IBM® DB2® Universal Database

Application Building Guide

Version 7

SC09-2948-00

IBM® DB2® Universal Database

Application Building Guide

Version 7

SC09-2948-00

Before using this information and the product it supports, be sure to read the general information under "Appendix E. Notices" on page 391.

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

Contents

Welcome to DB2 Application Development

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

The section "The DB2 Developer's Edition" guides you to the installation information for your particular development needs. This information will help you decide how to install DB2 from either the IBM DB2 Universal Developer's Edition, Version 7.1, or the IBM DB2 Personal Developer's Edition, Version 7.1.

The section "DB2 Application Development Books" on page ix describes the main books in the DB2 library for application development.

The section "DB2 Programming Interfaces" on page x presents key programming interface concepts for DB2 application development.

The section "DB2 Features" on page xviii 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 that run on OS/2, Linux, and Windows 32-bit operating systems. The DB2 Universal Developer's Edition provides DB2 products on these platforms as well as on AIX, HP-UX, PTX, Silicon Graphics IRIX, and the Solaris** Operating Environment**.

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 on 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 use for 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. Each CD-ROM contains the DB2 server, Administration Client, Application Development Client, and Run-Time Client for a supported operating system. 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 the product. The Personal Developer's Edition box contains the CD-ROMs for Personal Edition products for the 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 CD-ROM with a Netscape browser for several operating systems. This allows you to view the HTML information in case you do not have a browser installed on your machine.
- A DB2 publications CD-ROM containing DB2 books in PDF format.
- DB2 Extenders for supported platforms.
- OLAP Starter Kit for supported platforms. You must install a DB2 server before installing the OLAP Starter Kit.
- In addition, you get copies of other software that you may find useful for developing applications. This software may vary from time to time, and is accompanied by license agreements for use.

The Universal Developer's Edition also contains the following:

- A set of CD-ROMs containing Administration Clients for all platforms. These clients contain tools for administering databases, such as the Control Center and the Event Analyzer. These clients also allow you to run applications on any system.
- A set of CD-ROMs containing Application Development Clients for all platforms. These clients have application development tools, sample programs, and header files. Each DB2 AD Client includes everything you need to develop your applications.
- A set of CD-ROMs containing Run-Time Clients for all platforms. An application can be run from a Run-Time Client on any system. The Run-Time Client does not have some of the features of the Administration Client, such as the DB2 Control Center and Event Analyzer, and so takes up less space.
- A CD-ROM containing Net.Data for supported platforms.

Installation Information

Each DB2 product CD-ROM contains installation information in HTML directly viewable from the CD-ROM. The *Quick Beginnings* book for the supported platform is provided, which explains how to install DB2 servers, Administration Clients, Application Development Clients, and Run-Time Clients.

Additional information is available in the *Installation and Configuration Supplement*. This book is only viewable from the Client CD-ROMs.

The book you want will be in a subdirectory for the language you are using. The CD-ROM README.TXT file tells you where to find the book files on the CD-ROM.

With your browser running, click on the `index.htm` file in the book subdirectory. Here are the subdirectories for the *Quick Beginnings* and *Installation and Configuration Supplement* books.

db2i2 *DB2 for OS/2 Quick Beginnings*

db2ix *DB2 for UNIX Quick Beginnings*

db2i6 *DB2 for Windows Quick Beginnings*

db2iy *Installation and Configuration Supplement*

The DB2 publications CD-ROM contains PDF files for all the DB2 books shipped with the product. They are viewable directly from the CD-ROM with an Adobe Acrobat Reader. The DB2 books available in your language are in the appropriate language subdirectory. For the installation information, you can access the *Quick Beginnings* and *Installation and Configuration Supplement* books. Their file names begin with the sets of characters listed above.

For instructions on printing the PDF files, see "Printing the PDF Books" on page 380.

See the README.TXT file on the CD-ROM for full details on how to access the book files.

DB2 Application Development Books

The DB2 library is described in "Appendix D. Using the DB2 Library" on page 371. 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, as well as 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 manage or access DB2 databases. You can:

1. Use DB2 APIs to perform administrative functions such as backing up and restoring databases.
2. Embed static and dynamic SQL statements in your applications.
3. Code DB2 Call Level Interface (DB2 CLI) function calls in your applications to invoke dynamic SQL statements.
4. Develop Java applications and applets that call the Java Database Connectivity application programming interface (JDBC API).
5. 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.

6. 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.

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, bind and then compile static SQL statements before you run your application. Static SQL is best run on databases whose schema does 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 should 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 AD Client. 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.ibm.com/software/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.

It is also possible to run the SQL statements as static SQL in a CLI, ODBC or JDBC application. The CLI/ODBC/JDBC Static Profiling feature enables end users of an application to replace the use of dynamic SQL with static SQL in many cases. For more information, see:

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

You can build an ODBC application without using an ODBC driver manager, and simply use DB2's ODBC driver on any platform by linking your application with `libdb2` on UNIX, and `db2cli.lib` on OS/2 and Windows 32-bit operating systems. The DB2 CLI sample programs demonstrate this. They are located in `sqllib/samples/cli` on UNIX and `%DB2PATH%\samples\cli` on OS/2 and Windows 32-bit operating systems.

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 AD Client 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 42.

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

All DB2 CLI users share the same privileges. Embedded SQL offers the advantage of more granular security through granting execute privileges to particular users for a package.

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.ibm.com/software/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 xxiii.

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 Microsoft OLE DB to ODBC 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 development time, ease of use, 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.ibm.com/software/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.ibm.com/software/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.

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.

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 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.

OLE Automation UDFs and Stored Procedures

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 32-bit operating systems provides access to OLE automation objects using UDFs and stored procedures. To access OLE automation objects and invoke their methods, you must register the methods of the objects as UDFs or stored procedures. DB2 applications can then invoke the methods by calling the UDFs or stored procedures. 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.

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.

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.

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. On Windows 32-bit operating systems, it can be launched from these popular application development tools: Microsoft Visual Studio, Microsoft Visual Basic, and IBM VisualAge for Java, or launched as a separate application from the IBM DB2 Universal Database program group. It can also be started by executing the following file:

```
%DB2PATH%\bin\DB2SPB.exe
```

where `%DB2PATH%` points to the directory where DB2 is installed.

On AIX and Solaris, the Stored Procedure Builder can be started with the `db2spb` command.

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 Application Development (DB2 AD) Client for DB2 Universal Database Version 7.1 on the following platforms:

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

Notes:

1. DB2 for NUMA-Q supports the PTX operating system.
2. Windows 32-bit operating systems includes Windows NT, Windows 95, Windows 98, and Windows 2000. Whenever this book mentions Windows 32-bit operating systems, all of these operating systems are implied, except in the case of Systems Network Architecture (SNA) support, and REXX support. These are supported on Windows NT and Windows 2000 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). It also discusses user-defined functions (UDFs).
- 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 Application Development Client on OS/2 server and client workstations.
- *DB2 for UNIX Quick Beginnings*, which explains how to install the database manager, and the DB2 Application Development Client on UNIX server and client workstations.
- *DB2 for Windows Quick Beginnings*, which explains how to install the database manager, and the DB2 Application Development Client 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. Using the DB2 Library" on page 371.

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 7.1. 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 chapters are grouped as follows:

- Chapters 1 to 3: each contains general, introductory information for all platforms.
- Chapters 4 and 5: each contains specific programming information for all platforms.
- Chapters 6 to 13: each contains programming information specific to one platform.

All application developers should read the first three chapters, and then read the "Building Applications" chapter containing the specific programming information they will need, depending on the operating system and programming language they will be using.

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 Application Development Client

Note: The Application Development Client was known as the DB2 Software Development Kit (DB2 SDK) Client in previous versions of DB2.

The DB2 Application Development (DB2 AD) Client 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 AD Client installed. You can also 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 AD Clients 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).
- **Embedded SQL application support**, including programming libraries, include files and code samples.

- **DB2 Call Level Interface (DB2 CLI) application support**, including programming libraries, include files, and code samples to develop applications which are easily ported to ODBC and compiled with an ODBC SDK. An ODBC SDK 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. Only DB2 Clients for Windows 32-bit operating systems contain an ODBC driver manager.
- **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.
- **Java Development Kit (JDK) 1.1.8 and Java Runtime Environment (JRE) 1.1.8** from IBM, installed with DB2 for AIX and DB2 for Windows 32-bit operating systems, and shipped with DB2 for OS/2.
- **REXX language support** on AIX (32-bit applications only), OS/2, and Windows 32-bit operating systems. This support is not updated beyond DB2 Version 5.2.
- **ActiveX Data Objects (ADO) and Object Linking and Embedding (OLE) automation UDFs and Stored Procedures** on Windows 32-bit operating systems, including code samples implemented in Microsoft Visual Basic and Microsoft Visual C++. Also, code samples with Remote Data Objects (RDO) implemented in Microsoft Visual Basic.
- **Object Linking and Embedding Database (OLE DB) table functions** on Windows 32-bit operating systems.
- **DB2 Stored Procedure Builder (SPB)**, available on AIX, Solaris, and Windows 32-bit operating systems. This is 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. On Windows, it can be launched from these popular application development tools: Microsoft Visual Studio, Microsoft Visual Basic, and IBM VisualAge for Java, or launched as a separate application from the IBM DB2 Universal Database program group. On AIX and Solaris, it can be started with the `db2spb` command.
- **Interactive SQL** through the Command Center or 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 uses this interface. Information on

the set of Precompiler Services APIs 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 order it from 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 AD client 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 NUMA-Q
- DB2 for SCO UnixWare 7
- DB2 for Solaris
- DB2 for Windows NT
- Distributed Relational Database Architecture (DRDA)-compliant application servers, such as:
  - DB2 for OS/390
  - DB2 for AS/400
  - DB2 for VSE & VM (formerly SQL/DS for VM and VSE)
  - DRDA-compliant application servers from database vendors other than IBM.

Notes:

1. DB2 for NUMA-Q supports the PTX operating system.
2. DB2 for SCO UnixWare 7 is only available for DB2 Version 5.2.

Supported Software by Platform

This section lists the compilers and related software supported by DB2 for the platforms described in this book. The compiler information assumes that you are using the DB2 precompiler for that platform, and not the precompiler support that may be built into one of the listed compilers. 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 DB2 application development Web page at:

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

Notes:

1. The **DB2 Release Notes** may contain updated compiler and operating system information for supported platforms. The Release Notes are available in flat text and HTML formats in the following paths on your product CD-ROM, where `<language_directory>` is the directory for the language you are using, and `index.htm` is the main HTML file:

   Flat text file:

   doc/<language_directory>/release.txt (UNIX)

   Doc\<language_directory>\release.txt (OS/2 and Windows)

   HTML file:

   doc/<language_directory>/db2ir/index.htm (UNIX)

   Doc\<language_directory>\db2ir\index.htm (OS/2 and Windows)
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. **Fortran.** Fortran sample programs are not provided in DB2 version 7.1. For information on obtaining Fortran samples for DB2 version 6.1, visit the DB2 Application Development web page given above.
4. **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.
5. **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.

6. **Perl**. Release 0.71 of the DB2 UDB driver (DBD::DB2) for the Perl Database Interface (Perl DBI) is available for AIX, HP-UX, Linux, Solaris and Windows NT. This driver can be downloaded from:

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

7. **REXX**. IBM Object REXX for Windows NT/95 is no longer shipped with DB2. For information on obtaining Object REXX, visit:

   `http://www.ibm.com/software/ad/obj-rexx/`

AIX

DB2 for AIX supports the following operating system:

AIX/6000
Version 4.2.1 and later

(Version 4.3.3 and later for 64-bit)

DB2 for AIX supports the following programming languages and compilers:

C IBM C for AIX Version 3.6.6 (Version 3.6.6.3 for 64-bit)

C++ IBM C Set++ for AIX Version 3.6.6 (Version 3.6.6.3 for 64-bit)

IBM VisualAge C++ Version 4.0

COBOL
IBM COBOL Set for AIX Version 1.1

Micro Focus COBOL Version 4.0.20 (PRN 12.03 or later)

Micro Focus COBOL Version 4.1.10 (PRN 13.04 or later)

Fortran
IBM XL Fortran for AIX Versions 4.1 (for 32-bit) and 5.1.0 (for 32-bit and 64-bit)

Java Java Development Kit (JDK) Version 1.1.8 and Java Runtime Environment (JRE) Version 1.1.8 for AIX from IBM (installed with DB2)

Java Development Kit (JDK) Version 1.2.2 and Java Runtime Environment (JRE) Version 1.2.2 for AIX from IBM

Perl Release 0.71 of the DB2 UDB driver (DBD::DB2) for the Perl Database Interface (Perl DBI) (see note above)

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

IBM Object REXX for AIX Version 1.1

REXXSAA 4.00

Note: REXX support is for 32-bit only

HP-UX

DB2 for HP-UX supports the following operating systems:

HP-UX
Version 11.0 with the HP-UX Core OS Year 2000 Patch Bundle Version B.11.00.A1214 (Y2K-1100), or later patch bundles.

