:

```
# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$(HOME)/sqllib

CC = cc
COPY = cp
ERASE = rm -f

# The required compiler flags
CFLAGS= -I$(DB2PATH)/include

# The required libraries
LIBS= -L$(DB2PATH)/lib -Wl,-rpath,$(DB2PATH)/lib -ldb2
```

The `makefile` uses these variables when it compiles the stored procedure, `outsrv2`, and copies it to the function sub-directory:

```
#outsrv2 is an output Stored procedure, called by outcli2
outsrv2 : outsrv2.o ;
        $(CC) -o outsrv2 outsrv2.c $(CFLAGS) -shared $(LIBS) ;
        $(ERASE) $(DB2PATH)/function/outsrv2 ;
        $(COPY) outsrv2 $(DB2PATH)/function/outsrv2 ;
```

There is another build file that is called by makefiles that contain embedded SQL programs. It is called embprep (or embprepp for C++), and contains the precompile and bind steps for these embedded SQL programs. Making this a separate file which is called for each embedded SQL program saves repeating these steps within the body of the makefile itself. This file needs the user ID, password, and database name to connect to the database on the server. The makefile passes these values to embprep, or embprepp, when it calls it.

The database variable, DB, is hardcoded by default to the sample database, and can be changed by the user if another database is used. The user ID and password variables, UID and PWD, are not set to any value by default. These optional parameters do not need to be used if the user is already working in the same instance as the server database. However, if this is not the case, for example, if the user is remotely connecting to the server from a client machine, then he or she can modify the makefile by giving the appropriate values to the UID and PWD variables, and they will be automatically passed to the precompile and bind file. The following is an example of the embprep file being called by the Micro Focus makefile on Solaris to build the embedded SQL application, updat:

```
updat.cbl : updat.sqb;
     embprep updat $(DB) $(UID) $(PWD)
updat.o : updat.cbl checkerr.o
     $(CC) updat.cbl ;
updat : updat.o checkerr.o;
     $(LINK)  updat.o $(LINKOBJS)
```

As is the case with the DB2 CLI stored procedure example above, the makefile variables, contained within the ″$(″ and ″)″ characters, are defined at the beginning of the makefile.

Error-checking Utilities

DB2 provides a utility file consisting of functions for error-checking and printing out error information. A version of this file is provided for each language in the samples directory. When used with an application program, the utility provides helpful error information, and makes debugging a DB2 program much easier. Most of the error-checking utilities use the DB2 APIs GET SQLSTATE MESSAGE and GETERROR MESSAGE to obtain pertinant SQLSTATE and SQLCA information related to problems encountered in program execution. The DB2 CLI utility, samputil, does not use these DB2 APIs; instead it uses equivalent DB2 CLI statements. With all the error-checking utilities, descriptive error messages are printed out to allow the developer to quickly understand the problem.

Some DB2 programs, such as stored procedures and user-defined functions, do not need to use the utilities. For Java, an error-checking utility file is not used. It is unnecessary because the `SQLException` object will be thrown if an exception occurs.

Here are the error-checking utility files used by DB2-supported compilers for the different programming languages:

checkerr.cbl
: For COBOL sample programs

samputil.c
: For CLI sample programs

util.c For C sample programs

util.C For C++ sample programs

util.f For Fortran sample programs (UNIX)

util.for
: For Fortran sample programs (OS/2)

In order to use the utility functions, the utility file must first be compiled, and then its object file linked in during the creation of the target program's executable. Both the `makefile` and build files in the `samples` directories do this for the programs that require the error-checking utility. Here are the object files created for the different programming languages:

checkerr.o
: For COBOL sample programs

samputil.o
: For CLI sample programs

util.o For C, C++ and Fortran sample programs

The following example demonstrates how the error-checking utilities are used in DB2 programs. The `util.c` source file defines the `check_error` function. This function checks the SQLCODE flag and prints out any information that is available related to the specific error indicated by this flag. A C program that uses the `util` program, such as `updat` in the example below, defines a macro for this function as follows:

```
#define  CHECKERR(CE_STR)   if (check_error (CE_STR, &sqlca) != 0) return 1;
```

and then calls the macro with the appropriate string argument for each database command that is executed:

```
   strcpy (jobUpdate, "Clerk");
   EXEC SQL UPDATE staff SET job = :jobUpdate WHERE job = 'Mgr';
   CHECKERR ("UPDATE STAFF");
```

If the statement fails, then the CHECKERR macro ensures an appropriate error message is printed out for the user.

Developers are encouraged to use and build upon these error-checking utilities when creating their own DB2 programs.

Java Applets and Applications

To build Java applets and applications, you follow the same steps on all platforms, so there is one chapter for this information, “Chapter 4. Building Java Applets and Applications” on page 57. There is specific set up information needed for each platform and this is given in separate sections in this chapter. This setup information is in addition to the DB2 setup information given in “Chapter 2. Setup” on page 33.

The Java chapter explains how to build JDBC programs that use the JDBC driver, and SQLJ programs that use embedded SQL for Java support in addition to using the JDBC driver. The chapter explains how to build JDBC and SQLJ applets, applications and stored procedures. It also explains how to build Java user-defined functions, which cannot contain JDBC or SQLJ statements.

The `samples` directory contains a Java `makefile` and build files for the SQLJ programs. JDBC programs are easy to build on the command line so no build files are supplied for these.

DB2 API Applications

Each DB2 SDK includes sample programs that call DB2 APIs. The ″Building Applications″ Chapters later in the book explain how to build the sample programs for the supported compilers using DB2 API build files supplied with the DB2 SDK for that platform. You can also use the makefiles that are supplied. Both the makefiles and the build 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 following sample program is used in this book to demonstrate the steps for building and running DB2 API applications using the supported programming languages. The steps you follow may vary, depending on your environment:

`client` demonstrates the use of the following APIs: `SET CLIENT`, and `QUERY CLIENT`

For a detailed description of all DB2 API sample programs, see "Sample Program Tables" on page 11.

The source files for DB2 API sample programs, where supported, are in the appropriate programming language subdirectory of `sqllib/samples` (UNIX), and `%DB2PATH%\samples` (OS/2 and Windows 32-bit operating systems).

After you build the sample programs, they can be used as templates to create your own applications. You can build the DB2 API programs using either the makefile or the build files provided.

DB2 Call Level Interface (CLI) Applications

The DB2 SDK comes with sample programs that use DB2 Call Level Interface (DB2 CLI) function calls. You can study the samples to learn how to access DB2 databases using these function calls in your applications.

The sample programs, the build files, and a `makefile`, are contained in the directory `sqllib/samples/cli` on UNIX, or `%DB2PATH%\samples\cli` on OS/2 and Windows 32-bit operating systems. You may need to modify the compiler options in the build files and the `makefile` for your environment.

The following are the sample programs used in this book to demonstrate the steps for building and running DB2 CLI applications (the steps you follow may vary, depending on your environment):

`basiccon`
establishes a basic connection with the database.

`outcli2`
is the client program of a client/server example; the server program is `outsrv2`.

`outsrv2`
is the server program of a client/server example; the client program is `outcli2`.

`calludf`
uses the functions created by the user-defined function program, `udf`.

For a more detailed description of all the DB2 CLI sample programs, see Table 8 on page 22. The *CLI Guide and Reference* explains how the samples using DB2 CLI work.

Embedded SQL Applications

Note: Embedded SQL for Java (SQLJ) is not discussed here, but is fully discussed in “Chapter 4. Building Java Applets and Applications” on page 57.

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

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

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

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

`updat` uses static SQL to update a database.

`outcli` is the client program of a client/server example; the server program is `outsrv`.

`outsrv` is the server program of a client/server example; the client program is `outcli`.

`calludf`
uses the functions created by the user-defined function program, `udf`.

For a more detailed description of these sample programs, see “Sample Program Tables” on page 11.

The source files for these sample programs, where supported, are in the appropriate programming language subdirectory of `sqllib/samples` on UNIX, and `%DB2PATH%\samples` on OS/2 and Windows 32-bit operating systems.

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

“Sample Programs” on page 11 lists all of the sample programs. The *Application Development Guide* explains how the samples containing embedded SQL work.

User-Defined Functions (UDFs)

User-defined functions allow you to write your own extensions of SQL for your own requirements. Like stored procedures, user-defined functions are stored on the server to be accessed by client applications. UDFs do not contain embedded SQL statements.

The following sample programs are used in this book to demonstrate the steps for building and storing a UDF library on the server:

`udf` creates a library of user-defined functions (UDFs); `calludf` is the client application that calls these functions.

`UDFsrv` creates a library of user-defined functions (UDFs) for Java only; `UDFcli`, and `UDFclie` are the JDBC and SQLJ client applications, respectively, that call these functions.

Multi-threaded Applications

DB2 supports C and C++ multi-threaded applications. These applications allow the user to have several simultaneous processes operating, to handle asynchronous events, and to create event-driven applications without resorting to polling schemes. The following sample program is used to demonstrate building a DB2 multi-threaded application on all supported UNIX platforms except HP-UX 10 and Linux:

`thdsrver`
demonstrates thread creation and management; only available in the C and C++ languages.

C++ Considerations for UDFs and Stored Procedures

Function names can be ’overloaded’ in C++. Two functions with the same name can coexist if they have different arguments, as in:

```
int func( int i )
```

and

```
int func( char c )
```

C++ compilers ’type-decorate’ or ’mangle’ function names by default. This means that argument type names are appended to their function names to resolve them, as in `func__Fi` and `func__Fc` for the two earlier examples. The mangled names will be different on each platform, so code that explicitly uses a mangled name is not portable.

The type-decorated function name can be determined from the `.o` file (object file) or the shared library using the `nm` command. This command can produce considerable output, so we suggest you pipe the output through `grep` to look for the right line, as follows:

```
nm myprog.o | grep myfunc
```

where `myprog.o` is your program object file, and `myfunc` is the function in the program source file.

The output produced by the command includes a line similar to the following:

```
myfunc__FPlT1PsT3PcN35|     3792|unamex|         | ...
```

When registering such a UDF with CREATE FUNCTION, the EXTERNAL NAME clause must specify the mangled function name obtained from `nm`:

```
CREATE FUNCTION myfunco(...) RETURNS...
       ...
       EXTERNAL NAME '/whatever/path/myprog!myfunc__FPlT1PsT3PcN35'
       ...
```

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

```
CALL 'myprog!myfunc__FPlT1PsT3PcN35' ( ... )
```

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

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

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

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

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

Chapter 4. Building Java Applets and Applications

You can develop Java programs to access DB2 databases with the appropriate Java Development Kit (JDK) for your platform. The JDK includes Java Database Connectivity (JDBC), a dynamic SQL API for Java.

DB2 JDBC support is provided as part of the Java Enablement option on DB2 clients and servers. With this support, you can build and run JDBC applications and applets. These contain dynamic SQL only, and use a Java call interface to pass SQL statements to DB2.

DB2 Java embedded SQL (SQLJ) support is provided as part of the DB2 Software Developer's Kit (DB2 SDK). With DB2 SQLJ support, in addition to DB2 JDBC support, you can build and run SQLJ applets and applications. These contain static SQL and use embedded SQL statements that are bound to a DB2 database.

The SQLJ support provided by the DB2 SDK includes:

- The DB2 SQLJ translator, `sqlj`, which replaces embedded SQL statements in the SQLJ program with Java source statements, and generates a serialized profile which contains information about the SQL operations found in the SQLJ program.
- The DB2 SQLJ profile customizer, `db2profc`, which precompiles the SQL statements stored in the serialized profile, customizes them into runtime function calls, and generates a package in the DB2 database. For more information on the `db2profc` command, see the *Command Reference*.

- The DB2 SQLJ profile printer, `db2profp`, which prints the contents of a DB2 customized version of a profile in plain text.

For more information on DB2 programming in Java, refer to the ″Programming in Java″ chapter in the *Application Development Guide*.

For the latest, updated DB2 Java information, visit the Web Page at:

```
http://www.software.ibm.com/data/db2/java
```

Setting the Environment

AIX

To build Java applications on AIX with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The Java Development Kit (JDK) Version 1.1.2 or later for AIX from IBM (refer to `http://www.software.ibm.com/data/db2/java`).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 6.1 for AIX clients and servers. For previous versions of DB2 for AIX, this support has been provided by the DB2 Client Application Enabler for AIX, Version 2.1.2 or later.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where `/home/db2inst/jdk11` is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the `JDK11_PATH` field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to redirect the output to a file for easier viewing. The `JDK11_PATH` field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run JDBC and SQLJ programs on AIX with DB2 JDBC support, commands to update the AIX Java environment are included in the database manager files `db2profile` and `db2cshrc`. When a DB2 instance is created, `.profile` and/or `.cshrc` are modified so that:

1. LD_LIBRARY_PATH includes `sqllib/lib`.

2. CLASSPATH includes:
   - "." (the current directory)
   - the file `sqllib/java/db2java.zip`

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

HP-UX

To build Java applications on HP-UX with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The HP-UX Developer's Kit for Java Release 1.1.3 or later from Hewlett-Packard (refer to `http://www.software.ibm.com/data/db2/java`).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 6.1 for HP-UX clients and servers. For previous versions of DB2 for HP-UX, this support has been provided by the DB2 Client Application Enabler for HP-UX, Version 2.1.2 or later.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where `/home/db2inst/jdk11` is the path where the JDK is installed.

Note: Java Stored Procedures and UDFs are only available on HP-UX Version 11

You can check the DB2 database manager configuration to verify the correct value for the `JDK11_PATH` field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to redirect the output to a file for easier viewing. The `JDK11_PATH` field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run JDBC and SQLJ programs on HP-UX with DB2 JDBC support, commands to update the HP-UX Java environment are included in the database manager files `db2profile` and `db2cshrc`. When a DB2 instance is created, `.profile` and/or `.cshrc` are modified so that:

1. THREADS_FLAG is set to ″native″.
2. CLASSPATH includes:
   - ″.″ (the current directory)
   - the file `sqllib/java/db2java.zip`

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

Linux

To build Java applications on Linux with DB2 JDBC support, you need to install and configure the following on your development machine:

1. Java Development Kit (JDK) 1.1.7, Version 1a or later for Linux from the Blackdown Organization. (see `http://www.software.ibm.com/data/db2/java`).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 6.1 for Linux clients and servers. For DB2 for Linux Version 5.2 (beta), this support has been provided with the DB2 Client Application Enabler for Linux, Version 5.2.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where `/home/db2inst/jdk11` is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the `JDK11_PATH` field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to redirect the output to a file for easier viewing. The `JDK11_PATH` field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

Note: On Linux, the Java Virtual Machine implementation does not work well in programs that run in a ″setuid″ environment. The shared library that contains the Java interpreter, `libjava.so`, may fail to load. As a workaround, you can create a symbolic link to the JVM shared library in /usr/lib, with a command similar to the following (depending on where Java is installed on your machine):

```
ln -s /usr/local/jdk117_v1a/lib/i686/native_threads/libjava.so /usr/lib
```

For more information on this and other workarounds available, please visit:

```
http://www.software.ibm.com/data/db2/java/faq.html
```

To run JDBC and SQLJ programs on Linux with DB2 JDBC support, commands to update your Linux Java environment are included in the database manager files db2profile and db2cshrc. When a DB2 instance is created, .bashrc, .profile, and/or .cshrc are modified so that:

1. THREADS_FLAG is set to ″native″.
2. CLASSPATH includes:
   - ″.″ (the current directory)
   - the file sqllib/java/db2java.zip

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

OS/2

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 1.1.4 for OS/2 from IBM (refer to http://www.software.ibm.com/data/db2/java).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 6.1 for OS/2 clients and servers. For previous versions of DB2 for OS/2, this support has been provided by the DB2 Client Application Enabler for OS/2, Version 2.1.2 or later.

The JDK must be installed in an HPFS drive to allow long filenames with extensions greater than three characters, such as .java. Your Java working directory must also be on an HPFS drive. If you will be running Java stored procedures or UDFs on an OS/2 server, DB2 must be installed on an HPFS drive on the server in order to allow the stored procedure or UDF .class files to be placed in the %DB2PATH%\function directory.

To run DB2 Java stored procedures or UDFs, you need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH c:\jdk11
```

where `c:\jdk11` is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the `JDK11_PATH` field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to redirect the output to a file for easier viewing. The `JDK11_PATH` field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run JDBC and SQLJ programs on OS/2 with DB2 JDBC support, the CLASSPATH environment variable is automatically updated when DB2 is installed to include:

- ″.″ (the current directory)
- the file `%DB2PATH%\java\db2java.zip`

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
%DB2PATH%\java\sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
%DB2PATH%\java\runtime.zip
```

Silicon Graphics IRIX

To build Java applications on Silicon Graphics IRIX with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The Java Development Environment 3.1 (Sun JDK 1.1.5) and the Java Execution Environment 3.1 (Sun JRE 1.1.5) from Silicon Graphics, Inc. (refer to `http://www.software.ibm.com/data/db2/java`).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 6.1 for Silicon Graphics IRIX clients. For previous DB2 for Silicon Graphics IRIX clients, this support has been provided by the DB2 Client Application Enabler for Silicon Graphics IRIX, Version 2.1.2 or later.

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

Also, since you will be accessing a database on the server from a remote client that is running on a different operating system, you need to bind the database utilities, including the DB2 CLI, to the database. See “Binding” on page 41 for more information.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where `/home/db2inst/jdk11` is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the `JDK11_PATH` field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to redirect the output to a file for easier viewing. The `JDK11_PATH` field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run JDBC and SQLJ programs on Silicon Graphics IRIX with DB2 JDBC support, commands to update the Silicon Graphics IRIX Java environment are included in the database manager files `db2profile` and `db2cshrc`. When a DB2 instance is created, `.profile` and/or `.cshrc` are modified so that CLASSPATH includes:

- ″.″ (the current directory)
- the file `sqllib/java/db2java.zip`

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

Note: On Silicon Graphics IRIX, the connection context `close()` method may cause a trap. As a work-around, leave the connection context to be closed automatically during garbage collection.

Solaris

To build Java applications on Solaris with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The Java Development Kit (JDK) Version 1.1.4 or later for Solaris, and the Solaris Native Thread pack, from Sun Microsystems (refer to `http://www.software.ibm.com/data/db2/java`).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 6.1 for Solaris clients and servers. For previous versions of DB2 for Solaris, this support has been provided by the DB2 Client Application Enabler for Solaris, Version 2.1.2 or later.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH /home/db2inst/jdk11
```

where `/home/db2inst/jdk11` is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the `JDK11_PATH` field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to redirect the output to a file for easier viewing. The `JDK11_PATH` field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

Note: On Solaris, some Java Virtual Machine implementations do not work well in programs that run in a ″setuid″ environment. The shared library that contains the Java interpreter, `libjava.so`, may fail to load. As a workaround, you can create symbolic links for all needed JVM shared libraries in `/usr/lib`, with a command similar to the following (and depending on where Java is installed on your machine):

```
ln -s /opt/jdk1.1.4/lib/sparc/native_threads/*.so /usr/lib
```

For more information on this and other workarounds available, please visit:

```
http://www.software.ibm.com/data/db2/java/faq.html
```

To run JDBC and SQLJ programs on Solaris with DB2 JDBC support, commands to update the Solaris Java environment are included in the database manager files `db2profile` and `db2cshrc`. When a DB2 instance is created, `.profile` and/or `.cshrc` are modified so that:

1. THREADS_FLAG is set to ″native″.
2. CLASSPATH includes:
   - ″.″ (the current directory)
   - the file `sqllib/java/db2java.zip`

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
sqllib/java/runtime.zip
```

Windows 32-bit Operating Systems

To build Java applications on a Windows 32-bit platform with DB2 JDBC support, you need to install and configure the following on your development machine:

1. One of the following:
   - The Java Development Kit (JDK) 1.1.7 and Java Runtime Environment (JRE) 1.1.7 for Win32 from IBM. These are shipped with DB2 (refer to `http://www.software.ibm.com/data/db2/java`).
   - The Java Development Kit (JDK) Version 1.1 or later for Win32 from Sun Microsystems (refer to `http://www.software.ibm.com/data/db2/java`).
   - Microsoft Software Developer's Kit for Java, Version 3.1 (refer to `http://www.software.ibm.com/data/db2/java`).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 6.1 for Windows 32-bit operating systems clients and servers. For previous versions of DB2 for Windows, this support has been provided by the DB2 Client Application Enabler for Windows NT and Windows 95, Version 2.1.2 or later.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH c:\jdk11
```

where `c:\jdk11` is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the `JDK11_PATH` field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to redirect the output to a file for easier viewing. The `JDK11_PATH` field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run JDBC and SQLJ programs on a supported Windows platform with DB2 JDBC support, CLASSPATH is automatically updated when DB2 is installed to include:

- ″.″ (the current directory)
- the file `%DB2PATH%\java\db2java.zip`

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
%DB2PATH%\java\sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
%DB2PATH%\java\runtime.zip
```

To specify which Java Development Kit to use for DB2 SQLJ, DB2 installs the environment variable DB2JVIEW on Windows 32-bit operating systems. It applies to all DB2 SQLJ commands (db2profc, db2profp, profdb, profp and sqlj).

If the default setting of ″DB2JVIEW=0″ is used, or DB2JVIEW is not set, the Sun JDK will be used; that is, if you call ″profp″, it will run as ″java sqlj.runtime.profile.util.ProfilePrinter″. If ″DB2JVIEW=1″, the Microsoft SDK for Java will be used; that is, if you call ″profp″, it will run as ″jview sqlj.runtime.profile.util.ProfilePrinter″.

Java Sample Programs

DB2 provides sample programs, used in the following sections, to demonstrate building and running JDBC programs that exclusively use dynamic SQL, and SQLJ programs that use static SQL. On UNIX platforms, the Java samples are located in `sqllib/samples/java`. On OS/2 and Windows 32-bit operating systems, the samples are located in `%DB2PATH%\samples\java`. The samples directory also contains a `README`, a `makefile`, and build files.

The Java `makefile` builds all the supplied sample programs. It requires a compatible make executable program, which is not normally provided by Java Development Kits. See the comment at the beginning of the text of the `makefile` for more information. Some of the Java `makefile` commands differ from other languages: the `make clean` command removes all files produced by the java compiler, such as `.class` files. It also removes `core` dumps. The `make cleanall` command removes these files as well as any files produced by the SQLJ translator.

The JDBC programs are relatively simple to build on the command line so no build files are included for them. There are two SQLJ build files provided: `bldsqlj` builds SQLJ applets and applications; `bldsqljs` builds SQLJ stored procedures. These build files are demonstrated in the section “SQLJ Programs” on page 69.

OS/2

On OS/2, your working directory must be on an HPFS drive. Since DB2 sample programs are provided on OS/2 to be compatible with FAT drives, filenames have at most a three character extension. To comply with this restriction, Java sample file extensions have been truncated. After copying the Java files to your working directory, you can rename the files with the following commands:

```
move *.jav *.java
move *.htm *.html
move *.sql *.sqlj
```

JDBC Programs

Applets

`DB2Applt` demonstrates a dynamic SQL Java applet using the JDBC applet driver to access a DB2 database.

To build and run this applet by commands entered at the command line:

1. Ensure that a web server is installed and running on your DB2 machine (server or client).
2. Modify the `DB2Applt.html` file according to the instructions in the file.
3. Start the JDBC applet server on the TCP/IP port specified in `DB2Applt.html`; for example, if in `DB2Applt.html`, you specified `param name=port value='6789'`, then you would enter:
   ```
   db2jstrt 6789
   ```
4. Compile `DB2Applt.java` to produce the file `DB2Applt.class` with this command:
   ```
   javac DB2Applt.java
   ```
5. Ensure that your working directory is accessible by your web browser. If it is not, copy `DB2Applt.class` and `DB2Applt.html` into a directory that is accessible.

6. Copy the file `%DB2PATH%\java\db2java.zip` on OS/2 or Windows 32-bit operating systems, or `sqllib/java/db2java.zip` on UNIX, into the same directory as `DB2Applt.class` and `DB2Applt.html`.
7. On your client machine, start your web browser (which must support JDK 1.1) and load `DB2Applt.html`.

As an alternative to steps (1), (5) and (7), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer DB2Applt.html
```

You can also use the Java `makefile` to build this program.

Applications

`DB2Appl` demonstrates a dynamic SQL Java application using the JDBC Application driver to access a DB2 database.

To build and run this application by commands entered at the command line:

1. Compile `DB2Appl.java` to produce the file `DB2Appl.class` with this command:
   ```
   javac DB2Appl.java
   ```
2. Run the java interpreter on the application with this command:
   ```
   java DB2Appl
   ```

You can also use the Java `makefile` to build this program.

Client Applications for Stored Procedures

`DB2SpCli` is a client application that calls the Java stored procedure, `DB2Stp`, using the JDBC Application driver. Before building and running this client application, build the stored procedure on the server. See “Stored Procedures” on page 69.

To build and run this client program by commands entered at the command line:

1. Compile `DB2SpCli.java` to produce the file `DB2SpCli.class` with this command:
   ```
   javac DB2SpCli.java
   ```
2. Run the java interpreter on the client program with this command:
   ```
   java DB2SpCli
   ```

You can also use the Java `makefile` to build this program.

Client Applications for User-Defined Functions

UDFcli is the client program that calls the user-defined functions implemented in the user-defined function server program, UDFsrv, using the JDBC application driver. Before building and running this client application, build the user-defined function program, UDFsrv, on the server. See "User-Defined Functions (UDFs)" on page 78.

To build and run this client program by commands entered at the command line:

1. Compile UDFcli.java to produce the file UDFcli.class with this command:

   ```
   javac UDFcli.java
   ```
2. Run the java interpreter on the client program with this command:
   ```
   java UDFcli
   ```

You can also use the Java makefile to build this program.

Stored Procedures

DB2Stp demonstrates a dynamic SQL Java stored procedure using the JDBC Application driver. Stored procedures are compiled and stored on a server. When called by a client application, they access the server database and return information to the client application.

To build and run this stored procedure on the server by commands entered at the command line:

1. Compile DB2Stp.java to produce the file DB2Stp.class with this command:
   ```
   javac DB2Stp.java
   ```
2. Copy DB2Stp.class to the %DB2PATH%\function directory on OS/2 and Windows 32-bit operating systems, or to the sqllib/function directory on UNIX.
3. Compile and run the DB2SpCli client application to access the stored procedure. See "Client Applications for Stored Procedures" on page 68.

You can also use the Java makefile to build this program.

SQLJ Programs

The build file, bldsqlj, contains the commands to build an SQLJ applet or application. On UNIX this is a script file. On OS/2, it is the command file, bldsqlj.cmd, and on Windows, it is the batch file, bldsqlj.bat. The contents of the command and batch files are the same, and this version is presented first, followed by the UNIX script file. The applet and application building sections that follow will refer back to these build files.

Note: The SQLJ translator shipped with DB2 compiles the translated `.java` files into `.class` files. Therefore, the build files in this section do not use the java compiler.

In the following build file for OS/2 and Windows 32-bit operating systems, 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 Builds a Java embedded SQL (SQLJ) program.
rem Usage: bldsqlj prog_name [ db_name [ userid password ]]

if "%1" == "" goto error

rem Translate and compile the SQLJ source file.
sqlj %1.sqlj

rem Bind the package to the database.
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2profc -url=jdbc:db2:sample -prepoptions="package using %1" %1_SJProfile0
   goto continue
:case2
   db2profc -url=jdbc:db2:%2 -prepoptions="package using %1" %1_SJProfile0
   goto continue
:case3
   db2profc -url=jdbc:db2:%2 -user=%3 -password=%4 -prepoptions="package using %1"
     %1_SJProfile0
   goto continue
:continue

goto exit

:error
echo Usage: bldsqlj prog_name [ db_name [ userid password ]]

:exit
@echo on
```

Translator and Bind Options for `bldsqlj`

`sqlj`	The SQLJ translator (also compiles the program).
`%1.sqlj`	The SQLJ source file.
`%1.java`	The translated Java file from the SQLJ source file.
`db2profc`	The DB2 for Java profile customizer.
`-url`	Specifies a JDBC URL for establishing a database connection, such as `jdbc:db2:sample`.
`-user`	Specifies a user ID (optional parameter).
`-password`	Specifies a password (optional parameter).
`-prepoptions`	Specifies the package name for the database with the string "`package using %1`", where `%1` is the SQLJ source file name.
`%1_SJProfile0`	Specifies a profile for the program.

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

```
#! /bin/ksh
# bldsqlj script file
# Builds a Java embedded SQL (SQLJ) sample
# Usage: bldsqlj <prog_name> [ <db_name> [ <userid> <password> ]]

# Translate and compile the SQLJ source file.
sqlj $1.sqlj

# Bind the package to the database.
if (($# < 2))
then
   db2profc -url=jdbc:db2:sample -prepoptions="package using $1" $1_SJProfile0
elif (($# < 3))
then
   db2profc -url=jdbc:db2:$2 -prepoptions="package using $1" $1_SJProfile0
else
   db2profc -url=jdbc:db2:$2 -user=$3 -password=$4 -prepoptions="package using $1"
     $1_SJProfile0
fi
```

Translator and Bind Options for `bldsqlj`

`sqlj`	The SQLJ translator (also compiles the program).
`$1.sqlj`	The SQLJ source file.
`$1.java`	The translated Java file from the SQLJ source file.
`db2profc`	The DB2 for Java profile customizer.
`-url`	Specifies a JDBC URL for establishing a database connection, such as `jdbc:db2:sample`.
`-user`	Specifies a user ID (optional parameter).
`-password`	Specifies a password (optional parameter).
`-prepoptions`	Specifies the package name for the database with the string "`package using $1`", where $1 is the SQLJ source file name.
`$1_SJProfile0`	Specifies a profile for the program.

Applets

`Applt` demonstrates an SQLJ applet that accesses a DB2 database.

To build this applet with the build file, `bldsqlj`, and then run it:

1. Ensure that a web server is installed and running on your DB2 machine (server or client).
2. Modify the Applt.html file according to the instructions in the file.
3. Start the JDBC applet server on the TCP/IP port specified in Applt.html. For example, if in Applt.html, you specified param name=port value='6789', then you would enter:
   ```
   db2jstrt 6789
   ```
4. Build the applet with this command:
   ```
   bldsqlj Applt [ <db_name> [ <userid> <password> ]]
   ```
   where the optional parameter <db_name> allows you to access another database instead of the default sample database. The optional parameters <userid>, and <password> are needed if the database you are accessing is on a different instance, such as if you are accessing a server from a remote client machine.
5. Ensure that your working directory is accessible by your web browser. If it is not, copy the following files into a directory that is accessible:
   ```
   Applt.html,                        Applt.class,
   Applt_Cursor1.class,               Applt_Cursor2.class,
   Applt_SJProfileKeys.class,         Applt_SJProfile0.ser
   ```
6. Copy the files %DB2PATH%\java\db2java.zip and %DB2PATH%\java\runtime.zip on OS/2 and Windows 32-bit operating systems, or sqllib/java/db2java.zip and sqllib/java/runtime.zip on UNIX, into the same directory as your other Applt files.
7. On your client machine, start your web browser (which must support JDK 1.1) and load Applt.html.