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

C HP C Compiler version A.11.00.03

C++ HP-UX C++ Version A.12.00

COBOL
Micro Focus COBOL Version 4.1

Fortran
HP Fortran/9000 Version 10.0

HP-UX F77 B.11.00.01

Java HP-UX Developer's Kit for Java Release 1.1.8 from Hewlett-Packard

Perl Release 0.71 of the DB2 UDB driver (DBD::DB2) for the Perl Database Interface (Perl DBI) (see note above)

Linux

DB2 for Linux supports the following operating system:

Linux kernel Version 2.2.12 or later, glibc Version 2.1.2 or later, libstdc++ Version 2.9.0, rpm (required to install), and the pdksh package (required to run the DB2 command line processor)

DB2 for Linux supports the following programming languages and compilers:

C GNU/Linux gcc version egcs-2.91.66 (egcs-1.1.2 release)

C++ GNU/Linux g++ version egcs-2.91.66 (egcs-1.1.2 release)

Java IBM Developer kit and Runtime Environment for Linux, Version 1.1.8

Perl Release 0.71 of the DB2 UDB driver (DBD::DB2) for the Perl Database Interface (Perl DBI) (see note above)

OS/2

DB2 for OS/2 supports the following operating systems:

OS/2 WARP 3.0, WARP 4.0, and WARP 4.5

DB2 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 Version 2.0

Micro Focus COBOL Version 4.0.20

FORTRAN
WATCOM FORTRAN 77 32 Version 10.5

Java Java Development Kit (JDK) Version 1.1.8 and Java Runtime Environment (JRE) Version 1.1.8 for OS/2 from IBM (shipped with DB2)

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

PTX

DB2 for NUMA-Q supports the following operating system:

PTX Version 4.5

DB2 for NUMA-Q supports the following programming languages and compilers:

C ptx/C Versions 4.5

C++ ptx/C++ Version 5.2

Java ptx/JSE Version 3.0

Silicon Graphics IRIX

DB2 for Silicon Graphics IRIX supports the following operating system:

Silicon Graphics IRIX
Version 6.2 and later

DB2 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 Java2 Software Development Kit Version 1.2.1 (JDK 1.2.1) from Silicon Graphics, Inc.

Solaris

DB2 for Solaris supports the following operating system:

Solaris
Versions 2.6, Solaris 7, and Solaris 8

DB2 for Solaris supports the following programming languages and compilers:

C SPARCompiler C Versions 4.2 (for 32-bit) and 5.0 (for 32-bit and 64-bit)

C++ SPARCompiler C++ Version 4.2 (for 32-bit) and 5.0 (for 32-bit and 64-bit)

COBOL
Micro Focus COBOL Version 4.0

Fortran
SPARCompiler Fortran Versions 4.2 and 5.0

Java Java Development Kit (JDK) Versions 1.1.8 and 1.2 for Solaris from Sun Microsystems

Perl Release 0.71 of the DB2 UDB driver (DBD::DB2) for the Perl Database Interface (Perl DBI) (see note above)

Windows 32-bit Operating Systems

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

Microsoft Windows NT
Version 4.0 with Service Pack 4 or later.

Microsoft Windows 2000

Microsoft Windows 98

Microsoft Windows 95
Version 4.00.950 or later

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

Basic Microsoft Visual Basic Version 4.2 and Version 5.0 (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 4.2 and Version 5.0

COBOL
Micro Focus COBOL Version 4.0.20

IBM VisualAge COBOL Version 2.0

REXX IBM Object REXX for Windows NT/95 Version 1.1 (See note above)

Java Java Development Kit (JDK) 1.1.8 and Java Runtime Environment (JRE) 1.1.8 for Win32 from IBM (installed with DB2)

Java Development Kit (JDK) 1.2 for Win32 from Sun Microsystems

Microsoft Software Developer's Kit for Java, Version 3.1

Perl Release 0.71 of the DB2 UDB driver (DBD::DB2) for the Perl Database Interface (Perl DBI) (Available on Windows NT. See note above.)

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 platforms or 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 Application Development (DB2 AD) Client. You can use the sample programs as templates to create your own applications.

Sample program file extensions differ for each supported language, and for embedded SQL and non-embedded SQL programs within each language. File extensions may also differ for groups of programs within a language. These different sample file extensions are categorized in the following tables:

Sample File Extensions by Language
Table 1 on page 13.

Sample File Extensions by Program Group
Table 2 on page 14.

The following tables document the sample programs by type:

DB2 API Sample Programs with No Embedded SQL
Table 3 on page 16.

DB2 API Embedded SQL Sample Programs
Table 4 on page 19.

Embedded SQL Sample Programs with No DB2 APIs
Table 5 on page 20.

User-Defined Function Sample Programs
Table 6 on page 22

DB2 CLI Sample Programs
Table 7 on page 22.

Java JDBC Sample Programs
Table 8 on page 24.

Java SQLJ Sample Programs
Table 9 on page 25.

SQL Procedure Sample Programs
Table 10 on page 26.

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 31.

Notes:

1. Table 4 on page 19 contains programs that have both DB2 APIs and embedded SQL statements. For all DB2 API sample programs, please see both Table 3 on page 16 and Table 4 on page 19. For all embedded SQL sample programs (except for Java SQLJ), please see both Table 4 on page 19 and Table 5 on page 20.
2. Table 6 on page 22 of UDF sample programs does not contain DB2 CLI UDF programs. For these, please see Table 7 on page 22.

Table 1. 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`
JAVA	`samples/java`	`.sqlj`	`.java`
REXX	`samples/rexx`	`.cmd`	`.cmd`

Table 2. Sample File Extensions by Program Group

Sample Group	Directory	File Extension
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++)
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
SQL Procedures	samples/sqlproc	.db2 .c .sqc (Client Applications)
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 AIX, OS/2, and Windows 32-bit operating systems in the cobol subdirectory.

Micro Focus Cobol Samples
Are only supplied for AIX, HP-UX, OS/2, Solaris Operating Environment, and Windows 32-bit operating systems in the cobol_mf subdirectory.

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 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.

DB2 API Non-Embedded SQL Samples

Table 3. 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
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

Table 3. DB2 API Sample Programs with No Embedded SQL (continued)

Sample Program	Included APIs
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
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

Table 3. DB2 API Sample Programs with No Embedded SQL (continued)

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

DB2 API Embedded SQL Samples

Table 4. DB2 API Embedded SQL Sample Programs

Sample Program	Included APIs
asynrlog	• sqlurlog - Asynchronous Read Log
autocfg	• db2AutoConfig -- Autoconfig • db2AutoConfigMemory -- Autoconfig Free Memory • sqlfudb -- Update Database Configuration • sqlfusys -- Update Database Manager Configuration • sqlesetc -- Set Client • sqlaintp -- SQLCA Message
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
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

Table 4. DB2 API Embedded SQL Sample Programs (continued)

Sample Program	Included APIs
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
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
utilemb	• sqlaintp - Get Error Message • sqlogstt - Get SQLSTATE Message

Embedded SQL Samples With No DB2 APIs

Table 5. 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.

Table 5. Embedded SQL Sample programs with No DB2 APIs (continued)

Sample Program Name	Program Description
`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.
`columns`	Demonstrates the use of a cursor that is processed using dynamic SQL. This program lists a result set from SYSCAT.COLUMNS 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.
`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.
`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.
`spclient`	A client application that calls stored procedures in the `spserver` shared library.
`spcreate.db2`	A CLP script that contains the CREATE PROCEDURE statements to register the stored procedures created by the `spserver` program.
`spdrop.db2`	A CLP script that contains the DROP PROCEDURE statements necessary for deregistering the stored procedures created by the `spserver` program.
`spserver`	A server program demonstrating stored procedures. The client program is `spclient`.
`static`	Demonstrates static SQL to retrieve information.
`tabsql`	Demonstrates the use of advanced SQL table expressions.
`tbdefine`	Demonstrates creating and dropping tables.

Table 5. Embedded SQL Sample programs with No DB2 APIs (continued)

Sample Program Name	Program Description
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.
udfcli	Demonstrates calling a user-defined function (UDF) created by the udfsrv program, and stored on the server to access tables in the sample database.
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 6. 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.
udfsrv.c	Creates a library with the User-Defined Function ScalarUDF, to access the sample database tables.
UDFsrv.java	Demonstrates the use of Java User-Defined Functions (UDFs).

DB2 Call Level Interface Samples

Table 7. Sample CLI Programs in DB2 Universal Database

Sample Program Name	Program Description
Common Utility Files	
utilcli.c	Utility functions used in CLI samples.
utilapi.c	Utility functions that call DB2 APIs.
Application Level - Samples that deal with the application level of DB2 and CLI.	
apinfo.c	How to get and set application level information.
aphndls.c	How to allocate and free handles.
apsqlca.c	How to work with SQLCA data.

Table 7. Sample CLI Programs in DB2 Universal Database (continued)

Sample Program Name	Program Description
Installation Image Level - Samples that deal with the installation image level of DB2 and CLI.	
`ilinfo.c`	How to get and set installation level information (such as the version of the CLI driver).
Instance Level - Samples that deal with the instance level of DB2 and CLI.	
`ininfo.c`	How to get and set instance level information.
Database Level - Samples that deal with database objects in DB2.	
`dbconn.c`	How to connect and disconnect from a database.
`dbinfo.c`	How to get and set information at a database level.
`dbmconn.c`	How to connect and disconnect from multiple databases (uses DB2 APIs to create and drop second database).
`dbmuse.c`	How to perform transactions with multiple databases (uses DB2 APIs to create and drop second database).
`dbnative.c`	How to translate a statement that contains an ODBC escape clause to a data source specific format.
`dbuse.c`	How to work with database objects.
`dbusemx.sqc`	How to use a single database in conjunction with embedded SQL.
Table Level - Samples that deal with table objects in DB2.	
`tbconstr.c`	How to work with table constraints.
`tbconstr.c`	How to create, alter and drop tables.
`tbinfo.c`	How to get and set information at a table level.
`tbmod.c`	How to modify information in a table.
`tbread.c`	How to read information in a table.
Data Type Level - Samples that deal with data types.	
`dtinfo.c`	How to get information about data types.
`dtlob.c`	How to read and write LOB data.
`dtudt.c`	How to create, use, and drop user defined distinct types.
UDF Level - Samples that demonstrate user defined functions.	
`udfcli.c`	Client application which calls the user defined function in udfsrv.c.
`udfsrv.c`	User defined function ScalarUDF called by udfcli.c sample.
Stored Procedure Level - Samples that demonstrate stored procedures in CLI.	
`spcreate.db2`	CLP script to issue CREATE PROCEDURE statements.
`spdrop.db2`	CLP script to drop stored procedures from the catalog.
`spclient.c`	Client program used to call the server functions declared in spserver.c.

Table 7. Sample CLI Programs in DB2 Universal Database (continued)

Sample Program Name	Program Description
spserver.c	Stored procedure functions built and run on the server.
spcall.c	Program to call any stored procedure.

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 8. Java Database Connectivity (JDBC) Sample Programs

Sample Program Name	Program Description
DB2Appl.java	A JDBC application that queries the sample database using the invoking user's privileges.
DB2Applt.java	A JDBC applet that queries the 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.
DB2UdCli.java	A Java client application that calls the Java user-defined function, DB2Udf.
Dynamic.java	Demonstrates a cursor using dynamic SQL.
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.
Outcli.java	A Java client application that calls the SQLJ stored procedure, Outsrv.
PluginEx.java	A Java program that adds new menu items and toolbar buttons to the DB2 Web Control Center.
Spclient.java	A JDBC client application that calls PARAMETER STYLE JAVA stored procedures in the Spserver stored procedure class.
Spcreate.db2	A CLP script that contains the CREATE PROCEDURE statements to register the methods contained in the Spserver class as stored procedures.
Spdrop.db2	A CLP script that contains the DROP PROCEDURE statements necessary for deregistering the stored procedures contained in the Spserver class.
Spserver.java	A JDBC program demonstrating PARAMETER STYLE JAVA stored procedures. The client program is Spclient.java.

Table 8. Java Database Connectivity (JDBC) Sample Programs (continued)

Sample Program Name	Program Description
UDFcli.java	A JDBC client application that calls functions in the Java user-defined function library, UDFsrv.
UseThrds.java	Shows how to use threads to run an SQL statement asynchronously (JDBC version of CLI sample async.c).
V5SpCli.java	A Java client application that calls the DB2GENERAL stored procedure, V5Stp.java.
V5Stp.java	Demonstrates a DB2GENERAL stored procedure that updates the EMPLOYEE table on the server, and returns new salary and payroll information to the client. The client program is V5SpCli.java.
Varinp.java	Demonstrates variable input to Embedded Dynamic SQL statement calls using parameter markers.

Table 9. 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.
Stclient.sqlj	An SQLJ client application that calls PARAMETER STYLE JAVA stored procedures created by the SQLJ stored procedure program, Stserver.
Stcreate.db2	A CLP script that contains the CREATE PROCEDURE statements to register the methods contained in the Stserver class as stored procedures.
Stdrop.db2	A CLP script that contains the DROP PROCEDURE statements necessary for deregistering the stored procedures contained in the Stserver class.
Stserver.sqlj	An SQLJ program demonstrating PARAMETER STYLE JAVA stored procedures. The client program is Stclient.sqlj.
Static.sqlj	Uses static SQL to retrieve information.

Table 9. Embedded SQL for Java (SQLJ) Sample Programs (continued)

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

SQL Procedure Samples

Table 10. SQL Procedure Sample Programs

Sample Program Name	Program Description
`basecase.db2`	The UPDATE_SALARY procedure raises the salary of an employee identified by the ″empno″ IN parameter in the ″staff″ table of the ″sample″ database. The procedure determines the raise according to a CASE statement that uses the ″rating″ IN parameter.
`basecase.sqc`	Calls the UPDATE_SALARY procedure.
`baseif.db2`	The UPDATE_SALARY_IF procedure raises the salary of an employee identified by the ″empno″ IN parameter in the ″staff″ table of the ″sample″ database. The procedure determines the raise according to an IF statement that uses the ″rating″ IN parameter.
`baseif.sqc`	Calls the UPDATE_SALARY_IF procedure.
`dynamic.db2`	The CREATE_DEPT_TABLE procedure uses dynamic DDL to create a new table. The name of the table is based on the value of the IN parameter to the procedure.
`dynamic.sqc`	Calls the CREATE_DEPT_TABLE procedure.
`iterate.db2`	The ITERATOR procedure uses a FETCH loop to retrieve data from the ″department″ table. If the value of the ″deptno″ column is not 'D11', modified data is inserted into the ″department″ table. If the value of the ″deptno″ column is 'D11', an ITERATE statement passes the flow of control back to the beginning of the LOOP statement.
`iterate.sqc`	Calls the ITERATOR procedure.
`leave.db2`	The LEAVE_LOOP procedure counts the number of FETCH operations performed in a LOOP statement before the ″not_found″ condition handler invokes a LEAVE statement. The LEAVE statement causes the flow of control to exit the loop and complete the stored procedure.
`leave.sqc`	Calls the LEAVE_LOOP procedure.
`loop.db2`	The LOOP_UNTIL_SPACE procedure counts the number of FETCH operations performed in a LOOP statement until the cursor retrieves a row with a space (' ') value for column ″midinit″. The loop statement causes the flow of control to exit the loop and complete the stored procedure.

Table 10. SQL Procedure Sample Programs (continued)

Sample Program Name	Program Description
`loop.sqc`	Calls the LOOP_UNTIL_SPACE procedure.
`nestcase.db2`	The BUMP_SALARY procedure uses nested CASE statements to raise the salaries of employees in a department identified by the dept IN parameter from the ″staff″ table of the ″sample″ database.
`nestcase.sqc`	Calls the BUMP_SALARY procedure.
`nestif.db2`	The BUMP_SALARY_IF procedure uses nested IF statements to raise the salaries of employees in a department identified by the dept IN parameter from the ″staff″ table of the ″sample″ database.
`nestif.sqc`	Calls the BUMP_SALARY_IF procedure.
`repeat.db2`	The REPEAT_STMT procedure counts the number of FETCH operations performed in a repeat statement until the cursor can retrieve no more rows. The condition handler causes the flow of control to exit the repeat loop and complete the stored procedure.
`repeat.sqc`	Calls the REPEAT_STMT procedure.
`rsultset.c`	Calls the MEDIAN_RESULT_SET procedure, displays the median salary, then displays the result set generated by the SQL procedure. This client is written using the CLI API, which can accept result sets.
`rsultset.db2`	The MEDIAN_RESULT_SET procedure obtains the median salary of employees in a department identified by the ″dept″ IN parameter from the ″staff″ table of the ″sample″ database. The median value is assigned to the salary OUT parameter and returned to the ″rsultset″ client. The procedure then opens a WITH RETURN cursor to return a result set of the employees with a salary greater than the median. The procedure returns the result set to the client.
`spserver.db2`	The SQL procedures in this CLP script demonstrate basic error-handling, nested stored procedure calls, and returning result sets to the client application or the calling application. You can call the procedures using the ″spcall″ application, in the CLI samples directory. You can also use the ″spclient″ application, in the C and CPP samples directories, to call the procedures that do not return result sets.
`whiles.db2`	The DEPT_MEDIAN procedure obtains the median salary of employees in a department identified by the ″dept″ IN parameter from the ″staff″ table of the ″sample″ database. The median value is assigned to the salary OUT parameter and returned to the ″whiles″ client. The whiles client then prints the median salary.
`whiles.sqc`	Calls the DEPT_MEDIAN procedure.

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.
`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.

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

Sample Program Name	Program Description
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	Demonstrates rollup queries on a Microsoft Excel sales spreadsheet (implemented in Visual Basic).
names	Queries a Lotus Notes address book (implemented in Visual Basic).
inbox	Queries Microsoft Exchange inbox e-mail messages through OLE/Messaging (implemented in Visual Basic).
invoice	An OLE automation user-defined function that sends Microsoft Word invoice documents as e-mail attachments (implemented in Visual Basic).
bcounter	An OLE automation user-defined function demonstrating a scratchpad using instance variables (implemented in Visual Basic).
ccounter	A counter OLE automation user-defined function (implemented in Visual C++).
salarysrv	An OLE automation stored procedure that calculates the median salary of the STAFF table of the sample database (implemented in Visual Basic).
salarycltvc	A Visual C++ embedded SQL sample that calls the Visual Basic stored procedure, salarysrv.
salarycltvb	A Visual Basic DB2 CLI sample that calls the Visual Basic stored procedure, salarysrv.
testcli	An OLE automation embedded SQL client application that calls the stored procedure, tstsrv (implemented in Visual C++).
tstsrv	An OLE automation stored procedure demonstrating the passing of various types between client and stored procedure (implemented 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

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

Sample Program Name	Program Description
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
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.
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 AD Client, 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 361 if you need information about database instances.
- The DB2 AD Client 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.

Updating the Database Manager Configuration File

This file contains important settings for application development. You can change these settings by entering:

```
db2 update dbm cfg using <keyword> <value>
```

and you can view the settings by entering:

```
db2 get dbm cfg
```

See the *Command Reference* for more information on using these commands.

- For stored procedures, the keyword KEEPDARI has the default value yes. This keeps the stored procedure process alive. If you are developing a stored procedure, you may want to test loading the same stored procedure library a number of times. This default setting may interfere with reloading the library. Its best to change the value of this keyword to no while developing stored procedures, and then change it back to yes when you are ready to load the final version of your stored procedure.
- For Java, update the JDK11_PATH keyword. See "Setting the Environment" on page 64 for details.

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 36
- "Setting the Windows 32-bit Operating Systems Environment" on page 37

After you set up your operating system environment, you may want to create the sample database, which is used by the sample programs in this book. To create the database, see "Creating, Cataloging, and Binding the Sample Database" on page 40.

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 entered, 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 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 AD Client, 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 AD Client, 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 67.

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, no DB2 environment variables should be 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 set the DB2INSTDEF registry variable. It defines the default instance name which is used 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, PTX, and Solaris:

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

Silicon Graphics IRIX:

- PATH, to include several DB2 directories including `sqllib/bin`
- LD_LIBRARY_PATH, to include the directory `sqllib/lib` (needed for o32 object type applications)
- LD_LIBRARYN32_PATH, to include the directory `sqllib/lib32` (needed for n32 object type applications)

For the Java environment variables updated by DB2, see "Setting the Environment" on page 64.

Setting the Windows 32-bit Operating Systems Environment

When you install the DB2 AD Client on Windows NT or Windows 2000, the install program updates the 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 72.

When you install the DB2 AD Client 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 2000 control panel
- The Windows 95 or Windows 98 command window
- The Windows 95 or Windows 98 autoexec.bat file

Notes:

1. Exercise caution when changing these environment variables. **Do not** change the DB2PATH environment variable.
2. When using the variable %DB2PATH% in a command, put the full path in quotes, as in set LIB="%DB2PATH%\lib";%LIB%. In DB2 Version 7.1 the default installation value for this variable is \Program Files\sqllib, which contains a space, so not using quotes may cause an error.

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.

  For example, the Microsoft Visual C++ compiler environment setup file, Vc\bin\vcvars32.bat, has the following command:

  ```
  set INCLUDE=%MSVCDir%\INCLUDE;%MSVCDir%\...\ATL\INCLUDE;%INCLUDE%
  ```

  To use this file with DB2, first move %INCLUDE%, which sets the %DB2PATH%\INCLUDE path, from the end of the list to the beginning, as follows:

  ```
  set INCLUDE=%INCLUDE%;%MSVCDir%\INCLUDE;%MSVCDir%\...\ATL\INCLUDE;
  ```

- 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. See the *Administration Guide* for information on registry variables. See the *Command Reference* for information on 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 or a Windows 2000 environment is implemented as a Windows NT service or Windows 2000 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 Windows NT or Windows 2000 event logs, 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, if TCP/IP is being used, enter:

   ```
   db2set DB2COMM=tcpip
   ```

   and 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.

2. Start the database instance by entering:

```
db2start
```

The binding of the utilities to the sample database must be done from the client. For more information, see "Binding" on page 42.

Windows NT and Windows 2000

In a production system for DB2 for Windows NT or DB2 for Windows 2000, 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 or Windows 2000 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
  ```

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:

Windows NT

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.

Windows 2000

1. Click on the "Start" button.
2. Click on "Settings".
3. Click on "Control Panel".
4. Click on "Administrative Tools".

5. Click on "Services".
6. In the Services window, highlight "DB2 Security Server".
7. If it does not have the settings "Started" and "Automatic" listed, click on "Action" from the top menu.
8. Click on "Properties".
9. Make sure you are in the "General" tab.
10. Choose "Automatic" from the 'Startup Type' drop-down menu.
11. Click on "OK".
12. 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 and Windows, `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 and Windows, 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 and Windows, 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 *Data Movement Utilities Guide and 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 *Command Reference* 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 user userid using password
   ```

   where *userid* and *password* are the user ID and password of the instance where the sample database is located.

   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
   ```

   where *userid* and *password* are the user ID and password of the instance where the sample database is located.
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 IBM DB2.
3. From the IBM DB2 menu, select the DB2 Command Window.

   The command window displays.
4. Connect to the sample database by entering:

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

where *userid* and *password* are the user ID and password of the instance where the `sample` database is located.

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 AS/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 47, then the specific chapter that follows for the applications you are building.

For programming with Java, see "Chapter 4. Building Java Applets and Applications" on page 63.

For programming with SQL Procedures, see "Chapter 5. Building SQL Procedures" on page 89.

For programming with DB2 APIs, DB2 CLI, and Embedded SQL, see the 'Building Applications' chapter for your platform.

For further information, refer to the following books:

- The *Application Development Guide* for applications using embedded SQL, JDBC, and SQLJ, and for user-defined functions (UDFs).
- The *CLI Guide and Reference* for applications using DB2 CLI or ODBC.
- The *Administrative API Reference* for DB2 API applications.

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 apply to most DB2-supported platforms.

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

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

General Points for Building and Running DB2 Programs

1. Application Environment:
   - 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.
   - 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`.
   - 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. When you are building applications for production, the DB2 runtime paths that are built into the executables should be the install paths, and not the paths of the local DB2 instance where you are developing your applications. This book is designed to show you how to build applications in a development environment, and therefore documents the instance

copies of sqllib/include and sqllib/lib on UNIX, and %DB2PATH%\include and %DB2PATH%\lib on OS/2 and Windows 32-bit operating systems.

4. 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, generically, as build files.

Build files are provided by DB2 for each language on supported platforms where the types of programs they build are available, in the same directory as the sample programs for each language. Table 16 lists the build files for all languages on all supported platforms except for C++ on Windows 32-bit operating systems. The filename extensions have been left off. For OS/2, the extension is .cmd; for Windows 32-bit operating systems, the extension is .bat. There is no extension for UNIX platforms.

For Windows 32-bit operating systems, there are two supported C++ compilers: Microsoft Visual C++ and IBM VisualAge C++. For these compilers, respectively, an "m" or a "v" has been inserted after the "bld" for each build file name, except for bldclisp, which becomes either bldmclis or bldvclis. These build files are listed in Table 17 on page 49. The .bat extension has been left off.

Table 16. DB2 Build Files

Build File	Types of Programs Built
bldsqlj	Java Embedded SQLJ applications.

Table 16. DB2 Build Files (continued)

Build File	Types of Programs Built
bldsqljs	Java Embedded SQLJ stored procedures.
bldcli	DB2 CLI applications, with or without embedded SQL.
bldapi	DB2 CLI applications, with or without embedded SQL, that require linking in the `utilapi` utility file that contains DB2 APIs for creating and dropping a database.
bldclisp	DB2 CLI stored procedures (non-embedded SQL).
bldapp	Application programs with or without embedded SQL.
bldsrv	Embedded SQL stored procedures.
bldudf	User-defined functions (UDFs).
bldmt	Multi-threaded embedded SQL applications (only available for C/C++ on supported UNIX platforms).
bldevm	The event monitor sample program, `evm` (only available on AIX and OS/2).

Table 17. C/C++ Build Files for Windows 32-bit operating systems

Build File	Types of Programs Built
bldmcli	Microsoft Visual C++ DB2 CLI applications, with or without embedded SQL.
bldvcli	VisualAge C++ DB2 CLI applications, with or without embedded SQL.
bldmapi	Microsoft Visual C++ DB2 CLI applications, with or without embedded SQL, that require linking in the `utilapi` utility file that contains DB2 APIs for creating and dropping a database.
bldvapi	VisualAge C++ DB2 CLI applications, with or without embedded SQL, that require linking in the `utilapi` utility file that contains DB2 APIs for creating and dropping a database.
bldmclis	Microsoft Visual C++ DB2 CLI stored procedures (non-embedded SQL).
bldvclis	VisualAge C++ DB2 CLI stored procedures (non-embedded SQL).
bldmapp	Microsoft Visual C++ application programs with or without embedded SQL.
bldvapp	VisualAge C++ application programs with or without embedded SQL.
bldmsrv	Microsoft Visual C++ Embedded SQL stored procedures.
bldvsrv	VisualAge C++ Embedded SQL stored procedures.
bldmudf	Microsoft Visual C++ User-defined functions (UDFs).

Table 17. C/C++ Build Files for Windows 32-bit operating systems (continued)

Build File	Types of Programs Built
bldvudf	VisualAge C++ User-defined functions (UDFs).

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 is built, even if the source file is not modified. This is unlike a `makefile`. It means a developer can rebuild an existing program without having to delete previous object and executable files, or modifying the source.

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 used for embedded SQL programs call another file, `embprep`, that contains the precompile and bind steps for embedded SQL programs. These steps may require the optional parameters user ID and password, depending on where the embedded SQL program is being built.

If a developer is building the 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. 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 provided. The `embprep` file is also used by the makefile. See "Makefiles", below, for more on this 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 builds most of 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 program, `spserver`, into a shared library that can be called by the client application, `spclient`.

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, `spserver`, and copies the shared library to the function subdirectory:

```
spserver : utilcli.o
        $(CC) -o spserver spserver.c utilcli.o $(CFLAGS) $(LIBS) \
        -H512 -T512 -bE:spserver.exp -e outlanguage
        $(ERASE) $(DB2PATH)/function/spserver
        $(COPY) spserver $(DB2PATH)/function/spserver
```

For embedded SQL programs, the makefile calls the `embprep` file, which 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

uses the parameters for user ID, password, and database name to connect to the database on the server. The `makefile` passes these values to `embprep` 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 `embprep` precompile and bind file. The following is an example of the `embprep` file being called by the Micro Focus COBOL makefile on Windows NT to build the embedded SQL application, `updat`:

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

Error-checking Utilities

The DB2 AD Client provides several utility files. These files have functions for error-checking and printing out error information. The exception to this is the CLI utility file, `utilapi.c`, which calls the DB2 APIs to create and drop a database. Utility files are provided for each language in the samples directory. When used with an application program, the error-checking utility files provide helpful error information, and make debugging a DB2 program much easier. Most of the error-checking utilities use the DB2 APIs `GET SQLSTATE MESSAGE` and `GETERROR MESSAGE` to obtain pertinent SQLSTATE and SQLCA information related to problems encountered in program execution. The DB2 CLI utility file, `utilcli.c`, 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. They are also not necessary for Java 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 programs

utilcli.c
For CLI programs

utilapi.c
For C non-embedded SQL programs

utilemb.sqc
For C embedded SQL programs

utilapi.C
For C++ non-embedded SQL programs

utilemb.sqC
For C++ embedded SQL programs

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 utilities.

The following example demonstrates how the error-checking utilities are used in DB2 programs. The `utilemb.h` header file defines the `EMB_SQL_CHECK` macro which is substituted for the functions `SqlInfoPrint()` and `TransRollback()`:

```
#define EMB_SQL_CHECK( MSG_STR )           \
   if( SqlInfoPrint( MSG_STR, &sqlca, __LINE__, __FILE__)  != 0 ) TransRollback( );
```

`SqlInfoPrint()` checks the SQLCODE flag. It prints out any available information related to the specific error indicated by this flag. It also points to where the error occurred in the source code. `TransRollback()` allows the utility file to safely rollback a transaction where an error has occurred. It requires embedded SQL statements to connect to the database and execute a rollback. The following is an example of how the C++ program `cursor` calls the utility functions by using the macro, supplying the value `"DECLARE CURSOR"` for the `MSG_STR` parameter of the `SqlInfoPrint()` function:

```
Cursor::Fetch () {
   EXEC SQL DECLARE c1 CURSOR FOR
            SELECT name, dept FROM staff WHERE job='Mgr'
            FOR UPDATE OF job;
   EMB_SQL_CHECK("DECLARE CURSOR") ;
```

The `EMB_SQL_CHECK` macro ensures that if the `DECLARE` statement fails, the transaction will be safely rolled back, and an appropriate error message printed out.

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 63. 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

The DB2 AD Client 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 the DB2 application build files supplied with the DB2 AD Client 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 12.

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.

Note: When writing your API applications, all API input structures should be memset to 0 so that structure elements that are not explicitly set are initialized to 0. This is important when recompiling applications that have been coded against downlevel versions of APIs. When recompiling, the new definition of the structure is used but all the elements may not be initialized. The call to memset will ensure that they are initialized. Here is an example showing the input data structure, `pLoadInStruct`, memset to 0:

```
memset( pLoadInStruct, 0, sizeof(pLoadInStruct) );
```

DB2 Call Level Interface (CLI) Applications

The DB2 AD Client 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.

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 provided an ODBC driver manager is available on the application platform.

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):

tbinfo Demonstrates how to get and set information at a table level.

dbusemx
Demonstrates how to use a single database in conjunction with embedded SQL.

dbmconn
Demonstrates how to connect to, and disconnect from, multiple databases.

spclient
is the client program of a client/server example; the server program is `spserver`.

spserver
is the server program of a client/server example; the client program is `spclient`.

udfcli uses the functions created by the user-defined function program, `udfsrv`.

For a more detailed description of all the DB2 CLI sample programs, see Table 7 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 63.

The DB2 AD Client 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 AD Client 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 embedded SQL applications using the supported programming languages. The steps you follow may vary, depending on your environment:

updat uses static SQL to update a database.

The following samples are used to demonstrate embedded SQL client applications for stored procedures and user-defined functions (UDFs) using C and C++:

spclient
is the client program that demonstrates calling stored procedures; the server program is `spserver`.

spserver
is the server program demonstrating stored procedures; the client program is `spclient`.

udfcli uses the ScalarUDF function in the user-defined function library, `udfsrv`.

The following samples are used to demonstrate stored procedures and user-defined functions (UDFs) using COBOL:

outcli is the client program that demonstrates calling stored procedures; the server program is `outsrv`.

outsrv is the server program demonstrating stored procedures; the client program is `outcli`.

calludf
calls the functions in the user-defined function library, `udf`.

For a more detailed description of these sample programs, see "Sample Program Tables" on page 12.

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 12 lists all of the sample programs. The *Application Development Guide* explains how the samples containing embedded SQL work.

Stored Procedures

Stored procedures are built and stored on the server. They can be remotely accessed by client applications. The stored procedure functions then perform processing locally on the server database, and send back the results to the client. This reduces network traffic and improves overall performance.

A stored procedure can be run as fenced or unfenced. 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 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.

The stored procedures demonstrated in this book are stored on 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 unfenced 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*.

The following sample program is used to demonstrate the steps for building and storing a stored procedure library on the server using SQL Procedures:

`spserver.db2`
: is a CLP script containing SQL Procedures used to create a shared library on the server. It can be called by the CLP `call` command, or by the `spclient` application in the C, C++, and CLI directories.

The following sample program is used to demonstrate the steps for building and storing a stored procedure library on the server using C and C++:

`spserver`
: is the server program of a client/server example; the client program is `spclient`.

The following sample program is used to demonstrate the steps for building and storing a stored procedure library on the server using Java:

`Spserver`
: is the server program of a client/server example; the client program is `Spclient`.

The following sample program is used to demonstrate the steps for building and storing a stored procedure library on the server using COBOL:

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

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:

udfsrv creates a library of user-defined functions (UDFs). It is available for C and C++ only. The client application, `udfcli`, calls these functions.

udf creates a library of user-defined functions (UDFs). It is available for COBOL only. The client application, `calludf`, calls these functions.

UDFsrv creates a library of user-defined functions (UDFs). It is available 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 on supported UNIX platforms. These applications allow the user to have several simultaneous processes operating several simultaneous threads of execution. This allows the handling of asynchronous events, and the creating of event-driven applications without resorting to polling schemes. The following sample program is used to demonstrate building a DB2 multi-threaded application:

thdsrver
demonstrates thread creation and management.

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.

On OS/2 and Windows 32-bit operating systems, the type-decorated function name can be determined from the `.obj` (object) file.

With the VisualAge C++ compiler on OS/2 and Windows, you can use the `cppfilt` command to determine the type-decorated function name from the `.obj` (object) file, as follows:

```
cppfilt -b /p myprog.obj
```

where `myprog.obj` is your program object file.

With the Microsoft Visual C++ compiler on Windows, you can use the `dumpbin` command to determine the type-decorated function name from the `.obj` (object) file, as follows:

```
dumpbin /symbols myprog.obj
```

where `myprog.obj` is your program object file.

On UNIX platforms, the type-decorated function name can be determined from the `.o` (object) file, or from 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 all of these commands includes a line with the mangled function name. On UNIX, for example, this line is similar to the following:

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

Once you have obtained the mangled function name from one of the above commands, you can use it in the appropriate command. We demonstrate this below using the mangled function name obtained from the UNIX example above. A mangled function name obtained on OS/2 or Windows would be used the same way.

When registering a UDF with CREATE FUNCTION, the EXTERNAL NAME clause must specify the mangled function name:

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

Likewise, when calling a stored procedure, the `CALL` function must also specify 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 AD Client. 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 AD Client 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.

To run DB2 Java applications, you must install and invoke a Java Virtual Machine (JVM) that provides native threads support. To execute a Java application using native threads, you can use the `-native` option in your command. For example, to run the Java sample application, `App.class`, you can use the following command:

```
java -native App
```

You may be able to specify native threads as the default thread support for some Java Virtual Machines. The information in this chapter assumes native threads support is the default. Please refer to your JVM documentation for instructions on making native threads the default on your system.

To run DB2 Java applets, you may invoke a Java Virtual Machine that provides either native threads or green threads support.

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.ibm.com/software/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. One of the following:
   - The Java Development Kit (JDK) Version 1.1.8 and Java Runtime Environment (JRE) Version 1.1.8 for AIX from IBM. These are installed with DB2. (Refer to `http://www.ibm.com/software/data/db2/java`.)
   - The Java Development Kit (JDK) Version 1.2.2 and Java Runtime Environment (JRE) Version 1.2.2 for AIX from IBM. (Refer to `http://www.ibm.com/software/data/db2/java`.)
2. DB2 Java Enablement, provided on DB2 Universal Database Version 7.1 for AIX clients and servers.

The JDBC 1.22 driver is still the default driver on all operating systems, including AIX. To take advantage of the new features of JDBC 2.0, you must install both the JDBC 2.0 driver and JDK 1.2 support for your platform. To install the JDBC 2.0 driver for AIX, enter the `usejdbc2` command from the `sqllib/java12` directory. This command performs the following tasks:

- Creates an `sqllib/java11` directory for the 1.22 driver files
- Backs up the JDBC 1.22 driver files into the `sqllib/java11` directory

- Copies the JDBC 2.0 driver files from the sqllib/java12 directory into the appropriate directories

To switch back to the JDBC 1.22 driver, execute the usejdbc1 command from the sqllib/java12 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 /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 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 1.1.8 or later from Hewlett-Packard (refer to http://www.ibm.com/software/data/db2/java).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 7.1 for HP-UX clients and servers.

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 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. IBM Developer kit and Runtime Environment for Linux, Version 1.1.8. (see `http://www.ibm.com/software/data/db2/java`).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 7.1 for Linux clients and servers.

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 /usr/local/jdk118
```

where /usr/local/jdk118 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/jdk118/lib/linux/native_threads/libjava.so /usr/lib
```

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

```
http://www.ibm.com/software/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 1.1.8 and Java Runtime Environment (JRE) Version 1.1.8 for OS/2 from IBM. These are shipped with DB2. (Refer to `http://www.ibm.com/software/data/db2/java`.)
2. DB2 Java Enablement, provided on DB2 Universal Database Version 7.1 for OS/2 clients and servers.

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
```

PTX

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

1. ptx/JSE Version 3.0, which is equivalent to Sun Microsystem's JDK 1.2 (refer to http://www.ibm.com/software/data/db2/java).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 7.1 for NUMA-Q.

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 ptx/JSE 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 /opt/jse3.0
```

where /opt/jse3.0 is the path where the ptx/JSE 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 PTX with DB2 JDBC support, commands to update the PTX 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
```

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. Java2 Software Development Kit Version 1.2.1 (JDK 1.2.1) from Silicon Graphics, Inc. (refer to http://www.ibm.com/software/data/db2/java).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 7.1 for Silicon Graphics IRIX clients.

Note: SQLJ applications on Silicon Graphics IRIX can only be built with the o32 object type. To change the Java default object type to o32, set the environment variable SGI_ABI with the Korn shell using this command:

```
export SGI_ABI=-o32
```

and with the C shell using this command:

```
setenv SGI_ABI -o32
```

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 42 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 in the Solaris operating environment with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The Java Development Kit (JDK) Version 1.1.8 or 1.2 for Solaris from Sun Microsystems (refer to `http://www.ibm.com/software/data/db2/java`).
2. DB2 Java Enablement, provided on DB2 Universal Database Version 7.1 for Solaris clients and servers.

The JDBC 1.22 driver is still the default driver on all operating systems, including Solaris. To take advantage of the new features of JDBC 2.0, you must install both the JDBC 2.0 driver and JDK 1.2 support for your platform. To install the JDBC 2.0 driver for Solaris, enter the `usejdbc2` command from the sqllib/java12 directory. This command performs the following tasks:

- Creates an `sqllib/java11` directory for the 1.22 driver files
- Backs up the JDBC 1.22 driver files into the `sqllib/java11` directory
- Copies the JDBC 2.0 driver files from the `sqllib/java12` directory into the appropriate directories

To switch back to the JDBC 1.22 driver, execute the `usejdbc1` command from the `sqllib/java12` 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 /usr/java
```

where `/usr/java` 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: In the Solaris operating environment, 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 /usr/java/lib/*.so /usr/lib
```

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

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

To run JDBC and SQLJ programs in the Solaris operating environment 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) 1.1.8 and Java Runtime Environment (JRE) 1.1.8 for Win32 from IBM. These are installed with DB2. (Refer to http://www.ibm.com/software/data/db2/java.)
   - The Java Development Kit (JDK) Version 1.2 for Win32 from Sun Microsystems (refer to http://www.ibm.com/software/data/db2/java).
   - Microsoft Software Developer's Kit for Java, Version 3.1 (refer to http://www.ibm.com/software/data/db2/java).

2. DB2 Java Enablement, provided on DB2 Universal Database Version 7.1 for Windows 32-bit operating systems clients and servers.

The JDBC 1.22 driver is still the default driver on all operating systems, including Windows 32-bit. To take advantage of the new features of JDBC 2.0, you must install both the JDBC 2.0 driver and JDK 1.2 support for your platform. To install the JDBC 2.0 driver for Windows 32-bit operating systems, enter the `usejdbc2` command from the `sqllib\java12` directory. This command performs the following tasks:

- Creates an `sqllib\java11` directory for the 1.22 driver files
- Backs up the JDBC 1.22 driver files into the `sqllib\java11` directory
- Copies the JDBC 2.0 driver files from the `sqllib\java12` directory into the appropriate directories

To switch back to the JDBC 1.22 driver, execute the `usejdbc1` batch file from the `sqllib\java12` 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.

Note: If the path where the JDK is installed contains a directory name with one or more spaces, put the path in single quotes. For example:

```
db2 update dbm cfg using JDK11_PATH 'c:\Program Files\jdk11'
```

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. 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 77.

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 (or "net") 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 (or "app") 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

`Spclient` is a client application that calls the Java stored procedure class, `Spserver`, using the JDBC application driver. Before building and running this client application, build the stored procedure class on the server. See "Stored Procedures" on page 77.

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

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

   ```
   javac Spclient.java
   ```

2. Run the Java interpreter on the client program with this command:

   ```
   java Spclient
   ```

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 86.

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

`Spserver` demonstrates dynamic SQL PARAMETER STYLE JAVA stored procedures 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 program on the server by commands entered at the command line:

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

   ```
   javac Spserver.java
   ```

2. Copy `Spserver.class` to the `%DB2PATH%\function` directory on OS/2 and Windows 32-bit operating systems, or to the `sqllib/function` directory on UNIX.
3. Next, catalog the stored procedures by running the `Spcreate.db2` script on the server. First, connect to the database:

   ```
   db2 connect to sample
   ```

   If the stored procedures were previously cataloged, you can drop them with this command:

   ```
   db2 -td@ -vf Spdrop.db2
   ```

   Then catalog them with this command:

   ```
   db2 -td@ -vf Spcreate.db2
   ```

4. Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.
5. Compile and run the `Spclient` client application to access the stored procedure class. See "Client Applications for Stored Procedures" on page 76.

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

SQLJ Programs

Note: To build and run SQLJ programs with the IBM Java Development Kit for UNIX, OS/2, and Windows 32-bit operating systems, you must turn off the just-in-time compiler of the JDK with the following command for your operating system:

OS/2 and Windows:

```
SET JAVA_COMPILER=NONE
```

UNIX: `export JAVA_COMPILER=NONE`

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 bldsqlj -- OS/2 and Windows 32-bit operating systems
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
rem and bind the package to the database.
if "%2" == "" goto case1
if "%3" == "" goto case2
if "%4" == "" goto error
goto case3
:case1
   sqlj %1.sqlj
   db2profc -url=jdbc:db2:sample -prepoptions="package using %1" %1_SJProfile0
   goto continue
:case2
   sqlj     -url=jdbc:db2:%2 %1.sqlj
   db2profc -url=jdbc:db2:%2 -prepoptions="package using %1" %1_SJProfile0
   goto continue
:case3
   sqlj     -url=jdbc:db2:%2 -user=%3 -password=%4  %1.sqlj
   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 Precompile 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 serialized 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 -- UNIX platforms
# Builds a Java embedded SQL (SQLJ) sample
# Usage: bldsqlj <prog_name> [ <db_name> [ <userid> <password> ]]

# Translate and compile the SQLJ source file
# and bind the package to the database.
if (($# < 2))
then
   sqlj $1.sqlj
   db2profc -url=jdbc:db2:sample -prepoptions="package using $1" $1_SJProfile0
elif (($# < 3))
```

```
then
   sqlj     -url=jdbc:db2:$2 $1.sqlj
   db2profc -url=jdbc:db2:$2 -prepoptions="package using $1" $1_SJProfile0
else
   sqlj     -url=jdbc:db2:$2 -user=$3 -password=$4  $1.sqlj
   db2profc -url=jdbc:db2:$2 -user=$3 -password=$4 -prepoptions="package using $1"
     $1_SJProfile0
fi
```

Translator and Precompile 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 serialized 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

`Stclient` is the client program that calls the SQLJ stored procedure class, `Stserver`, using the JDBC Application driver. Before building and running this client application, build the stored procedure class on the server. See "Stored Procedures" on page 82.

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

```
bldsqlj Stclient [ <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 Stclient
```

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 86.

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. Since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, the source file name, is required. If no database name is supplied, the program uses the default sample database.

```
@echo off
rem bldsqljs -- OS/2 and Windows 32-bit operating systems
rem Builds a Java embedded SQL (SQLJ) stored procedure
rem Usage: bldsqljs prog_name [ db_name ]

if "%1" == "" goto error

rem Translate and compile the SQLJ source file
rem and bind the package to the database.
if "%2" == "" goto case1
goto case2
:case1
   sqlj %1.sqlj
   db2profc -url=jdbc:db2:sample -prepoptions="package using %1" %1_SJProfile0
   goto continue
:case2
   sqlj     -url=jdbc:db2:%2 %1.sqlj
   db2profc -url=jdbc:db2:%2 -prepoptions="package using %1" %1_SJProfile0
: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 ]
:exit
@echo on
```

Translator and Precompile 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`.
-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 serialized 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. Since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, the source file name, is required. If no database name is supplied, the program uses the default `sample` database.

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

# 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
# and bind the package to the database.
if (($# < 2))
then
   sqlj $1.sqlj
   db2profc -url=jdbc:db2:sample -prepoptions="package using $1" $1_SJProfile0
else
   sqlj     -url=jdbc:db2:$2 $1.sqlj
   db2profc -url=jdbc:db2:$2 -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 Precompile 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`.
-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 serialized profile for the program.

`Stserver` demonstrates PARAMETER STYLE JAVA stored procedures 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 class with the build file, `bldsqljs`:

1. Enter the following command:

   ```
   bldsqljs Stserver [ <db_name> ]
   ```

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

2. Next, catalog the stored procedures by running the `Stcreate.db2` script on the server. First, connect to the database:

   ```
   db2 connect to sample
   ```

   If the stored procedures were previously cataloged, you can drop them with this command:

   ```
   db2 -td@ -vf Stdrop.db2
   ```

Then catalog them with this command:

```
db2 -td@ -vf Stcreate.db2
```

3. Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.
4. Compile and run the `Stclient` client application to call the stored procedures. See "Client Programs for Stored Procedures" on page 82.

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 76. To compile and run the SQLJ client application, `UDFclie`, see "Client Programs for User-Defined Functions" on page 82.

`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.ibm.com/software/data/db2/db2lotus/gojava.htm
   ```

Chapter 5. Building SQL Procedures

This chapter provides detailed information for building DB2 SQL Procedures.

DB2 SQL Procedure sample programs are in the `sqllib/samples/sqlproc` directory on UNIX platforms, and in the `%DB2PATH%\samples\sqlproc` directory on OS/2 and Windows 32-bit operating systems. For a description of the sample programs, see Table 10 on page 26.

Setting the SQL Procedures Environment

These instructions are in addition to the instructions for setting up the DB2 environment in "Chapter 2. Setup" on page 33.

For SQL procedures support you have to install the Application Development Client and a DB2 supported C or C++ compiler on the server. For information on installing the Application Development Client, refer to the *Quick Beginnings* book for your platform. For the C and C++ compilers supported by DB2 on your platform, see "Supported Software by Platform" on page 7.

Note: On the OS/2 FAT file system, you are limited to a schema name for SQL Procedures of eight characters or less. You have to use the HPFS file system for schema names longer than eight characters.

Configuring the Compiler Environment

To create SQL procedures, configure DB2 to use a supported C or C++ compiler on the server by the following steps:

- Create an executable file that sets the environment for the compiler. This will be a command file on OS/2, a script file on UNIX, or a batch file on Windows. The compiler may require path, include, and library environment variables.
- Set the DB2_SQLROUTINE_COMPILER_PATH DB2 registry variable to the executable file with the following command:

  ```
  db2set DB2_SQLROUTINE_COMPILER_PATH=executable_file
  ```

where *executable_file* is the full path name for the C compiler environment file.

If you do not set the DB2_SQLROUTINE_COMPILER_PATH DB2 registry variable, DB2 sets it to a default file. Depending on your operating system, this file will have one of the following paths and file names:

OS/2: `%DB2PATH%\function\routine\sr_cpath.cmd`

UNIX: `$HOME/sqllib/function/routine/sr_cpath`

Windows:
`%DB2PATH%\function\routine\sr_cpath.bat`

You can use this default file as long as you modify it to reflect the settings required for the server operating system and the C or C++ compiler you are using.

Note: On Windows NT and Windows 2000, you do not have to set the DB2_SQLROUTINE_COMPILER_PATH DB2 registry variable if you store the environment variables for your compiler as SYSTEM variables.

Customizing Compiler Options

DB2 provides default values for one of the compilers it supports on each platform. To use other compilers, set the SQL procedure compiler options using the DB2_SQLROUTINE_COMPILE_COMMAND DB2 registry variable. To specify customized C or C++ compiler options for SQL procedures, store the entire command line, including all options, in the DB2 registry with the following command:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=compiler_command
```

where *compiler_command* is the C or C++ compile command, including the options and parameters required to create stored procedures.

In the compiler command, use the keyword `SQLROUTINE_FILENAME` to replace the filename for the generated SQC, C, PDB, DEF, EXP, messages log and shared library files. For AIX only, use the keyword `SQLROUTINE_ENTRY` to replace the entry point name.

As examples of the default values for the DB2_SQLROUTINE_COMPILE_COMMAND for the supported C or C++ compilers, here are the default compiler values on AIX, Solaris, and Windows 32-bit operating systems. Also given are suggested changes to return debugging information. Similar changes can be made to return debugging information on other platforms.

AIX This is the default compiler command value for IBM C Set++ for AIX Version 3.6.6:

```
xlC_r -+ -H512 -T512 -I$HOME/sqllib/include SQLROUTINE_FILENAME.c \
  -bE:SQLROUTINE_FILENAME.exp -e SQLROUTINE_ENTRY \
  -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -lc -ldb2
```

To return debug information, change the default to add the `-g` option to the DB2_SQLROUTINE_COMPILE_COMMAND as follows:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="xlC_r -+ -H512 -T512 -g \
  -I$HOME/sqllib/include SQLROUTINE_FILENAME.c \
  -bE:SQLROUTINE_FILENAME.exp -e SQLROUTINE_ENTRY \
  -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -lc -ldb2"
```

where ″\″ is only used to indicate a carriage return.

Note: To compile 64-bit SQL procedures on AIX, add the `-q64` option to the above commands.

Solaris

This is the default compiler command value for SPARCompiler C++ Versions 4.2 and 5.0:

```
cc -# -Kpic -I$HOME/sqllib/include SQLROUTINE_FILENAME.c -G \
  -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -R$HOME/sqllib/lib -ldb2
```

To return debug information, change the default to add the `-g` option to the DB2_SQLROUTINE_COMPILE_COMMAND as follows:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="cc -# -Kpic -g \
  -I$HOME/sqllib/include SQLROUTINE_FILENAME.c -G \
  -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib \
  -R$HOME/sqllib/lib -ldb2"
```

where ″\″ is only used to indicate a carriage return.

Note: To compile 64-bit SQL procedures on Solaris, add the `-xarch=v9` option to the above commands.

Windows 32-bit operating systems

This is the default compiler command value for Microsoft Visual C++ Versions 5.0 and 6.0:

```
cl -Od -W2 /TC -D_X86_=1 -I%DB2PATH%\include SQLROUTINE_FILENAME.c
  /link -dll -def:SQLROUTINE_FILENAME.def /out:SQLROUTINE_FILENAME.dll
  %DB2PATH%\lib\db2api.lib
```

To return debug information, change the default as follows:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="cl -Od -W2 /TC -D_X86_=1
  -Z7 -I%DB2PATH%\include SQLROUTINE_FILENAME.c /link -dll
  -def:SQLROUTINE_FILENAME.def /out:SQLROUTINE_FILENAME.dll
  -debug:full -pdb:none -debugtype:cv %DB2PATH%\lib\db2api.lib"
```

Note: You must enter the compiler command value on one line for Windows 32-bit operating systems.

To return to the default compiler options, set the DB2 registry value for DB2_SQLROUTINE_COMPILE_COMMAND to null with the following command:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=
```

Retaining Intermediate Files

When you issue a CREATE PROCEDURE statement, DB2 creates a number of intermediate files that are normally deleted if DB2 successfully completes the statement. If an SQL procedure does not perform as expected, you might find it useful to examine the SQC, C, PDB, and message log files created by DB2. To keep the files that DB2 creates during the successful execution of a CREATE PROCEDURE statement, you must set the value of the DB2_SQLROUTINE_KEEP_FILES DB2 registry variable to "1", "y" or "yes", as in the following command:

```
db2set DB2_SQLROUTINE_KEEP_FILES=1
```

Depending on your operating system, the intermediate files are retained in one of the following directories:

UNIX

```
$HOME/sqllib/function/routine/sqlproc/database_name/schema_name
```

where *database_name* and *schema_name* are the database and schema used to create the SQL procedures.