As an alternative to steps (1), (5) and (7), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer Applt.html
```

You can also use the Java makefile to build this program.

Applications

App demonstrates an SQLJ application that accesses a DB2 database.

To build this application with the build file, bldsqlj, enter this command:

```
bldsqlj App [ <db_name> [ <userid> <password> ]]
```

where the optional parameter <db_name> allows you to access another database instead of the default sample database. The optional parameters

<userid>, and <password> are needed if the database you are accessing is on a different instance, such as if you are accessing a server from a remote client machine.

Run the Java interpreter on the application with this command:

```
java App
```

You can also use the Java makefile to build this program.

Client Programs for Stored Procedures

StpCli is the client program that calls the SQLJ stored procedure Stp using the JDBC Application driver. Before building and running this client application, build the stored procedure on the server. See “Stored Procedures” on page 75.

To build this client program with the build file bldsqlj, enter this command:

```
bldsqlj StpCli [ <db_name> [ <userid> <password> ]]
```

where the optional parameter <db_name> allows you to access another database instead of the default sample database. The optional parameters <userid>, and <password> are needed if the database you are accessing is on a different instance, such as if you are accessing a server from a remote client machine.

Run the Java interpreter on the client application with this command:

```
java StpCli
```

You can also use the Java makefile to build this program.

Client Programs for User-Defined Functions

UDFclie is the client program that calls the user-defined functions implemented in the server program, UDFsrv, using the JDBC application driver. Before building and running this client application, build the UDFsrv program on the server. See “User-Defined Functions (UDFs)” on page 78.

To build this SQLJ client program with the build file, bldsqlj, enter this command:

```
bldsqlj UDFclie [ <db_name> [ <userid> <password> ]]
```

where the optional parameter <db_name> allows you to access another database instead of the default sample database. The optional parameters

<userid>, and <password> are needed if the database you are accessing is on a different instance, such as if you are accessing a server from a remote client machine.

Run the Java interpreter on client application with this command:

```
   java UDFclie
```

You can also use the Java makefile to build this program.

Stored Procedures

The build file, bldsqljs, contains the commands to build an SQLJ stored procedure. On UNIX this is a script file. On OS/2, it is the command file, bldsqljs.cmd, and on Windows, it is a batch file, bldsqljs.bat. The contents of the command and batch files are the same, and this version is presented first, followed by the UNIX script file.

In the following build file for OS/2 and Windows 32-bit operating systems, 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 Builds a Java embedded SQL (SQLJ) stored procedure.
rem Usage: bldsqljs prog_name [ db_name [ userid password ]]

if "%1" == "" goto error

rem Translate and compile the SQLJ source file.
sqlj %1.sqlj

rem Bind the package to the database.
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   db2profc -url=jdbc:db2:sample -prepoptions="package using %1" %1_SJProfile0
   goto continue
:case2
   db2profc -url=jdbc:db2:%2 -prepoptions="package using %1" %1_SJProfile0
   goto continue
:case3
   db2profc -url=jdbc:db2:%2 -user=%3 -password=%4 -prepoptions="package using
     %1" %1_SJProfile0
   goto continue
:continue
```

```
rem Copy the *.class and *.ser files to the 'function' directory.
copy %1*.class %DB2PATH%\function
copy %1*.ser %DB2PATH%\function

goto exit

:error
echo Usage: bldsqljs prog_name [ db_name [ userid password ]]

:exit
@echo on
```

Translator and Bind Options for `bldsqljs`

`sqlj`	The SQLJ translator (also compiles the program).
`%1.sqlj`	The SQLJ source file.
`%1.java`	The translated Java file from the SQLJ source file.
`db2profc`	The DB2 for Java profile customizer.
`-url`	Specifies a JDBC URL for establishing a database connection, such as `jdbc:db2:sample`.
`-user`	Specifies a user ID (optional parameter).
`-password`	Specifies a password (optional parameter).
`-prepoptions`	Specifies the package name for the database with the string "`package using %1`", where `%1` is the SQLJ source file name.
`%1_SJProfile0`	Specifies a profile for the program.

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

```
#! /bin/ksh
# bldsqljs script file
# Builds a Java embedded SQL (SQLJ) stored procedure.
# Usage: bldsqljs <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Translate and compile the SQLJ source file.
sqlj $1.sqlj

# Bind the package to the database.
if (($# < 2))
then
   db2profc -url=jdbc:db2:sample -prepoptions="package using $1" $1_SJProfile0
elif (($# < 3))
then
   db2profc -url=jdbc:db2:$2 -prepoptions="package using $1" $1_SJProfile0
else
   db2profc -url=jdbc:db2:$2 -user=$3 -password=$4 -prepoptions="package using
     $1" $1_SJProfile0
fi

# Copy the *.class and *.ser files to the 'function' directory.
rm -f $DB2PATH/function/$1*.class
rm -f $DB2PATH/function/$1*.ser
cp $1*.class $DB2PATH/function
cp $1*.ser $DB2PATH/function
```

Translator and Bind Options for `bldsqljs`

sqlj	The SQLJ translator (also compiles the program).
$1.sqlj	The SQLJ source file.
$1.java	The translated Java file from the SQLJ source file.
db2profc	The DB2 for Java profile customizer.
-url	Specifies a JDBC URL for establishing a database connection, such as `jdbc:db2:sample`.
-user	Specifies a user ID (optional parameter).
-password	Specifies a password (optional parameter).
-prepoptions	Specifies the package name for the database with the string "`package using $1`", where $1 is the SQLJ source file name.
$1_SJProfile0	Specifies a profile for the program.

Stp demonstrates an SQLJ stored procedure using the JDBC Application driver to access a DB2 database. Stored procedures are compiled and stored on a server. When called by a client application, they access the server database and return information to the client application.

To build this stored procedure with the build file, `bldsqljs`, enter the following command:

```
bldsqljs Stp [ <db_name> [ <userid> <password> ]]
```

where the optional parameter `<db_name>` allows you to access another database instead of the default `sample` database. The optional parameters `<userid>`, and `<password>` are needed if the database you are accessing is on a different instance, such as if you are accessing a server from a remote client machine.

Compile and run the `StpCli` client application to call the stored procedure. See “Client Programs for Stored Procedures” on page 74.

You can also use the Java `makefile` to build this program.

User-Defined Functions (UDFs)

`UDFsrv` demonstrates Java user-defined functions. Since UDF programs do not contain SQL statements, a Java UDF program cannot contain SQLJ statements. Once the `UDFsrv` libary is created on the server, it may be accessed by a client application. DB2 provides both a JDBC client application, `UDFcli`, and an SQLJ client application, `UDFclie`. Either can be used to access the `UDFsrv` library.

To build and run the UDF program on the server by commands entered at the command line:

1. Compile `UDFsrv.java` to produce the file `UDFsrv.class` with this command:
   ```
   javac UDFsrv.java
   ```
2. Copy `UDFsrv.class` to the `%DB2PATH%\function` directory on OS/2 and Windows 32-bit operating systems, or to the `sqllib/function` directory on UNIX.
3. To access the `UDFsrv` library, you can use either JDBC or SQLJ client applications. To compile and run the JDBC client application, `UDFcli`, see “Client Applications for User-Defined Functions” on page 69. To compile and run the SQLJ client application, `UDFclie`, see “Client Programs for User-Defined Functions” on page 74.

`UDFcli` and `UDFclie` contain the `CREATE FUNCTION` SQL statement that you use to register the UDFs contained in `UDFsrv` with the database. `UDFcli` and `UDFclie` also contain SQL statements that make use of the UDFs, once they have been registered.

General Points for DB2 Java Applets

1. For a larger JDBC or SQLJ applet that consists of several Java classes, you may choose to package all its classes in a single JAR file. For an SQLJ applet, you would also have to package its serialized profiles along with its classes. If you choose to do this, add your JAR file into the `archive` parameter in the ″applet″ tag. For details, see the JDK Version 1.1 documentation.

   For SQLJ applets: some browsers do not yet have support for loading a serialized object from a resource file associated with the applet. For example, you will get the following error message when trying to load the applet `Applt` in those browsers:

   ```
   java.lang.ClassNotFoundException: Applt_SJProfile0
   ```

   As a work-around, there is a utility which converts a serialized profile into a profile stored in Java class format. The utility is a Java class called `sqlj.runtime.profile.util.SerProfileToClass`. It takes a serialized profile resource file as input and produces a Java class containing the profile as output. Your profile can be converted using one of the following commands:

   ```
   profconv Applt_SJProfile0.ser
   ```

   or

   ```
   java sqlj.runtime.profile.util.SerProfileToClass Applt_SJProfile0.ser
   ```

   The class `Applt_SJProfile0.class` is created as a result. Replace all profiles in `.ser` format used by the applet with profiles in `.class` format, and the problem should go away.
2. You may wish to place the file `db2java.zip` (and for SQLJ applets, also the file `runtime.zip`) into a directory that is shared by several applets that may be loaded from your Web site. These files are in the `%DB2PATH%\java` directory on OS/2 and Windows 32-bit operating systems, and in the `sqllib/java` directory on UNIX. You may need to add a `codebase` parameter into the ″applet″ tag in the HTML file to identify the directory. For details, see the JDK Version 1.1 documentation.
3. Since DB2 Version 5.2, signal handling has been added to the JDBC applet server (listener), `db2jd`, to make it more robust. As a result, one cannot use the CTRL-C command to kill `db2jd`. Therefore, the only way to terminate the listener is to kill the process.
4. For information on running DB2 Java applets on a webserver, specifically the Domino Go Webserver, see:

   ```
   http://www.software.ibm.com/data/db2/db2lotus/gojava.htm
   ```

Chapter 5. Building AIX Applications

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

For the latest DB2 application development updates for AIX, visit the Web page at:

```
http://www.software.ibm.com/data/db2/udb/ad
```

Important Considerations

This section gives AIX-specific information for building DB2 applications on various supported compilers. It includes:

- Installing and Running IBM and Micro Focus COBOL
- Entry Points for Stored Procedures and UDFs
- Stored Procedures and the CALL statement
- UDFs and the CREATE FUNCTION statement

Installing and Running IBM and Micro Focus COBOL

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

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

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

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

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

Entry Points for Stored Procedures and UDFs

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

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

```
-e funcname
```

makes `funcname` the default entry point. For information on how this relates to the CREATE FUNCTION statement, see "UDFs and the CREATE FUNCTION Statement" on page 85.

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

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

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

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

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

The AIX compiler documentation has additional information on export files.

Stored Procedures and the CALL Statement

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

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

- Using the `-e` option.

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

  ```
  -e modify
  ```

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

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

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

  The library `mystored` is loaded into memory, and the function `modify` is picked up by DB2 as the default entry point, and is executed.
- Using an export file specified using the `-bE:` option.

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

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

  In the link step, you specify:

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

  which identifies the export file `mystored.exp`.

  The export file looks like this:

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

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

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

and

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

UDFs and the CREATE FUNCTION Statement

The *Application Development Guide* describes how to code your UDF. The *SQL Reference* describes how to register your UDF with DB2 using the CREATE FUNCTION statement. This section explains the relation between compiling and linking your UDF and the information you provide in the EXTERNAL NAME clause of the CREATE FUNCTION statement.

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

- Using the -e option.

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

  ```
  -e modify
  ```

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

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

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

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

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

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

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

  In the link step, you specify:

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

  which identifies the export file myudfs.exp.

The export file looks like this:

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

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

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

and

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

IBM C

This section explains how to use IBM C with the following kinds of DB2 interfaces:

- DB2 APIs
- DB2 CLI
- Embedded SQL

DB2 API Applications

The script file `bldxlcapi` in `sqllib/samples/c` contains the commands to build a DB2 API program. The one parameter it takes, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldxlcapi script file
# Build sample programs that do not contain embedded SQL
# Usage: blxlcapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.c error-checking utility.
xlc -I$DB2PATH/include -c util.c
# Compile the program.
xlc -I$DB2PATH/include -c $1.c
# Link the program.
xlc -o $1 $1.o util.o -ldb2 -L$DB2PATH/lib
```

Compile and Link Options for bldxlc	
The script file contains the following compile options:	
`xlc`	The IBM C compiler.
`-I$DB2PATH/include`	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
The script file contains the following link options:	
`xlc`	Use the compiler as a front end for the linker.
`-o $1`	Specify the executable program.
`$1.o`	Specify the object file.
`util.o`	Include the utility object file for error checking.
`-ldb2`	Link to the database manager library.
`-L$DB2PATH/lib`	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
Refer to your compiler documentation for additional compiler options.	

To build the sample program `client` from the source file `client.c`, enter:

```
bldxlcapi client
```

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

```
client
```

DB2 CLI Applications

The script file `bldcli` in `sqllib/samples/cli` contains the commands to build a DB2 CLI program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldcli script file -- AIX
# Builds a CLI program with IBM C.
# Usage: bldcli <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
xlc -I$DB2PATH/include -c samputil.c

# Compile the program.
xlc -I$DB2PATH/include -c $1.c

# Link the program.
xlc -o $1 $1.o samputil.o -L$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcli

The script file contains the following compile options:

xlc The IBM C compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

xlc Use the compiler as a front end for the linker.

-o $1 Specify the executable program.

$1.o Specify the object file.

samputil.o
Include the utility object file for error checking.

-L$DB2PATH/lib
Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.

-ldb2 Link with the database manager library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `basiccon` from the source file `basiccon.c`, enter:

```
   bldcli basiccon
```

The result is an executable file basiccon. You can run the executable file by entering the executable name:

```
basiccon
```

DB2 CLI Stored Procedures

The script file bldclisp in sqllib/samples/cli contains the commands to build a DB2 CLI stored procedure. The parameter, $1, specifies the name of your source file.

The script file uses the source file name, $1, for the shared library name, and for the entry point to the shared library. If you are building stored procedures that have the entry point function name different from the source file name, you can modify the script file to accept a second parameter for the entry point by changing the link option to **-e** $2.

```
#! /bin/ksh
# bldclisp script file -- AIX
# Builds a CLI stored procedure in IBM C.
# Usage: bldclisp <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
xlc -I$DB2PATH/include -c samputil.c

# Compile the program.
xlc -I$DB2PATH/include -c $1.c

# Link the program.
xlc -o $1 $1.o samputil.o -L$DB2PATH/lib -ldb2 -lm -H512 -T512 -bE:$1.exp -e $1

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldclisp

The script file contains the following compile options:

xlc The IBM C compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: $HOME/sqllib/include.

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldclisp

The script file contains the following link options:

`xlc`	Use the compiler as a front end for the linker.
`-o $1`	Specify the executable program.
`$1.o`	Specify the object file.
`samputil.o`	Include the utility object file for error checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
`-ldb2`	Link with the database manager library.
`-H512`	Specify output file alignment.
`-T512`	Specify output file text segment starting address.
`-bE:$.exp`	Specify an export file. The export file contains a list of the stored procedures.
`-e $1`	Specify the default entry point to the shared library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `outsrv2` from source file `outsrv2.c`, enter:

```
bldclisp outsrv2
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide.*

Note: An unfenced stored procedure runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure, which runs in an address space isolated from the database manager. With unfenced stored procedures there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures when you need to maximize the

performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Application Development Guide* for more information.

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

Once you build the stored procedure outsrv2, you can build the CLI client application outcli2 that calls the stored procedure. You can build outcli2 by using the script file bldcli. Refer to “DB2 CLI Applications” on page 87 for details.

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

outcli2 *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 outsrv2, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

The script file bldxlc, in sqllib/samples/c, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
```

```
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
xlc -I$DB2PATH/include -c util.c

# Compile the program.
xlc -I$DB2PATH/include -c $1.c

# Link the program.
xlc -o $1 $1.o util.o -ldb2 -L$DB2PATH/lib
```

Compile and Link Options for bldxlc

The script file contains the following compile options:

`xlc` The IBM C compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldxlc	
The script file contains the following link options:	
xlc	Use the compiler as a front end for the linker.
-o $	Specify the executable program.
$1.o	Specify the program object file.
util.o	Include the utility object file for error checking.
-ldb2	Link to the database manager library.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
Refer to your compiler documentation for additional compiler options.	

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

```
bldxlc updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

The script file uses the source file name, $1, for the shared library name, and for the entry point to the shared library. If you are building stored procedures with the entry point function name different from the source file name, you can modify the script file to accept another parameter for the entry point. We recommend renaming the database parameter to $3, the user ID paramaeter to $4, and the password parameter to $5. Then you can change the entry point link option to `-e $2`, and specify the additional parameter on the command line when you run the script file.

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset
 # Compile the util.c error-checking utility.
xlc -I$DB2PATH/include -c util.c

# Compile the program.
xlc -I$DB2PATH/include -c $1.c

# Link the program using the export file $1.exp,
# creating shared library $1 with entry point $1.
xlc -o $1 $1.o util.o -ldb2 -L$DB2PATH/lib \
    -H512 -T512 -bE:$1.exp -e $1

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldxlcsrv

The script file contains the following compile options:

xlc	The IBM C compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

xlc	Use the compiler as a front end for the linker.
-o $1	Specify the output as a shared library file.
$1.o	Specify the stored procedure object file.
util.o	Include the utility object file for error checking.
-ldb2	Link with the database manager library.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
-H512	Specify output file alignment.
-T512	Specify output file text segment starting address.
-bE:$1.exp	Specify an export file. The export file contains a list of the stored procedures.
-e $1	Specify the default entry point to the shared library.

Refer to your compiler documentation for additional compiler options.

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

```
bldxlcsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide.*

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldxlc`. Refer to "IBM C" on page 86 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

User-Defined Functions (UDFs)

The script file `bldxlcudf`, in `sqllib/samples/c`, contains the commands to build a UDF. UDFs are compiled like stored procedures. They cannot contain SQL statements. This means to build a UDF program, you never need to connect to a database, precompile, and bind the program.

The first parameter, $1, specifies the name of your source file. The second parameter, $2, specifies the default entry point to the shared library. The script file uses the source file name, $1, for the shared library name.

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the program.
xlc -I$DB2PATH/include -c $1.c

# Link the program.
xlc -o $1 $1.o -ldb2 -ldb2apie -L$DB2PATH/lib -H512 -T512 -bE:$1.exp -e $2

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldxlcudf

The script file contains the following compile options:

`xlc` The IBM C compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldxlcudf

The script file contains the following link options:

xlc	Use the compiler as a front end for the linker.
-o $1	Specify the output as a shared library file.
$1.o	Specify the shared library object file.
-ldb2	Link with the database manager library.
-ldb2apie	Link with the DB2 API Engine library to allow the use of LOB locators.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
-H512	Specify output file alignment.
-T512	Specify output file text segment starting address.
-bE:$1.exp	Specify an export file. The export file contains a list of the UDFs.
-e $2	Specify the default entry point to the shared library.

Refer to your compiler documentation for additional compiler options. Refer to "UDFs and the CREATE FUNCTION Statement" on page 85 for more information on creating UDFs.

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

```
   bldxlcudf udf increase
```

The script file copies the UDF to the server in the path `sqllib/function`.

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

Once you build `udf`, you can build the client application, `calludf`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI `calludf` program from the source file `calludf.c`, in `sqllib/samples/cli`, using the script file `bldcli`. Refer to "DB2 CLI Applications" on page 87 for details.

You can build the embedded SQL `calludf` program from the source file `calludf.sqc`, in `sqllib/samples/c`, using the script file `bldxlc`. Refer to "IBM C" on page 86 for details.

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

```
calludf
```

The calling application calls functions from the udf library.

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

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

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

Multi-threaded Applications

C multi-threaded applications on AIX Version 4 need to be compiled and linked with the xlc_r compiler instead of the xlc compiler or, for C++, with the xlC_r compiler instead of the xlC compiler. If you are using AIX 4.3 or later, use the xlc_r7 or xlC_r7 compiler. The _r versions pass -D_REENTRANT to the C preprocessor and -lpthreads to the linker, so it is not necessary to do this explicitly. Please see the /etc/xlC.cfg file for more information.

The script file bldxlcmt, in sqllib/samples/c, contains the commands to build an embedded SQL multi-threaded program. If you want to build a DB2 API or DB2 CLI multi-threaded program, comment out the connect, precompile, bind, and disconnect commands. For DB2 CLI, also substitute the samputil.c and samputil.o files for util.c and util.o.

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

```
#! /bin/ksh
# bldxlcmt script file -- AIX
# Builds an embedded SQL multi-threaded program with IBM C.
# Usage: bldxlcmt <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$HOME/sqllib

# Connect to a database.
if (($# < 2))
then
  db2 connect to sample
elif (($# < 3))
```

```
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

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

# Compile the util.c error-checking utility.
xlc_r -I$DB2PATH/include -c util.c
# Compile the program.
xlc_r -I$DB2PATH/include -c $1.c
# Link the program
xlc_r -o $1 $1.o util.o -L$DB2PATH/lib -ldb2
```

Besides the `xlc_r` compiler, discussed above, the other compile and link options are the same as those used for the embedded SQL script file, `bldxlc`. For information on these options, see “Embedded SQL Applications” on page 91.

To build the multi-threaded sample program, `thdsrver`, from the source file `thdsrver.sqc`, enter:

```
   bldxlcmt thdsrver
```

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

```
   thdsrver
```

IBM C Set++

This section contains the following topics:
- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)
- Multi-threaded Applications

DB2 API Applications

The script file `bldcsetapi` in `sqllib/samples/c` contains the commands to build a DB2 API program. The one parameter it takes, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldcsetapi script file
# Builds a C++ DB2 API program that does not contain embedded SQL.
# Usage: bldcsetapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.C error-checking utility.
xlC -I$DB2PATH/include -c util.C
# Compile the program.
xlC -I$DB2PATH/include -c $1.C
# Link the program.
xlC -o $1 $1.o util.o -ldb2 -L$DB2PATH/lib
```

Compile and Link Options for bldcsetapi

The script file contains the following compile options:

xlC	The IBM C Set ++ compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

xlC	Use the compiler as a front end for the linker.
-o $1	Specify the executable program.
-o $1	Specify the program object file.
util.o	Include the utility object file for error checking.
-ldb2	Link with the database manager library.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path `/usr/lib:/lib`.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.C`, enter:

```
   bldcsetapi client
```

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

```
   client
```

Embedded SQL Applications

The script file `bldcset`, in `sqllib/samples/cpp`, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
xlC -I$DB2PATH/include -c util.C

# Compile the program.
xlC -I$DB2PATH/include -c $1.C
```

```
# Link the program.
xlC -o $1 $1.o util.o -ldb2 -L$DB2PATH/lib
```

Compile and Link Options for bldcset

The script file contains the following compile options:

xlC	The IBM C Set++ compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

xlC	Use the compiler as a front end for the linker.
-o $1	Specify the executable program.
$1.o	Specify the program object file.
util.o	Include the utility object file for error checking.
-ldb2	Link with the database manager library.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.

Refer to your compiler documentation for additional compiler options.

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

```
bldcset updat
```

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

```
updat
```

Embedded SQL Stored Procedures

Note: Please see the information for building C++ stored procedures and UDFs in “C++ Considerations for UDFs and Stored Procedures” on page 55.

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

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

# Compile the program.
xlC -I$DB2PATH/include -c $1.C

# Link the program using the export file $1.exp,
# creating a shared library called $1
makeC++SharedLib -p 1024 -o $1 $1.o -L$DB2PATH/lib -ldb2 -E $1.exp

# Copy the shared library to the DB2 instance sqllib/function subdirectory.
# Note: this assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldcsetsrv

The script file contains the following compile options:

xlC	The IBM C Set++ compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

makeC++SharedLib	Linker script for stored procedures with static constructors.
-p 1024	Set the priority to the arbitrary value of 1024.
-o $1	Specify the output as a shared library file.
$1.o	Specify the program object file.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
-ldb2	Link with the database manager library.
-E $1.exp	Specify an export file. The export file contains a list of the stored procedures.

Refer to your compiler documentation for additional compiler options.

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

```
bldcsetsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the "Stored Procedures" chapter in the *Application Development Guide.*

Note: An unfenced stored procedure runs in the same address space as the database manager and results in increased performance when

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldcset`. Refer to “Embedded SQL Applications” on page 102 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

User-Defined Functions (UDFs)

Note: Please see the information for building C++ UDFs and stored procedures in “C++ Considerations for UDFs and Stored Procedures” on page 55.

The script file `bldxlCudf`, in `sqllib/samples/cpp`, contains the commands to build a UDF. UDFs cannot contain embedded SQL statements. Therefore, to build a UDF program, you never need to connect to a database or precompile and bind the program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the program. The extension ".c" is for a C source file.
# Change the extension to ".C" if compiling a C++ source file.
xlC -I$DB2PATH/include -c $1.c

# Link the program.
makeC++SharedLib -p 1024  -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2apie -E $1.exp

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldxlCudf

The script file contains the following compile options:

xlC
The IBM C Set++ compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

-c
Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldxlCudf
The script file contains the following link options:
makeC++SharedLib Linker script for stored procedures with static constructors.
-p 1024 Set the priority to the arbitrary value of 1024.
-o $1 Specify the output as a shared library file.
$1.o Specify the program object file.
-L$DB2PATH/lib Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
-ldb2 Link with the database manager library.
-ldb2apie Link with the DB2 API Engine library to allow the use of LOB locators.
-E $1.exp Specify an export file. The export file contains a list of the stored procedures.
Refer to your compiler documentation for additional compiler options. Refer to "UDFs and the CREATE FUNCTION Statement" on page 85 for more information on creating UDFs.

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

```
bldxlCudf udf
```

The script file copies the UDF to the server in the path `sqllib/function`.

Note: If you wish to build a C++ UDF program that has a `.C` extension, you must modify the script file `bldxlCudf` to accept programs with this extension.

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

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build the `calludf` program from the `calludf.sqC` source file in `sqllib/samples/cpp` using the script file `bldcset`. Refer to "Embedded SQL Applications" on page 102 for details.

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

```
calludf
```

The calling application calls functions from the udf library.

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

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

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

Multi-threaded Applications

C++ multi-threaded applications on AIX Version 4 need to be compiled and linked with the xlC_r compiler instead of the xlC compiler or, for C, with the xlc_r compiler instead of the xlc compiler. If you are using AIX 4.3 or later, use the xlC_r7 or xlc_r7 compiler. The _r versions pass -D_REENTRANT to the C preprocessor and -lpthreads to the linker, so it is not necessary to do this explicitly. Please see the /etc/xlC.cfg file for more information.

The script file bldxlCmt, in sqllib/samples/cpp, contains the commands to build an embedded SQL multi-threaded program. If you want to build a DB2 API multi-threaded program, comment out the connect, precompile, bind, and disconnect commands.

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

```
#! /bin/ksh
# bldxlCmt script file -- AIX
# Builds a multi-threaded program containing embedded SQL
# with IBM C++.
# Usage: bldxlCmt <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$HOME/sqllib

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

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

# Compile the util.C error-checking utility.
xlC_r -I$DB2PATH/include -c util.C
# Compile the program.
xlC_r -I$DB2PATH/include -c $1.C
# Link the program
xlC_r -o $1 $1.o util.o -L$DB2PATH/lib -ldb2
```

Besides the `xlC_r` compiler, discussed above, the other compile and link options are the same as those used in the embedded SQL script file, `bldxlC`. For information on these options, see "Embedded SQL Applications" on page 102.

To build the multi-threaded sample program, `thdsrver`, from the source file `thdsrver.sqC`, enter:

```
bldxlCmt thdsrver
```

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

```
thdsrver
```

VisualAge C++

The VisualAge C++ compiler differs from other compilers on AIX. To compile a program with VisualAge C++ Version 4.0, you must first make a configuration file. See the documentation that comes with the compiler to learn more about this.

DB2 provides configuration files for the different types of DB2 programs you can build with the VisualAge C++ compiler. To use a DB2 configuration file, you first set an environment variable to the program name you wish to compile. Then you compile the program with a command supplied by VisualAge C++. Here are the configuration files provided by DB2, and the sections describing how they can be used to compile your programs:

api.icc
DB2 API configuration file. For details, see "DB2 API Applications" on page 111.

cli.icc
DB2 CLI configuration file. For details, see “DB2 CLI Applications” on page 113.

clis.icc
DB2 CLI stored procedure configuration file. For details, see “DB2 CLI Stored Procedures” on page 114.

emb.icc
Embedded SQL configuration file. For details, see “Embedded SQL Applications” on page 117.

stp.icc
Embedded SQL stored procedure configuration file. For details, see “Embedded SQL Stored Procedures” on page 119.

udf.icc
User-defined function configuration file. For details, see “User-Defined Functions (UDFs)” on page 122.

DB2 API Applications

The configuration file, api.icc, in sqllib/samples/c, allows you to build DB2 API programs in C. There is also a C++ DB2 API configuration file in sqllib/samples/cpp. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// api.icc configuration file for DB2 API programs
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export API=prog_name'
// To use on OS/2 and Windows, enter: 'set API=prog_name'
// Then compile the program by entering: 'vacbld api.icc'

if defined( $API )
{
  prog_name = $API
}
else
{
  error "Environment Variable API is not defined."
}

infile = prog_name".c"
util   = "util.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2api.lib"
  option opts = link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
}

option opts
{
  target type(exe) outfile
  {
    source infile
    source util
    source lib
  }
}
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the DB2 API sample program `client` from the source file `client.c`, do the following:

1. Set the API environment variable to the program name by entering:
   ```
   export API=client
   ```
2. If you have an `api.ics` file in your working directory, produced by building a different program with the `api.icc` file, delete the `api.ics` file with this command:
   ```
   rm api.ics
   ```

   An existing `api.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld api.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The result is an executable file, `client`. You can run the program by entering the executable name:

```
client
```

DB2 CLI Applications

The configuration file, `cli.icc`, in `sqllib/samples/cli`, allows you to build DB2 CLI programs. This file can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// cli.icc configuration file for DB2 CLI applications
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export CLI=prog_name'
// To use on OS/2 and Windows, enter: 'set CLI=prog_name'
// Then compile the program by entering: 'vacbld cli.icc'

if defined( $CLI )
{
  prog_name = $CLI
}
else
{
  error "Environment Variable CLI is not defined."
}

infile   = prog_name".c"
samputil = "samputil.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2cli.lib"
  option opts = link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
}

option opts
{
  target type(exe) outfile
  {
    source infile
    source samputil
    source lib
  }
}
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: __TOS_AIX__, __TOS_OS2__, __TOS_WIN__.

To use the configuration file to build the DB2 CLI sample program basiccon from the source file basiccon.c, do the following:

1. Set the CLI environment variable to the program name by entering:
   ```
   export CLI=basiccon
   ```
2. If you have a cli.ics file in your working directory, produced by building a different program with the cli.icc file, delete the cli.ics file with this command:
   ```
   rm cli.ics
   ```
   An existing cli.ics file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld cli.icc
   ```
   Note: The vacbld command is provided by VisualAge C++ Version 4.0.

The result is an executable file, basiccon. You can run the program by entering the executable name:

```
basiccon
```

DB2 CLI Stored Procedures

The configuration file, clis.icc, in sqllib/samples/cli, allows you to build DB2 CLI stored procedures. This file can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// clis.icc configuration file for DB2 CLI stored procedures
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export CLIS=prog_name'
// To use on OS/2 and Windows, enter: 'set CLIS=prog_name'
// Then compile the program by entering: 'vacbld clis.icc'

if defined( $CLIS )
{
  prog_name = $CLIS
}
else
{
  error "Environment Variable CLIS is not defined."
}

infile = prog_name".c"
samputil = "samputil.c"
expfile = prog_name".exp"
```

```
if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else /* if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ ) */
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2cli.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
  cpcmd       = "copy"
  funcdir     = db2path"\\function"
}

option opts
{
  target type(dll) outfile
  {
    source infile
    source samputil
    source lib
  }
}

if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}

run after cpcmd " " outfile " " funcdir
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the DB2 CLI stored procedure `outsrv2` from the source file `outsrv2.c`, do the following:

1. Set the CLIS environment variable to the program name by entering:
   ```
   export CLIS=outsrv2
   ```
2. If you have a `clis.ics` file in your working directory, produced by building a different program with the `clis.icc` file, delete the `clis.ics` file with this command:
   ```
   rm clis.ics
   ```
   An existing `clis.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld clis.icc
   ```
   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The stored procedure is copied to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv2`, you can build the CLI client application `outcli2` that calls the stored procedure. You can build `outcli2` by using the configuration file, `cli.icc`. Refer to “DB2 CLI Applications” on page 113 for details.

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

`outcli2` *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 `outsrv2`, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

The configuration file, `emb.icc`, in `sqllib/samples/c`, allows you to build DB2 embedded SQL applications in C. There is also a C++ configuration file for embedded SQL applications in `sqllib/samples/cpp`. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// emb.icc configuration file for embedded SQL applications
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export EMB=prog_name'
// To use on OS/2 and Windows, enter: 'set EMB=prog_name'
// Then compile the program by entering: 'vacbld emb.icc'

if defined( $EMB )
{
  prog_name = $EMB
}
else
{
  error "Environment Variable EMB is not defined."
}

// To connect to another database, replace "sample"
// For user ID and password, update 'user' and 'passwd'
// and take out the comment in the line: 'run before "embprep "'
dbname = "sample"
user   = ""
passwd = ""

// Precompiling the source program file
run before "embprep " prog_name " " dbname // " " user " " passwd

infile = prog_name".c"
util   = "util.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2api.lib"
  option opts = link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
}

option opts
{
  target type(exe) outfile
  {
    source infile
    source util
    source lib
  }
}
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the embedded SQL application `updat` from the source file `updat.sqc`, do the following:

1. Set the EMB environment variable to the program name by entering:
   ```
   export EMB=updat
   ```
2. If you have an `emb.ics` file in your working directory, produced by building a different program with the `emb.icc` file, delete the `emb.ics` file with this command:
   ```
   rm emb.ics
   ```

   An existing `emb.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld emb.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The result is an executable file, `updat`. You can run the program by entering the executable name:

```
updat
```

Embedded SQL Stored Procedures

The configuration file, `stp.icc`, in `sqllib/samples/c`, allows you to build DB2 embedded SQL stored procedures in C. There is also a C++ configuration file for embedded SQL stored procedures in `sqllib/samples/cpp`. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// stp.icc configuration file for embedded SQL stored procedures
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export STP=prog_name'
// To use on OS/2 and Windows, enter: 'set STP=prog_name'
// Then compile the program by entering: 'vacbld emb.icc'

if defined( $STP )
{
  prog_name = $STP
}
else
{
  error "Environment Variable STP is not defined."
}

// To connect to another database, replace "sample"
// For user ID and password, update 'user' and 'passwd'
// and take out the comment in the line: 'run before "embprep "'
dbname = "sample"
user   = ""
passwd = ""

// Precompiling the source program file
run before "embprep " prog_name " " dbname // " " user " " passwd

infile = prog_name".c"
util   = "util.c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2api.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
  cpcmd       = "copy"
  funcdir     = db2path"\\function"
}

option opts
{
  target type(dll) outfile
  {
    source infile
    source util
    source lib
  }
}

if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}

run after cpcmd " " outfile " " funcdir
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the embedded SQL stored procedure `outsrv` from the source file `outsrv.sqc`, do the following:

1. Set the STP environment variable to the program name by entering:
   ```
   export STP=outsrv
   ```
2. If you have an `stp.ics` file in your working directory, produced by building a different program with the `stp.icc` file, delete the `stp.ics` file with this command:
   ```
   rm stp.ics
   ```
   An existing `stp.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld stp.icc
   ```
   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The stored procedure is copied to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the configuration file `emb.icc`. Refer to “Embedded SQL Applications” on page 117 for details.

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

`outcli` *remote_database userid password*

where

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

userid
: Is a valid user ID.

password
: Is a valid password.

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

User-Defined Functions (UDFs)

The configuration file, udf.icc, in sqllib/samples/c, allows you to build user-defined functions in C. There is also a C++ configuration file for user-defined functions in sqllib/samples/cpp. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// udf.icc configuration file for user-defined functions
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export UDF=prog_name'
// To use on OS/2 and Windows, enter: 'set UDF=prog_name'
// Then compile the program by entering: 'vacbld udf.icc'

if defined( $UDF )
{
  prog_name = $UDF
}
else
{
  error "Environment Variable UDF is not defined."
}

infile = prog_name".c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a", "libdb2apie.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2api.lib", "db2apie.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
  cpcmd       = "copy"
  funcdir     = db2path"\\function"
}

option opts
```

```
{
  target type(dll) outfile
  {
    source infile
    source lib
  }
}
if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}

run after cpcmd " " outfile " " funcdir
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the user-defined function program `udf` from the source file `udf.c`, do the following:

1. Set the UDF environment variable to the program name by entering:
   ```
   export UDF=udf
   ```
2. If you have a `udf.ics` file in your working directory, produced by building a different program with the `udf.icc` file, delete the `udf.ics` file with this command:
   ```
   rm udf.ics
   ```

   An existing `udf.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld udf.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The UDF library is copied to the server in the path `sqllib/function`.

If necessary, set the file mode for the user-defined function so the DB2 instance can run it.

Once you build `udf`, you can build the client application, `calludf`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI `calludf` program from the source file `calludf.c`, in `sqllib/samples/cli`, by using the configuration file `cli.icc`. Refer to “DB2 CLI Applications” on page 113 for details.

You can build the embedded SQL `calludf` program from the source file `calludf.sqc`, in `sqllib/samples/c`, by using the configuration file `emb.icc`. Refer to “Embedded SQL Applications” on page 117 for details.

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

```
calludf
```

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

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

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

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

IBM COBOL Set for AIX

This section includes the following topics:

- Using the Compiler
- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures

Using the Compiler

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

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

  ```
  $HOME/sqllib/include/cobol_i
  ```

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

If you are NOT using the ″System/390 host data type support″ feature of the IBM COBOL Set for AIX compiler, or you are using an earlier version of this compiler, then the DB2 include files for your applications are in the following directory:

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

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

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

DB2 API Applications

The script file `bldcobapi`, in `sqllib/samples/cobol`, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldcobapi script file
# Builds a COBOL DB2 API program not containing embedded SQL
# Usage:  bldcobapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the checkerr.cbl error checking utility.
cob2 -qpgmname\(mixed\) -qlib -I$DB2PATH/include/cobol_a \
     -c checkerr.cbl

# Compile the program.
cob2 -qpgmname\(mixed\) -qlib -I$DB2PATH/include/cobol_a \
     -c $1.cbl

# Link the program.
cob2 -o $1 $1.o checkerr.o -ldb2 -L$DB2PATH/lib
```

Compile and Link Options for bldcobapi

The script file contains the following compile options:

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

The script file contains the following link options:

cob2	Use the compiler as a front end for the linker.
-o $1	Specify the executable program.
$1.o	Specify the program object file.
checkerr.o	Include the utility object file for error-checking.
-ldb2	Link with the database manager library.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.cbl`, enter:

```
bldcobapi client
```

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

```
client
```

Embedded SQL Applications

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Compile the checkerr.cbl error checking utility.
cob2 -qpgmname\(mixed\) -qlib -I$DB2PATH/include/cobol_a \
     -c checkerr.cbl

# Compile the program.
cob2 -qpgmname\(mixed\) -qlib -I$DB2PATH/include/cobol_a \
     -c $1.cbl

# Link the program.
cob2 -o $1 $1.o checkerr.o -ldb2 -L$DB2PATH/lib
```

Compile and Link Options for bldcob
The script file contains the following compile options: **cob2** The IBM COBOL Set compiler. **-qpgmname\(mixed\)** Instructs the compiler to permit CALLs to library entry points with mixed-case names. **-qlib** Instructs the compiler to process COPY statements. **-I$DB2PATH/include/cobol_a** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include/cobol_a`. **-c** Perform compile only; no link. This book assumes that compile and link are separate steps.
The script file contains the following link options: **cob2** Use the compiler as a front end for the linker. **-o $1** Specify the executable program. **$1.o** Specify the program object file. **checkerr.o** Include the utility object file for error-checking. **-ldb2** Link with the database manager library. **-L$DB2PATH/lib** Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. Refer to your compiler documentation for additional compiler options.

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

```
bldcob updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

The script file uses the source file name, $1, for the shared library name, and for the entry point to the shared library. If you are building stored procedures where the entry point function name is different from the source file name, you can modify the script file to accept another parameter for the entry point. We recommend renaming the database parameter to $3, the user ID parameter to $4, and the password parameter to $5. Then you can change the entry point link option to `-e` $2, and specify the additional parameter on the command line when you run the script file.

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

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

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

# Disconnect from the database.
db2 connect reset

# Compile the checkerr.cbl error checking utility.
cob2 -qpgmname\(mixed\) -qlib -I$DB2PATH/include/cobol_a \
     -c checkerr.cbl

# Compile the program.
cob2 -qpgmname\(mixed\) -qlib -c -I$DB2PATH/include/cobol_a $1.cbl

# Link the program using the export file $1.exp
# creating shared library $1 with entry point $1.
cob2 -o $1 $1.o checkerr.o -H512 -T512 -e $1 -bE:$1.exp \
     -L$DB2PATH/lib -ldb2
```

```
# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldcobsrv
The script file contains the following compile options: **cob2** The IBM COBOL Set compiler. **-qpgmname\(mixed\)** Instructs the compiler to permit CALLs to library entry points with mixed-case names. **-qlib** Instructs the compiler to process COPY statements. **-c** Perform compile only; no link. This book assumes that compile and link are separate steps. **-I$DB2PATH/include/cobol_a** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include/cobol_a`.
The script file contains the following link options: **cob2** Use the compiler to link edit. **-o $1** Specify the output as a shared library file. **$1.o** Specify the stored procedure object file. **checkerr.o** Include the utility object file for error-checking. **-H512** Specify output file alignment. **-T512** Specify output file text segment starting address. **-e $1** Specify the default entry point to the shared library. **-bE:$1.exp** Specify an export file. The export file contains a list of the stored procedures. **-L$DB2PATH/lib** Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. **-ldb2** Link with the database manager library. Refer to your compiler documentation for additional compiler options.

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

```
bldcobsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldcob`. Refer to "IBM COBOL Set for AIX" on page 124 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

Micro Focus COBOL

This section includes the following topics:

- Using the Compiler
- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures

Using the Compiler

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

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

     ```
     /usr/lpp/db2_06_01/lib/db2mkrts.args
     ```
- You must include the DB2 COBOL COPY file directory in the Micro Focus COBOL environment variable COBCPY. The COBCPY environment variable specifies the location of the COPY files. The DB2 COPY files for Micro Focus COBOL reside in `sqllib/include/cobol_mf` under the database instance directory.

  To include the directory, enter:

  ```
  export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
  ```

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

DB2 API Applications

The script file `bldmfapi`, in `sqllib/samples/cobol_mf`, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldmfapi script file
# Builds a COBOL DB2 API program not containing embedded SQL
# Usage: bldmfapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY

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

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

# Link the program.
cob -x -o $1 $1.o checkerr.o -ldb2 -ldb2gmf -L$DB2PATH/lib
```

Compile and Link Options for bldmfapi

The script file contains the following compile options:

cob	The COBOL compiler.
-c	Perform compile only; no link.
-x	Produce an executable program.

The script file contains the following link options:

cob	Use the compiler as a front end for the linker.
-x	Produce an executable program.
-o $1	Specify the executable program.
$1.o	Specify the program object file.
-ldb2	Link to the database manager library.
-ldb2gmf	Link to the DB2 exception-handler library for Micro Focus COBOL.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.cbl`, enter:

```
bldmfapi client
```

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

```
client
```

Embedded SQL Applications

The script file bldmfcob, in sqllib/samples/cobol_mf, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY

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

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

```
# Link the program.
cob -x -o $1 $1.o checkerr.o -ldb2 -ldb2gmf -L$DB2PATH/lib
```

Compile and Link Options for bldmfcob

The script file contains the following compile options:

cob	The COBOL compiler.
-c	Perform compile only; no link.
-x	Produce an executable program.

The script file contains the following link options:

cob	Use the compiler to link edit.
-x	Produce an executable program.
-o $1	Specify the name of the executable program.
$1.o	Specify the program object file.
-ldb2	Link to the database manager library.
-ldb2gmf	Link to the DB2 exception-handler library for Micro Focus COBOL.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.

Refer to your compiler documentation for additional compiler options.

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

```
bldmfcob updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

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

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

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

The script file uses the source file name, $1, for the shared library name, and for the entry point to the shared library. If you are building stored procedures where the entry point function name is different from the source file name, you can modify the script file to accept another parameter for the entry point. We recommend renaming the database parameter to $3, the user ID parameter to $4, and the password parameter to $5. Then you can change the entry point link option to -e $2, and specify the additional parameter on the command line when you run the script file.

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

```
fi

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

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY
# Compile the checkerr.cbl error checking utility.
cob -c -x checkerr.cbl
# Compile the program.
cob -c -x $1.cbl
# Create the wrapper program for the stored procedure.
wrpmfcobs $1
# Link the program using the export file ${1}_wrap.exp,
# creating a shared library called $1 with the main
# entry point ${1}_wrap.
cob -x -o $1 ${1}_wrap.c $1.o -Q -bE:${1}_wrap.exp -Q "-e $1" \
-Q -bI:$DB2PATH/lib/db2g.imp -ldb2gmf -L$DB2PATH/lib
# Copy the shared library to the DB2 instance sqllib/function subdirectory.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldmfcobs

The script file contains the following compile options:

cob	The COBOL compiler.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.
-x	Produce an executable program.

Compile and Link Options for bldmfcobs
The script file contains the following link options:
cob Use the compiler to link edit.
-x Produce an executable program.
-o $1 Specify the executable program.
-o ${1}_wrap.c Specify the wrapper program.
$1.o Specify the program object file.
-Q -bE:${1}_wrap.exp Specify an export file. The export file contains a list of the stored procedure entry points. If a stored procedure is called `x`, then its entry point will be `x_wrap`.
-Q "-e $1" Specify the default entry point to the shared library.
-Q -bI:$DB2PATH/lib/db2g.imp Provides a list of entry points to the DB2 application library.
-ldb2gmf Link to the DB2 exception-handler library for Micro Focus COBOL.
-L$DB2PATH/lib Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
Refer to your compiler documentation for additional compiler options.

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

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

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

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

int main(void)
{
  return 0;
}

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

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

```
   bldmfcobs outsrv
```

The script file copies the stored procedure to the server in the path sqllib/function. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the sqllib/function/unfenced directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide.*

Note: An unfenced stored procedure runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure, which runs in an address space isolated from the database manager. With unfenced stored procedures there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures when you need to maximize the

performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Application Development Guide* for more information.

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldcob`. Refer to “Micro Focus COBOL” on page 132 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

Exiting the Stored Procedure

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

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

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

IBM XL Fortran for AIX

This section contains the following topics:

- Using the Compiler
- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures

Using the Compiler

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

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

  The precompiler always treats lines with a `D` or `d` in column one as comments.
- The compiler is case insensitive by default. You can make it case sensitive by using a compiler option.

  SQL keywords are always case insensitive. If you make the compiler case sensitive, you must enter all Fortran keywords in lowercase. Additionally, identifier references must match the case of declarations.
- A single tab introduces source lines such that the following character is positioned at column 7. The compiler treats tabs in locations other than between columns 1-6, and in character constants, as blanks.
- You cannot use the following data declaration keywords in host variable declarations: `POINTER`, `BYTE`, `STATIC`, and `AUTOMATIC`.
- Pass by-value arguments using `%VAL()` and by-reference arguments using `%REF()`. The *Administrative API Reference* uses this syntax in the Fortran DB2 API examples.
- You cannot use the XL Fortran for AIX free-format option in .sqf files.
- The DB2 precompiler is case insensitive, but XL Fortran for AIX may not be, depending on compiler options. Therefore, do not use host variables with the same spelling and expect the case of the letters in the variable to make them unique. For example, the precompiler treats `NAME`, `name`, and `Name` as equal.

  Similarly, the following keywords are recognized by the precompiler in a case insensitive manner:

  ```
  @PROCESS                 ENDIF                    INTERFACE
  AUTOMATIC                ENDFORALL                LOGICAL
  BLOCKDATA                ENDINTERFACE             MODULE
  BYTE                     ENDSELECT                PARAMETER
  CHARACTER                ENDTYPE                  POINTER
  COMPLEX                  ENDWHERE                 PROGRAM
  CONTAINS                 ENTRY                    REAL
  DOUBLECOMPLEX            FORMAT                   STATIC
  DOUBLEPRECISION          FUNCTION                 SUBROUTINE
  END                      IF                       TYPE
  ENDDO                    IMPLICIT                 USE
  ENDFILE                  INTEGER
  ```
- The precompiler allows only digits, blanks, and tab characters within columns 1-5 on continuation lines.

- You cannot use the \ character to include string delimiters within strings. For example, use the strings `'the''character'` or `"the""character"` instead of `'the\'character'` or `"the\"character"`.
- Fortran `.sqf` source files do not support Hollerith constants.

DB2 API Applications

The script file `bldxlfapi`, in `sqllib/samples/fortran`, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldxlfapi script file
# Build sample Fortran program not containing embedded SQL.
# Usage:  bldxlfapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.f error-checking utility.
xlf -I$DB2PATH/include -c util.f

# Compile the program.
xlf -I$DB2PATH/include -c $1.f

# Link the program.
xlf -o $1 $1.o util.o -ldb2 -L$DB2PATH/lib
```

Compile and Link Options for bldxlfapi

The script file contains the following compile options:

`xlf` The IBM XL Fortran compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldxlfapi

The script file contains the following link options:

xlf	Use the compiler as a front end for the linker.
-o $1	Specify the executable program.
$1.o	Specify the program object file.
util.o	Include the utility object file for error-checking.
-ldb2	Link with the database manager library.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.f`, enter:

```
bldxlfapi client
```

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

```
client
```

Embedded SQL Applications

The script file `bldxlf`, in `sqllib/samples/fortran`, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
```

```
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

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

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

# Disconnect from the database.
db2 connect reset

# Compile the util.f error-checking utility.
xlf -I$DB2PATH/include -c util.f

# Compile the program.
xlf -I$DB2PATH/include -c $1.f

# Link the program.
xlf -o $1 $1.o util.o -ldb2 -L$DB2PATH/lib
```

Compile and Link Options for bldxlf

The script file contains the following compile options:

`xlf` The IBM XL Fortran compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

`xlf` Use the compiler as a front end for the linker.

`-o $1` Specify the executable program.

`$1.o` Specify the program object file.

`util.o` Include the utility object file for error-checking.

`-ldb2` Link with the database manager library.

`-L$DB2PATH/lib`
Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.

Refer to your compiler documentation for additional compiler options.

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

```
bldxlf updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

The script file uses the source file name, $1, for the shared library name, and for the entry point to the shared library. If you are building stored procedures where the entry point function name is different from the source file name, you can modify the script file to accept another parameter for the entry point. We recommend renaming the database parameter to $3, the user ID parameter to $4, and the password parameter to $5. Then you can change the entry point link option to -e $2, and specify the additional parameter on the command line when you run the script file.

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

# Precompile the program.
```

```
db2 prep $1.sqf bindfile

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

# Disconnect from the database.
db2 connect reset

# Compile the util.f error-checking utility.
xlf -I$DB2PATH/include -c util.f

# Compile the program.
xlf -I$DB2PATH/include -c $1.f

# Link the program using the export file $1.exp,
# creating shared library $1 with entry point $1
xlf -o $1 $1.o util.o -ldb2 -L$DB2PATH/lib \
    -H512 -T512 -bE:$1.exp -e $1

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldxlfsrv

The script file contains the following compile options:

`xlf` The IBM XL Fortran compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldxlfsrv

The script file contains the following link options:

xlf	Use the compiler as a front end for the linker.
-o $1	Specify the output as a shared library file.
$1.o	Specify the stored procedure object file.
util.o	Include the utility object file for error-checking.
-ldb2	Link with the database manager library.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
-H512	Specify output file alignment.
-T512	Specify output file text segment starting address.
-bE:$1.exp	Specify an export file. The export file contains a list of the stored procedures.
-e $1	Specify the default entry point to the shared library.

Refer to your compiler documentation for additional compiler options.

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

```
bldxlfsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the "Stored Procedures" chapter in the *Application Development Guide*.

Note: An unfenced stored procedure runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure, which runs in an address space isolated from the database manager. With unfenced stored procedures there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures when you need to maximize the

performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Application Development Guide* for more information.

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldxlf`. Refer to “Embedded SQL Applications” on page 143 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

REXX

You do not precompile or bind REXX programs.

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

Enter:

```
export LIBPATH=$LIBPATH:/lib:/usr/lib:/usr/lpp/db2_06_01/sqllib/lib
```

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

1. At the shell command prompt, enter `rexx` *name* where *name* is the name of your REXX program.
2. If the first line of your REXX program contains a ″magic number″, (#!), and identifies the directory where the REXX/6000 interpreter resides, you

can run your REXX program by entering its name at the shell command prompt. For example, if the REXX/6000 interpreter file is in the /usr/bin directory, include the following as the very first line of your REXX program:

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

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

```
   chmod +x name
```

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

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

- Add the line, "#! /usr/bin/rexx", to the top of the program source file, if it's not already there. Then, run the program directly by entering:

  ```
  updat.cmd
  ```

- Specify the REXX interpreter and the program by entering:

  ```
  rexx updat.cmd
  ```

For further information on REXX and DB2, refer to the chapter, ″Programming in REXX″, in the *Application Development Guide*.

Chapter 6. Building HP-UX Applications

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

For the latest DB2 application development updates for HP-UX, visit the Web page at:

```
http://www.software.ibm.com/data/db2/udb/ad
```

Notes:

1. You may get a warning similar to the following when compiling programs on HP-UX:

   ```
   (Warning) At least one PA 2.0 object file (<filename>.o) was detected.
   The linked object may not run on a PA 1.x system.
   ```

   where `<filename>` is the program file you are compiling.

   Unless you have a PA_RISC 1.x system, this warning does not apply. You may add the option `+DAportable` to both the compile and link steps. This option will generate code compatible across PA_RISC 1.x and 2.0 workstations, and will suppress the warning. However, the use of this option has negative performance implications, and therefore, has not been included in the script files in this chapter.
2. If you are migrating DB2 from HP-UX Version 10 or earlier to HP-UX Version 11, your DB2 programs must be re-precompiled with DB2 on HP-UX Version 11 (if they include embedded SQL), and must be re-compiled. This includes all DB2 applications, stored procedures,

user-defined functions and user exit programs. As well, DB2 programs that are compiled on HP-UX Version 11 may not run on HP-UX Version 10 or earlier. DB2 programs that are compiled and run on HP-UX Version 10 may connect remotely to HP-UX Version 11 servers.

HP-UX C

This section explains how to use HP-UX C with the following interfaces:
- DB2 APIs
- DB2 CLI
- Embedded SQL

DB2 API Applications

The script file, `bldccapi`, in `sqllib/samples/c`, contains the commands to build a sample C program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldccapi script file
# Builds a sample C program not containing embedded SQL
# Usage: bldccapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.c error-checking utility.
cc -Aa +e -I$DB2PATH/include -c util.c

# Compile the program.
cc -Aa +e -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L$DB2PATH/lib -ldb2
```

Compile and Link Options for bldccapi
The script file contains the following compile options: **cc** The C compiler. **-Aa** Use ANSI standard mode (for the C compiler only). **+e** Enables HP value-added features while compiling in ANSI C mode. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include` **-c** Perform compile only; no link. This book assumes that compile and link are separate steps.
The script file contains the following link options: **cc** Use the compiler as a front end for the linker. **-o $1** Specify the executable. **$1.o** Specify the program object file. **util.o** Include the utility object file for error checking. **-L$DB2PATH/lib** Specify the location of the DB2 runtime shared libraries. For example:`$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. **-ldb2** Link with the DB2 library. Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.c`, enter:

```
bldccapi client
```

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

```
client
```

DB2 CLI Applications

The script file `bldcli` in `sqllib/samples/cli` contains the commands to build a DB2 CLI program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldcli script file -- HP-UX
# Builds a CLI program with HP-UX C.
# Usage: bldcli <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc -Aa +e -I$DB2PATH/include -c samputil.c

# Compile the program.
cc -Aa +e -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o samputil.o -L$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcli

The script file contains the following compile options:

cc Use the C compiler.

-Aa Use ANSI standard mode.

+e Enables HP value-added features while compiling in ANSI C mode.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

cc Use the compiler as a front end for the linker.

-o $1 Specify the executable program.

-o $1.o
Specify the object file.

samputil.o
Include the utility object file for error checking.

-L$DB2PATH/lib
Specify the location of the DB2 runtime shared libraries. For example, `$HOME/sqllib/lib`

-ldb2 Link with the database manager library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `basiccon` from the source file `basiccon.c`, enter:

```
bldcli basiccon
```

The result is an executable file basiccon. You can run the executable file by entering the executable name:

```
basiccon
```

DB2 CLI Stored Procedures

The script file bldclisp in sqllib/samples/cli contains the commands to build a DB2 CLI stored procedure. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldclisp script file -- HP-UX
# Builds a CLI stored procedure in HP-UX C.
# Usage: bldclisp <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc +u1 +z -Aa +e -I$DB2PATH/include -c samputil.c

# Compile the program.
cc +u1 +z -Aa +e -I$DB2PATH/include -c $1.c

# Link the program.
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldclisp	
The script file contains the following compile options:	
cc	The C compiler.
+u1	Allow unaligned data access. Use only if your application uses unaligned data.
+z	Generate position-independent code.
-Aa	Use ANSI standard mode (for the C compiler only).
+e	Enables HP value-added features while compiling in ANSI C mode.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.
The script file contains the following link options:	
ld	Use the linker to link edit.
-b	Create a shared library rather than a normal executable.
-o $1	Specify the executable.
$1.o	Specify the object file.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `-L$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-ldb2	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

To build the sample program `outsrv2` from source file `outsrv2.c`, enter:

```
bldclisp outsrv2
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv2`, you can build the CLI client application `outcli2` that calls the stored procedure. You can build `outcli2` by using the script file `bldcli`. Refer to "DB2 CLI Applications" on page 153 for details.

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

```
outcli2 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 `outsrv2`, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

The script file, `bldcc`, in `sqllib/samples/c`, contains the commands to build an embeddded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
cc -Aa +e -I$DB2PATH/include -c util.c

# Compile the program.
cc -Aa +e -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcc

The script file contains the following compile options:

cc The C compiler.

-Aa Use ANSI standard mode (for the C compiler only).

+e Enables HP value-added features while compiling in ANSI C mode.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `-I$DB2PATH/include`

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldcc

The script file contains the following link options:

cc	Use the compiler to link edit.
-o $1	Specify the executable.
$1.o	Specify the program object file.
util.o	Include the utility object file for error checking.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: -L$DB2PATH/lib. If you do not specify the -L option, /usr/lib:/lib is assumed.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldcc updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib
```

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

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

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

# Disconnect from the database.
db2 connect reset

# Compile the program.
cc +u1 +z -Aa +e -I$DB2PATH/include -c $1.c

# Link the program to create a shared library
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory of the DB2
# instance. This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldccsrv

The script file contains the following compile options:

cc	The C compiler.
+u1	Allow unaligned data access. Use only if your application uses unaligned data.
-Aa	Use ANSI standard mode (for the C compiler only).
+z	Generate position-independent code.
+e	Enables HP value-added features while compiling in ANSI C mode.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `-I$DB2PATH/include`.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldccsrv

The script file contains the following link options:

ld	Use the linker to link edit.
-b	Create a shared library rather than a normal executable.
-o $1	Specify the output as a shared library file.
$1.o	Specify the program object file.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldccsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldcc`. Refer to “HP-UX C” on page 152 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
: Is a valid password.

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

User-Defined Functions (UDFs)

The script file `bldccudf`, in `sqllib/samples/c`, contains the commands to build a UDF. UDFs are compiled like stored procedures. They cannot contain embedded SQL statements. This means to build a UDF program, you never need to connect to a database, precompile, and bind the program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the program.
cc +u1 +z -Aa +e -I$DB2PATH/include -c $1.c

# Link the program and create a shared library.
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the DB2 instance sqllib/function subdirectory.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldccudf

The script file contains the following compile options:

cc	The C compiler.
+u1	Allow unaligned data access. Use only if your application uses unaligned data.
-Aa	Use ANSI standard mode (for the C compiler only).
+z	Generate position-independent code.
+e	Enables HP value-added features while compiling in ANSI C mode.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldccudf

The script file contains the following link options:

`ld` Use the linker to link edit.

`-b` Create a shared library rather than a normal executable.

`-o $1` Specify the output as a shared library file.

`$1.o` Specify the program object file.

`-L$DB2PATH/lib`
Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.

`-ldb2` Link with the DB2 library.

`-ldb2apie`
Link with the DB2 API Engine library to allow the use of LOB locators.

Refer to your compiler documentation for additional compiler options.

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

```
bldccudf udf
```

The script file copies the UDF to the server in the path `sqllib/function`.

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

Once you build `udf`, you can build the client application, `calludf`, that calls it. DB2 CLI and embedded SQL versions of this program are provided. You can build the DB2 CLI `calludf` program from the `calludf.c` source file in `sqllib/samples/cli` using the DB2 CLI script file `bldcli`. Refer to "DB2 CLI Applications" on page 153 for details.

You can build the embedded SQL `calludf` program from the `calludf.sqc` source file in `sqllib/samples/c` using the script file `bldcc`. Refer to "Embedded SQL Applications" on page 157 for details.

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

```
calludf
```

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

Multi-threaded Applications

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

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

The script file, `bldccmt`, in `sqllib/samples/c`, contains the commands to build an embedded SQL multi-threaded program. If you want to build a DB2 API or DB2 CLI multi-threaded program, comment out the connect, precompile, bind, and disconnect commands. For DB2 CLI, also substitute the `samputil.c` and `samputil.o` files for `util.c` and `util.o`.

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

```
#! /bin/ksh
# bldccmt script file -- HP-UX
# Builds a multi-threaded embedded SQL program with HP-UX C.
# Usage: bldccmt <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

# Precompile the program.
db2 prep $1.sqc bindfile
# Bind the program to the database.
db2 bind $1.bnd
# Disconnect from the database.
```

```
db2 connect reset

# Compile the util.c error-checking utility.
cc -Aa +e -I$DB2PATH/include -D_POSIX_C_SOURCE=199506L -c util.c

# Compile the program.
cc -Aa +e -I$DB2PATH/include -D_POSIX_C_SOURCE=199506L -c $1.c

# Link the program
cc -o $1 $1.o util.o -L$DB2PATH/lib -ldb2 -lpthread
```

Besides the -D_POSIX_C_SOURCE=199506L compile option, and the -lpthread link option, discussed above, the other compile and link options are the same as those used for the embedded SQL script file, bldcc. For information on these options, see "Embedded SQL Applications" on page 157.

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

```
bldccmt thdsrver
```

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

```
thdsrver
```

HP-UX C++

This section covers the followintg topics:

- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)
- Multi-threaded Applications

DB2 API Applications

The script file bldCCapi, in sqllib/samples/cpp, contains the commands to build a sample C++ program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldCCapi script file
# Builds a C++ program not containing embedded SQL
# Usage: bldCCapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.C error-checking utility.
CC +a1 -ext -I$DB2PATH/include -c util.C

# Compile the program.
CC +a1 -ext -I$DB2PATH/include -c $1.C

# Link the program.
CC -o $1 $1.o util.o -L$DB2PATH/lib -ldb2
```

Compile and Link Options for bldCCapi

The script file contains the following compile options:

CC	The C compiler.
+a1	Instruct the compiler to use ANSI C/C++.
-ext	Allow various C++ extensions including ″long long″ support.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

CC	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Specify the program object file.
util.o	Include the utility object file for error checking.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.C`, enter:

```
bldCCapi client
```

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

```
client
```

Embedded SQL Applications

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

# Compile the util.C error-checking utility.
CC +a1 -ext -I$DB2PATH/include -c util.C

# Compile the program.
CC +a1 -ext -I$DB2PATH/include -c $1.C

# Link the program.
CC -o $1 $1.o util.o -L$DB2PATH/lib -ldb2
```

Compile and Link Options for bldCC

The script file contains the following compile options:

CC	The C compiler.
+a1	Instruct the compiler to use ANSI C/C++.
-ext	Allow various C++ extensions including ″long long″ support.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

CC	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Specify the program object file.
util.o	Include the utility object file for error checking.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldCC updat
```

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

```
updat
```

Embedded SQL Stored Procedures

Note: Please see the information for building C++ stored procedures and UDFs in “C++ Considerations for UDFs and Stored Procedures” on page 55.