OS/2 and Windows

```
%DB2PATH%\function\routine\sqlproc\database_name\schema_name
```

where *database_name* and *schema_name* are the database and schema used to create the SQL procedures.

Customizing Precompile and Bind Options

The precompile and bind options can be customized by setting the DB2_SQLROUTINE_PREPOPTS DB2 registry variable. These options cannot be customized at procedure level. To specify customized precompilation options for SQL procedures, put the list of precompile options to be used by the DB2 precompiler in the DB2 registry with the following command:

```
db2set DB2_SQLROUTINE_PREPOPTS=options
```

where *options* specifies the list of precompile options to be used by the DB2 precompiler. Only the following options are allowed:

```
BLOCKING {UNAMBIG | ALL | NO}
DATETIME {DEF | USA | EUR | ISO | JIS | LOC}
DEGREE {1 | degree-of-parallelism | ANY}
DYNAMICRULES {BIND | RUN}
EXPLAIN {NO | YES | ALL}
EXPLAINSNAP {NO | YES | ALL}
INSERT {DEF | BUF}
ISOLATION {CS |RR |UR |RS |NC}
QUERYOPT optimization-level
SYNCPOINT {ONEPHASE | TWOPHASE | NONE}
```

Creating SQL Procedures

The DB2 Command Line Processor scripts in the `sqllib/samples/sqlproc` directory on UNIX, and the `%DB2PATH%\samples\sqlproc` directory on OS/2 and Windows, (those ending with the `.db2` extension) execute the CREATE PROCEDURE statement that creates the stored procedure on the server. Before executing this statement you must connect to the database. Each CLP script has a corresponding client application file of the same name, with either a `.sqc`, or a `.c` extension.

Before running a CREATE PROCEDURE CLP script, connect to the sample database with the command:

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

where *userid* and *password* are the user ID and password of the instance where the `sample` database is located.

To execute the CREATE PROCEDURE statement contained in the `rsultset.db2` script file, enter the following command:

```
db2 -td@ -vf rsultset.db2
```

Next, stop and restart DB2. Now, you can call the SQL procedures, as explained in "Calling SQL Procedures", below.

Calling SQL Procedures

You can call SQL procedures by using the command line processor (CLP) `call` command or by building client applications.

Using the CALL Command

To use the `call` command, you must enter the stored procedure name plus any IN or INOUT arguments that are required by that stored procedure. You do not enter OUT parameters.

First, create the SQL procedure by following the steps in "Creating SQL Procedures".

To call the SQL procedure, you must first connect to the database:

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

where *userid* and *password* are the user ID and password of the instance where the `sample` database is located.

The parameters for a stored procedure are given in the `CREATE PROCEDURE` statement for the stored procedure in the program source file. For example, in the source file, `whiles.db2`, the `CREATE PROCEDURE` statement for the `DEPT_MEDIAN` procedure begins:

```
CREATE PROCEDURE DEPT_MEDIAN
(IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
```

To call this procedure, you need to put in a valid SMALLINT value for the IN parameter, `deptNumber`. You can obtain a valid value from the corresponding table in the sample database, or by checking the client calling program source file for the value it uses. In `whiles.sqc`, you will find the value ″51″ is used:

```
printf("Use CALL with Host Variables to invoke the Server Procedure "
   "named %s\n", procname);
dept = 51;                    /* get median for dept. 51 */
```

Enter the `call` command with the procedure name and the value for the IN parameter. The procedure's parameters must be enclosed in parentheses, and quotes must be used, as follows:

```
db2 "call DEPT_MEDIAN (51)"
```

You should receive this result:

```
MEDIANSALARY: 1.76545000000000e+04
```

Keep the following points in mind when using the `call` command:

- There is a maximum of 1023 characters for a result column.
- The stored procedure being called must be defined in the catalog.
- LOBs and binary data (FOR BIT DATA, VARBINARY, LONGVARBINARY, GRAPHIC, VARGRAPHIC, LONGVARGRAPHIC) are not supported.

OS/2 DB2 CLI Client Applications

The command file `bldcli.cmd` in `%DB2PATH%\samples\sqlproc` contains the commands to build a DB2 CLI client application for SQL procedures. See "DB2 CLI Applications" on page 216 for detailed information on `bldcli.cmd`.

To build the DB2 CLI client application, `rsultset`, from the source file `rsultset.c`, enter:

```
bldcli rsultset
```

This command creates the executable file, `rsultset`.

To call the stored procedure, run the sample client application by entering the executable file name, the name of the database to which you are connecting, and the user ID and password of the database instance:

`rsultset` *database userid password*

OS/2 Embedded SQL Client Applications

The command file `bldapp.cmd` in `%DB2PATH%\samples\sqlproc` contains the commands to build an embedded SQL client application for SQL procedures. See "DB2 API and Embedded SQL Applications" on page 221 for detailed information on `bldapp.cmd`.

To build the embedded SQL client application, `basecase`, from the source file `basecase.sqc`, enter the command file name, the executable name, the database to which you are connecting, and the user ID and password of the database instance:

```
bldapp basecase database userid password
```

The result is an executable file, `basecase`.

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

`basecase` *database userid password*

UNIX DB2 CLI Client Applications

The script file `bldcli` in `sqllib/samples/sqlproc` contains the commands to build a DB2 CLI client application for SQL procedures. For detailed information on the `bldcli` script file, see the ″DB2 CLI Applications″ section in the ″Building Applications″ chapter for your UNIX platform.

To build the DB2 CLI client application, `rsultset`, from the source file `rsultset.c`, enter:

```
bldcli rsultset
```

This command creates the executable file, `rsultset`.

To call the stored procedure, run the sample client application by entering the executable file name, the name of the database to which you are connecting, and the user ID and password of the database instance:

`rsultset` *database userid password*

UNIX Embedded SQL Client Applications

The script file `bldapp` in `sqllib/samples/sqlproc` contains the commands to build an embedded SQL client application for SQL procedures. For detailed

information on the bldapp script file, see the "DB2 API and Embedded SQL Applications" section in the "Building Applications" chapter for your UNIX platform.

To build the embedded SQL client application, basecase, from the source file basecase.sqc, enter the script file name, the executable name, the database to which you are connecting, and the user ID and password of the database instance:

```
bldapp basecase database userid password
```

The result is an executable file, basecase.

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

basecase *database userid password*

Windows DB2 CLI Client Applications

The %DB2PATH%\samples\sqlproc directory contains two build files for building DB2 CLI client applications: bldmcliis for the Microsoft Visual C++ compiler, and bldvcli is for the IBM VisualAge C++ compiler. For detailed information on bldmcli, see "DB2 CLI Applications" on page 322. For detailed information on bldvcli, see "DB2 CLI Applications" on page 337.

To build the DB2 CLI client application, rsultset, from the source file rsultset.c, depending on the compiler you are using, enter either:

```
bldmcli rsultset
```

or

```
bldvcli rsultset
```

These commands create the executable file, rsultset.

To call the stored procedure, run the sample client application by entering the executable file name, the name of the database to which you are connecting, and the user ID and password of the database instance:

rsultset *database userid password*

Windows Embedded SQL Client Applications

The %DB2PATH%\samples\sqlproc directory contains two build files for building embedded SQL client applications: bldmapp is for the Microsoft Visual C++ compiler, and bldvapp is for the IBM VisualAge C++ compiler. For detailed information on bldmapp, see "DB2 API and Embedded SQL Applications" on page 328. For detailed information on bldvapp, see "DB2 API and Embedded SQL Applications" on page 342.

To build the embedded SQL client application, `basecase`, from the source file `basecase.sqc`, enter script file name, the executable name, the database to which you are connecting, and the user ID and password of the database instance. Depending on the compiler you are using, this command would be either:

```
bldmapp basecase database userid password
```

or

```
bldvapp basecase database userid password
```

The result is an executable file, `basecase`.

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

`basecase` *database userid password*

Chapter 6. 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.ibm.com/software/data/db2/udb/ad
```

Note: To build 64-bit applications with the build files in this chapter, you can either uncomment the indicated command in each build file, or set the 64-bit object mode environment with the following command:

```
export OBJECT_MODE=64
```

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 103.

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 tells you how to compile and link your stored procedure in line with 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 would be a list of the stored procedure functions, with the default entry point listed first:

  ```
  modify
  remove
  add
  ```

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

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

  and

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

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 CLI
- DB2 APIs
- Embedded SQL

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.

This is the only required parameter, and the only one needed for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

If the program contains embedded SQL, indicated by the `.sqc` extension, then the `embprep` script is called to precompile the program, producing a program file with a `.c` extension.

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

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
fi

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
```

```
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi

# Compile the error-checking utility.
xlc $CFLAGS_64 -I$DB2PATH/include -c utilcli.c

# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c

# Link the program.
xlc $CFLAGS_64 -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcli
Compile Options: **xlc** The IBM C compiler. **$CFLAGS_64** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include` **-c** Perform compile only; no link. This script has separate compile and link steps.
Link Options: **xlc** Use the compiler as a front end for the linker. **$CFLAGS_64** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-o $1** Specify the executable program. **$1.o** Specify the object file. **utilcli.o** Include the utility object file for error checking. **-L$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 DB2 library. Refer to your compiler documentation for additional compiler options.

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

```
bldcli tbinfo
```

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

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, dbusemx, from the source file dbusemx.sqc:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldcli dbusemx
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldcli dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldcli dbusemx database userid password
   ```

The result is an executable file, dbusemx.

There are three ways to run this embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:
   ```
   dbusemx
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   dbusemx database
   ```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   dbusemx database userid password
   ```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions on more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs.

The script file bldapi in sqllib/samples/cli contains the commands to build a DB2 CLI program with DB2 APIs. This file compiles and links in the utilapi utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the bldcli script.

Please see “DB2 CLI Applications” on page 104 for the compile and link options common to both `bldapi` and `bldcli`.

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

```
bldapi dbmconn
```

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

```
dbmconn
```

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; `$2`, specifies the stored procedure function that is the entry point to the shared library.

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

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

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi

# Compile the error-checking utility.
xlc $CFLAGS_64 -I$DB2PATH/include -c utilcli.c

# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c

# Link the program.
xlc $CFLAGS_64 -o $1 $1.o utilcli.o -L$DB2PATH/lib \
  -ldb2 -lm -H512 -T512 -bE:$1.exp -e $2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldclisp
Compile Options: **xlc** The IBM C compiler. **$CFLAGS_64** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`. **-c** Perform compile only; no link. This book assumes that compile and link are separate steps.
Link Options: **xlc** Use the compiler as a front end for the linker. **$CFLAGS_64** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-o $1** Specify the executable program. **$1.o** Specify the object file. **utilcli.o** Include the utility object file for error checking. **-L$DB2PATH/lib** Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`. **-ldb2** Link with the DB2 library. **-lm** Link with the math library. **-H512** Specify output file alignment. **-T512** Specify output file text segment starting address. **-bE:$.exp** Specify an export file. The export file contains a list of the stored procedures. **-e $2** Specify the default entry point to the shared library. Refer to your compiler documentation for additional compiler options.

To build the sample program `spserver` from the source file `spserver.c`, enter the build file name, program name, and the name of the stored procedure function that is the entry point to the shared library:

```
bldclisp spserver outlanguage
```

The script file copies the stored procedure to the sqllib/function directory.

Next, catalog the stored procedures by running the spcreate.db2 script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then you can catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, spserver, you can build the CLI client application, spclient, that calls the stored procedures within the shared library.

You can build spclient by using the script file, bldcli. Refer to "DB2 CLI Applications" on page 104 for details.

To access the shared library, run the sample client application by entering:

spclient *database userid password*

where

database
Is the name of the database to which you want to connect. The name could be sample, or its alias, or another database name.

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, spserver, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

DB2 API and Embedded SQL Applications

The build file, bldapp, in sqllib/samples/c, contains the commands to build a DB2 application program.

The first parameter, $1, specifies the name of your source file. This is the only required parameter, and the only one needed for DB2 API programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- AIX
# Builds a C application program.
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi

# If embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  xlc $CFLAGS_64 -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  xlc $CFLAGS_64 -I$DB2PATH/include -c utilapi.c
fi

# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c

if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  xlc $CFLAGS_64 -o $1 $1.o utilemb.o -ldb2 -L$DB2PATH/lib
```

```
else
  # Link the program with utilapi.o
  xlc $CFLAGS_64 -o $1 $1.o utilapi.o -ldb2 -L$DB2PATH/lib
fi
```

Compile and Link Options for bldapp
Compile Options: **xlc** The IBM C compiler. **$CFLAGS_64** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`. **-c** Perform compile only; no link. Compile and link are separate steps.
Link Options: **xlc** Use the compiler as a front end for the linker. **$CFLAGS_64** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-o $** Specify the executable program. **$1.o** Specify the program object file. **utilemb.o** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.o** If not an embedded SQL program, include the DB2 API utility object file for error checking. **-ldb2** Link to the database manager library. **-L$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 DB2 API non-embedded SQL sample program, `client`, from the source file `client.c`, enter:

```
bldapp client
```

The result is an executable file, `client`.

To run the executable file, enter the executable name:

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqc`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The script file `bldsrv`, 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 stored procedure function that is the entry point to the shared library. The third parameter, `$3`, specifies the name of the database to which you want to connect. Since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Only the first two parameters, source file name and entry point, are required. Database name is optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldsrv script file -- AIX
# Builds a C stored procedure
# Usage: bldsrv <prog_name> <entry_point> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $3

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi

# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c

# Link the program using the export file $1.exp,
# creating shared library $1 with entry point $2.
xlc $CFLAGS_64 -o $1 $1.o -ldb2 -L$DB2PATH/lib -H512 -T512 -bE:$1.exp -e $2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldsrv

Compile Options:

xlc The IBM C compiler.

$CFLAGS_64
Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.

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

-c Perform compile only; no link. Compile and link are separate steps.

Compile and Link Options for bldsrv	
Link options:	
xlc	Use the compiler as a front end for the linker.
$CFLAGS_64	Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-o $1	Specify the output as a shared library file.
$1.o	Specify the stored procedure object file.
-ldb2	Link with the DB2 library.
-L$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 `spserver` from source file `spserver.sqc`, if connecting to the `sample` database, enter the build file name, program name, and the name of the stored procedure function that is the entry point to the shared library:

```
bldsrv spserver outlanguage
```

If connecting to another database, also enter the database name:

```
bldsrv spserver outlanguage database
```

The script file copies the stored procedure to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the client application, `spclient`, that accesses the shared library.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 109 for details.

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

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

User-Defined Functions (UDFs)

The script file `bldudf`, 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 do not 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 stored procedure function that is the entry point to the shared library. The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldudf script file -- AIX
# Builds a C UDF library
# Usage: bldudf <prog_name> <entry_point>

# Set DB2PATH to where DB2 will be accessed.
```

```
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi

# Compile the program.
xlc $CFLAGS_64 -I$DB2PATH/include -c $1.c

# Link the program.
xlc $CFLAGS_64 -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.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldudf

Compile Options:

xlc The IBM C compiler.

$CFLAGS_64
Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.

-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 bldudf	
Link Options:	
xlc	Use the compiler as a front end for the linker.
$CFLAGS_64	Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-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 103 for more information on creating UDFs.	

To build the user-defined function program, `udfsrv`, from the source file `udfsrv.c`, enter the build file name, program name, and UDF function that is the entry point to the shared library:

```
bldudf udfsrv ScalarUDF
```

The script file copies the UDF to the `sqllib/function` directory.

If necessary, set the file mode for the UDF so the client application can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI udfcli program from the source file udfcli.c, in sqllib/samples/cli, using the script file bldcli. Refer to "DB2 CLI Applications" on page 104 for details.

You can build the embedded SQL udfcli program from the source file udfcli.sqc, in sqllib/samples/c, using the script file bldapp. Refer to "DB2 API and Embedded SQL Applications" on page 109 for details.

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

```
udfcli
```

The calling application calls the ScalarUDF function from the udfsrv library.

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 for 32-bit applications, use the xlc_r7 or xlC_r7 compiler. The _r versions (as well as the other multi-threaded compiler front ends) set the appropriate preprocessor defines for multi-threaded compilation, and supply the appropriate threaded library names to the linker.

Additional information about compiler and link flag settings using the multi-threaded compiler front ends can be obtained from /etc/xlC.cfg when using the 3.1 compiler, or /etc/ibmcxx.cfg when using the 3.6 or newer compilers.

The script file bldmt, in sqllib/samples/c, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- AIX
# Builds a C multi-threaded embedded SQL program.
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# Precompile and bind the program.
embprep $1 $2 $3 $4
```

```
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi

# Compile the program.
xlc_r $CFLAGS_64 -I$DB2PATH/include -c $1.c

# Link the program.
xlc_r $CFLAGS_64 -o $1 $1.o -L$DB2PATH/lib -ldb2
```

Besides the `xlc_r` compiler, discussed above, and the absence of a utility file linked in, the compile and link options are the same as those used for the embedded SQL script file, `bldapp`. For information on these options, see "DB2 API and Embedded SQL Applications" on page 109.

To build the multi-threaded sample program, `thdsrver`, from the source file `thdsrver.sqc`, enter:

```
bldmt 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 and Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)
- Multi-threaded Applications

DB2 API and Embedded SQL Applications

The build file, `bldapp`, in `sqllib/samples/cpp`, contains the commands to build DB2 API and embedded SQL applications.

The first parameter, `$1`, specifies the name of your source file. This is the only required parameter, and the only one needed for DB2 API programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

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

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

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  xlC $CFLAGS_64 -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  xlC $CFLAGS_64 -I$DB2PATH/include -c utilapi.C
fi

# Compile the program.
xlC $CFLAGS_64 -I$DB2PATH/include -c $1.C

if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  xlC $CFLAGS_64 -o $1 $1.o utilemb.o -ldb2 -L$DB2PATH/lib
else
  # Link the program with utilapi.o
  xlC $CFLAGS_64 -o $1 $1.o utilapi.o -ldb2 -L$DB2PATH/lib
fi
```

Compile and Link Options for bldapp
Compile Options: **xlC** The IBM C Set++ compiler. **$CFLAGS_64** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`. **-c** Perform compile only; no link. Compile and link are separate steps.
Link options: **xlC** Use the compiler as a front end for the linker. **$CFLAGS_64** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-o $1** Specify the executable program. **-o $1** Specify the program object file. **utilapi.o** Include the API utility object file for non-embedded SQL programs. **utilemb.o** Include the embedded SQL utility object file for embedded SQL programs. **-ldb2** Link with the database manager library. **-L$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 non-embedded SQL sample program `client` from the source file `client.C`, enter:

```
bldapp client
```

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

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqC`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

Note: Please see the information for building C++ stored procedures in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file `bldsrv`, 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 stored procedure function that is the entry point to the shared library. The third parameter, `$3`, specifies the name of the database to which you want to connect. Since the stored procedure must be build on the same instance where the database resides, you do not need parameters for user ID and password.

Only the first two parameters, source file name and entry point, are required. Database name is optional. If no database name is supplied, the program uses the default `sample` database.

```
#! /bin/ksh
# bldsrv script file -- AIX
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
  LFLAGS_64=-X64
else
  CFLAGS_64=
  LFLAGS_64=
fi

# Compile the program.
xlC $CFLAGS_64 -I$DB2PATH/include -c $1.C

# Link using export file $1.exp, creating shared library $1
makeC++SharedLib $LFLAGS_64 -p 1024 -o $1 $1.o -L$DB2PATH/lib -ldb2 -E $1.exp

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldsrv

Compile Options:

xlC The IBM C Set++ compiler.

$CFLAGS_64
Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.

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

-c Perform compile only; no link. Compile and link are separate steps.

Compile and Link Options for bldsrv
Link options: **makeC++SharedLib** Linker script for stored procedures with static constructors. **$LFLAGS_64** Contains "-X64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-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. **-e $2** Specify an entry point to the shared library. Refer to your compiler documentation for additional compiler options.

To build the sample program `spserver` from the source file `spserver.sqC`, if connecting to the `sample` database, enter the build file name, program name, and the name of the stored procedure function that is the entry point to the shared library:

```
bldsrv spserver outlanguage
```

If connecting to another database, also enter the database name:

```
bldsrv spserver outlanguage database
```

The script file copies the shared library to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the client application `spclient` that calls the stored procedures within it.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 119 for details.

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

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, which executes a number of stored procedure functions on the server database, and then returns the output to the client application.

User-Defined Functions (UDFs)

Note: Please see the information for building C++ UDFs in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file `bldudf`, in `sqllib/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, precompile, and bind the program.

Parameter `$1` specifies the name of your source file. Parameter `$2` specifies the user-defined function that is the entry point to the shared library. The script file uses the source file name, `$1`, for the shared library name.

```
#! /bin/ksh
# bldudf script file -- AIX
# Builds a C++ UDF library
```

```
# Usage: bldudf <prog_name>

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

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
  LFLAGS_64=-X64
else
  CFLAGS_64=
  LFLAGS_64=
fi

# Compile the program.
if [[ -f $1".c" ]]
then
  xlC $CFLAGS_64 -I$DB2PATH/include -c $1.c
elif [[ -f $1".C" ]]
then
  xlC $CFLAGS_64 -I$DB2PATH/include -c $1.C
fi

# Link using export file $1.exp, creating shared library $1
makeC++SharedLib $LFLAGS_64 -p 1024  -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2apie
   -E $1.exp

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldudf

Compile Options:

xlC The IBM C Set++ compiler.

$CFLAGS_64
Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.

-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 bldudf
Link Options: **makeC++SharedLib** Linker script for stored procedures with static constructors. **$LFLAGS_64** Contains "-X64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-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 103 for more information on creating UDFs.

To build the user-defined function program `udfsrv` from the source file `udfsrv.c`, enter the build file name, program name, and UDF function that is the entry point to the shared library:

```
bldudf udfsrv ScalarUDF
```

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 `udfsrv`, you can build the client application, `udfcli`, that calls it. You can build the `udfcli` program from the `udfcli.sqC` source file in `sqllib/samples/cpp` using the script file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 119 for details.

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

```
udfcli
```

The calling application calls the ScalarUDF function in the udfsrv library.

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 for 32-bit applications, use the xlC_r7 or xlc_r7 compiler. The _r versions (as well as the other multi-threaded compiler front ends) set the appropriate preprocessor defines for multi-threaded compilation and supply the appropriate threaded library names to the linker.

Additional information about compiler and link flag settings using the multi-threaded compiler front ends can be obtained from /etc/xlC.cfg when using the 3.1 compiler, or /etc/ibmcxx.cfg when using the 3.6 or newer compilers.

The script file bldmt, in sqllib/samples/cpp, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- AIX
# Builds a C++ multi-threaded embedded SQL program
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# Precompile and bind the program.
embprep $1 $2 $3 $4

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-q64
else
  CFLAGS_64=
fi

# Compile the program.
xlC_r $CFLAGS_64 -I$DB2PATH/include -c $1.C
```

```
# Link the program.
xlC_r $CFLAGS_64 -o $1 $1.o -L$DB2PATH/lib -ldb2
```

Besides the `xlC_r` compiler, discussed above, and no utility file linked in, the compile and link options are the same as those used in the embedded SQL script file, `bldapp`. For information on these options, see "DB2 API and Embedded SQL Applications" on page 119.

To build the multi-threaded sample program, `thdsrver`, from the source file `thdsrver.sqC`, enter:

```
bldmt thdsrver
```

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

```
thdsrver
```

VisualAge C++ Version 4.0

This VisualAge C++ compiler is for AIX, OS/2, and Windows 32-bit operating systems. The information in this section applies to all these platforms.

The VisualAge C++ compiler differs from other compilers documented in this book. 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:

cli.icc
: DB2 CLI configuration file. For details, see "DB2 CLI Applications" on page 130.

cliapi.icc
: DB2 CLI with DB2 APIs configuration file. For details, see "DB2 CLI Applications with DB2 APIs" on page 132.

clis.icc
: DB2 CLI stored procedure configuration file. For details, see "DB2 CLI Stored Procedures" on page 133.

api.icc
DB2 API configuration file. For details, see "DB2 API Applications" on page 136.

emb.icc
Embedded SQL configuration file. For details, see "Embedded SQL Applications" on page 137.

stp.icc
Embedded SQL stored procedure configuration file. For details, see "Embedded SQL Stored Procedures" on page 139.

udf.icc
User-defined function configuration file. For details, see "User-Defined Functions (UDFs)" on page 142.

DB2 CLI Applications

The configuration file, `cli.icc`, in `sqllib/samples/cli` on AIX, and in `%DB2PATH%\samples\cli` on OS/2 and Windows 32-bit operating systems, allows you to build DB2 CLI programs.

```
// 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"
utilcli = "utilcli.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 utilcli
    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 `tbinfo` from the source file `tbinfo.c`, do the following:

1. Set the CLI environment variable to the program name by entering:

   ```
   export CLI=tbinfo
   ```

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, `tbinfo`. You can run the program by entering the executable name:

```
tbinfo
```

Building and Running Embedded SQL Applications

You can use the `cli.icc` configuration file to compile an embedded SQL program after the program is precompiled with the `embprep` file on AIX, `embprep.cmd` on OS/2, or `embprep.bat` on Windows 32-bit operating systems. This file precompiles the source file and binds the program to the database.

There are three ways to precompile the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:

```
embprep dbusemx
```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   embprep dbusemx database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   embprep dbusemx database userid password
   ```

The result is a precompiled C file, `dbusemx.c`.

After it is precompiled, the C file can be compiled with the `cli.icc` file, as follows:

1. Set the CLI environment variable to the program name by entering:

   ```
   export CLI=dbusemx
   ```

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.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   dbusemx
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   dbusemx database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   dbusemx database userid password
   ```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions with more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs. The configuration file, `cliapi.icc`, in

sqllib/samples/cli on AIX, and in %DB2PATH%\samples\cli on OS/2 and Windows 32-bit operating systems, allows you to build DB2 CLI programs with DB2 APIs.

This file compiles and links in the utilapi utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the cli.icc configuration file.

To build the DB2 CLI sample program, dbmconn, from the source file dbmconn.c, do the following:

1. Set the CLIAPI environment variable to the program name by entering:
   ```
   export CLIAPI=dbmconn
   ```
2. If you have a cliapi.ics file in your working directory, produced by building a different program with the cliapi.icc file, delete the cliapi.ics file with this command:
   ```
   rm cliapi.ics
   ```
   An existing cliapi.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 cliapi.icc
   ```
   Note: The vacbld command is provided by VisualAge C++ Version 4.0.