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

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

# Compile the program. First ensure it is coded with extern "C".
CC +a1 +z -ext -I$DB2PATH/include -c $1.C
# Link the program to create a shared library.
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2
# Copy the shared library to the DB2 instance sqllib/function subdirectory.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldCCsrv

The script file contains the following compile options:

CC	The C++ compiler.
+a1	Instruct the compiler to use ANSI C/C++.
+z	Generate position-independent code.
-ext	Allow various C++ extensions including ″long long″ support.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$DB2PATH/include`
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

ld	Use the linker to link edit.
-b	Create a shared library rather than a normal executable.
-o $1	Specify the executable.
$1.o	Specify the program object file.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `-L$DB2PATH/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldCCsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide.*

Note: An unfenced stored procedure runs in the same address space as the database manager and results in increased performance when

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldCC`. Refer to “Embedded SQL Applications” on page 168 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

User-Defined Functions (UDFs)

Note: Please see the information for building C++ UDFs and stored procedures in “C++ Considerations for UDFs and Stored Procedures” on page 55.

The script file `bldCCudf`, in `sqllib/samples/c`, contains the commands to build a UDF. User-defined programs cannot contain embedded SQL statements. This means to build a UDF program, you never need to connect to a database, precompile, and bind the program.

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

```
#! /bin/ksh
# bldCCudf script file -- HP-UX
# Builds a C++ user-defined function library.
# Usage: bldCCudf <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the program. The extension ".c" is for a C source file.
# Change the extension to ".C" if compiling a C++ source file.
CC -ext +a1 +z -I$DB2PATH/include -c $1.c

# Link the program.
CC -b -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# Note: this assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldCCudf

The script file contains the following compile options:

CC	The C++ compiler.
-ext	Allow various C++ extensions including ″long long″ support.
+a1	Instruct the compiler to use ANSI C/C++.
+z	Generate position-independent code.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: $DB2PATH/include
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldCCudf

The script file contains the following link options:

`CC`	Use the compiler as a front end for the linker.
`-b`	Create a shared library rather than a normal executable.
`-o $1`	Specify the executable.
`$1.o`	Specify the program object file.
`-L$DB2PATH/lib`	Specify the location of the DB2 runtime shared libraries. For example: `-L$DB2PATH/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
`-ldb2`	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldCCudf udf
```

The script file copies the UDF to the server in the path `sqllib/function`.

Note: If you wish to build a C++ UDF program that has a `.C` extension, you must modify the script file `bldCCudf` to accept programs with this extension.

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

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build the `calludf` program from the `calludf.sqC` source file in `sqllib/samples/cpp` using the script file `bldCC`. Refer to "Embedded SQL Applications" on page 168 for details.

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

```
calludf
```

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

Multi-threaded Applications

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

Multi-threaded applications on HP-UX version 11 need to have _REENTRANT defined for their compilation. The HP-UX documentation recommends compiling with -D_POSIX_C_SOURCE=199506L. This will also ensure _REENTRANT is defined. For the HP-UX C++ compiler, `-D_HPUX_SOURCE` must also be used in order to define `rand_r`. Applications also need to be linked with `-lpthread`.

The script file, `bldCCmt`, in `sqllib/samples/cpp`, contains the commands to build an embedded SQL multi-threaded program. If you want to build a DB2 API multi-threaded program, comment out the connect, precompile, bind, and disconnect commands.

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

```
#! /bin/ksh
# bldCCmt script file -- HP-UX
# Builds an embedded SQL multi-threaded program with HP-UX C++.
# Usage: bldCCmt <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

# Compile the util.C error-checking utility.
```

```
CC -ext +a1 +z -I$DB2PATH/include \
   -D_HPUX_SOURCE -D_POSIX_C_SOURCE=199506L -c util.C
# Compile the program.
CC -ext +a1 +z -I$DB2PATH/include \
   -D_HPUX_SOURCE -D_POSIX_C_SOURCE=199506L -c $1.C
# Link the program
CC -o $1 $1.o util.o -L$DB2PATH/lib -ldb2 -lpthread
```

Besides the `-D_HPUX_SOURCE` and `-D_POSIX_C_SOURCE=199506L` compile options, and the `-lpthread` link option, discussed above, the other compile and link options are the same as those used for the embedded SQL script file, `bldCC`. For information on these options, see “Embedded SQL Applications” on page 168.

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

```
bldCCmt thdsrver
```

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

```
thdsrver
```

Micro Focus COBOL

This section contains the following topics:

- Using the Compiler
- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures

Using the Compiler

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

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

     ```
     /opt/IBMdb2/V6.1/lib/db2mkrts.args
     ```

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

  To include the directory, enter:

  ```
  export COBCPY=$COBCPY:/opt/IBMdb2/V6.1/include/cobol_mf
  ```

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

DB2 API Applications

The script file `bldmfapi`, in `sqllib/samples/cobol_mf`, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldmfapi script file
# Builds a COBOL program not containing embedded SQL
# Usage: bldmfapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

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

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

# Link the program.
cob -x $1.o checkerr.o -L$DB2PATH/lib -ldb2 -ldb2gmf
```

Compile and Link Options for bldmfapi

The script file contains the following compile options:

cob	The Micro Focus COBOL compiler.
-cx	Compile to object module.

Compile and Link Options for bldmfapi

The script file contains the following link options:

cob Use the compiler to link edit.

-x Specify an executable program.

$1.o Include the program object file.

checkerr.o
Include the utility object file for error checking.

-L$DB2PATH/lib
Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`

-ldb2 Link to the DB2 library.

-ldb2gmf
Link to the DB2 exception-handler library for Micro Focus COBOL.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.cbl`, enter:

```
bldmfapi client
```

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

```
client
```

Embedded SQL Applications

The script file `bldmfcc`, in `sqllib/samples/cobol_mf`, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Connect to a database.
```

```
if (($# < 2))
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

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

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

# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

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

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

# Link the program.
cob -x $1.o checkerr.o -L$DB2PATH/lib -ldb2 -ldb2gmf
```

Compile and Link Options for bldmfcc

The script file contains the following compile options:

cob	The Micro Focus COBOL compiler.
-cx	Compile to object module.

Compile and Link Options for bldmfcc

The script file contains the following link options:

cob Use the compiler as a front end for the linker.

-x Specify an executable program.

$1.o Include the program object file.

checkerr.o
Include the utility object file for error checking.

-L$DB2PATH/lib
Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`.

-ldb2 Link to the DB2 library.

-ldb2gmf
Link to the DB2 exception-handler library for Micro Focus COBOL.

Refer to your compiler documentation for additional compiler options.

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

```
bldmfcc updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

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

```
#! /bin/ksh
# bldmfsp script file
# Builds a COBOL stored procedure.
# Usage: bldmfsp <stored_proc_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

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

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

# Link the program.
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2gmf \
   -L$COBDIR/coblib -lcobol -lcrtn

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldmfsp

The script file contains the following compile options:

cob	The COBOL compiler.
+z	Generate position-independent code.
-cx	Compile to object module.

Compile and Link Options for bldmfsp

The script file contains the following link options:

`ld` Use the linker to link edit.

`-b` Create a shared library rather than a normal executable file.

`-o $1` Specify the executable.

`$1.o` Include the program object file.

`-L$DB2PATH/lib`
Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`

`-ldb2` Link to the DB2 shared library.

`-ldb2gmf`
Link to the DB2 exception-handler library for Micro Focus COBOL.

`-L$COBDIR/coblib`
Specify the location of the COBOL runtime libraries.

`-lcobol`
Link to the COBOL library.

`-lcrtn` Link to the crtn library.

Refer to your compiler documentation for additional compiler options.

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

```
bldmfsp outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide.*

Note: An unfenced stored procedure runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure, which runs in an address space isolated from the database manager. With unfenced stored procedures there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures when you need to maximize the

performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Application Development Guide* for more information.

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldmfcc`. Refer to "Embedded SQL Applications" on page 178 for details.

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

```
outcli
```

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

Exiting the Stored Procedure

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

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

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

HP Fortran/9000 and HP-UX F77

DB2 API Applications

The script file, `bldf77api`, in `sqllib/samples/fortran`, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldf77api script file
# Builds a Fortran program not containing embedded SQL
# Usage: bldf77api <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.f error-checking utility.
f77 -w -c -I$DB2PATH/include util.f

# Compile the program.
f77 -w -c -I$DB2PATH/include $1.f

# Link the program.
f77 $1.o util.o -Wl,-L$DB2PATH/lib -ldb2 -o $1
```

Compile and Link Options for bldf77api

The script file contains the following compile options:

f77	The Fortran compiler.
-w	Suppress warning messages.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

The script file contains the following link options:

f77	Use the compiler to link edit.
$1.o	Include the program object file.
util.o	Include the object file for error checking.
-Wl,	Instruct the compiler to pass `-L$DB2PATH/lib` directly to the linker.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`
-ldb2	Link with the DB2 library.
-o $1	Specify the executable.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.f`, enter:

```
bldf77api client
```

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

```
client
```

Embedded SQL Applications

The script file, `bldf77`, in `sqllib/samples/fortran`, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Compile the util.f error-checking utility.
f77 -w -c -I$DB2PATH/include util.f

# Compile the program.
f77 -w -c -I$DB2PATH/include $1.f

# Link the program.
f77 $1.o util.o -Wl,-L$DB2PATH/lib -ldb2 -o $1
```

Compile and Link Options for bldf77	
The script file contains the following compile options:	
f77	The Fortran compiler.
-w	Suppress warning messages.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `-I$HOME/sqllib/include`
The script file contains the following link options:	
f77	Use the compiler to link edit.
$1.o	Include the program object file.
util.o	Include the utility object file for error checking.
-Wl,	Instruct the compiler to pass `-L$DB2PATH/lib` directly to the linker.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.
-o $1	Specify the executable.
Refer to your compiler documentation for additional compiler options.	

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

```
bldf77 updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Compile the program.
f77 -w -n -c +z -I$DB2PATH/include $1.f

# Link the program.
ld -b -E -o $1 $1.o -L$DB2PATH/lib

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldf77sp

The script file contains the following compile options:

f77	The Fortran compiler.
-w	Suppress warning messages.
-n	Generate a shared object file.
-c	Perform compile only; no link.
+z	Generate position-independent code.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
-o $1	Specify the executable.

Refer to your compiler documentation for additional compiler options.

The script file contains the following link options:

ld	Use the linker to link edit.
-b -E	Specify the default export file for the stored procedure.
-o $1	Specify the executable.
$1.o	Include the program object file.
-L$DB2PATH/lib	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`

Refer to your compiler documentation for additional compiler options.

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

```
bldf77srv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

Note: An unfenced stored procedure runs in the same address space as the database manager and results in increased performance when

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldf77`. Refer to “HP Fortran/9000 and HP-UX F77” on page 183 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

Chapter 7. Building Linux Applications

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

For the latest DB2 application development updates for Linux, visit the Web page at:

```
http://www.software.ibm.com/data/db2/udb/ad
```

Linux C

This section explains how to use the Linux C compiler with the following DB2 interfaces:

- DB2 APIs
- DB2 CLI
- Embedded SQL

DB2 API Applications

The script file `bldccapi` in `sqllib/samples/c` contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldccapi script file
# Builds a DB2 API program with Linux C.
# Usage: bldccapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.c error-checking utility.
cc -I$DB2PATH/include -c util.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L$DB2PATH/lib \
   -Wl,-rpath,$DB2PATH/lib -ldb2
```

Compile and Link Options for bldccapi

The script file contains the following compile options:

`cc` The C compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

`-c` Perform compile only; no link. The script file has separate compile and link steps.

The script file contains the following link options:

`cc` Use the compiler as a front end for the linker.

`-o $1` Specify the executable.

`$1.o` Include the program object file.

`util.o` Include the utility object file for error checking.

`-L$DB2PATH/lib`
Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.

`-Wl,-rpath,$DB2PATH/lib`
Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.

`-ldb2` Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.c`, enter:

```
bldccapi client
```

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

```
client
```

DB2 CLI Applications

The script file `bldcli` in `sqllib/samples/cli` contains the commands to build a DB2 CLI program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldcli script file -- Linux
# Builds a DB2 CLI program with Linux C.
# Usage: bldcli <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc -I$DB2PATH/include -c samputil.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o samputil.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcli

The script file contains the following compile options:

cc The C compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. The script file has separate compile and link steps.

Compile and Link Options for bldcli
The script file contains the following link options:

`cc`	Use the compiler as a front end for the linker.
`-o $1`	Specify the executable.
`$1.o`	Include the program object file.
`samputil.o`	Include the utility object file for error checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
`-Wl,-rpath,$DB2PATH/lib`	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
`-ldb2`	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `basiccon` from the source file `basiccon.c`, enter:

```
bldcli basiccon
```

The result is an executable file `basiccon`. You can run the executable file by entering the executable name:

```
basiccon
```

DB2 CLI Stored Procedures

The script file `bldclisp` in `sqllib/samples/cli` contains the commands to build a DB2 CLI stored procedure. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldclisp script file -- Linux
# Builds a CLI stored procedure in Linux C.
# Usage: bldclisp <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc -I$DB2PATH/include -c samputil.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o samputil.o -shared -L$DB2PATH/lib -ldb2

# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldclisp

The script file contains the following compile options:

cc The C compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

-c Perform compile only; no link. This script file has separate compile and link steps.

Compile and Link Options for bldclisp

The script file contains the following link options:

cc	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
samputil.o	Include the utility object file for error-checking.
-shared	Generate a shared library.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `outsrv2` from the source file `outsrv2.c`, enter:

```
bldclisp outsrv2
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide.*

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

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

Once you build the stored procedure outsrv2, you can build the CLI client application outcli2 that calls the stored procedure. You can build outcli2 by using the script file bldcli. Refer to “DB2 CLI Applications” on page 193 for details.

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

```
outcli2 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 outsrv2, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$(HOME)/sqllib

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

# Compile the util.c error-checking utility.
cc -I$DB2PATH/include -c util.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L$DB2PATH/lib \
   -Wl,-rpath,$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcc

The script file contains the following compile options:

cc The C compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. This script file has separate compile and link steps.

Compile and Link Options for bldcc	
The script file contains the following link options:	
cc	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Specify the object file.
util.o	Include the utility object file for error checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-Wl,-rpath,$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

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

```
bldcc updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$(HOME)/sqllib

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

# Compile the util.c error-checking utility.
cc -I$DB2PATH/include -c util.c
# Compile the program.
cc -I$DB2PATH/include -c $1.c
# Link the program and create a shared library
cc -shared -o $1 $1.o -L$DB2PATH/lib -ldb2
# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldccsrv

The script file contains the following compile options:

cc The C compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. This script file has separate compile and link steps.

Compile and Link Options for bldccsrv	
The script file contains the following link options:	
cc	Use the compiler as a front end for the linker.
-shared	Generate a shared library.
-o $1	Specify the executable.
$1.o	Include the program object file.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-ldb2	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

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

```
bldccsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldcc`. Refer to “Embedded SQL Applications” on page 197 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

User-Defined Functions (UDFs)

The script file `bldccudf` in `sqllib/samples/c` contains the commands to build a UDF. A UDF does not contain embedded SQL statements. So to build a UDF progam, you do not need to connect to a database or precompile and bind the program.

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

```
#! /bin/ksh
# bldccudf script file
# Builds a C user-defined function library.
# Usage: bldccudf <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$(HOME)/sqllib

# Compile the program.
cc -I$DB2PATH/include -c $1.c
# Link the program and create a shared library.
cc -o $1 $1.o -shared -L$DB2PATH/lib -ldb2 -ldb2apie
# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldccudf

The script file contains the following compile options:

`cc` The C compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

`-c` Perform compile only; no link. This script file has separate compile and link steps.

The script file contains the following link options:

`cc` Use the compiler as a front end for the linker.

`-o $1` Specify the executable.

`$1.o` Include the program object file.

`-shared`
Generate a shared library.

`-L$DB2PATH/lib`
Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.

`-ldb2` Link with the DB2 library.

`-ldb2apie`
Link with the DB2 API Engine library to allow the use of LOB locators.

Refer to your compiler documentation for additional compiler options.

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

```
bldccudf udf
```

The script file copies the UDF to the server in the path `sqllib/function`.

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

Once you build `udf`, you can build the client application, `calludf`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI `calludf` program from the `calludf.c` source file in `sqllib/samples/cli` using the DB2 CLI script file `bldcli`. Refer to “DB2 CLI Applications” on page 193 for details.

You can build the embedded SQL calludf program from the calludf.sqc source file in sqllib/samples/c using the script file bldcc. Refer to “Embedded SQL Applications” on page 197 for details.

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

```
calludf
```

The calling application calls functions from the udf library.

Linux C++

This section covers the following topics:

- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions

DB2 API Applications

The script file bldCCapi, in sqllib/samples/cpp, contains the commands to build a sample DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldCCapi script file
# Builds a DB2 API program.
# Usage: bldCCapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.C error-checking utility.
g++ -I$DB2PATH/include -c util.C
# Compile the program.
g++ -I$DB2PATH/include -c $1.C
# Link the program.
g++ -o $1 $1.o util.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
```

Compile and Link Options for bldCC

The script file contains the following compile options:

g++	The C++ compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
-c	Perform compile only; no link. This script file has separate compile and link steps.

The script file contains the following link options:

g++	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
util.o	Include the utility object file for error checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-Wl,-rpath,$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.C`, enter:

```
bldCCapi client
```

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

```
client
```

Embedded SQL Applications

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

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

required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default sample database.

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

# Compile the util.C error-checking utility.
g++ -I$DB2PATH/include -c util.C
# Compile the program.
g++ -I$DB2PATH/include -c $1.C
# Link the program.
g++ -o $1 $1.o util.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
```

Compile and Link Options for bldCC

The script file contains the following compile options:

g++ The C++ compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. This script file has separate compile and link steps.

Compile and Link Options for bldCC

The script file contains the following link options:

g++	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
util.o	Include the utility object file for error checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-Wl,-rpath,$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldCC updat
```

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

```
updat
```

Embedded SQL Stored Procedures

Note: Please see the information for building C++ stored procedures and UDFs in "C++ Considerations for UDFs and Stored Procedures" on page 55.

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

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$HOME/sqllib

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

# Compile the program.
g++ -I$DB2PATH/include -c $1.C
# Link the program and create a shared library.
g++ -shared -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldCCsrv

The script file contains the following compile options:

g++ The C++ compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. This script file has separate compile and link steps.

Compile and Link Options for bldCCsrv

The script file contains the following link options:

g++	Use the compiler as a front end for the linker.
-shared	Generate a shared library.
-o $1	Specify the executable.
$1.o	Include the program object file.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldCCsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldCC`. Refer to “Embedded SQL Applications” on page 205 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

User-Defined Functions (UDFs)

Note: Please see the information for building C++ UDFs and stored procedures in “C++ Considerations for UDFs and Stored Procedures” on page 55.

The script file `bldCCudf` in `sqllib/samples/cpp` contains the commands to build a UDF. A UDF does not contain embedded SQL statements. So to build a UDF progam, you do not need to connect to a database or precompile and bind the program.

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

```
#! /bin/ksh
# bldCCudf script file -- Linux
# Builds a C++ user-defined function library.
# Usage: bldCCudf <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$HOME/sqllib

# Compile the program. The extension ".c" is for a C source file.
# Change the extension to ".C" if compiling a C++ source file.
g++ -I$DB2PATH/include -c $1.c

# Link the program.
g++ -o $1 $1.o -shared -L$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldCCudf

The script file contains the following compile options:

g++ The C++ compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

-c Perform compile only; no link. This script file has separate compile and link steps.

The script file contains the following link options:

g++ Use the compiler as a front end for the linker.

-o $1 Specify the executable.

$1.o Include the program object file.

-shared
Generate a shared library.

-L$DB2PATH/lib
Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.

-ldb2 Link with the DB2 library.

-ldb2apie
Link with the DB2 API Engine library to allow the use of LOB locators.

Refer to your compiler documentation for additional compiler options.

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

```
bldCCudf udf
```

The script file copies the UDF to the server in the path `sqllib/function`.

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

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build the `calludf` program from the `calludf.sqC` source file in `sqllib/samples/cpp` using the script file `bldCC`. Refer to "Embedded SQL Applications" on page 205 for details.

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

```
calludf
```

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

Chapter 8. Building OS/2 Applications

This chapter provides detailed information for building 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.

For the latest DB2 application development updates for OS/2, visit the Web page at:

```
http://www.software.ibm.com/data/db2/udb/ad
```

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

IBM VisualAge C++ for OS/2 Version 3

This section includes the following topics:

- DB2 API Applications
- DB2 CLI Applications
- DB2 CLI Stored Procedures
- Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)

Note: The VisualAge C++ compiler is used for both C and C++ sample programs supplied in the %DB2PATH\samples\c and %DB2PATH\samples\cpp directories. The command files in both these directories contain commands to accept either a C or C++ source file, depending on the file extension. By default, the C++ commands are commented out in the command files in %DB2PATH\samples\c, and the C commands are commented out in the command files in %DB2PATH\samples\cpp. This section demonstrates building programs using the C command files.

DB2 API Applications

The command file bldvaapi.cmd, in %DB2PATH%\samples\c, and in %DB2PATH%\samples\cpp, contains the commands to build a DB2 API program. The parameter, %1, specifies the name of your source file.

```
@echo off
rem  bldvaapi command file
rem  Builds a C or C++ DB2 API program.
rem  Usage: bldvaapi <prog_name>

if "%1" == "" goto error

rem Compile the error-checking utility.
icc -c util.c
rem For C++, comment out the above line and uncomment the following:
rem icc -c util.cxx

rem  Compile the program.
icc -C+ -O- -Ti+ %1.c
rem For C++, comment out the above line and uncomment the following:
rem icc -C+ -O- -Ti+ %1.cxx

rem  Link step
ilink /NOFREE /NOI /DEBUG /ST:64000 /PM:VIO %1.obj util.obj,,,db2api;

goto exit

:error
echo Usage: bldvaapi <prog_name>

:exit
@echo on
```

Compile and Link Options for bldvaapi

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:64000	Specify a stack size of at least 64000.
/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 `client` from the C source file `client.c` in `%DB2PATH%\samples\c`, or from the C++ source file `client.cxx` in `%DB2PATH%\samples\cpp`, enter:

```
   bldvaapi client
```

The result is an executable file, `client`. You can run the executable file by entering the executable name:

```
   client
```

DB2 CLI Applications

The command file `bldcli`, in `%DB2PATH%\samples\cli`, contains the commands to build a DB2 CLI program.

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

```
@echo off
rem  bldcli command file - OS/2
rem  Builds a CLI program with IBM VisualAge C++.
rem  Usage: bldcli <prog_name>

if "%1" == "" goto error

rem Compile the error-checking utility.
icc -C+ -O- -Ti+ samputil.c

rem  Compile the program.
icc -C+ -O- -Ti+ %1.c

rem  Link the program.
ilink /NOFREE /NOI /DEBUG /ST:64000 /PM:VIO %1.obj
    samputil.obj,%1.exe,NUL,db2cli.lib;

goto exit

:error
echo Usage: bldcli <prog_name>

:exit
@echo on
```

Compile and Link Options for bldcli

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

Compile and Link Options for bldcli	
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:64000	Specify a stack size of at least 64 000.
/PM:VIO	Enable the program to run in an OS/2 window.
%1.obj	Include the object file.
samputil.obj	Include the utility object file for error checking.
%1.exe	Specify the executable.
NUL	Accept the default value.
db2cli.lib	Link with the DB2 CLI library.
Refer to your compiler documentation for additional compiler options.	

To build the sample program basiccon from the source file basiccon.c, enter:

```
bldcli basiccon
```

The result is an executable file, basiccon.exe. You can run the executable file by entering the executable name (without the extension):

```
basiccon
```

DB2 CLI Stored Procedures

The command file bldclisp, in %DB2PATH%\samples\cli, contains the commands to build a CLI stored procedure. The command file builds the stored procedure into a DLL on the server.

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  bldclisp command file - OS/2
rem  Builds a CLI stored procedure using the IBM VisualAge C++ compiler.
rem  Usage: bldclisp <prog_name>

if "%1" == "" goto error
```

```
rem Compile the error-checking utility.
icc -C+ -Ti+ -Ge- -Gm+ -W2 samputil.c

rem Compile the program.
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c

rem Link the program and produce a DLL.
ilink /NOFREE /MAP /NOI /DEBUG /ST:64000 %1.obj,%1.dll,,db2cli.lib,%1.def;

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

goto exit

:error
echo Usage: bldclisp <prog_name>

:exit
@echo on
```

Compile and Link Options for bldclisp

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 run-time library that is statically linked.
-Gm+	Link with multi-tasking libraries.
-W2	Output warning, error, and severe and unrecoverable error messages.

Compile and Link Options for bldclisp

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:64000`
Specify a stack size of at least 64000.

`%1.obj` Include the object file.

`%1.dll` Create a dynamic link library.

`db2cli.lib`
Link with the DB2 CLI library.

`%1.def` Module definition file.

Refer to your compiler documentation for additional compiler options.

To build the `outsrv2` stored procedure from the source file `outsrv2.c`, enter:

```
   bldclisp outsrv2
```

The command file uses the module definition file, `outsrv2.def`, contained in the same directory as the sample programs, to build the stored procedure. The command file copies the stored procedure DLL, `outsrv2.dll`, on the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the name of the stored procedure DLL, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move the DLL to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

Once you build the stored procedure `outsrv2`, you can build the client application `outcli2` that calls the stored procedure. You can build `outcli2` by using the command file `bldcli`. Refer to “DB2 CLI Applications” on page 215 for details.

To run the stored procedure, enter:

```
outcli2 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, `outsrv2`, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

The command file `bldvaemb.cmd`, in `%DB2PATH%\samples\c`, and in `%DB2PATH%\samples\cpp`, contains the commands to build an embedded SQL 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 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.
db2 prep %1.sqc bindfile
rem For C++, comment out the above line, and uncomment the following:
rem db2 prep %1.sqx bindfile

rem Compile the util.c error checking utility.
icc -c util.c
rem For C++, comment out the above line, and uncomment the following:
rem icc -c util.cxx

rem Compile the program.
icc -C+ -O- -Ti+ %1.c
rem For C++, comment out the above line, and uncomment the following:
rem icc -C+ -O- -Ti+ %1.cxx

rem Link the program.
ilink /NOFREE /NOI /DEBUG /ST:64000 /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:64000	Specify a stack size of at least 64000.
/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`, from the C source file `updat.sqc`, or the C++ source file, `updat.sqx`, enter:

```
bldvaemb updat
```

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

```
updat
```

Embedded SQL Stored Procedures

The command file `bldvastp`, in `%DB2PATH%\samples\c`, and in `%DB2PATH%\samples\cpp`, contains the commands to build an embedded SQL 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 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.
db2 prep %1.sqc bindfile
rem For C++, comment out the above line and uncomment the following:
rem db2 prep %1.sqx bindfile

rem Compile the program.
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c
rem For C++, comment out the above line and uncomment the following:
rem icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.cxx

rem Link the program.
ilink /NOFREE /NOI /DEBUG /ST:64000 %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 run-time library that is statically linked.
-Gm+	Link with multi-tasking 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.
/NOI	No Ignore Case. Force case sensitive identifiers.
/DEBUG	Include debugging information.
/ST:64000	Specify a stack size of at least 64000.
%1.dll	Create a 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 stored procedure `outsrv` from the C source file `outsrv.sqc`, or the C++ source file `outsrv.sqx`, enter:

```
   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`, on the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the name of the stored procedure DLL, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move the DLL to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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 command file `bldvaemb`. Refer to “Embedded SQL Applications” on page 220 for details.

To call the stored procedure, 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.

User-Defined Functions (UDFs)

The command file bldvaudf, in %DB2PATH%\samples\c, and in %DB2PATH%\samples\cpp, contains the commands to build a UDF.

UDFs cannot contain embedded SQL statements. Therefore, to build a UDF program, you do not need to connect to a database to precompile and bind the program.

The command file takes one parameter, %1, which specifies the name of your source file. It 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.
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c
rem For C++, comment out the above line and uncomment the following:
rem icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.cxx

rem Link the program.
ilink /NOFREE /MAP /NOI /DEBUG /ST:64000 %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 run-time library that is statically linked.
-Gm+	Link with multi-tasking libraries.
-W2	Output warning, error, and severe and unrecoverable error messages.

Compile and Link Options for bldvaudf

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:64000	Specify a stack size of at least 64000.
%1.dll	Create a 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, from the C source file, udf.c, enter:

```
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.

Once you build udf, you can build the client application, calludf, that calls it. DB2 CLI as well as embedded SQL C and C++ versions of this program are provided.

You can build the DB2 CLI calludf program from the calludf.c source file in %DB2PATH%\samples\cli using the command file bldcli.cmd. Refer to "DB2 CLI Applications" on page 215 for details.

You can build the C embedded SQL calludf program from the calludf.sqc source file in %DB2PATH%\samples\c using the command file bldvaemb. Refer to "Embedded SQL Applications" on page 220 for details.

You can build the C++ embedded SQL calludf program from the calludf.sqx source file in %DB2PATH%\samples\cpp using the command file bldvaemb. Refer to "Embedded SQL Applications" on page 220 for details.

To run the UDF, enter:

```
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:

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

You do not have to type the SQL statement in uppercase.

IBM VisualAge C++ for OS/2 Version 4.0

The VisualAge C++ compiler differs from other compilers on OS/2. To compile a program with VisualAge C++ Version 4.0, you must first make a configuration file. See the documentation that comes with the compiler to learn more about this.

DB2 provides configuration files for the different types of DB2 programs you can build with the VisualAge C++ compiler. To use a DB2 configuration file, you first set an environment variable to the program name you wish to compile. Then you compile the program with a command supplied by VisualAge C++. Here are the configuration files provided by DB2, and the sections describing how they can be used to compile your programs:

`api.icc`
: DB2 API configuration file. For details, see “DB2 API Applications” on page 229.

`cli.icc`
: DB2 CLI configuration file. For details, see “DB2 CLI Applications” on page 230.

`clis.icc`
: DB2 CLI stored procedure configuration file. For details, see “DB2 CLI Stored Procedures” on page 232.

`emb.icc`
: Embedded SQL configuration file. For details, see “Embedded SQL Applications” on page 235.

`stp.icc`
: Embedded SQL stored procedure configuration file. For details, see “Embedded SQL Stored Procedures” on page 236.

udf.icc
User-defined function configuration file. For details, see “User-Defined Functions (UDFs)” on page 239.

DB2 API Applications

The configuration file, api.icc, in %DB2PATH%\samples\c, allows you to build DB2 API programs in C. There is also a C++ DB2 API configuration file in %DB2PATH%\samples\cpp. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// api.icc configuration file for DB2 API programs
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export API=prog_name'
// To use on OS/2 and Windows, enter: 'set API=prog_name'
// Then compile the program by entering: 'vacbld api.icc'

if defined( $API )
{
  prog_name = $API
}
else
{
  error "Environment Variable API is not defined."
}

infile = prog_name".c"
util   = "util.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2api.lib"
  option opts = link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
}

option opts
{
  target type(exe) outfile
  {
    source infile
    source util
    source lib
  }
}
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the DB2 API sample program `client` from the source file `client.c`, do the following:

1. Set the API environment variable to the program name by entering:
   ```
   set API=client
   ```
2. If you have an `api.ics` file in your working directory, produced by building a different program with the `api.icc` file, delete the `api.ics` file with this command:
   ```
   del api.ics
   ```

   An existing `api.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld api.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The result is an executable file, `client`. You can run the program by entering the executable name:

```
client
```

DB2 CLI Applications

The configuration file, `cli.icc`, in `%DB2PATH%\samples\cli`, allows you to build DB2 CLI programs. This file can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// cli.icc configuration file for DB2 CLI applications
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export CLI=prog_name'
// To use on OS/2 and Windows, enter: 'set CLI=prog_name'
// Then compile the program by entering: 'vacbld cli.icc'

if defined( $CLI )
{
  prog_name = $CLI
}
else
{
  error "Environment Variable CLI is not defined."
}
```

```
infile   = prog_name".c"
samputil = "samputil.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2cli.lib"
  option opts = link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
}

option opts
{
  target type(exe) outfile
  {
    source infile
    source samputil
    source lib
  }
}
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the DB2 CLI sample program `basiccon` from the source file `basiccon.c`, do the following:

1. Set the CLI environment variable to the program name by entering:

   ```
   set CLI=basiccon
   ```

2. If you have a `cli.ics` file in your working directory, produced by building a different program with the `cli.icc` file, delete the `cli.ics` file with this command:

   ```
   del cli.ics
   ```

   An existing `cli.ics` file produced for the same program you are going to build again does not have to be deleted.

3. Compile the sample program by entering:

   ```
   vacbld cli.icc
   ```

Note: The vacbld command is provided by VisualAge C++ Version 4.0.

The result is an executable file, basiccon. You can run the program by entering the executable name:

```
basiccon
```

DB2 CLI Stored Procedures

The configuration file, clis.icc, in %DB2PATH%\samples\cli, allows you to build DB2 CLI stored procedures. This file can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// clis.icc configuration file for DB2 CLI stored procedures
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export CLIS=prog_name'
// To use on OS/2 and Windows, enter: 'set CLIS=prog_name'
// Then compile the program by entering: 'vacbld clis.icc'

if defined( $CLIS )
{
  prog_name = $CLIS
}
else
{
  error "Environment Variable CLIS is not defined."
}

infile = prog_name".c"
samputil = "samputil.c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else /* if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ ) */
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2cli.lib"
  option opts = link( exportList, expfile ),
              link( libsearchpath, db2path"\\lib" ),
              incl( searchPath, db2path"\\include" )
  cpcmd       = "copy"
  funcdir     = db2path"\\function"
}

option opts
{
  target type(dll) outfile
  {
    source infile
    source samputil
    source lib
  }
}

if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}

run after cpcmd " " outfile " " funcdir
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the DB2 CLI stored procedure `outsrv2` from the source file `outsrv2.c`, do the following:

1. Set the CLIS environment variable to the program name by entering:

   ```
   set CLIS=outsrv2
   ```

2. If you have a `clis.ics` file in your working directory, produced by building a different program with the `clis.icc` file, delete the `clis.ics` file with this command:

   ```
   del clis.ics
   ```

   An existing `clis.ics` file produced for the same program you are going to build again does not have to be deleted.

3. Compile the sample program by entering:

   ```
   vacbld clis.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The stored procedure is copied to the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv2`, you can build the CLI client application `outcli2` that calls the stored procedure. You can build `outcli2` by using the configuration file, `cli.icc`. Refer to “DB2 CLI Applications” on page 230 for details.

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

`outcli2` *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 `outsrv2`, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

The configuration file, emb.icc, in %DB2PATH%\samples\c, allows you to build DB2 embedded SQL applications in C. There is also a C++ configuration file for embedded SQL applications, in %DB2PATH%\samples\cpp. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// emb.icc configuration file for embedded SQL applications
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export EMB=prog_name'
// To use on OS/2 and Windows, enter: 'set EMB=prog_name'
// Then compile the program by entering: 'vacbld emb.icc'

if defined( $EMB )
{
  prog_name = $EMB
}
else
{
  error "Environment Variable EMB is not defined."
}

// To connect to another database, replace "sample"
// For user ID and password, update 'user' and 'passwd'
// and take out the comment in the line: 'run before "embprep "'
dbname = "sample"
user   = ""
passwd = ""

// Precompiling the source program file
run before "embprep " prog_name " " dbname // " " user " " passwd

infile = prog_name".c"
util   = "util.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2api.lib"
  option opts = link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
}

option opts
```

```
{
  target type(exe) outfile
  {
    source infile
    source util
    source lib
  }
}
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the embedded SQL application `updat` from the source file `updat.sqc`, do the following:

1. Set the EMB environment variable to the program name by entering:
   ```
   set EMB=updat
   ```
2. If you have an `emb.ics` file in your working directory, produced by building a different program with the `emb.icc` file, delete the `emb.ics` file with this command:
   ```
   del emb.ics
   ```

   An existing `emb.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld emb.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The result is an executable file, `updat`. You can run the program by entering the executable name:

```
updat
```

Embedded SQL Stored Procedures

The configuration file, `stp.icc`, in `%DB2PATH%\samples\c`, allows you to build DB2 embedded SQL stored procedures in C. There is also a C++ configuration file for embedded SQL stored procedures in `%DB2PATH%\samples\cpp`. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// stp.icc configuration file for embedded SQL stored procedures
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export STP=prog_name'
// To use on OS/2 and Windows, enter: 'set STP=prog_name'
// Then compile the program by entering: 'vacbld emb.icc'

if defined( $STP )
{
```

```
  prog_name = $STP
}
else
{
  error "Environment Variable STP is not defined."
}

// To connect to another database, replace "sample"
// For user ID and password, update 'user' and 'passwd'
// and take out the comment in the line: 'run before "embprep "'
dbname = "sample"
user   = ""
passwd = ""

// Precompiling the source program file
run before "embprep " prog_name " " dbname // " " user " " passwd

infile = prog_name".c"
util   = "util.c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2api.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
  cpcmd       = "copy"
  funcdir     = db2path"\\function"
}

option opts
{
  target type(dll) outfile
  {
    source infile
    source util
    source lib
  }
}

if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}

run after cpcmd " " outfile " " funcdir
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the embedded SQL stored procedure `outsrv` from the source file `outsrv.sqc`, do the following:

1. Set the STP environment variable to the program name by entering:
   ```
   set STP=outsrv
   ```
2. If you have an `stp.ics` file in your working directory, produced by building a different program with the `stp.icc` file, delete the `stp.ics` file with this command:
   ```
   del stp.ics
   ```

   An existing `stp.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld stp.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The stored procedure is copied to the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide.*

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the configuration file `emb.icc`. Refer to "Embedded SQL Applications" on page 235 for details.

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

`outcli` *remote_database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

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

User-Defined Functions (UDFs)

The configuration file, `udf.icc`, in `%DB2PATH%\samples\c`, allows you to build user-defined functions in C. There is also a C++ configuration file for user-defined functions in `%DB2PATH%\samples\cpp`. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// udf.icc configuration file for user-defined functions
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export UDF=prog_name'
// To use on OS/2 and Windows, enter: 'set UDF=prog_name'
// Then compile the program by entering: 'vacbld udf.icc'

if defined( $UDF )
{
  prog_name = $UDF
}
else
{
  error "Environment Variable UDF is not defined."
}

infile = prog_name".c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a", "libdb2apie.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2api.lib", "db2apie.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
  cpcmd       = "copy"
  funcdir     = db2path"\\function"
}

option opts
{
  target type(dll) outfile
  {
    source infile
    source lib
  }
}
if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}

run after cpcmd " " outfile " " funcdir
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the user-defined function program udf from the source file udf.c, do the following:

1. Set the UDF environment variable to the program name by entering:
   ```
   set UDF=udf
   ```
2. If you have a udf.ics file in your working directory, produced by building a different program with the udf.icc file, delete the udf.ics file with this command:
   ```
   del udf.ics
   ```

   An existing udf.ics file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld udf.icc
   ```

   Note: The vacbld command is provided by VisualAge C++ Version 4.0.

The UDF library is copied to the server in the path %DB2PATH%\function.

If necessary, set the file mode for the user-defined function so the DB2 instance can run it.

Once you build udf, you can build the client application, calludf, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI calludf program from the source file calludf.c, in %DB2PATH%\samples\cli, by using the configuration file cli.icc. Refer to "DB2 CLI Applications" on page 230 for details.

You can build the embedded SQL calludf.sqc program from the source file calludf.sqc, in %DB2PATH%\samples\c, by using the configuration file emb.icc. Refer to "Embedded SQL Applications" on page 235 for details.

To call the UDF, run the sample calling application by entering the executable name:
```
calludf
```

The calling application calls functions from the udf library.

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

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

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

IBM VisualAge COBOL for OS/2

This section contains the following topics:
- Using the Compiler
- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures

Using the Compiler

These points will help you use the IBM VisualAge COBOL compiler with DB2.

Workaround for Creating Bind Files

When creating applications using DB2 for OS/2 and IBM Cobol, the DB2 precompiler will often fail to create bind files. This is due to a file handle limit within OS/2.

The fix is to make OS/2 allow more file handles on the machine perfoming the compiling. The line:

```
SET SHELLHANDLESINC=20
```

should be inserted into the CONFIG.SYS file on the machine where DB2 for OS/2 is installed. Alternately, one can use the NODATA option when compiling (this is an IBM Cobol option).

Embedded SQL and DB2 API Calls

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the IBM VisualAge 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 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".
  ```

DB2 API Applications

The command file bldapicb.cmd, in %DB2PATH%\samples\cobol, contains the commands to build a DB2 API program. The parameter, %1, specifies the name of your source file.

```
@echo off
rem  bldapicb command file
rem  Builds a COBOL program that does not contain embedded SQL
rem  Usage: bldapicb <prog_name>

if "%1" == "" goto error

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:64000 /PM:VIO /NOI /DEBUG

goto exit

:error
echo Usage: bldapicb prog_name

:exit
@echo on
```

Compile and Link Options for bldapicb

The command file contains the following compile options:

cob2	The IBM VisualAge 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.
db2api.lib	Link with the DB2 library.
/ST:64000	Specify a stack size of at least 64000.
/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 `client.cbl`, enter:

```
bldapicb client
```

The result is an executable file `client`. You can run the executable file against the SAMPLE database by entering the executable name:

```
client
```

Embedded SQL Applications

The command file `bldibmcb.cmd`, in `%DB2PATH%\samples\cobol`, contains the commands to build an embedded SQL 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:64000 /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 VisualAge 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.
db2api.lib	Link with the DB2 library.
/ST:64000	Specify a stack size of at least 64000.
/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`, from the source file `updat.sqb`, enter:

```
bldibmcb updat
```

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

```
updat
```

Embedded SQL 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:64000 /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 VisualAge 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:64000	Specify a stack size of at least 64000.
/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` stored procedure, from the source file `outsrv.sqb`, enter:

```
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`, on the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the name of the stored procedure DLL, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move the DLL to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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 command file `bldibmcb`. Refer to “Embedded SQL Applications” on page 244 for details.

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

```
outcli
```

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

Micro Focus COBOL

This section contains the following topics:

- Using the Compiler
- DB2 API Applications
- Embedded SQL Applications

- Embedded SQL Stored Procedures

Using the Compiler

DB2 does not support the link386 linker that comes with the Micro Focus COBOL compiler. To link DB2 Micro Focus COBOL programs, you must use the `ilink` linker that is available from IBM compiler products. The `cbllink` command, used in the script files in this section, calls the `ilink` linker.

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, the default.
- 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%
  ```

Calls to all DB2 application programming interfaces 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 run-time 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.

DB2 API Applications

The command file `bldmfapi`, in `%DB2PATH%\samples\cobol_mf`, contains the commands to build a DB2 API program. The parameter, `%1`, specifies the name of your source file.

```
@echo off
rem bldmfapi command file (for programs that do not contain embedded SQL)
rem Usage: bldmfapi <prog_name>

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
@echo on
```

Compile and Link Options for bldmfapi

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 M. F. COBOL.

Refer to your compiler documentation for additional compiler options.

To build the sample program client, from the source file client.cbl, enter:

```
bldmfapi client
```

The result is an executable file, client.exe. You can run the executable file against the sample database by entering the executable name (without the extension):

```
client
```

Embedded SQL Applications

The command file bldmfcob, in %DB2PATH%\samples\cobol_mf, contains the commands to build an embedded SQL 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 Builds an embedded SQL 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 M. F. COBOL. Refer to your compiler documentation for additional compiler options.

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

```
bldmfcob updat
```

The result is an executable file, updat.exe. You can run the executable file against the sample database by entering the executable name (without the extension):

```
updat
```

Embedded SQL Stored Procedures

The command file bldmfcbs, in %DB2PATH%\samples\cobol_mf, contains the commands to build an embedded SQL 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 Builds an embedded SQL stored procedure using the 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 bldmfcbs

The command file contains the following compile option:

`cobol`	The Micro Focus COBOL compiler.

Compile and Link Options for bldmfcbs

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 M. F. COBOL.

Refer to your compiler documentation for additional compiler options.

To build the stored procedure `outsrv` from the source file `outsrv.sqb`, 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`, on the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the name of the stored procedure DLL, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move the DLL to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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 “Embedded SQL Applications” on page 251 for details.

To run the stored procedure, 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.

FORTRAN 77

This section contains the following topics:

- Using the Compiler
- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures

Using the 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 *Administrative 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.

DB2 API Applications

The command file bldapi, in %DB2PATH%\samples\fortran, contains the commands to build a DB2 API program. The parameter, %1, specifies the name of your source file.

```
@echo off
rem  bldapi command file
rem  Builds a FORTRAN DB2 API program
rem  Usage: bldapi <prog_name>
```

```
if "%1" == "" goto error

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=64000

goto exit

:error
echo Usage: bldapi prog_name

:exit
@echo on
```

Compile and Link Options for bldapi

The command file contains the following compile options:

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

Compile and Link Options for bldapi

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.

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=64000
Specify a stack size of at least 64000.

Refer to your compiler documentation for additional compiler options.

To build the sample program, `client`, from the source file, `client.f`, enter:

```
bldapi client
```

The result is an executable file, `client`. You can run the executable file against the SAMPLE database by entering the executable name:

```
client
```

Embedded SQL Applications

The command file, `bldfor`, in `%DB2PATH%\samples\fortran`, contains the commands to build an embedded SQL 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=64000

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 Include full debugging information.

d2 Perform run-time checking.

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.

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=64000
Specify a stack size of at least 64000.

Refer to your compiler documentation for additional compiler options.

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

```
bldfor updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

```
   c$pragma aux <stored_proc_name> parm caller[]  (data_reference,
   c                           data_reference, data_reference, data_reference)
```

`caller []` allows the parameters to be put on a stack instead of in registers. This complies with the calling convention that DB2 APIs use.

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 embedded SQL stored procedure
rem Usage: bldforsr <stored_proc_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=64000
```

```
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 Create a 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 library.

option

stack=64000
Specify a stack size of at least 64000.

Refer to your compiler documentation for additional compiler options.

To build the `outsrv.sqf` stored procedure, enter:

```
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=64000
```

The command file copies the stored procedure DLL, `outsrv.dll`, on the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the name of the stored procedure DLL, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move the DLL to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the "Stored Procedures" chapter in the *Application Development Guide*.

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

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 command file `bldfor`. Refer to “Embedded SQL Applications” on page 258 for details.

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

```
outcli
```

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

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`, enter:

```
   updat
```

For further information on REXX and DB2, refer to the chapter, ″Programming in REXX″, in the *Application Development Guide*.

Chapter 9. Building Silicon Graphics IRIX Applications

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

For the latest DB2 application development updates for Silicon Graphics IRIX, visit the Web page at:

```
http://www.software.ibm.com/data/db2/udb/ad
```

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

Also, since you will be accessing a database on the server from a remote client that is running on a different operating system, you need to bind the database utilities, including the DB2 CLI, to the database. See “Binding” on page 41 for more information.

DB2 Library Support

Silicon Graphics IRIX provides three separate and incompatible object types: o32 (the default), n32 (the new 32 object type), and n64 (the 64 object type). DB2 does not yet support n64, but does support the o32 and n32 object types.

The operating system also has two separate and incompatible versions of thread APIs: the sproc interface and the POSIX threads API. DB2 provides support for the POSIX threads API.

Applications which use the sproc interface can use the non-threaded versions of the DB2 library, `libdb2`, which are not thread safe. Care should be taken when using the sproc interface because `libdb2` is not sproc safe.

To accommodate this range of functionality, DB2 provides the following library support:

`lib/libdb2.so`
: o32 with no threads

`lib/libdb2_th.so`
: o32 with POSIX threads

`lib32/libdb2.so`
: n32 with no threads

`lib32/libdb2_th.so`
: n32 with POSIX threads

To use the `n32` object type, programs must be compiled and linked with the `-n32` option, as well as linked with the `lib32/libdb2.so` or `lib32/libdb2_th.so` library. To use the default `o32` object type, programs must be linked to either the `lib/libdb2.so` or `lib/libdb2_th.so` libraries without the `-n32` option.

MIPSpro C

This section explains how to use MIPSpro C with the following kinds of DB2 interfaces:

- DB2 APIs
- DB2 CLI
- Embedded SQL

DB2 API Applications

The script file `bldccapi`, in `sqllib/samples/c`, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldccapi script file
# Builds a DB2 API program
# Usage: bldccapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.c error-checking utility.
cc -I$DB2PATH/include -c util.c
# Compile util.c for n32 object type support.
# cc -n32 -I$DB2PATH/include -c util.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c
# Compile the program for n32 object type support.
# cc -n32 -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L$DB2PATH/lib -rpath $DB2PATH/lib -lm -ldb2
# Link the program for n32 object type support
# cc -n32 -o $1 $1.o util.o -L$DB2PATH/lib32 -rpath $DB2PATH/lib32 -lm -ldb2
```

Compile and Link Options for bldccapi

The script file contains the following compile options:

cc Use the C compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldccapi

The script file contains the following link options:

`cc`	Use the compiler as a front end for the linker.
`-o $1`	Specify the executable.
`$1.o`	Include the program object file.
`util.o`	Include the utility object file for error checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
`-rpath $DB2PATH/lib`	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
`-lm`	Link with the math library.
`-ldb2`	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.c`, enter:

```
bldccapi client
```

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

```
client userid password
```

where

userid Is a valid user ID.

password
Is a valid password.

DB2 CLI Applications

The script file `bldcli` in `sqllib/samples/cli` contains the commands to build a DB2 CLI program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldcli script file -- Silicon Graphics IRIX
# Builds a CLI program with MIPSpro C.
# Usage: bldcli <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc -I$DB2PATH/include -c samputil.c
# Compile the utility for n32 object type support.
# cc -n32 -I$DB2PATH/include -c samputil.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c
# Compile the program for n32 object type support.
# cc -n32 -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o samputil.o -L$DB2PATH/lib -rpath $DB2PATH/lib -lm -ldb2
# Link the program for n32 object type support.
# cc -n32 -o $1 $1.o samputil.o -L$DB2PATH/lib32 -rpath $DB2PATH/lib32 -lm -ldb2
```

Compile and Link Options for bldcli

The script file contains the following compile options:

cc Use the C compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldcli

The script file contains the following link options:

`cc`	Use the compiler as a front end for the linker.
`-o $1`	Specify the executable.
`$1.o`	Include the program object file.
`samputil.o`	Include the utility object file for error checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
`-rpath $DB2PATH/lib`	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
`-lm`	Link with the math library.
`-ldb2`	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `basiccon` from the source file `basiccon.c`, enter:

```
   bldcli basiccon
```

The result is an executable file `basiccon`. You can run the executable file by entering:

```
   basiccon userid password
```

where

userid Is a valid user ID.

password
Is a valid password.

DB2 CLI Client Applications for Stored Procedures

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

To build the DB2 CLI stored procedure `outsrv2` on a DB2-supported platform server, refer to the ″Building Applications″ chapter for that platform in this book. For other servers accessible by DB2 clients, see “Supported Servers” on page 6.

Once you build the stored procedure `outsrv2`, you can build the client application that calls the stored procedure, `outcli2`, from the source file `outcli2.c`, by using the script file `bldcli`. Refer to “DB2 CLI Applications” on page 268 for details.

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

```
outcli2 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 `outsrv2`, which gives it a value, and then returns the variable to the client application.

DB2 CLI Client Applications for UDFs

User-defined functions (UDFs) are your own scalar and table functions that you compile and store on the server. The server runs on another platform. To build the user-defined function program, `udf`, on a DB2-supported platform server, refer to the ″Building Applications″ chapter for that platform in this book. For other servers accessible by DB2 clients, see “Supported Servers” on page 6.

Once you build `udf`, you can build the DB2 CLI client application, `calludf`, that calls it, from the `calludf.c` source file in `sqllib/samples/cli`, using the DB2 CLI script file `bldcli`. Refer to “DB2 CLI Applications” on page 268 for details.

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

```
calludf userid password
```

where

userid Is a valid user ID.

password
Is a valid password.

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

Embedded SQL Applications

The script file bldcc, in sqllib/samples/c, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$HOME/sqllib

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

# Compile the util.c error-checking utility.
cc -I$DB2PATH/include -c util.c
# Compile util.c for n32 object type support.
# cc -n32 -I$DB2PATH/include -c util.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c
# Compile the program for n32 object type support.
# cc -n32 -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L$DB2PATH/lib -rpath $DB2PATH/lib -lm -ldb2
# Link the program for n32 object type support.
# cc -n32 -o $1 $1.o util.o -L$DB2PATH/lib32 -rpath $DB2PATH/lib32 -lm -ldb2
```

Compile and Link Options for bldcc

The script file contains the following compile options:

cc	Use the C compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

cc	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
util.o	Include the utility object file for error checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-rpath $DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-lm	Link with the math library.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldcc updat remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

```
updat userid password
```

where

userid Is a valid user ID.

password
Is a valid password.

Embedded SQL Client Applications for Stored Procedures

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

To build the embedded SQL stored procedure `outsrv` on a DB2-supported platform server, refer to the ″Building Applications″ chapter for that platform in this book. For other servers accessible by DB2 clients, see “Supported Servers” on page 6.

Once you build the stored procedure `outsrv`, you can build the client application that calls the stored procedure. You can build `outcli` from the source file `outcli.sqc`, by using the script file `bldcc`. Refer to “Embedded SQL Applications” on page 272 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

Client Applications for User-defined Functions (UDFs)

User-defined functions (UDFs) are your own scalar and table functions that you compile and store on the server. The server runs on another platform. To build the user-defined function program, `udf`, on a DB2-supported platform server, refer to the ″Building Applications″ chapter for that platform in this book. For other servers accessible by DB2 clients, see “Supported Servers” on page 6.

Once you build `udf`, you can build the embedded SQL client application, `calludf`, that calls it, from the `calludf.sqc` source file in `sqllib/samples/c` using the script file `bldcc`. Refer to “Embedded SQL Applications” on page 272 for details.

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

```
calludf userid password
```

where

userid Is a valid user ID.

password
Is a valid password.

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

Multi-threaded Applications

Multi-threaded applications on Silicon Graphics IRIX need to be linked with the POSIX threads version of the DB2 library for either the `o32` or `n32` object types, using the `-ldb2_th` link option.

The script file `bldccmt`, in `sqllib/samples/c`, contains the commands to build an embedded SQL multi-threaded program. If you want to build a DB2 API or DB2 CLI multi-threaded program, comment out the connect, precompile, bind, and disconnect commands. For DB2 CLI, also substitute the `samputil.c` and `samputil.o` files for `util.c` and `util.o`.

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

```
#! /bin/ksh
# bldccmt script file -- Silicon Graphics IRIX
# Builds a multi-threaded C program containing embedded SQL
# Usage: bldccmt <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$HOME/sqllib

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

# Compile the util.c error-checking utility.
cc -I$DB2PATH/include -c util.c
# Compile util.c for n32 object type support.
# cc -n32 -I$DB2PATH/include -c util.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c
# Compile the program for n32 object type support.
# cc -n32 -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L$DB2PATH/lib -rpath $DB2PATH/lib -lm -ldb2_th
# Link the program for n32 object type support.
# cc -n32 -o $1 $1.o util.o -L$DB2PATH/lib32 -rpath $DB2PATH/lib32 -lm -ldb2_th
```

Besides the `-ldb2_th` link option, discussed above, the other compile and link options are the same as those used for the embedded SQL script file, `bldcc`. For information on these options, see "Embedded SQL Applications" on page 272.

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

```
   bldccmt thdsrver
```

The result is an executable file, `thdsrver`.

To run the executable file against the sample database, enter the executable name:

```
thdsrver
```

MIPSpro C++

This section includes the following topics:

- DB2 API Applications
- Embedded SQL Applications

DB2 API Applications

The script file bldCCapi, in sqllib/samples/cpp, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldCCapi script file
# Builds a DB2 API program
# Usage: bldCCapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.c error-checking utility.
CC -I$DB2PATH/include -c util.c
# Compile util.c for n32 object type support.
# CC -n32 -I$DB2PATH/include -c util.c

# Compile the program.
CC -I$DB2PATH/include -c $1.c
# Compile the program for n32 object type support.
# CC -n32 -I$DB2PATH/include -c $1.c

# Link the program.
CC -o $1 $1.o util.o -L$DB2PATH/lib -rpath $DB2PATH/lib -lm -ldb2
# Link the program for n32 object type support
# CC -n32 -o $1 $1.o util.o -L$DB2PATH/lib32 -rpath $DB2PATH/lib32 -lm -ldb2
```

Compile and Link Options for bldCCapi

The script file contains the following compile options:

`CC` Use the C++ compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

`CC` Use the compiler as a front end for the linker.

`-o $1` Specify the executable.

`$1.o` Include the program object file.

`util.o` Include the utility object file for error checking.

`-L$DB2PATH/lib`
Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.

`-rpath $DB2PATH/lib`
Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.

`-lm` Link with the math library.

`-ldb2` Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.C`, enter:

```
bldCCapi client
```

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

```
client userid password
```

where

userid Is a valid user ID.

password
Is a valid password.

Embedded SQL Applications

The script file bldCC, in sqllib/samples/cpp, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

# Compile the util.c error-checking utility.
CC -I$DB2PATH/include -c util.c
# Compile util.c for n32 object type support.
# CC -n32 -I$DB2PATH/include -c util.c

# Compile the program.
CC -I$DB2PATH/include -c $1.c
# Compile the program for n32 object type support.
# CC -n32 -I$DB2PATH/include -c $1.c

# Link the program.
CC -o $1 $1.o util.o -L$DB2PATH/lib -rpath $DB2PATH/lib -lm -ldb2
# Link the program for n32 object type support
# CC -n32 -o $1 $1.o util.o -L$DB2PATH/lib32 -rpath $DB2PATH/lib32 -lm -ldb2
```

Compile and Link Options for bldCC

The script file contains the following compile options:

CC	Use the C++ compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

CC	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
util.o	Include the utility object file for error checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-rpath $DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-lm	Link with the math library.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldCC updat remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

```
updat userid password
```

where

userid Is a valid user ID.

password
Is a valid password.

Embedded SQL Client Applications for Stored Procedures

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

To build the embedded SQL stored procedure `outsrv` on a DB2-supported platform server, refer to the ″Building Applications″ chapter for that platform in this book. For other servers accessible by DB2 clients, see “Supported Servers” on page 6.

Once you build the stored procedure `outsrv`, you can build the client application that calls the stored procedure. You can build `outcli` from the source file `outcli.sqC`, by using the script file `bldCC`. Refer to “Embedded SQL Applications” on page 279 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

Embedded SQL Client Application for UDFs

User-defined functions (UDFs) are your own scalar functions that you compile and store on the server. The server runs on another platform. To build the user-defined function program, `udf`, on a DB2-supported platform server, refer to the ″Building Applications″ chapter for that platform in this book. For other servers accessible by DB2 clients, see “Supported Servers” on page 6.

Once you build `udf`, you can build the embedded SQL client application, `calludf`, that calls it, from the `calludf.sqC` source file in `sqllib/samples/cpp` using the script file `bldCC`. Refer to “Embedded SQL Applications” on page 279 for details.

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

```
calludf userid password
```

where

userid Is a valid user ID.

password
Is a valid password.

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

Multi-threaded Applications

Multi-threaded applications on Silicon Graphics IRIX need to be linked with the POSIX threads version of the DB2 library for either the `o32` or `n32` object types, using the `-ldb2_th` link option.

The script file `bldCCmt`, in `sqllib/samples/cpp`, contains the commands to build an embedded SQL multi-threaded program. If you want to build a DB2 API multi-threaded program, comment out the connect, precompile, bind, and disconnect commands.

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

```
#! /bin/ksh
# bldCCmt script file -- Silicon Graphics IRIX
# Builds a multi-threaded C++ program containing embedded SQL
# Usage: bldCCmt <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the instance path.
DB2PATH=$HOME/sqllib

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

# Compile the util.c error-checking utility.
CC -I$DB2PATH/include -c util.c
# Compile util.c for n32 object type support.
# CC -n32 -I$DB2PATH/include -c util.c

# Compile the program.
CC -I$DB2PATH/include -c $1.c
# Compile the program for n32 object type support.
# CC -n32 -I$DB2PATH/include -c $1.c

# Link the program.
CC -o $1 $1.o util.o -L$DB2PATH/lib -rpath $DB2PATH/lib -lm -ldb2_th
# Link the program for n32 object type support.
# CC -n32 -o $1 $1.o util.o -L$DB2PATH/lib32 -rpath $DB2PATH/lib32 -lm -ldb2_th
```

Besides the `-ldb2_th` link option, discussed above, the other compile and link options are the same as those used for the embedded SQL script file, `bldCC`. For information on these options, see “Embedded SQL Applications” on page 279.

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

```
   bldCCmt thdsrver
```

The result is an executable file, `thdsrver`.

To run the executable file against the sample database, enter the executable name:

```
thdsrver
```

MIPSpro Fortran-77

This section includes the following topics:

- DB2 API Applications
- Embedded SQL Applications

DB2 API Applications

The script file, bldf77api, in sqllib/samples/fortran, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldf77api script file
# Builds a DB2 API program
# Usage: bldf77api <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.f error-checking utility.
f77 -I$DB2PATH/include -w -c util.f
# Compile util.f for n32 object type support.
# f77 -n32 -I$DB2PATH/include -w -c util.f

# Compile the program.
f77 -I$DB2PATH/include -w -c $1.f
# Compile the program for n32 object type support.
# f77 -n32 -I$DB2PATH/include -w -c $1.f

# Link the program.
f77 -o $1 $1.o util.o -L$DB2PATH/lib -rpath $DB2PATH/lib -ldb2
# Link the program for n32 object type support.
# f77 -n32 -o $1 $1.o util.o -L$DB2PATH/lib32 -rpath $DB2PATH/lib32 -ldb2
```

Compile and Link Options for bldf77api

The script file contains the following compile options:

f77	The Fortran compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
-w	Suppress warning messages.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

f77	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
util.o	Include the utility object file for error-checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`.
-rpath $DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.f`, enter:

```
bldf77api client
```

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

```
client
```

Embedded SQL Applications

The script file, `bldf77`, in `sqllib/samples/fortran`, contains the commands to build an embedded SQL program.

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

required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default sample database.

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

# Compile the util.f error-checking utility.
f77 -I$DB2PATH/include -w -c util.f
# Compile util.f for n32 object type support.
# f77 -n32 -I$DB2PATH/include -w -c util.f

# Compile the program.
f77 -I$DB2PATH/include -w -c $1.f
# Compile the program for n32 object type support.
# f77 -n32 -I$DB2PATH/include -w -c $1.f

# Link the program.
f77 -o $1 $1.o util.o -L$DB2PATH/lib -rpath $DB2PATH/lib -ldb2
# Link the program for n32 object type support.
# f77 -n32 -o $1 $1.o util.o -L$DB2PATH/lib32 -rpath $DB2PATH/lib32 -ldb2
```

Compile and Link Options for bldf77

The script file contains the following compile options:

`f77`	The Fortran compiler.
`-I$DB2PATH/include`	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
`-w`	Suppress warning messages.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

`f77`	Use the compiler as a front end for the linker.
`-o $1`	Specify the executable.
`$1.o`	Include the program object file.
`util.o`	Include the utility object file for error checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`.
`-rpath $DB2PATH/lib`	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
`-ldb2`	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldf77 updat remote_database userid password
```

where

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

userid Is a valid user ID.

password
: Is a valid password.

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

```
updat userid password
```

where

userid Is a valid user ID.

password
Is a valid password.

Embedded SQL Client Applications for Stored Procedures

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

To build the Fortran embedded SQL stored procedure `outsrv` on a DB2-supported platform server, refer to the ″Building Applications″ chapter for that platform in this book. For other servers accessible by DB2 clients, see “Supported Servers” on page 6.

Once you build the stored procedure `outsrv`, you can build `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldf77`. Refer to “Embedded SQL Applications” on page 285 for details.

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

```
outcli
```

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

Chapter 10. Building Solaris Applications

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

For the latest DB2 application development updates for Solaris, visit the Web page at:

```
http://www.software.ibm.com/data/db2/udb/ad
```

Note: Because the DB2 library, libdb2.so, is threadsafe, and due to the way threads are implemented on Solaris, you might consider using the -mt multi-threaded option when linking executables also linked with -ldb2. If the -mt switch is not used, the application may see an error such as the following when the application is run:

```
libc internal error: _rmutex_unlock: rmutex not held
```

SPARCompiler C

This section explains how to use SPARCompiler C with the following kinds of DB2 interfaces:

- DB2 APIs
- DB2 CLI
- Embedded SQL

DB2 API Applications

The script file, `bldccapi`, in `sqllib/samples/c`, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldccapi script file
# Builds a C DB2 API program.
# Usage:  bldccapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.c error-checking utility.
cc -I$DB2PATH/include -c util.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
```

Compile and Link Options for bldccapi

The script file contains the following compile options:

`cc` The C compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldccapi

The script file contains the following link options:

`cc`	Use the compiler as a front end for the linker.
`-o $1`	Specify the executable.
`$1.o`	Include the program object file.
`util.o`	Include the utility object file for error checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
`-R$DB2PATH/lib`	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
`-ldb2`	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.c`, enter:

```
bldccapi client
```

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

```
client
```

DB2 CLI Applications

The script file `bldcli` in `sqllib/samples/cli` contains the commands to build a DB2 CLI program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldcli script file -- Solaris
# Builds a DB2 CLI program.
# Usage: bldcli <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc -I$DB2PATH/include -c samputil.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o samputil.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcli

The script file contains the following compile options:

`cc` Use the C compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

The script file contains the following link options:

`cc` Use the compiler as a front end for the linker.

`-o $1` Specify the executable program.

`$1.o` Include the program object file.

`samputil.o`
Include the utility object file for error checking.

`-L$DB2PATH/lib`
Specify the location of the DB2 static and shared libraries at link-time. For example, `$HOME/sqllib/lib`

`-R$DB2PATH/lib`
Specify the location of the DB2 shared libraries at run-time. For example, `$HOME/sqllib/lib`

`-ldb2` Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

To build the sample program basiccon from the source file basiccon.c, enter:

```
bldcli basiccon
```

The result is an executable file basiccon. You can run the executable file by entering the executable name:

```
basiccon
```

DB2 CLI Stored Procedures

The script file bldclisp in sqllib/samples/cli contains the commands to build a DB2 CLI stored procedure. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldclisp script file -- Solaris
# Builds a DB2 CLI stored procedure.
# Usage: bldclisp <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the error-checking utility.
cc -Kpic -I$DB2PATH/include -c samputil.c

# Compile the program.
cc -Kpic -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o samputil.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -G

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldclisp

The script file contains the following compile options:

cc The C compiler.

-Kpic Generate position-independent code for shared libraries.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: $HOME/sqllib/include.

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldclisp

The script file contains the following link options:

cc	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
samputil.o	Include the utility object file for error-checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.
-G	Generate a shared library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `outsrv2` from source file `outsrv2.c`, enter:

```
bldclisp outsrv2
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure outsrv2, you can build the CLI client application outcli2.c that calls the stored procedure. You can build outcli2 by using the script file bldcli. Refer to “DB2 CLI Applications” on page 291 for details.

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

```
   outcli2 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 outsrv2, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

The script file, bldcc, in sqllib/samples/c, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Connect to a database.
if (($# < 2))
then
   db2 connect to sample
```

```
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

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

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

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
cc -I$DB2PATH/include -c util.c

# Compile the program.
cc -I$DB2PATH/include -c $1.c

# Link the program.
cc -o $1 $1.o util.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcc

The script file contains the following compile options:

`cc` The C compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldcc
The script file contains the following link options:

cc	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
util.o	Include the utility object file for error-checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldcc updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

# Disconnect from the database.
db2 connect reset

# Compile the util.c error-checking utility.
cc -Kpic -I$DB2PATH/include -c util.c

# Compile the program.
cc -Kpic -I$DB2PATH/include -c $1.c

# Link the program and create a shared library
cc -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldccsrv

The script file contains the following compile options:

cc The C compiler.

-Kpic Generate position-independent code for shared libraries.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `-I$DB2PATH/include`

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldccsrv

The script file contains the following link options:

cc	Use the compiler as a front end for the linker.
-G	Generate a shared library.
-o $1	Specify the executable.
$1.o	Include the program object file.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldccsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv`, you can build the client application, `outcli`, that calls the stored procedure. You can build `outcli` using the script file `bldcc`. Refer to “Embedded SQL Applications” on page 295 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

User-Defined Functions (UDFs)

The script file, `bldccudf`, in `sqllib/samples/c`, contains the commands to build a UDF. UDFs do not contain embedded SQL statements. Therefore, to build a UDF program, you do not connect to a database or precompile and bind the program.

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

```
#! /bin/ksh
# bldccudf script file
# Builds a C user-defined function library.
# Usage: bldccudf <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the program.
cc -Kpic -I$DB2PATH/include -c $1.c

# Link the program and create a shared library.
cc -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -ldb2apie -G

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldccudf

The script file contains the following compile options:

cc
The C compiler.

-Kpic
Generate position-independent code for shared libraries.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

-c
Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldccudf

The script file contains the following link options:

cc	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.
-ldb2apie	Link with the DB2 API Engine library to allow the use of LOB locators.
-G	Generate a shared library.

Refer to your compiler documentation for additional compiler options.

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

```
bldccudf udf
```

The script file copies the UDF to the server in the path `sqllib/function`.

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

Once you build `udf`, you can build the client application, `calludf`, that calls it. DB2 CLI and embedded SQL versions of this program are provided. You can build the DB2 CLI `calludf` program from the `calludf.c` source file in `sqllib/samples/cli` using the script file `bldcli`. Refer to “DB2 CLI Applications” on page 291 for details.

You can build the embedded SQL `calludf` program from the `calludf.sqc` source file in `sqllib/samples/c` using the script file `bldcc`. Refer to “Embedded SQL Applications” on page 295 for details.

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

```
calludf
```

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

Multi-threaded Applications

Multi-threaded applications using SPARCompiler C on Solaris need to be compiled and linked with -mt. This will pass -D_REENTRANT to the preprocessor, and -lthread to the linker. Posix threads also require -lpthread to be passed to the linker. In addition, using the compiler option -D_POSIX_PTHREAD_SEMANTICS allows posix variants of functions such as getpwnam_r().

The script file, bldccmt, in sqllib/samples/c, contains the commands to build an embedded SQL multi-threaded program. If you want to build a DB2 API or DB2 CLI multi-threaded program, comment out the connect, precompile, bind, and disconnect commands. For DB2 CLI, also substitute the samputil.c and samputil.o files for util.c and util.o.

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

```
#! /bin/ksh
# bldccmt script file -- Solaris
# Builds an embedded SQL multi-threaded program.
# Usage: bldccmt <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

# Compile the util.c error-checking utility.
cc -mt -D_POSIX_PTHREAD_SEMANTICS -I$DB2PATH/include -c util.c
# Compile the program.
```

```
cc -mt -D_POSIX_PTHREAD_SEMANTICS -I$DB2PATH/include -c $1.c
# Link the program
cc -mt -o $1 $1.o util.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -lpthread
```

Besides the -mt, -D_POSIX_PTHREAD_SEMANTICS, and -lpthread options, discussed above, the other compile and link options are the same as those used for the embedded SQL script file, bldcc. For information on these options, see “Embedded SQL Applications” on page 295.

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

```
   bldccmt thdsrver
```

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

```
   thdsrver
```

SPARCompiler C++

This section includes the following topics:

- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)

DB2 API Applications

The script file, bldCCapi, in sqllib/samples/cpp, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldCCapi script file
# Builds a C++ DB2 API program.
# Usage:  bldCCapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.C error-checking utility.
CC -I$DB2PATH/include -c util.C
# Compile the program.
CC -I$DB2PATH/include -c $1.C
# Link the program.
CC -o $1 $1.o util.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
```

Compile and Link Options for bldCC	
The script file contains the following compile options:	
CC	The C++ compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.
The script file contains the following link options:	
CC	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
util.o	Include the utility object file for error-checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

To build the sample program `client` from the source file `client.C`, enter:

```
bldCCapi client
```

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

```
client
```

Embedded SQL Applications

The script file, `bldCC`, in `sqllib/samples/cpp`, contains the commands to build an embedded SQL application.

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

required. Database name, user ID, and password are optional. If no database name is supplied, the program uses the default sample database.

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

# Compile the util.C error-checking utility.
CC -I$DB2PATH/include -c util.C
# Compile the program.
CC -I$DB2PATH/include -c $1.C
# Link the program.
CC -o $1 $1.o util.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
```

Compile and Link Options for bldCC

The script file contains the following compile options:

CC The C++ compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: $HOME/sqllib/include

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldCC
The script file contains the following link options:

CC Use the compiler as a front end for the linker.

-o $1 Specify the executable.

$1.o Include the program object file.

util.o Include the utility object file for error-checking.

-L$DB2PATH/lib
Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.

-R$DB2PATH/lib
Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.

-ldb2 Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldCC updat
```

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

```
updat
```

Embedded SQL Stored Procedures

Note: Please see the information for building C++ stored procedures and UDFs in “C++ Considerations for UDFs and Stored Procedures” on page 55.

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

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

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

# Compile the util.C error-checking utility.
CC -Kpic -I$DB2PATH/include -c util.C
# Compile the program.
# Ensure the program is coded with extern "C".
CC -Kpic -I$DB2PATH/include -c $1.C
# Link the program and create a shared library
CC -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldCCsrv

The script file contains the following compile options:

CC
The C++ compiler.

-Kpic
Generate position-independent code for shared libraries.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `-I$DB2PATH/include`

-c
Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldCCsrv

The script file contains the following link options:

`CC`	Use the compiler as a front end for the linker.
`-G`	Generate a shared library.
`-o $1`	Specify the executable.
`$1.o`	Include the program object file.
`-L$DB2PATH/lib`	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
`-R$DB2PATH/lib`	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
`-ldb2`	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

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

```
bldCCsrv outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldCC`. Refer to "Embedded SQL Applications" on page 305 for details.

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

```
outcli remote_database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

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

User-Defined Functions (UDFs)

Note: Please see the information for building C++ UDFs and stored procedures in "C++ Considerations for UDFs and Stored Procedures" on page 55.

The script file, `bldCCudf`, in `sqllib/samples/cpp`, contains the commands to build a UDF. UDFs do not contain embedded SQL statements. Therefore, to build a UDF program, you do not connect to a database or precompile and bind the program.

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

```
#! /bin/ksh
# bldCCudf script file
# Builds a C++ user-defined function library.
# Usage: bldCCudf <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the program.
CC -Kpic -I$DB2PATH/include -c $1.c

# Link the program and create a shared library.
CC -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldCCudf

The script file contains the following compile options:

CC	The C++ compiler.
-Kpic	Generate position-independent code for shared libraries.
-I$DB2PATH/include	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldCCudf	
The script file contains the following link options:	
CC	Use the compiler as a front end for the linker.
-G	Generate a shared library.
-o $1	Specify the executable.
$1.o	Include the program object file.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.
-ldb2apie	Link with the DB2 API Engine library to allow the use of LOB locators.
Refer to your compiler documentation for additional compiler options.	

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

```
bldCCudf udf
```

The script file copies the UDF to the server in the path `sqllib/function`.

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

Once you build `udf`, you can build the client application, `calludf`, that calls it. You can build the `calludf` program from the `calludf.sqC` source file in `sqllib/samples/cpp`, using the script file `bldCC`. Refer to "Embedded SQL Applications" on page 305 for details.

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

```
calludf
```

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

Multi-threaded Applications

Multi-threaded applications using SPARCompiler C++ on Solaris need to be compiled and linked with `-mt`. This will pass `-D_REENTRANT` to the preprocessor, and -lthread to the linker. POSIX threads also require -lpthread to be passed to the linker. In addition, using the compiler option -D_POSIX_PTHREAD_SEMANTICS allows POSIX variants of functions such as getpwnam_r().

The script file, bldCCmt, in sqllib/samples/cpp, contains the commands to build an embedded SQL multi-threaded program. If you want to build a DB2 API multi-threaded program, comment out the connect, precompile, bind, and disconnect commands.

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

```
#! /bin/ksh
# bldCCmt script file -- Solaris
# Builds an embedded SQL multi-threaded program.
# Usage: bldCCmt <_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

# Compile the util.C error-checking utility.
CC -mt -D_POSIX_PTHREAD_SEMANTICS -I$DB2PATH/include -c util.C
# Compile the program.
CC -mt -D_POSIX_PTHREAD_SEMANTICS -I$DB2PATH/include -c $1.C
# Link the program
CC -mt -o $1 $1.o util.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -lpthread
```

Besides the -mt, -D_POSIX_PTHREAD_SEMANTICS, and -lpthread options, discussed above, the other compile and link options are the same as those used for the embedded SQL script file, bldCC. For information on these options, see “Embedded SQL Applications” on page 305.

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

```
bldCCmt thdsrver
```

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

```
thdsrver
```

Micro Focus COBOL

This section contains the following topics:

- Using the Compiler
- DB2 API Applications
- DB2 Embedded SQL Applications

Using the Compiler

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

- When you precompile your application using the command line processor command db2 prep, use the target mfcob option (the default).
- In order to use the built-in precompiler front-end, run-time interpreter or Animator debugger, add the DB2 Generic API entry points to the Micro Focus run-time module rts32 by executing the MKRTS command provided by Micro Focus, as follows:
  1. Log in as root.
  2. Execute MKRTS with the arguments supplied in the following directory:
     ```
     /opt/IBMdb2/V6.1/lib/db2mkrts.args
     ```
- You must include the DB2 COBOL COPY file directory in the Micro Focus COBOL environment variable COBCPY. The COBCPY environment variable specifies the location of COPY files. The DB2 COPY files for Micro Focus COBOL reside in sqllib/include/cobol_mf under the database instance directory.

  To include the directory, enter:
  ```
  export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
  ```

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

DB2 API Applications

The script file bldmfapi, in sqllib/samples/cobol_mf, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldmfapi script file.
# Usage: bldmfapi <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY

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

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

# Link the program.
cob -x $1.o checkerr.o -L$DB2PATH/lib -ldb2 -ldb2gmf
```

Compile and Link Options for bldmfapi

The script file contains the following compile options:

cob	The Micro Focus COBOL compiler.
-cx	Compile to object module.

The script file contains the following link options:

cob	Use the compiler as a front end for the linker.
-x	Specify an executable program.
$1.o	Include the program object file.
checkerr.o	Include the utility object file for error-checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: $HOME/sqllib/lib.
-ldb2	Link with the DB2 library.
-ldb2gmf	Link with the DB2 exception-handler library for Micro Focus COBOL.

Refer to your compiler documentation for additional compiler options.

To build the sample program `client` from the source file `client.cbl`, enter:

```
bldmfapi client
```

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

```
client
```

Embedded SQL Applications

The script file `bldmfcc`, in `sqllib/samples/cobol_mf`, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY

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

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

# Link the program.
cob -x $1.o checkerr.o -L$DB2PATH/lib -ldb2 -ldb2gmf
```

Compile and Link Options for bldmfcc
The script file contains the following compile options: **cob** The Micro Focus COBOL compiler. **-cx** Compile to object module.
The script file contains the following link options: **cob** Use the compiler as a front end for the linker. **-x** Specify an executable program. **$1.o** Include the program object file. **checkerr.o** Include the utility object file for error-checking. **-L$DB2PATH/lib** Specify the location of the DB2 static and shared libraries at link-time. For example: $HOME/sqllib/lib. **-ldb2** Link with the DB2 library. **-ldb2gmf** Link with the DB2 exception-handler library for Micro Focus COBOL. Refer to your compiler documentation for additional compiler options.

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

```
bldmfcc updat
```

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

```
updat
```

SPARCompiler Fortran

This section includes the following topics:

- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures

DB2 API Applications

The script file `bldf77api`, in `sqllib/samples/fortran`, contains the commands to build a DB2 API program. The parameter, $1, specifies the name of your source file.

```
#! /bin/ksh
# bldf77api script file
# Builds a Fortran DB2 API program
# Usage: bldf77api <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Compile the util.f error-checking utility.
f77 -I$DB2PATH/include -c util.f

# Compile the program.
f77 -I$DB2PATH/include -c $1.f

# Link the program.
f77 -o $1 $1.o util.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
```

Compile and Link Options for bldf77api

The script file contains the following compile options:

`f77` The Fortran compiler.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

`-c` Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldf77api	
The script file contains the following link options:	
f77	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
util.o	Include the utility object file for error checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

To build the sample program `client` from the source file `client.f`, enter:

```
bldf77api client
```

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

```
client
```

Embedded SQL Applications

The script file `bldf77`, in `sqllib/samples/fortran`, contains the commands to build an embedded SQL program.

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

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

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Connect to a database.
if (($# < 2))
```

```
then
   db2 connect to sample
elif (($# < 3))
then
   db2 connect to $2
else
   db2 connect to $2 user $3 using $4
fi

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

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

# Disconnect from the database.
db2 connect reset

# Compile the util.f error-checking utility.
f77 -I$DB2PATH/include -c util.f

# Compile the program.
f77 -I$DB2PATH/include -c $1.f

# Link the program.
f77 -o $1 $1.o util.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
```

Compile and Link Options for bldf77

The script file contains the following compile options:

f77 The Fortran compiler.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

-c Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldf77	
The script file contains the following link options:	
f77	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
util.o	Include the utility object file for error-checking.
-L$DB2PATH/lib	Specify the location of the DB2 static and shared libraries at link-time. For example: >$HOME/sqllib/lib.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: >$HOME/sqllib/lib.
-ldb2	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

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

```
bldf77 updat
```

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

```
updat
```

Embedded SQL Stored Procedures

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

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

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

```
#! /bin/ksh
# bldf77sp script file
# Builds a Fortran stored procedure
# Usage: bldf77 <stored_proc_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

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

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

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

# Disconnect from the database.
db2 connect reset

# Build the stored procedure.
f77 -G $1.f -o $1 -I$DB2PATH/include

# Copy the shared library to the sqllib/function subdirectory of the DB2 instance.
# This assumes the user has write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile Options for bldf77sp

The script file contains the following compile options:

f77 The Fortran compiler.

-G $1.f
Generate a shared library from the precompiled source file.

-o $1 Specify the shared library.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.

Refer to your compiler documentation for additional compiler options.

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

```
bldf77sp outsrv
```

The script file copies the stored procedure to the server in the path `sqllib/function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `sqllib/function/unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the script file `bldf77`. Refer to “Embedded SQL Applications” on page 319 for details.

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

```
outcli
```

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

Chapter 11. Building Applications for Windows 32-bit Operating Systems

This chapter provides detailed information for building applications on Windows 32-bit operating systems. 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.

For the latest DB2 application development updates for Windows 32-bit operating systems, visit the Web page at:

```
http://www.software.ibm.com/data/db2/udb/ad
```

Note: All applications on Windows 32-bit applications, 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 *Application Development Guide.*

For DB2 for Windows 32-bit operating systems, the WCHARTYPE CONVERT option is supported for applications compiled with the Microsoft Visual C++ compiler. However, do not use the CONVERT option with this compiler 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 run-time environment does not handle substitution characters for certain double byte characters. This could result in run time conversion errors.

The WCHARTYPE CONVERT option is not supported for applications compiled with the IBM VisualAge C++ compiler. For this 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 INSERT INTO TABLENAME VALUES(:mb);
```

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

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

Object Linking and Embedding Database (OLE DB) Table Functions

DB2 supports OLE DB table functions. For these functions, there is no application building needed besides creating the CREATE FUNCTION DDL. OLE DB table function sample files are provided by DB2 in the %DB2PATH%\samples\oledb directory. These are Command Line Processor (CLP) files. They can be built with the following steps:

1. db2 connect to database_name

2. db2 -t -v -f file_name.db2
3. db2 terminate

where `database_name` is the database you are connecting to, and `file_name` is the name of the CLP file, with extension `.db2`.

For a full description of OLE DB table functions, see the *Application Development Guide*.

Microsoft Visual Basic

Note: The DB2 SDK for Windows 32-bit operating systems does not supply a precompiler for Microsoft Visual Basic.

This section covers the following topics:
- ActiveX Data Objects (ADO)
- Remote Data Objects (RDO)
- Object Linking and Embedding (OLE) Automation

ActiveX Data Objects (ADO)

ActiveX Data Objects (ADO) allow you to write an application to access and manipulate data in a database server through an OLE DB provider. The primary benefits of ADO are high speed, ease of use, low memory overhead, and a small disk footprint.

To use ADO with Microsoft Visual Basic, you need to establish a reference to the ADO type library. Do the following:
1. Select ″References″ from the Project menu
2. Check the box for ″Microsoft ActiveX Data Objects <version_number> Library″
3. Click ″OK″.

where <version_number> is the current version the ADO library.

Once this is done, ADO objects, methods, and properties will be accessible through the VBA Object Browser and the IDE Editor.

A full Visual Basic program includes forms and other graphical elements, and you need to view it inside the Visual Basic environment. Here are Visual Basic commands as part of a program to access the DB2 `sample` database, cataloged in ODBC:

Establish a connection:

```
Dim db As Connection
Set db = New Connection
```

Set client-side cursors supplied by the local cursor library:

```
db.CursorLocation = adUseClient
```

Set the provider so ADO will use the Microsoft ODBC Driver, and open database ″sample″ with no user id/password; that is, use the current user:

```
db.Open "SAMPLE"
```

Create a record set:

```
Set adoPrimaryRS = New Recordset
```

Use a select statement to fill the record set:

```
adoPrimaryRS.Open "select EMPNO,LASTNAME,FIRSTNME,MIDINIT,EDLEVEL,JOB
from EMPLOYEE Order by EMPNO", db
```

From this point, the programmer can use the ADO methods to access the data such as moving to the next record set:

```
adoPrimaryRS.MoveNext
```

Deleting the current record in the record set:

```
adoPrimaryRS.Delete
```

As well, the programmer can do the following to access an individual field:

```
Dim Text1 as String
Text1 = adoPrimaryRS!LASTNAME
```

DB2 provides Visual Basic ADO sample programs in the `%DB2PATH%\samples\ADO\VB` directory.

Remote Data Objects (RDO)

Remote Data Objects (RDO) provide an information model for accessing remote data sources through ODBC. RDO offers a set of objects that make it easy to connect to a database, execute queries and stored procedures, manipulate results, and commit changes to the server. It is specifically designed to access remote ODBC relational data sources, and makes it easier to use ODBC without complex application code, and is a primary means of accessing a relational database that is exposed with an ODBC driver. RDO implements a thin code layer over the Open Database Connectivity (ODBC) API and driver manager that establishes connections, creates result sets and cursors, and executes complex procedures using minimal workstation resources.

To use RDO with Microsoft Visual Basic, you need to establish a reference to your Visual Basic project. Do the following:

1. Select ″References″ from the Project menu
2. Check the box for ″Microsoft Remote Data Object <Version Number>″
3. Click ″OK″.

where <version_number> is the current RDO version.

A full Visual Basic program includes forms and other graphical elements, and you need to view it inside the Visual Basic environment. Here are Visual Basic commands as part of a DB2 program that connects to the `sample` database, opens a record set that selects all the columns from the `EMPLOYEE` table, and then displays the employee names to a message window, one by one:

```
Dim rdoEn As rdoEngine
Dim rdoEv As rdoEnvironment
Dim rdoCn As rdoConnection
Dim Cnct$
Dim rdoRS As rdoResultset
Dim SQLQueryDB As String
```

Assign the connection string:

```
 Cnct$ = "DSN=SAMPLE;UID=;PWD=;"
```

Set the RDO environment:

```
Set rdoEn = rdoEngine
Set rdoEv = rdoEn.rdoEnvironments(0)
```

Connect to the database:

```
 Set rdoCn = rdoEv.OpenConnection("", , , Cnct$)
```

Assign the SELECT statement for the record set:

```
SQLQueryDB = "SELECT * FROM EMPLOYEE"
```

Open the record set and execute the query:

```
Set rdoRS = rdoCn.OpenResultset(SQLQueryDB)
```

While not at the end of the record set, display Message Box with LastName, Firstname from table, one employee at a time:

```
While Not rdoRS.EOF
MsgBox rdoRS!LASTNAME & ", " & rdoRS!FIRSTNME
```

Move to the next row in the record set:

```
rdoRS.MoveNext
Wend
```

Close the program:

```
rdoRS.Close
rdoCn.Close
rdoEv.Close
```

DB2 provides Visual Basic RDO sample programs in the %DB2PATH%\samples\RDO directory.

Object Linking and Embedding (OLE) Automation

This section describes Object Linking and Embedding (OLE) automation UDFs in Microsoft Visual Basic as well as accessing 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, see the *Application Development Guide*.

OLE Automation UDFs

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. For information on building and running the OLE automation UDF samples, please see the readme.txt file in %DB2PATH%\samples\ole.

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.

For information on setting up and running the automation controller and the projects using it, please see the `readme.txt` file in `%DB2PATH%\samples\ole`.

Microsoft Visual C++

This section includes the following topics:

- ActiveX Data Objects (ADO)
- Object Linking and Embedding (OLE) Automation
- DB2 API Applications
- DB2 CLI Applications
- DB2 CLI Stored Procedures
- Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)

Note: The Visual C++ compiler is used for both C and C++ sample programs supplied in the `%DB2PATH\samples\c` and `%DB2PATH\samples\cpp` directories. The batch files in both these directories contain commands to accept either a C or C++ source file, depending on the file extension. By default, the C++ commands are commented out in the batch files in `%DB2PATH\samples\c`, and the C commands are commented out in the batch files in `%DB2PATH\samples\cpp`. Except for the first two topics where batch files are not used, this section demonstrates building programs using the C batch files.

ActiveX Data Objects (ADO)

DB2 ADO programs using Visual C++ can be compiled the same as regular C++ programs, once you make the following change.

To have your C++ source program run as an ADO program, you can put the following import statement at the top of your source program file:

```
#import "C:\program files\common files\system\ado\msado<VERSION NUMBER>.dll" \
no_namespace \
rename( "EOF", "adoEOF")
```

where `<VERSION NUMBER>` is the version number of the ADO library.

When the program is compiled, the user will need to verify that the `msado<VERSION NUMBER>.dll` is in the path specified. An alternative is to add `C:\program files\common files\system\ado` to the environment variable `LIBPATH`, and then use this shorter import statement in your source file:

```
#import <msado<VERSION NUMBER>.dll> \
no_namespace \
rename( "EOF", "adoEOF")
```

This is the method used in the DB2 sample program, `BLOBAccess.dsp`.

With this IMPORT statement, your DB2 program will have access to the ADO library. You can now compile your Visual C++ program as you would any other program. If you are also using another programming interface, such as DB2 APIs, or DB2 CLI, refer to the appropriate section in this chapter for additional information on building your program.

DB2 provides Visual C++ ADO sample programs in the `%DB2PATH%\samples\ADO\VC` directory.

Object Linking and Embedding (OLE) Automation

This section describes Object Linking and Embedding (OLE) automation UDFs in 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 *Application Development Guide*.

OLE Automation UDFs

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 “User-Defined Functions (UDFs)” on page 345.

DB2 provides self-containing samples of OLE automation UDFs in Microsoft Visual C++, located in the directory `%DB2PATH%\samples\ole\msvc`. For information on building and running the OLE automation UDF samples, please see the `readme.txt` file in `%DB2PATH%\samples\ole`.

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 directory `%DB2PATH%\samples\ole\msvc` contains a DB2 client program, `salaryclt`, implemented in Microsoft Visual C++, which invokes the ″median″ stored procedure.

For information on setting up and running the automation controller and the projects using it, please see the `readme.txt` file in `%DB2PATH%\samples\ole`.

DB2 API Applications

The batch file `bldmsapi.bat`, in `%DB2PATH%\samples\c`, and `%DB2PATH%\samples\cpp`, contains the commands to build a DB2 API program.

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

```
@echo off
rem bldmsapi.bat file
rem Builds a DB2 API program in C or C++
rem using the Microsoft Visual C++ compiler.
rem Usage: bldmsapi prog_name

if "%1" == "" goto error

rem  Compile the C program.
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c util.c
rem For C++, comment out the above line, and uncomment the following:
rem cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx util.cxx

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

goto exit

:error
echo Usage: bldmsapi prog_name

:exit
@echo on
```

Compile and Link Options for bldmsapi

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.
-D_X86_=1	Compiler option necessary for Windows 32-bit operating systems to run on Intel-based computers.
-DWIN32	Compiler option necessary for Windows 32-bit operating systems.

Compile and Link Options for bldmsapi

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 the executable.

%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, client, from either the source file client.c, in %DB2PATH%\samples\c, or from the source file client.cxx, in %DB2PATH%\samples\cpp, enter:

```
bldmsapi client
```

The result is an executable file, client.exe. You can run the executable file by entering the executable name (without the extension) on the command line:

```
client
```

DB2 CLI Applications

The batch file bldmcli.bat, in %DB2PATH%\samples\cli, contains the commands to build a DB2 CLI program.

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

```
@echo off
rem  bldmcli batch file - Windows 32-bit Operating Systems
rem  Builds a CLI program with Microsoft Visual C++.
rem  Usage: bldmcli prog_name

if "%1" == "" goto error

rem Compile the error-checking utility.
cl -Z7 -Od -c -W1 -D_X86=1 -DWIN32 samputil.c

rem  Compile the program.
cl -Z7 -Od -c -W1 -D_X86=1 -DWIN32 %1.c

rem  Link the program.
link -debug:full -debugtype:cv -OUT:%1.exe %1.obj samputil.obj db2cli.lib
```

```
goto exit

:error
echo Usage: bldmcli prog_name

:exit
@echo on
```

Compile and Link Options for bldmcli

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.
-W1	Set warning level.
-D_X86_=1	Compiler option necessary for Windows 32-bit operating systems to run on Intel-based computers.
-DWIN32	Compiler option necessary for Windows 32-bit operating systems.

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 the executable.
%1.obj	Include the object file.
samputil.obj	Include the utility object file for error checking.
db2cli.lib	Link with the DB2 CLI library.

Refer to your compiler documentation for additional compiler options.

To build the sample program basiccon, from the source file basiccon.c, enter:

```
bldmcli basiccon
```

The result is an executable file, `basiccon.exe`. You can run the executable file against the SAMPLE database by entering the executable name (without the extension):

```
basiccon
```

DB2 CLI Stored Procedures

The batch file `bldmclis.bat`, in `%DB2PATH%\samples\cli`, contains the commands to build a CLI stored procedure. The batch file builds the stored procedure into a DLL on the server.

The 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 bldmclis.bat file - Windows 32-bit Operating Systems
rem Builds a CLI stored procedure using the Microsoft Visual C++ compiler.
rem Usage: bldmclis prog_name

if "%1" == "" goto error

rem Compile the program.
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c  samputil.c

rem Link the program.
link -debug:full -debugtype:cv -out:%1.dll %1.obj samputil.obj db2cli.lib
  -def:%1.def

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

goto exit

:error
echo Usage: bldmclis prog_name

:exit
@echo on
```

Compile and Link Options for bldmclis

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.
-D_X86_=1	Compiler option necessary for Windows 32-bit operating systems to run on Intel-based computers.
-DWIN32	Compiler option necessary for Windows 32-bit operating systems.

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.dll	Build a .DLL file.
%1.obj	Include the object file.
samputil.obj	Include the utility object file for error-checking.
db2cli.lib	Link with the DB2 CLI library.
-def:%1.def	Use the module definition file.

Refer to your compiler documentation for additional compiler options.

To build the `outsrv2` stored procedure from the source file `outsrv2.c`, enter:

```
bldmclis outsrv2
```

The batch file uses the module definition file `outsrv2.def`, contained in the same directory as the CLI sample programs, to build the stored procedure. The batch file copies the stored procedure DLL, `outsrv2.dll`, to the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the stored procedure DLL name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

Once you build the stored procedure, `outsrv2`, you can build the client application that calls the stored procedure. You can build `outcli2` using the batch file `bldmcli`. See “DB2 CLI Applications” on page 335 for details.

To run the stored procedure, enter:

```
outcli2 remote_database userid password
```

where

remote_database
: is the name of the database to which you want to connect, such as SAMPLE.

userid is a valid user ID.

password
: is a valid password.

The client application passes a variable to the server program, `outsrv2`, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

The batch file `bldmsemb.bat`, in `%DB2PATH%\samples\c`, and in `%DB2PATH%\samples\cpp`, contains the commands to build an embedded SQL 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 bldmsemb.bat file
rem Builds a 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.
db2 prep %1.sqc bindfile
rem For C++, comment out the above line and uncomment the following:
rem db2 prep %1.sqx bindfile

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

rem Disconnect from the database.
db2 connect reset

rem  Compile the program.
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c util.c
rem For C++, comment out the above line and uncomment the following:
rem cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx util.cxx

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.
-D_X86_=1	Compiler option necessary for Windows 32-bit operating systems to run on Intel-based computers.
-DWIN32	Compiler option necessary for Windows 32-bit operating systems.

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`, from the C source file `updat.sqc` in `%DB2PATH%\samples\c`, or from the C++ source file `updat.sqx` in `%DB2PATH%\samples\cpp`, enter:

```
bldmsemb updat
```

The result is an executable file `updat.exe`. You can run the executable file against the SAMPLE database by entering the executable name (without the extension):

```
   updat
```

Embedded SQL Stored Procedures

The batch file `bldmsstp.bat`, in `%DB2PATH%\samples\c`, and in `%DB2PATH%\samples\cpp`, contains the commands to build an embedded SQL stored procedure. The batch file builds 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 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.
db2 prep %1.sqc bindfile
rem For C++, comment out the above line and uncomment the following:
rem db2 prep %1.sqx bindfile

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

rem Disconnect from the database.
db2 connect reset

rem Compile the program.
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c
```

```
rem For C++, comment out the above line and uncomment the following:
rem cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx

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.
-D_X86_=1	Compiler option necessary for Windows 32-bit operating systems to run on Intel-based computers.
-DWIN32	Compiler option necessary for Windows 32-bit operating systems.

Compile and Link Options for bldmsstp

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.

-out:%1.dll
Build a .DLL file.

%1.obj Include the object file.

db2api.lib
Link with the DB2 library.

-def:%1.def
Module definition file.

Refer to your compiler documentation for additional compiler options.

To build the outsrv stored procedure from either the C source file, outsrv.sqc, or the C++ source file, outsrv.sqx, enter:

```
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. For DB2DARI parameter style stored procedures where the invoked procedure matches the stored procedure DLL name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the %DB2PATH%\function\unfenced directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the "Stored Procedures" chapter in the *Application Development Guide*.

Note: An unfenced stored procedure runs in the same address space as the database manager and results in increased performance when compared to a fenced stored procedure, which runs in an address space isolated from the database manager. With unfenced stored procedures there is a danger that user code could accidentally or maliciously damage the database control structures. Therefore, you should only run unfenced stored procedures when you need to maximize the

performance benefits. Ensure these programs are thoroughly tested before running them as unfenced. Refer to the *Application Development Guide* for more information.

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 "Embedded SQL Applications" on page 339 for details.

To run the stored procedure, 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.

User-Defined Functions (UDFs)

The batch file bldmsudf, in %DB2PATH%\samples\c, and in %DB2PATH%\samples\cpp, contains the commands to build a UDF.

UDFs cannot contain embedded SQL statements. Therefore, to build a UDF program, you do not need to connect to a database to precompile and bind the program.

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

```
@echo off
rem bldmsudf.bat file
rem Builds a C or C++ user-defined function (UDF).
rem Usage: bldmsudf udf_prog_name

if "%1" == "" goto error

rem Compile the program.
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c
rem For C++, comment out the above line and uncomment the following:
rem cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx

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.
`-D_X86_=1`	Compiler option necessary for Windows 32-bit operating systems to run on Intel-based computers.
`-DWIN32`	Compiler option necessary for Windows 32-bit operating systems.

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.
`-out:%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.
`-def:%1.def`	Module definition file.

Refer to your compiler documentation for additional compiler options.

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

Once you build udf, you can build the client application, calludf, that calls it. DB2 CLI, as well as embedded SQL C and C++ versions of this program are provided.

You can build the DB2 CLI calludf program from the calludf.c source file in %DB2PATH%\samples\cli using the batch file bldmscli. Refer to “DB2 CLI Applications” on page 335 for details.

You can build the embedded SQL C calludf program from the calludf.sqc source file in %DB2PATH%\samples\c using the batch file bldmsemb. Refer to “Embedded SQL Applications” on page 339 for details.

You can build the embedded SQL C++ calludf program from the calludf.sqx source file in %DB2PATH%\samples\cpp using the batch file bldmsemb. Refer to “Embedded SQL Applications” on page 339 for details.

To run the UDF, enter:

```
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++ Version 3.5

This section contains the following topics:

- DB2 API Applications
- DB2 CLI Applications
- DB2 CLI Stored Procedures
- Embedded SQL Applications
- Embedded SQL Stored Procedures

- User-Defined Functions (UDFs)

Note: The VisualAge C++ compiler is used for both C and C++ sample programs supplied in the %DB2PATH\samples\c and %DB2PATH\samples\cpp directories. The batch files in both these directories contain commands to accept either a C or C++ source file, depending on the file extension. By default, the C++ commands are commented out in the batch files in %DB2PATH\samples\c, and the C commands are commented out in the batch files in %DB2PATH\samples\cpp. Except for the first topic where batch files are not used, this section demonstrates building programs using the C batch files.

DB2 API Applications

The batch file bldvaapi.bat, in %DB2PATH%\samples\c, and in %DB2PATH%\samples\cpp, contains the commands to build a DB2 API program.

Ensure the LIB environment variable points to %DB2PATH%\lib like this:

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

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

```
@echo off
rem bldvaapi.bat file
rem Builds a DB2 API program using the IBM VisualAge C++ compiler.
rem USAGE: bldvaapi <prog_name>

rem  Compile the program.
icc -c -Ti -W1 %1.c util.c
rem For C++, comment out the above line and uncomment the following:
rem icc -c -Ti -W1 %1.c util.cxx

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

goto exit

:error
echo Usage: bldvaapi <prog_name>
:exit
@echo on
```

Compile and Link Options for bldvaapi

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 `client` from the C source file `client.c`, or the C++ file `client.cxx`, enter:

```
bldvaapi client
```

The result is an executable file `client.exe`. You can run the executable file against the SAMPLE database by entering the executable name (without the extension):

```
client
```

DB2 CLI Applications

The batch file `bldvcli.bat`, in `%DB2PATH%\samples\cli`, contains the commands to build a DB2 CLI program in IBM VisualAge C++.

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

```
@echo off
rem  bldvcli batch file - Windows 32-bit Operating Systems
rem  Builds a CLI program with IBM VisualAge C++.
rem  Usage: bldvcli prog_name

if "%1" == "" goto error

rem Compile the error-checking utility.
icc -c -Ti -W1 samputil.c

rem  Compile the program.
icc -c -Ti -W1 %1.c

rem  Link the program.
ilink /MAP /DEBUG /ST:32000 /PM:VIO %1.obj samputil.obj db2cli.lib

goto exit

:error
echo Usage: bldvcli prog_name

:exit
@echo on
```

Compile and Link Options for bldvcli

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.

Compile and Link Options for bldvcli

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.

`samputil.obj`
Include the utility object file for error checking.

`db2cli.lib`
Link with the DB2 CLI library.

Refer to your compiler documentation for additional compiler options.

To build the sample program `basiccon`, from the source file `basiccon.c`, enter:

```
bldvcli basiccon
```

The result is an executable file `basiccon.exe`. You can run the executable file against the SAMPLE database by entering the executable name (without the extension):

```
basiccon
```

DB2 CLI Stored Procedures

The batch file `bldvclis.bat`, in `%DB2PATH%\samples\cli`, contains the commands to build a CLI stored procedure. The batch file builds the stored procedure into a DLL on the server.

The 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 bldvclis.bat file - Windows 32-bit Operating Systems
rem Builds a CLI stored procedure using the IBM VisualAge C++ compiler.
rem Usage: bldvclis prog_name

if "%1" == "" goto error

rem Compile the program.
icc -c+ -Ti -Ge- -Gm+ -W1 %1.c  samputil.c

rem Import the library and create an export 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 as in: "@16". In outsrv2va.def, for example, the IBM VisualAge C++
rem compiler requires "EXPORTS _outsrv2@16" and not "EXPORTS outsrv2".
ilib /GI %1va.def

rem Link the program and produce a DLL.
ilink /ST:64000 /PM:VIO /MAP /DLL %1.obj samputil.obj %1.exp db2cli.lib

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

goto exit

:error
echo Usage: bldvclis prog_name

:exit
@echo on
```

Compile and Link Options for bldvclis

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 run-time library that is statically linked.
-Gm+	Link with multi-tasking libraries.
-W1	Output warning, error, and severe and unrecoverable error messages.

Compile and Link Options for bldvclis

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 64 000.
/PM:VIO	Enable the program to run in a window or in a full screen.
/MAP	Generate a map file.
/DLL	Build a .DLL file.
%1.obj	Include the object file.
%1.exp	Include the VisualAge export file.
db2cli.lib	Link with the DB2 CLI library.

Refer to your compiler documentation for additional compiler options.

To build the `outsrv2` stored procedure from the source file `outsrv2.c`, enter:

```
bldvclis outsrv2
```

The batch file uses the module definition file, `outsrv2va.def`, contained in the same directory as the CLI sample programs, to build the stored procedure. The batch file copies the stored procedure DLL, `outsrv2.dll`, to the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the stored procedure DLL name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the "Stored Procedures" chapter in the *Application Development Guide*.

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

Once you build the stored procedure, `outsrv2`, you can build the client application that calls the stored procedure. You can build `outcli2` using the batch file `bldvcli`. See "DB2 CLI Applications" on page 349 for details.

To run the stored procedure, enter:

```
outcli2 remote_database userid password
```

where

remote_database
is the name of the database to which you want to connect, such as SAMPLE.

userid is a valid user ID.

password
is a valid password.

The client application passes a variable to the server program, `outsrv2`, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

The batch file `bldvaemb.bat`, in `%DB2PATH%\samples\c`, and in `%DB2PATH%\samples\cpp`, contains the commands to build an embedded SQL program.

Ensure the LIB environment variable points to `%DB2PATH%\lib` like this:

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

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
rem using 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.
db2 prep %1.sqc bindfile
rem For C++, comment out the above line and uncomment the following:
rem db2 prep %1.sqx bindfile

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

rem Disconnect from the database.
db2 connect reset

rem  Compile the program.
icc -c -Ti -W1 %1.c util.c
rem For C++, comment out the above line and uncomment the following:
rem icc -c -Ti -W1 %1.cxx util.cxx

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.

Compile and Link Options for bldvaemb

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, from the C source file updat.sqc, or the C++ source file updat.sqx, enter:

```
bldvaemb updat
```

The result is an executable file, updat.exe. You can run the executable file against the SAMPLE database by entering the executable name (without the extension):

```
updat
```

Embedded SQL Stored Procedures

The batch file bldvastp.bat, in %DB2PATH%\samples\c, and in %DB2PATH%\samples\cpp, contains the commands to build an embedded SQL stored procedure. The batch file compiles the stored procedure into a DLL, and stores it 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 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.
db2 prep %1.sqc bindfile
rem For C++, comment out the above line and uncomment the following:
rem db2 prep %1.sqx bindfile

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

rem Disconnect from the database.
db2 connect reset

rem Compile the program.
icc -c+ -Ti -Ge- -Gm+ -W1 %1.c
rem For C++, comment out the above line and uncomment the following:
rem icc -c+ -Ti -Ge- -Gm+ -W1 %1.cxx

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 run-time library that is statically linked.
-Gm+	Link with multi-tasking 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 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 stored procedure from the C source file, outsrv.sqc, or the C++ source file, outsrv.sqx, enter:

```
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. For DB2DARI parameter style stored procedures where the invoked procedure matches the stored procedure DLL name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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 “Embedded SQL Applications” on page 354 for details.

To run the stored procedure, 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.

User-Defined Functions (UDFs)

The batch file `bldvaudf`, in `%DB2PATH%\samples\c`, and in `%DB2PATH%\samples\cpp`, contains the commands to build a UDF.

UDFs cannot contain embedded SQL statements. Therefore, to build a UDF program, you do not need to connect to a database to precompile and bind the program.

The 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)
rem using the IBM VisualAge C++ compiler.
rem Usage: bldvaudf <udf_program_name>

rem Compile the program.
icc -Ti -c+ -Ge- -Gm+ -W1 %1.c
rem For C++, comment out the above line and uncomment the following:
rem icc -Ti -c+ -Ge- -Gm+ -W1 %1.cxx

rem Generate an import library and export file using a definition file.
rem Function(s) in the .def file are prepended with an underscore, and
rem appended with the @ sign and number of bytes of arguments (in decimal).
rem Parameters of less than four bytes are rounded up to four bytes.
rem Structure size is rounded up to a multiple of four bytes.
rem For example, function fred prototyped as: "int fred(int, int, short);"
rem would appear as:  "_fred@12" in the .def file.
rem These decorated function names can also be found in %1.map
rem after running the following ilink command without %1va.exp.
ilib /gI %1va.def

rem Create 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 run-time library that is statically linked.
-Gm+	Link with multi-tasking libraries.
-W1	Output warning, error, and severe and unrecoverable error messages.

Compile and Link Options for bldvaudf

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, from the source file, udf.c in %DB2PATH%\samples\c, enter:

```
bldvaudf 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.

Once you build udf, you can build the client application, calludf, that calls the UDF. DB2 CLI as well as embedded SQL C and C++ versions of this program are provided.

You can build the DB2 CLI calludf program from the calludf.c source file in %DB2PATH%\samples\cli using the batch file bldvcli. Refer to “DB2 CLI Applications” on page 349 for details.

You can build the C embedded SQL calludf program from the calludf.sqc source file in %DB2PATH%\samples\c using the batch file bldvaemb. Refer to “Embedded SQL Applications” on page 354 for details.

You can build the C++ embedded SQL `calludf` program from the `calludf.sqx` source file in `%DB2PATH%\samples\cpp` using the batch file `bldvaemb`. Refer to “Embedded SQL Applications” on page 354 for details.

To run the UDF, enter:

```
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:

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

You do not have to type the SQL statement in uppercase.

IBM VisualAge C++ Version 4.0

The VisualAge C++ compiler differs from other compilers on Windows 32-bit operating systems. To compile a program with VisualAge C++ Version 4.0, you must first make a configuration file. See the documentation that comes with the compiler to learn more about this.

DB2 provides configuration files for the different types of DB2 programs you can build with this VisualAge C++ compiler. To use a DB2 configuration file, you first set an environment variable to the program name you wish to compile. Then you compile the program with a command supplied by VisualAge C++ Version 4.0. Here are the configuration files provided by DB2, and the sections describing how they can be used to compile your programs:

api.icc
: DB2 API configuration file. For details, see “DB2 API Applications” on page 363.

cli.icc
: DB2 CLI configuration file. For details, see “DB2 CLI Applications” on page 364.

clis.icc
: DB2 CLI stored procedure configuration file. For details, see “DB2 CLI Stored Procedures” on page 366.

emb.icc
: Embedded SQL configuration file. For details, see “Embedded SQL Applications” on page 369.

stp.icc
: Embedded SQL stored procedure configuration file. For details, see “Embedded SQL Stored Procedures” on page 370.

udf.icc
: User-defined function configuration file. For details, see “User-Defined Functions (UDFs)” on page 373.

DB2 API Applications

The configuration file, api.icc, in %DB2PATH%\samples\c, allows you to build DB2 API programs in C. There is also a C++ DB2 API configuration file in %DB2PATH%\samples\cpp. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// api.icc configuration file for DB2 API programs
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export API=prog_name'
// To use on OS/2 and Windows, enter: 'set API=prog_name'
// Then compile the program by entering: 'vacbld api.icc'

if defined( $API )
{
  prog_name = $API
}
else
{
  error "Environment Variable API is not defined."
}

infile = prog_name".c"
util   = "util.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2api.lib"
  option opts = link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
}

option opts
```

```
{
  target type(exe) outfile
  {
    source infile
    source util
    source lib
  }
}
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the DB2 API sample program `client` from the source file `client.c`, do the following:

1. Set the API environment variable to the program name by entering:
   ```
   set API=client
   ```
2. If you have an `api.ics` file in your working directory, produced by building a different program with the `api.icc` file, delete the `api.ics` file with this command:
   ```
   del api.ics
   ```

   An existing `api.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld api.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The result is an executable file, `client`. You can run the program by entering the executable name:

```
client
```

DB2 CLI Applications

The configuration file, `cli.icc`, in `%DB2PATH%\samples\cli`, allows you to build DB2 CLI programs. This file can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// cli.icc configuration file for DB2 CLI applications
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export CLI=prog_name'
// To use on OS/2 and Windows, enter: 'set CLI=prog_name'
// Then compile the program by entering: 'vacbld cli.icc'

if defined( $CLI )
{
  prog_name = $CLI
}
```

```
else
{
  error "Environment Variable CLI is not defined."
}

infile   = prog_name".c"
samputil = "samputil.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2cli.lib"
  option opts = link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
}

option opts
{
  target type(exe) outfile
  {
    source infile
    source samputil
    source lib
  }
}
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the DB2 CLI sample program `basiccon` from the source file `basiccon.c`, do the following:

1. Set the CLI environment variable to the program name by entering:

   ```
   set CLI=basiccon
   ```
2. If you have a `cli.ics` file in your working directory, produced by building a different program with the `cli.icc` file, delete the `cli.ics` file with this command:

   ```
   del cli.ics
   ```

An existing cli.ics file produced for the same program you are going to build again does not have to be deleted.

3. Compile the sample program by entering:
   ```
   vacbld cli.icc
   ```
   Note: The vacbld command is provided by VisualAge C++ Version 4.0.

The result is an executable file, basiccon. You can run the program by entering the executable name:

```
basiccon
```

DB2 CLI Stored Procedures

The configuration file, clis.icc, in %DB2PATH%\samples\cli, allows you to build DB2 CLI stored procedures. This file can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// clis.icc configuration file for DB2 CLI stored procedures
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export CLIS=prog_name'
// To use on OS/2 and Windows, enter: 'set CLIS=prog_name'
// Then compile the program by entering: 'vacbld clis.icc'

if defined( $CLIS )
{
  prog_name = $CLIS
}
else
{
  error "Environment Variable CLIS is not defined."
}

infile = prog_name".c"
samputil = "samputil.c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else /* if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ ) */
{
  db2path     = $DB2PATH
```

```
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2cli.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
  cpcmd       = "copy"
  funcdir     = db2path"\\function"
}

option opts
{
  target type(dll) outfile
  {
    source infile
    source samputil
    source lib
  }
}

if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}

run after cpcmd " " outfile " " funcdir
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the DB2 CLI stored procedure `outsrv2` from the source file `outsrv2.c`, do the following:

1. Set the CLIS environment variable to the program name by entering:
   ```
   set CLIS=outsrv2
   ```
2. If you have a `clis.ics` file in your working directory, produced by building a different program with the `clis.icc` file, delete the `clis.ics` file with this command:
   ```
   del clis.ics
   ```
   An existing `clis.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld clis.icc
   ```
   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The stored procedure is copied to the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the "Stored Procedures" chapter in the *Application Development Guide*.

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

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

Once you build the stored procedure `outsrv2`, you can build the CLI client application `outcli2` that calls the stored procedure. You can build `outcli2` by using the configuration file, `cli.icc`. Refer to “DB2 CLI Applications” on page 364 for details.

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

`outcli2` *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 `outsrv2`, which gives it a value and then returns the variable to the client application.

Embedded SQL Applications

The configuration file, `emb.icc`, in `%DB2PATH%\samples\c`, allows you to build DB2 embedded SQL applications in C. There is also a C++ configuration file for embedded SQL applications, in `%DB2PATH%\samples\cpp`. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// emb.icc configuration file for embedded SQL applications
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export EMB=prog_name'
// To use on OS/2 and Windows, enter: 'set EMB=prog_name'
// Then compile the program by entering: 'vacbld emb.icc'

if defined( $EMB )
{
  prog_name = $EMB
}
else
{
  error "Environment Variable EMB is not defined."
}

// To connect to another database, replace "sample"
// For user ID and password, update 'user' and 'passwd'
// and take out the comment in the line: 'run before "embprep "'
dbname = "sample"
user   = ""
passwd = ""

// Precompiling the source program file
run before "embprep " prog_name " " dbname // " " user " " passwd

infile = prog_name".c"
util   = "util.c"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".exe"
  group lib   = "db2api.lib"
  option opts = link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
}

option opts
```

```
{
  target type(exe) outfile
  {
    source infile
    source util
    source lib
  }
}
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the embedded SQL application updat from the source file updat.sqc, do the following:

1. Set the EMB environment variable to the program name by entering:
   ```
   set EMB=updat
   ```
2. If you have an emb.ics file in your working directory, produced by building a different program with the emb.icc file, delete the emb.ics file with this command:
   ```
   del emb.ics
   ```
   An existing emb.ics file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld emb.icc
   ```
   Note: The vacbld command is provided by VisualAge C++ Version 4.0.

The result is an executable file, updat. You can run the program by entering the executable name:

```
updat
```

Embedded SQL Stored Procedures

The configuration file, stp.icc, in %DB2PATH%\samples\c, allows you to build DB2 embedded SQL stored procedures in C. There is also a C++ configuration file for embedded SQL stored procedures in %DB2PATH%\samples\cpp. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// stp.icc configuration file for embedded SQL stored procedures
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export STP=prog_name'
// To use on OS/2 and Windows, enter: 'set STP=prog_name'
// Then compile the program by entering: 'vacbld emb.icc'

if defined( $STP )
{
```

```
  prog_name = $STP
}
else
{
  error "Environment Variable STP is not defined."
}

// To connect to another database, replace "sample"
// For user ID and password, update 'user' and 'passwd'
// and take out the comment in the line: 'run before "embprep "'
dbname = "sample"
user   = ""
passwd = ""

// Precompiling the source program file
run before "embprep " prog_name " " dbname // " " user " " passwd

infile = prog_name".c"
util   = "util.c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2api.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
  cpcmd       = "copy"
  funcdir     = db2path"\\function"
}

option opts
{
  target type(dll) outfile
  {
    source infile
    source util
    source lib
  }
}

if defined( $__TOS_AIX__ )
{
  rmcmd      = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}

run after cpcmd " " outfile " " funcdir
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the embedded SQL stored procedure `outsrv` from the source file `outsrv.sqc`, do the following:

1. Set the STP environment variable to the program name by entering:
   ```
   set STP=outsrv
   ```
2. If you have an `stp.ics` file in your working directory, produced by building a different program with the `stp.icc` file, delete the `stp.ics` file with this command:
   ```
   del stp.ics
   ```

   An existing `stp.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld stp.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++ Version 4.0.

The stored procedure is copied to the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the shared library name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide.*

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

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

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls the stored procedure. You can build `outcli` using the configuration file `emb.icc`. Refer to “Embedded SQL Applications” on page 369 for details.

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

`outcli` *remote_database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

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

User-Defined Functions (UDFs)

The configuration file, `udf.icc`, in `%DB2PATH%\samples\c`, allows you to build user-defined functions in C. There is also a C++ configuration file for user-defined functions in `%DB2PATH%\samples\cpp`. These files can be used on AIX, OS/2 and Windows 32-bit operating systems.

```
// udf.icc configuration file for user-defined functions
// for VisualAge C++ Version 4.0
// To use on AIX, enter: 'export UDF=prog_name'
// To use on OS/2 and Windows, enter: 'set UDF=prog_name'
// Then compile the program by entering: 'vacbld udf.icc'

if defined( $UDF )
{
  prog_name = $UDF
}
else
{
  error "Environment Variable UDF is not defined."
}

infile = prog_name".c"
expfile = prog_name".exp"

if defined( $__TOS_AIX__ )
{
  // Set db2path to where DB2 will be accessed.
  // The default is the standard instance path.
  db2path     = $HOME"/sqllib"
  outfile     = prog_name
  group lib   = "libdb2.a", "libdb2apie.a"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"/lib" ),
                incl( searchPath, db2path"/include" )
  cpcmd       = "cp"
  funcdir     = db2path"/function"
}
else // if defined( $__TOS_OS2__ ) | defined( $__TOS_WIN__ )
{
  db2path     = $DB2PATH
  outfile     = prog_name".dll"
  if defined( $__TOS_WIN__ )
  {
    expfile = prog_name"v4.exp"
  }
  group lib   = "db2api.lib", "db2apie.lib"
  option opts = link( exportList, expfile ),
                link( libsearchpath, db2path"\\lib" ),
                incl( searchPath, db2path"\\include" )
  cpcmd       = "copy"
  funcdir     = db2path"\\function"
}

option opts
{
  target type(dll) outfile
  {
    source infile
    source lib
  }
}
if defined( $__TOS_AIX__ )
{
  rmcmd       = "rm -f"
  run after rmcmd " " funcdir "/" outfile
}

run after cpcmd " " outfile " " funcdir
```

VisualAge C++ Version 4.0 defines one of the following environment variables depending on the operating system on which it is installed: `__TOS_AIX__`, `__TOS_OS2__`, `__TOS_WIN__`.

To use the configuration file to build the user-defined function program udf from the source file udf.c, do the following:

1. Set the UDF environment variable to the program name by entering:
   ```
   set UDF=udf
   ```
2. If you have a udf.ics file in your working directory, produced by building a different program with the udf.icc file, delete the udf.ics file with this command:
   ```
   del udf.ics
   ```
   An existing udf.ics file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld udf.icc
   ```
   Note: The vacbld command is provided by VisualAge C++ Version 4.0.

The UDF library is copied to the server in the path %DB2PATH%\function.

If necessary, set the file mode for the user-defined function so the DB2 instance can run it.

Once you build udf, you can build the client application, calludf, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI calludf program from the source file calludf.c, in %DB2PATH%\samples\cli, by using the configuration file cli.icc. Refer to "DB2 CLI Applications" on page 364 for details.

You can build the embedded SQL calludf program from the source file calludf.sqc, in %DB2PATH%\samples\c, by using the configuration file emb.icc. Refer to "Embedded SQL Applications" on page 369 for details.

To call the UDF, run the sample calling application by entering the executable name:
```
calludf
```

The calling application calls functions from the udf library.

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

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

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

IBM VisualAge COBOL

This section contains the following topics:

- Using the Compiler
- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures

Using the Compiler

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the IBM VisualAge 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, the default.
- 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 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".
  ```

DB2 API Applications

The batch file `bldvcapi.bat`, in `%DB2PATH%\samples\cobol`, contains the commands to build a DB2 API program. The parameter, `%1`, specifies the name of your source file.

```
@echo off
rem bldvcapi.bat file
rem Build a DB2 API program using the IBM VisualAge COBOL compiler.
rem Usage: bldvcapi <prog_name>

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: bldvcapi <prog_name>

:exit
@echo on
```

Compile and Link Options for bldvcapi

The batch file contains the following compile options:

cob2 The IBM VisualAge 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 `client`, from the source file `client.cbl`, enter:

```
bldvcapi client
```

The result is an executable file, `client`. You can run the executable file by entering the executable name:

```
client
```

Embedded SQL Applications

The batch file bldvacob.bat, in %DB2PATH%\samples\cobol, contains the commands to build an embedded SQL 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 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 VisualAge 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`, enter:

```
bldvacob updat
```

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

```
updat
```

Embedded SQL Stored Procedures

The batch file `bldvacbs.bat`, in `%DB2PATH%\samples\cobol`, contains the commands to build an embedded SQL 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 Builds a COBOL stored procedure using the IBM VisualAge 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.
lib /nol /gi:%1 %1.obj
ilink /free /nol /dll db2api.lib %1.exp %1.obj iwzrwin3.obj

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 VisualAge 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:

ilink	Use the IBM VisualAge COBOL linker.
/free	Free format.
/nol	No logo.
/dll	Create the DLL with the source program name.
db2api.lib	Link with the DB2 library.
%1.exp	Include the export file.
%1.obj	Include the program object file.
iwzrwin3.obj	Include the object file provided by IBM VisualAge COBOL.

Refer to your compiler documentation for additional compiler options.

To build the `outsrv` stored procedure from the `outsrv.sqb` source file, enter:

```
bldvacbs outsrv
```

The batch file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the stored procedure DLL name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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 batch file `bldvacob`. See “Embedded SQL Applications” on page 378 for details.

To run the stored procedure, 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.

Micro Focus COBOL

This section includes the following topics:
- Using the Compiler
- DB2 API Applications
- Embedded SQL Applications
- Embedded SQL Stored Procedures

Using the 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, the default.
- 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%
  ```

Calls to all DB2 application programming interfaces 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 run-time 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.

DB2 API Applications

The batch file `bldapicb`, in `%DB2PATH%\samples\cobol_mf`, contains the commands to build a sample COBOL program. The parameter, `%1`, specifies the name of your source file.

```
@echo off
rem bldapicb.bat file
rem Build a DB2 API program using the Micro Focus COBOL compiler.
rem Usage: bldapicb <prog_name>

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

rem  Compile the program.
cobol %1.cbl;

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

goto exit

:error
```

```
echo Usage: bldapicb <prog_name>

:exit
@echo on
```

Compile and Link Options for bldapicb
The batch file contains the following compile option: **cobol** The Micro Focus COBOL compiler.
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, client from the source file client.cbl, enter:

```
bldapicb client
```

The result is an executable file, client.exe. You can run the executable file against the SAMPLE database by entering the executable name (without the extension):

```
client
```

Embedded SQL Applications

The batch file bldmfcob, in %DB2PATH%\samples\cobol_mf, contains the commands to build an embedded SQL 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;

rem  Compile the program.
cobol %1.cbl;

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 option:

cobol The Micro Focus COBOL compiler.

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, from the source file updat.sqb, enter:

```
   bldmfcob updat
```

The result is an executable file, updat.exe. You can run the executable file against the SAMPLE database by entering the executable name (without the extension):

```
   updat
```

Embedded SQL Stored Procedures

The batch file bldmfcbs, in %DB2PATH%\samples\cobol_mf, contains the commands to build an embedded SQL 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 Builds a COBOL stored procedure using the 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 /case;

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

rem Copy the stored procedure to the %DB2PATH%\function directory.
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.

/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`, 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 batch file copies the stored procedure DLL, `outsrv.dll`, to the server in the path `%DB2PATH%\function`. For DB2DARI parameter style stored procedures where the invoked procedure matches the stored procedure DLL name, this location indicates that the stored procedure is fenced. If you want this type of stored procedure to be unfenced, you must move it to the `%DB2PATH%\function\unfenced` directory. For all other types of DB2 stored procedures, you indicate whether it is fenced or not fenced with the CREATE FUNCTION statement in the calling program. For a full discussion on creating and using the different types of DB2 stored procedures, please see the ″Stored Procedures″ chapter in the *Application Development Guide*.

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

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 “Embedded SQL Applications” on page 384 for details.

To run the stored procedure, 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.

Object REXX

Object REXX is an object-oriented version of the REXX language. Object-oriented extensions have been added to classic 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 classic 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:
   ```
   rexx updat.cmd
   ```

For further information on REXX and DB2, refer to the chapter, ″Programming in REXX″, in the *Application Development Guide*.

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.

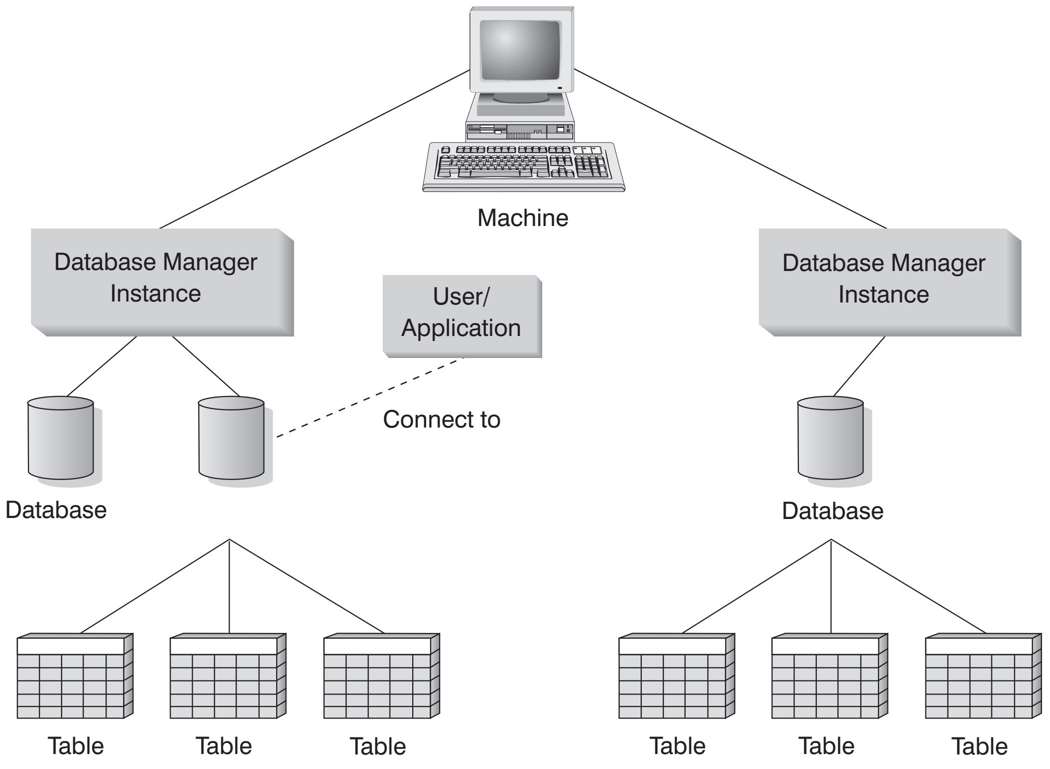

Figure 1. Database Manager Instances

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 6.

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 6.

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

instance name
: For UNIX platforms, this is a valid username that you specify when you create the database manager instance.

  For OS/2 and Windows NT, this is an alphanumeric string of up to eight characters. The DB2 instance is created for you during install.

instance directory
: The home directory where the instance is located.

  For UNIX platforms, the instance directory is `$HOME/sqllib`, where $HOME is the home directory of the instance owner.

  For OS/2 and Windows NT, the instance 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 for your platform.

Appendix B. Migrating Your Applications

When you upgrade to DB2 Universal Database Version 6 from a Version 2 or later installation of DB2, DB2 Client Application Enabler, or DB2 Software Developer's Kit, your database and node directories are migrated automatically. To migrate from DB2 Version 1, you must first migrate to DB2 Universal Database Version 5. Then you can migrate from Version 5 to Version 6. To migrate your existing databases, use the tools described in the *Administration Guide*.

Notes:

1. **HP-UX**. If you are migrating DB2 from HP-UX Version 10 or earlier to HP-UX Version 11, your DB2 programs must be re-precompiled with DB2 on HP-UX Version 11 (if they include embedded SQL), and must be re-compiled. This includes all DB2 applications, stored procedures, user-defined functions and user exit programs. As well, DB2 programs that are compiled on HP-UX Version 11 may not run on HP-UX Version 10 or earlier. DB2 programs that are compiled and run on HP-UX Version 10 may connect remotely to HP-UX Version 11 servers.
2. **Linux**. DB2 does not support migration from DB2 Universal Database for Linux Version 5.2 (Beta) to DB2 Universal Database for Linux Version 6.
3. **Micro Focus COBOL**. Any existing applications precompiled with DB2 Version 2.1.1 or earlier and compiled with Micro Focus COBOL should be re-precompiled with the current version of DB2, and then recompiled with Micro Focus COBOL. If these applications built with the earlier versions of the IBM precompiler are not re-precompiled, there is a possibility of database corruption if abnormal termination occurs.

Note: The following, as well as the ″Questions″ and ″Conditions″ sections, apply to UNIX platforms only.

If you have applications from DB2 Version 1, DB2 Version 2, or DB2 Version 5, and you want them to run in both a database instance of the previous version as well as a DB2 Version 6 instance on the same machine, you may need to make some changes to your environment. To determine what changes to make, answer the following questions, and then review the ″Conditions″ section to see if any of the conditions apply to your situation.

An AIX system is used to explain the points raised. The same concepts apply to other UNIX platforms, but the details may differ, such as environment variables and specific commands. If you are unfamiliar with these details for your operating system, please see the *Administration Guide* or the ″Migrated from Previous Versions″ chapter in the *DB2 for UNIX Quick Beginnings* book.

Questions

Question 1: How was the previous version application linked to the DB2 client run-time library, for example, libdb2.a on AIX?

To determine the embedded shared library search path for an executable, use one of the following system commands:

AIX /usr/bin/dump -H executable

HP-UX
/usr/bin/chatr executable

Silicon Graphics IRIX
/bin/elfdump -Lv executable

Solaris
/usr/bin/dump -Lv executable

where executable is the name of the application executable.

The following is a sample dump listing from a DB2 Version 1 for AIX application:

```
dbcat:

  ***Loader Section***
                      Loader Header Information
VERSION#         #SYMtableENT     #RELOCent        LENidSTR
0x00000001       0x00000012       0x00000029       0x00000064

#IMPfilID        OFFidSTR         LENstrTBL        OFFstrTBL
0x00000004       0x000003bc       0x00000077       0x00000420


  ***Import File Strings***
INDEX  PATH                          BASE                MEMBER
0      /usr/lpp/db2_01_01_0000/lib:/usr/lpp/xlC/lib:/usr/lib:/lib



1                                    libc.a              shr.o
2                                    libC.a              shr.o
3                                    libdb2.a            shr.o
```

Line 0 (zero) shows the directory paths that the executable searches to find the shared libraries to which it is linked. Lines 1, 2, and 3 show the shared libraries to which the application is linked.

Depending on how the application was built, you may see the following paths: `/usr/lpp/db2_01_01_0000/lib`, `INSTHOME/sqllib/lib` (where `INSTHOME` is the home directory of the database instance owner), or just the `/usr/lib:/lib` combination.

Question 2: How are the DB2 run-time libraries configured on your system?

When either of DB2 Versions 1, 2, 5 or 6 is installed, there is an optional step which creates symbolic links from the system default shared library path `/usr/lib` to the DB2 install path which contains the DB2 client run-time libraries.

For Version 1, the install path is `/usr/lpp/db2_01_01_0000/lib`. For Version 2, the install path is `/usr/lpp/db2_02_01/lib`. For Version 5, the install path is `/usr/lpp/db2_05_00/lib`. For Version 6, the install path is `/usr/lpp/db2_06_01/lib`. In all cases, the run-time shared libraries are named libdb2.a.

Only one version of these libraries can be the default at any one time. DB2 provides this default so that when you build an application, it does not depend on a particular version of DB2.

Question 3: Do you specify different search paths in your environment?

You can override the shared library search path coded in your application using the LIBPATH environment variable on AIX, SHLIB_PATH on HP-UX, and LD_LIBRARY_PATH on Silicon Graphics IRIX and Solaris. You can see the library search path using the appropriate system command for your platform given in the answer to Question 1.

Conditions

Once you have the answers to the questions above, you may need to make changes to your environment. Read the conditions listed below. If one of the conditions applies to your situation, make the necessary changes.

Condition 1: If a Version 5 application loads a shared library out of the AIX default shared library path `/usr/lib/libdb2.a`, and

- If there is a symbolic link from `/usr/lib/libdb2.a` to `/usr/lpp/db2_05_00/lib/libdb2.a`, and the database server is DB2 Universal Database Version 6 for AIX, do one of the following:

  - Change the symbolic link to point to `/usr/lpp/db2_06_01/lib/libdb2.a`.*DB2 for UNIX Quick Beginnings* has information about setting links between libraries. As root, you can change links using the "db2ln" command as follows:

    ```
    /usr/lpp/db2_06_01/cfg/db2ln
    ```
  - Set the LIBPATH environment variable to point to `/usr/lpp/db2_06_01/lib` or `INSTHOME/sqllib/lib`, where INSTHOME is the home directory of the Version 6 DB2 instance owner.
  - Configure a TCP/IP connection from the application (client) instance to the server instance. Refer to the *Installation and Configuration Supplement* for information about configuring TCP/IP.
- If there is a symbolic link from `/usr/lib/libdb2.a` to `/usr/lpp/db2_06_01/lib/libdb2.a`, and the database server is DB2 Version 5, configure a TCP/IP connection from the application (client) instance to the server instance. Refer to the *Installation and Configuration Supplement* for information about configuring TCP/IP.

Condition 2: If a Version 5 application loads a shared library out of the $HOME path of a DB2 Version 5 instance owner (`$HOME/sqllib/lib/libdb2.a`), and the database server is DB2 Universal Database Version 6 for AIX, do one of the following:

- Migrate the application instance to the same version as the database server instance.
- Set the LIBPATH environment variable to point to `/usr/lpp/db2_06_01/lib` or `INSTHOME/sqllib/lib`, where `INSTHOME` is the home directory of the Version 6 instance owner.
- Configure a TCP/IP connection from the application (client) instance to the server instance. Refer to the *Installation and Configuration Supplement* for information about configuring TCP/IP.

Condition 3: If a Version 5 application loads a shared library out of the DB2 Version 5 install path (`/usr/lpp/db2_05_00/lib/libdb2.a`), and the database server is DB2 Universal Database Version 6 for AIX, do one of the following:

- Set the LIBPATH environment variable to point to `/usr/lpp/db2_06_01/lib` or `INSTHOME/sqllib/lib`, where `INSTHOME` is the home directory of the database instance owner.
- Configure a TCP/IP connection from the application (client) instance to the server instance. Refer to the *Installation and Configuration Supplement* for information about configuring TCP/IP.

Condition 4: If a Version 5 application loads a shared library out of the DB2 Universal Database Version 6 for AIX install path (`/usr/lpp/db2_06_01/lib/libdb2.a`), and the database server is DB2 Version 5, configure a TCP/IP connection from the application (client) instance to the server instance. Refer to the *Installation and Configuration Supplement* for information about configuring TCP/IP.

Other Migration Considerations

Consider the following points when you develop your applications. They might help make your applications more portable:

- On UNIX, use only the default path, `/usr/lib:/lib`, in your applications. On OS/2 and Windows 32-bit operating systems, ensure the LIB environment varible points to `%DB2PATH%\lib` by using: `set LIB=%DB2PATH%\lib;%LIB%`. Also, create symbolic links between the default path and the version of DB2 you are using. Ensure that the link is to the minimum level of DB2 required by your applications. Refer to the *Quick Beginnings* book for your platform for information about setting links.
- If your application requires a particular version of DB2, code the path that specifies the DB2 version in your application. For example, if your AIX application requires DB2 Version 2, code /usr/lpp/db2_02_01/lib. Ordinarily, you do not need to do this.
- Generally, the path in your application should not point to the instance owner's copy of the `sqllib/lib` directory on UNIX, or the `%DB2PATH%\lib` directory on OS/2 and Windows 32-bit operating systems. This makes applications highly dependent on specific user names and environments.
- Generally, do not use the LIBPATH environment variable, or the LIB environment variable on Windows 32-bit operating systems, to alter search paths in a particular environment. The variable overrides the search paths specified in the applications running in that environment. Applications might not be able to find the libraries or the files that they need.
- In DB2 Universal Database Version 6, all character array items with string semantics have type char, instead of other variations, such as unsigned char. Any applications you code with DB2 Universal Database Version 6 should follow this practice.

  If you have DB2 Version 1 applications which use unsigned char, your compiler might produce warnings or errors because of type clashes between unsigned char in Version 1 applications and char in Version 6 function prototypes. If this occurs, use the compiler option `-DSQLOLDCHAR` to eliminate the problem.
- Refer to the *SQL Reference* for a list of incompatibilities between DB2 Universal Database Version 6 and previous versions of DB2. Refer to the *Administrative API Reference* for a list of API incompatibilities between DB2 Universal Database Version 6 and previous versions of DB2.

Appendix C. Problem Determination

You may encounter the following kinds of problems when building or running your applications:

- Client or server problems, such as failing to connect to the database during a build or when running your application.
- Operating system problems, such as not being able to find files during a build.
- Compiler option problems during a build.
- Syntax and coding problems during a build or when running your application.

You can use the following sources of information to resolve these problems:

Build files
For build problems such as connecting to a database, precompiling, compiling, linking, and binding, you can use the build 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.

Application Development Guide
Refer to the *Application Development Guide* for syntax and other coding problems.

CLI Guide and Reference
Refer to the *CLI Guide and Reference* for syntax, the CLI Trace facility, configuration keywords, and coding problems related to CLI programs.

SQL Reference
Refer to the *SQL Reference* for syntax of SQL statements and functions.

SQLCA data structure
If your application issues SQL statements or calls database manager APIs, it must check for error conditions by examining the SQLCA data structure.

The SQLCA data structure returns error information in the SQLCODE and SQLSTATE fields. The database manager updates the structure after every SQL statement is executed, and after most database manager API calls.

Your application can retrieve and print the error information or display it on the screen. Refer to the *Application Development Guide* for more information.

Online error messages

Different components of DB2, including the database manager, database administration utility, installation and configuration process, and command line processor, generate online error messages. Each of these messages has a unique prefix and a four or five digit message number following the prefix. A single letter is displayed after the message number indicating the severity of the error.

You can use the command line processor to see the help for the message by entering:

```
db2 "? xxxnnnn"
```

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

For the full list and description of DB2 error messages, see the *Message Reference*.

Diagnostic tools and error log

These are provided for build or runtime problems you cannot resolve using the other sources of information. The diagnostic tools include a trace facility, system log, and message log, among others. DB2 puts error and warning conditions in an error log based on priority and origin. Refer to the *Troubleshooting Guide* for more information. There is also a CLI trace facility specifically for debugging CLI programs. For more information, refer to the *CLI Guide and Reference*.

Appendix D. How the DB2 Library Is Structured

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

To access product information online, you can use the Information Center. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web. See "Accessing Information with the Information Center" on page 412 for details.

Completing Tasks with SmartGuides

SmartGuides help you complete some administration tasks by taking you through each task one step at a time. SmartGuides are available through the Control Center and the Client Configuration Assistant. The following table lists the SmartGuides.

Note: Create Database, Index, and Configure Multisite Update SmartGuide are available for the partitioned database environment.

SmartGuide	Helps You to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click **Add**.
Back up Database	Determine, create, and schedule a backup plan.	From the Control Center, click with the right mouse button on the database you want to back up and select **Backup**->**Database using SmartGuide**.
Configure Multisite Update SmartGuide	Perform a multi-site update, a distributed transaction, or a two-phase commit.	From the Control Center, click with the right mouse button on the **Database** icon and select **Multisite Update**.
Create Database	Create a database, and perform some basic configuration tasks.	From the Control Center, click with the right mouse button on the **Databases** icon and select **Create**->**Database using SmartGuide**.

SmartGuide	Helps You to...	How to Access...
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**.
Index	Advise which indexes to create and drop for all your queries.	From the Control Center, click with the right mouse button on the **Index** icon and select **Create**->**Index using SmartGuide**.
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 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**.

Accessing Online Help

Online help is available with all DB2 components. The following table describes the various types of help. You can also access DB2 information through the Information Center. For information see “Accessing Information with the Information Center” on page 412.

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, while `? catalog database` displays help for the CATALOG DATABASE command.

Type of Help	Contents	How to Access...
Control Center Help Client Configuration Assistant Help Event Analyzer Help Command 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 the **Help** push button or press the F1 key.
Message Help	Describes the cause of a message, and any action you should take.	From the command line processor in interactive mode, enter: ? *XXXnnnnn* where *XXXnnnnn* is a valid message identifier. For example, `? SQL30081` displays help about the SQL30081 message. To view message help one screen at a time, enter: ? *XXXnnnnn* | `more` 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.
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. **Note:** SQL help is not available on UNIX-based platforms.

Type of Help	Contents	How to Access...
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 the first two digits of the SQL state. For example, `? 08003` displays help for the 08003 SQL state, while `? 08` displays help for the 08 class code.

DB2 Information – Hardcopy and Online

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

Cross-platform books
These books contain the common DB2 information for all platforms.

Platform-specific books
These books are for DB2 on a specific platform. For example, there are separate *Quick Beginnings* books for DB2 on OS/2, on Windows NT, and on the UNIX-based platforms.

Cross-platform sample programs in HTML
These samples are the HTML version of the sample programs that are installed with the SDK. They are for informational purposes and do not replace the actual programs.

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

On OS/2 and Windows platforms, HTML documentation files can be installed under the doc\html subdirectory. Depending on the language of your system, some files may be in that language, and the remainder are in English.

On UNIX platforms, you can install multiple language versions of the HTML documentation files under the doc/%L/html subdirectories. Any documentation that is not available in a national language is shown in English.

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

View	See “Viewing Online Information” on page 411.
Search	See “Searching Online Information” on page 414.
Print	See “Printing the PostScript Books” on page 414.
Order	See “Ordering the Printed Books” on page 416.

Name	**Description**	**Form Number** **File Name for Online Book**	**HTML Directory**
	Cross-Platform Books		
Administration Guide	*Administration Guide, Design and Implementation* contains information required to design, implement, and maintain a database. It also describes database access using the Control Center(whether local or in a client/server environment), auditing, database recovery, distributed database support, and high availability. *Administration Guide, Performance* contains information that focuses on the database environment, such as application performance evaluation and tuning. You can order both volumes of the *Administration Guide* in the English language in North America using the form number SBOF-8922.	Volume 1 SC09-2839 db2d1x60 Volume 2 SC09-2840 db2d2x60	db2d0
Administrative 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.	SC09-2841 db2b0x60	db2b0
Application Building Guide	Provides environment setup information and step-by-step instructions about how to compile, link, and run DB2 applications on Windows, OS/2, and UNIX-based platforms. This book combines the *Building Applications* books for the OS/2, Windows, and UNIX-based environments.	SC09-2842 db2axx60	db2ax

Name	Description	Form Number File Name for Online Book	HTML Directory
APPC, CPI-C and SNA Sense Codes	Provides general information about APPC, CPI-C, and SNA sense codes that you may encounter when using DB2 Universal Database products. **Note:** Available in HTML format only.	No form number db2apx60	db2ap
Application Development Guide	Explains how to develop applications that access DB2 databases using embedded SQL or JDBC, how to write stored procedures, user-defined types, user-defined functions, and how to use triggers. It also discusses programming techniques and performance considerations. This book was formerly known as the *Embedded SQL Programming Guide*.	SC09-2845 db2a0x60	db2a0
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.	SC09-2843 db2l0x60	db2l0
Command Reference	Explains how to use the command line processor, and describes the DB2 commands you can use to manage your database.	SC09-2844 db2n0x60	db2n0
Data Movement Utilities Guide and Reference	Explains how to use the Load, Import, Export, Autoloader, and Data Propogation utilities to work with the data in the database.	SC09-2858 db2dmx60	db2dm
DB2 Connect Personal Edition Quick Beginnings	Provides planning, installing, and configuring information for DB2 Connect Personal Edition.	GC09-2830 db2c1x60	db2c1
DB2 Connect User's Guide	Provides concepts, programming and general usage information about the DB2 Connect products.	SC09-2838 db2c0x60	db2c0

Name	Description	Form Number File Name for Online Book	HTML Directory
Connectivity Supplement	Provides setup and reference information on how 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 on how to use DRDA application servers with DB2 Connect application requesters. **Note:** Available in HTML and PostScript formats only.	No form number db2h1x60	db2h1
Glossary	Provides a comprehensive list of all DB2 terms and definitions. **Note:** Available in HTML format only.	No form number db2t0x50	db2t0
Installation and Configuration Supplement	Guides you through the planning, installation, and set up of platform-specific DB2 clients. This supplement contains information on binding, setting up client and server communications, DB2 GUI tools, DRDA AS, distributed installation, and the configuration of distributed requests and access methods to heterogeneous data sources.	GC09-2857 db2iyx60	db2iy
Message Reference	Lists messages and codes issued by DB2, and describes the actions you should take.	GC09-2846 db2m0x60	db2m0
Replication Guide and Reference	Provides planning, configuration, administration, and usage information for the IBM Replication tools supplied with DB2.	SC26-9642 db2e0x60	db2e0
SQL Getting Started	Introduces SQL concepts, and provides examples for many constructs and tasks.	SC09-2856 db2y0x60	db2y0
SQL Reference, Volume 1 and Volume 2	Describes SQL syntax, semantics, and the rules of the language. Also includes information about release-to-release incompatibilities, product limits, and catalog views. You can order both volumes of the *SQL Reference* in the English language in North America with the form number SBOF-8923.	SBOF-8923 Volume 1 db2s1x60 Volume 2 db2s2x60	db2s0

Name	Description	Form Number File Name for Online Book	HTML Directory
System Monitor Guide and Reference	Describes how to collect different kinds of information about databases and the database manager. Explains how to use the information to understand database activity, improve performance, and determine the cause of problems.	SC09-2849 db2f0x60	db2f0
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	S10J-8169	db2p0
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database, Version 6.0, including information about Java-based tools.	SC09-2851 db2q0x60	db2q0
	Platform-Specific Books		
Administering Satellites Guide and Reference	Provides planning, configuration, administration, and usage information for satellites.	GC09-2821 db2dsx60	db2ds
DB2 Personal Edition Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on the OS/2, Windows 95, and Windows NT operating systems.	GC09-2831 db2i1x60	db2i1
DB2 for OS/2 Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the OS/2 operating system. Also contains installing and setup information for many supported clients.	GC09-2834 db2i2x60	db2i2
DB2 for UNIX Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on UNIX-based platforms. Also contains installing and setup information for many supported clients.	GC09-2836 db2ixx60	db2ix
DB2 for Windows NT Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the Windows NT operating system. Also contains installing and setup information for many supported clients.	GC09-2835 db2i6x60	db2i6

Name	Description	Form Number File Name for Online Book	HTML Directory
DB2 Enterprise - Extended Edition for UNIX Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for UNIX. Also contains installing and setup information for many supported clients.	GC09-2832 db2v3x60	db2v3
DB2 Enterprise - Extended Edition for Windows NT Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for Windows NT. Also contains installing and setup information for many supported clients.	GC09-2833 db2v6x60	db2v6
DB2 Connect Enterprise Edition for OS/2 and Windows NT Quick Beginnings	Provides planning, migration, installation, and configuration information for DB2 Connect Enterprise Edition on the OS/2 and Windows NT operating systems. Also contains installation and setup information for many supported clients. This book was formerly part of the *DB2 Connect Enterprise Edition Quick Beginnings.*	GC09-2828 db2c6x60	db2c6
DB2 Connect Enterprise Edition for UNIX Quick Beginnings	Provides planning, migration, installation, configuration, and usage information for DB2 Connect Enterprise Edition in UNIX-based platforms. Also contains installation and setup information for many supported clients. This book was formerly part of the *DB2 Connect Enterprise Edition Quick Beginnings.*	GC09-2829 db2cyx60	db2cy
DB2 Data Links Manager for AIX Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for AIX.	GC09-2837 db2z0x60	db2z0
DB2 Data Links Manager for Windows NT Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for Windows NT.	GC09-2827 db2z6x60	db2z6
DB2 Query Patroller Administration Guide	Provides administration information on DB2 Query Patrol.	SC09-2859 db2dwx60	db2dw

Name	Description	Form Number File Name for Online Book	HTML Directory
DB2 Query Patroller Installation Guide	Provides installation information on DB2 Query Patrol.	SC09-2860 db2iwx60	db2iw
DB2 Query Patroller User's Guide	Describes how to use the tools and functions of the DB2 Query Patrol.	SC09-2861 db2wwx60	db2ww
	Cross-Platform Sample Programs in HTML		
Sample programs in HTML	Provides the sample programs in HTML format for the programming languages on all platforms supported by DB2 for informational purposes (not all samples are available in all languages). Only available when the SDK is installed. See *Application Building Guide* for more information on the actual programs. **Note:** Available in HTML format only.	No form number	db2hs/c db2hs/cli db2hs/clp db2hs/cpp db2hs/cobol db2hs/cobol_mf db2hs/fortran db2hs/java db2hs/rexx

Notes:

1. The character in the sixth position of the file name indicates the language of a book. For example, the file name `db2d0e60` 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
Brazilian Portuguese	b
Bulgarian	u
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p

Portuguese	v
Russian	r
Simp. Chinese	c
Slovenian	l
Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

2. For late breaking information that could not be included in the DB2 books:
   - On UNIX-based platforms, see the Release.Notes file. This file is located in the DB2DIR/Readme/%L directory, where `%L` is the locale name and DB2DIR is:
     - /usr/lpp/db2_06_01 on AIX
     - /opt/IBMdb2/V6.1 on HP-UX, Solaris, SCO UnixWare 7, and Silicon Graphics IRIX
     - /usr/IBMdb2/V6.1 on Linux.
   - On other platforms, see the RELEASE.TXT file. This file is located in the directory where the product is installed.
   - Under Windows `Start` menu

Viewing Online Information

The manuals included with this product are in Hypertext Markup Language (HTML) softcopy format. Softcopy format enables you to search or browse the information, and provides hypertext links to related information. It also makes it easier to share the library across your site.

You can view the online books or sample programs with any browser that conforms to HTML Version 3.2 specifications.

To view online books or sample programs on all platforms other than SCO UnixWare 7:

- If you are running DB2 administration tools, use the Information Center. See "Accessing Information with the Information Center" on page 412 for details.
- Select the `Open Page` menu item of your Web browser. The page you open contains descriptions of and links to DB2 information:
  - On UNIX-based platforms, open the following page:

    ```
    file:/INSTHOME/sqllib/doc/%L/html/index.htm
    ```

    where *%L* is the locale name.

– On other platforms, open the following page:

```
sqllib\doc\html\index.htm
```

The path is located on the drive where DB2 is installed.

If you have not installed the Information Center, you can 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.

To view online books or sample programs on the SCO UnixWare 7:

- DB2 Universal Database for SCO UnixWare 7 uses the native SCOhelp utility to search the DB2 information. You can access SCOhelp by the following methods:
  – entering the "scohelp" command on the command line,
  – selecting the `Help` menu in the Control Panel of the CDE desktop or
  – selecting `Help` in the Root menu of the Panorama desktop

  For more information on SCOhelp, refer to the *Installation and Configuration Supplement*.

Accessing Information with the Information Center

The Information Center provides quick access to DB2 product information. The Information Center is available on all platforms on which the DB2 administration tools are available.

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

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

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

Tasks	Lists tasks you can perform using DB2.
Reference	Lists DB2 reference information, such as keywords, commands, and APIs.
Books	Lists DB2 books.
Troubleshooting	Lists categories of error messages and their recovery actions.
Sample Programs	Lists sample programs that come with the

	DB2 Software Developer's Kit. If the Software Developer's Kit is not installed, this tab is not displayed.
Web	Lists DB2 information on the World Wide Web. To access this information, you must have a connection to the Web from your system.

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

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

For a full text search, click the Search button of the Information Center follow the Search DB2 Books link in each HTML file.

The HTML search server is usually started automatically. If a search in the HTML information does not work, you may have to start the search server by double-clicking its icon on the Windows or OS/2 desktop.

Refer to the release notes if you experience any other problems when searching the HTML information.

Note: Search function is not available in the Linux and Silicon Graphics environments.

Setting Up a Document Server

By default, the DB2 information is installed on your local system. This means that each person who needs access to the DB2 information must install the same files. To have the DB2 information stored in a single location, use the following instructions:

1. Copy all files and subdirectories from \sqllib\doc\html on your local system to a Web server. Each book has its own subdirectory containing all the necessary HTML and GIF files that make up the book. Ensure that the directory structure remains the same.
2. Configure the Web server to look for the files in the new location. For information, see the NetQuestion Appendix in *Installation and Configuration Supplement*.

3. If you are using the Java version of the Information Center, you can specify a base URL for all HTML files. You should use the URL for the list of books.
4. Once you are able to view the book files, you should bookmark commonly viewed topics. Among those, you will probably want to bookmark the following pages:
   - List of books
   - Tables of contents of frequently used books
   - Frequently referenced articles, such as the ALTER TABLE topic
   - The Search form

For information about setting up a search, see the NetQuestion Appendix in *Installation and Configuration Supplement* book.

Searching Online Information

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. This function is not available in the Linux or Silicon Graphics IRIX environments.
- Click on **Index** at the bottom of any page in an HTML book. Use the index to find a specific topic in the book.
- Display the table of contents or index of the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. See “Accessing Information with the Information Center” on page 412 for details.

Printing the PostScript Books

If you prefer to have printed copies of the manuals, you can decompress and print PostScript versions. For the file name of each book in the library, see the table in “DB2 Information – Hardcopy and Online” on page 404. Specify the full path name for the file you intend to print.

On OS/2 and Windows platforms:

1. Copy the compressed PostScript files to a hard drive on your system. The files have a file extension of .exe and are located in the x:\doc\language\books\ps directory, where x: is the letter representing the

CD-ROM drive and *language* is the two-character country code that represents your language (for example, EN for English).

2. Decompress the file that corresponds to the book that you want. Each compressed book is a self-extracting executable file. To decompress the book, simply run it as you would run any other executable program. The result from this step is a printable PostScript file with a file extension of .ps.
3. Ensure that your default printer is a PostScript printer capable of printing Level 1 (or equivalent) files.
4. Enter the following command from a command line:
   ```
   print filename.ps
   ```

On UNIX-based platforms:

1. Mount the CD-ROM. Refer to your *Quick Beginnings* manual for the procedures to mount the CD-ROM.
2. Change to /cdrom/doc/%L/ps directory on the CD-ROM, where */cdrom* is the mount point of the CD-ROM and *%L* is the name of the desired locale. The manuals will be installed in the previously-mentioned directory with file names ending with .ps.Z.
3. Decompress and print the manual you require using the following command:
   - For AIX:
     ```
     zcat filename | qprt -P PSPrinter_queue
     ```
   - For HP-UX, Solaris, or SCO UnixWare 7:
     ```
     zcat filename | lp -d PSPrinter_queue
     ```
   - For Linux:
     ```
     zcat filename | lpr -P PSPrinter_queue
     ```
   - For Silicon Graphics IRIX:
     ```
     zcat < filename | lp -d PSPrinter_queue
     ```

   where *filename* is the full path name and extension of the compressed PostScript file and *PSprinter_queue* is the name of the PostScript printer queue.

   For example, to print the English version of *DB2 for UNIX Quick Beginnings* on AIX, you can use the following command:
   ```
   zcat /cdrom/doc/en/ps/db2ixe60.ps.Z || qprt -P ps1
   ```

Ordering the Printed Books

You can order the printed DB2 manuals either as a set or individually. There are three sets of books available. The form number for the entire set of DB2 books is SBOF-8926-00. The form number for the books listed under the heading ″Cross-Platform Books″ is SBOF-8924-00.

Note: These form numbers only apply if you are ordering books that are printed in the English language in North America.

You can also order books individually by the form number listed in "DB2 Information – Hardcopy and Online" on page 404. To order printed versions, contact your IBM authorized dealer or marketing representative, or phone 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.

Appendix E. Notices

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

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

IBM Director of Licensing
IBM Corporation, North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

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

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

Trademarks

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

ACF/VTAM
ADSTAR
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
Extended Services
FFST
First Failure Support Technology
IBM
IMS
LAN Distance
MVS/ESA
MVS/XA
OS/400
OS/390
OS/2
PowerPC
QMF
RACF
RISC System/6000
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. in the United States, other countries, or both.

HP-UX is a trademark of Hewlett-Packard.

Java, HotJava, Solaris, Solstice, and Sun are trademarks of Sun Microsystems, Inc.

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 accessing the following page:

`http://www.ibm.com/support/`

then performing a search using the keyword “handbook”.

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

World Wide Web

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

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

Anonymous FTP Sites

ftp.software.ibm.com

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

Internet Newsgroups
comp.databases.ibm-db2, bit.listserv.db2-l
These newsgroups are available for users to discuss their experiences with DB2 products.

CompuServe
GO IBMDB2 to access the IBM DB2 Family forums
All DB2 products are supported through these forums.

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

Part Number: CT7UMNA

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

CT7UMNA

SC09-2842-00