The result is an executable file, dbmconn. You can run the program by entering the executable name:

```
dbmconn
```

DB2 CLI Stored Procedures

The configuration file, clis.icc, in sqllib/samples/cli on AIX, and in %DB2PATH%\samples\cli on OS/2 and Windows 32-bit operating systems, allows you to build DB2 CLI stored procedures.

```
// 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"
utilcli = "utilcli.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 utilcli
    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 spserver from the source file spserver.c, do the following:

1. Set the CLIS environment variable to the program name by entering:

   ```
   export CLIS=spserver
   ```

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 on AIX, and in the path %DB2PATH%\function on OS/2 and Windows 32-bit operating systems.

Next, catalog the stored procedures by running the spcreate.db2 script on the server. First, connect to the database with the user ID and password of the instance where the database is located:

```
db2 connect to sample userid password
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure spserver, you can build the CLI client application spclient that calls the stored procedure. You can build spclient by using the configuration file, cli.icc. Refer to "DB2 CLI Applications" on page 130 for details.

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

spclient *database userid password*

where

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 accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

DB2 API Applications

The configuration file, `api.icc`, in `sqllib/samples/c` and in `sqllib/samples/cpp` on AIX, and in `%DB2PATH%\samples\c` and in `%DB2PATH%\samples\c` on OS/2 and Windows 32-bit operating systems, allows you to build DB2 API programs in C or C++.

```
// 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   = "utilapi.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
```

Embedded SQL Applications

The configuration file, `emb.icc`, in `sqllib/samples/c` and `sqllib/samples/cpp` on AIX, and in `%DB2PATH%\samples\c` and `%DB2PATH%\samples\cpp` on OS/2 and Windows 32-bit operating systems, allows you to build DB2 embedded SQL applications in C and C++.

```
// 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 = "utilemb.sqc"

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 and sqllib/samples/cpp on AIX, and in %DB2PATH%\samples\c and %DB2PATH%\samples\cpp on OS/2 and Windows 32-bit operating systems, allows you to build DB2 embedded SQL stored procedures in C and C++.

```
// 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"
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 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 spserver from the source file spserver.sqc, do the following:

1. Set the STP environment variable to the program name by entering:

   ```
   export STP=spserver
   ```

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 on AIX, and in the path %DB2PATH%\function on OS/2 and Windows 32-bit operating systems.

Next, catalog the stored procedures by running the spcreate.db2 script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure, spserver, you can build the client application, spclient, that calls the stored procedure. You can build spclient using the configuration file, emb.icc. Refer to “Embedded SQL Applications” on page 137 for details.

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

`spclient` *database userid password*

where

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 accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

User-Defined Functions (UDFs)

The configuration file, `udf.icc`, in `sqllib/samples/c` and `sqllib/samples/cpp` on AIX, and in `%DB2PATH%\samples\c` and `%DB2PATH%\samples\cpp` on OS/2 and Windows 32-bit operating systems, allows you to build user-defined functions in C and C++.

```
// 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 `udfsrv` from the source file `udf.c`, do the following:

1. Set the UDF environment variable to the program name by entering:

   ```
   export UDF=udfsrv
   ```

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 `udfsrv`, you can build the client application, `udfcli`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI `udfcli` program from the source file `udfcli.c`, in `sqllib/samples/cli` on AIX, and in `%DB2PATH%\samples\cli` on OS/2 and Windows 32-bit operating systems, by using the configuration file `cli.icc`. Refer to "DB2 CLI Applications" on page 130 for details.

You can build the embedded SQL `udfcli` program from the source file `udfcli.sqc`, in `sqllib/samples/c` on AIX, and in `%DB2PATH%\samples\cli` on OS/2 and Windows 32-bit operating systems, by using the configuration file `emb.icc`. Refer to "Embedded SQL Applications" on page 137 for details.

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

```
udfcli
```

The calling application calls the `ScalarUDF` function from the `udfsrv` library.

IBM COBOL Set for AIX

This section includes the following topics:

- Using the Compiler
- DB2 API and 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 and Embedded SQL Applications

The build file, bldapp, in sqllib/samples/cobol, contains the commands to build a DB2 application program.

The first parameter, $1, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, bldapp passes the parameters to the precompile and bind file, embprep. If no database name is supplied, the default sample database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- AIX
# Builds an IBM COBOL application program
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

```
DB2PATH=$HOME/sqllib

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqb" ]]
then
  embprep $1 $2 $3 $4
fi

# 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 bldapp

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. Compile and link are separate steps.

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 non-embedded SQL sample program `client` from the source file `client.cbl`, enter:

```
bldapp client
```

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

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqb`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The script file `bldsrv`, in `sqllib/samples/cobol`, 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. Since the stored procedure must be build on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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. 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
# bldsrv script file -- AIX
# Builds an IBM COBOL stored procedure
# Usage:  bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# 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 bldsrv
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`.
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`, connecting to the sample database, enter:

```
bldsrv outsrv
```

If connecting to another database, also include the database name:

```
bldsrv outsrv database
```

The script file copies the stored procedure to the server in the path `sqllib/function`.

If necessary, set the file mode for the stored procedure so the client application can access 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 `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 145 for details.

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

```
outcli database userid password
```

where

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 accesses the shared library, `outsrv`, and executes the stored procedure function of the same name on the server database, and then returns the output to the client application.

Micro Focus COBOL

This section includes the following topics:

- Using the Compiler
- DB2 API and 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 and Embedded SQL Applications

The build file, `bldapp`, in `sqllib/samples/cobol_mf`, contains the commands to build a DB2 application program.

The first parameter, `$1`, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- AIX
# Builds a Micro Focus COBOL application program
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqb" ]]
then
  embprep $1 $2 $3 $4
fi

# 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	
Compile Options:	
cob	The COBOL compiler.
-c	Perform compile only; no link.
-x	Produce an executable program.

Compile and Link Options for bldmfapi	
Link Options:	
cob	Use the compiler as a front end for the linker.
-x	Produce an executable program.
-o $1	Specify the executable program.
$1.o	Specify the program object file.
-ldb2	Link to the DB2 library.
-ldb2gmf	Link to the DB2 exception-handler library for Micro Focus COBOL.
-L$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 non-embedded SQL sample program, `client`, from the source file `client.cbl`, enter:

```
bldapp client
```

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

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqb`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldapp updat
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldapp updat database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

```
updat
```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

Notes:

1. Before building a stored procedure on AIX 4.3 using the Micro Focus 4.1 compiler, execute the following commands:

   ```
   db2stop
   db2set DB2LIBPATH=$LIBPATH
   db2set DB2ENVLIST="COBDIR LIBPATH"
   db2set
   db2start
   ```

   Ensure that `db2stop` stops the database and `LIBPATH` is set properly in your shell environment. The last `db2set` command is issued to display your settings: make sure `DB2LIBPATH` and `DB2ENVLIST` are set correctly.

2. 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, `bldsrv`, 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 `bldsrv`. 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 `bldsrv`, in `sqllib/samples/cobol_mf`, 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. Since the stored procedure must be build on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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. 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
# bldsrv script file -- AIX
# Builds a Micro Focus COBOL stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY

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

# Create the wrapper program for the stored procedure.
wrapsrv $1

# Link the program using export file ${1}_wrap.exp
# creating shared library $1 with 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 sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldsrv	
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 bldsrv
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, `wrapsrv`, causes the Micro Focus COBOL routine, `cobinit()`, to be called right before the stored procedure is executed. Its contents are shown below.

```
#! /bin/ksh
# wrapsrv script file
# Creates the wrapper program for Micro Focus COBOL stored procedures
# Usage: wrapsrv <stored_proc>

# Note: The client program calls "<stored_proc>_wrap" not "<stored_proc>"

# 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`, if connecting to the sample database, enter:

```
bldsrv outsrv
```

If connecting to another database, also enter the database name:

```
bldsrv outsrv database
```

The script file copies the shared library to the server in the path `sqllib/function`.

If necessary, set the file mode for the shared library so the client application can access it.

Once you build the stored procedure `outsrv`, you can build the client application `outcli` that calls it. You can build `outcli` using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 151 for details.

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

```
outcli database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `outsrv`, and executes the stored procedure function of the same name on the server database. The output is then returned to the client application.

Exiting the Stored Procedure

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

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

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

REXX

You do not precompile or bind REXX programs.

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

Enter:

```
export LIBPATH=$LIBPATH:/lib:/usr/lib:/usr/lpp/db2_07_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 7. Building HP-UX Applications

This chapter provides detailed information for building DB2 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 DB2 application development Web page at:

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

Notes:

1. The `+DAportable` option is used in the compile and link steps of the DB2 build files and makefiles. This option generates code compatible across PA_RISC 1.1 and 2.0 workstations. The use of this option comes with a slight performance cost. To improve performance, you can remove the `+DAportable` option from the build files and makefiles provided in the `sqllib/samples` directory. Without this option, you may get a warning similar to the following when building HP-UX programs:

   ```
   (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 1.1 or 2.0 system, this warning does not apply.
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 includes the following topics:

- DB2 CLI Applications
- DB2 CLI Stored Procedures
- DB2 API and Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)
- Multi-threaded Applications

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.

This is the only required parameter, and the only one needed for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

If the program contains embedded SQL, indicated by the `.sqc` extension, then the `embprep` script is called to precompile the program, producing a program file with a `.c` extension.

```
#! /bin/ksh
# bldcli script file -- HP-UX
# Builds a CLI program with HP-UX C.
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]
# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
cc +DAportable -Aa +e -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc +DAportable -Aa +e -I$DB2PATH/include -c $1.c

# Link the program.
cc +DAportable -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcli

Compile Options:

cc Use the C compiler.

+DAportable
Generates code compatible across PA_RISC 1.x and 2.0 workstations.

-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. Compile and link are separate steps.

Compile and Link Options for bldcli	
Link Options:	
cc	Use the compiler as a front end for the linker.
+DAportable	Use code compatible across PA_RISC 1.x and 2.0 workstations.
-o $1	Specify the executable program.
-o $1.o	Specify the object file.
utilcli.o	Include the utility object file for error checking.
-L$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 `tbinfo` from the source file `tbinfo.c`, enter:

```
bldcli tbinfo
```

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

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldcli dbusemx
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldcli dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldcli dbusemx database userid password
   ```

The result is an executable file, `dbusemx`.

There are three ways to run this embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:

   ```
   dbusemx
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   dbusemx database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   dbusemx database userid password
   ```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions on more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs.

The script file bldapi in sqllib/samples/cli contains the commands to build a DB2 CLI program with DB2 APIs. This file compiles and links in the utilapi utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the bldcli script. Please see "DB2 CLI Applications" on page 162 for the compile and link options common to both bldapi and bldcli.

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

```
bldapi dbmconn
```

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

```
dbmconn
```

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.

Only the first parameter, source file name, is required. Database name is optional. If no database name is supplied, the program uses the default sample database.

```
#! /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 +DAportable +u1 +z -Aa +e -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc +DAportable +u1 +z -Aa +e -I$DB2PATH/include -c $1.c

# Link the program.
ld -b -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldclisp

Compile Options:

cc The C compiler.

+DAportable
Generates code compatible across PA_RISC 1.x and 2.0 workstations.

+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.

Compile and Link Options for bldclisp
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 spserver from source file spserver.c, if connecting to the sample database, enter:

```
bldclisp spserver
```

The script file copies the shared library to the server in the path sqllib/function.

Next, catalog the stored procedures by running the spcreate.db2 script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, spserver, you can build the CLI client application spclient that accesses the shared library.

You can build spclient by using the script file, bldcli. Refer to "DB2 CLI Applications" on page 162 for details.

To access the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, which executes a number of stored procedure functions on the server database, and then returns the output to the client application.

DB2 API and Embedded SQL Applications

The script file, `bldapp`, in `sqllib/samples/c`, contains the commands to build a DB2 application program.

The first parameter, `$1`, specifies the name of your source file. This is the only required parameter, and the only one needed for DB2 API programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- HP-UX
# Builds a C application program
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  cc +DAportable -Aa +e -I$DB2PATH/include -c utilemb.c
else
```

```
  # Compile the utilapi.c error-checking utility.
  cc +DAportable -Aa +e -I$DB2PATH/include -c utilapi.c
fi

# Compile the program.
cc +DAportable -Aa +e -I$DB2PATH/include -c $1.c

if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  cc +DAportable -o $1 $1.o utilemb.o -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  cc +DAportable -o $1 $1.o utilapi.o -L$DB2PATH/lib -ldb2
fi
```

Compile and Link Options for bldapp

Compile Options:

cc The C compiler.

+DAportable
Generates code compatible across PA_RISC 1.x and 2.0 workstations.

-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.

Link Options:

cc Use the compiler to link edit.

+DAportable
Use code compatible across PA_RISC 1.x and 2.0 workstations.

-o $1 Specify the executable.

$1.o Specify the program object file.

utilemb.o
If an embedded SQL program, include the embedded SQL utility object file for error checking.

utilapi.o
If a non-embedded SQL program, include the DB2 API 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 DB2 API non-embedded SQL sample program, `client`, from the source file `client.c`, enter:

```
bldapp client
```

The result is an executable file, `client`.

To run the executable file, enter the executable name:

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqc`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The script file `bldsrv`, 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. Since the stored procedure must be build on the same instance where the database resides, you do not have to specify parameters for user ID and password.

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

```
#! /bin/ksh
# bldsrv script file -- HP-UX
# Builds a C stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# Compile the program.
cc +DAportable +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
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldsrv

Compile Options:

cc The C compiler.

+DAportable
Generates code compatible across PA_RISC 1.x and 2.0 workstations.

+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 bldsrv
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 `spserver` from the source file `spserver.sqc`, if connecting to the `sample` database, enter:

```
bldsrv spserver
```

If connecting to another database, also enter the database name:

```
bldsrv spserver database
```

The script file copies the shared library to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the client application `spclient` that accesses it.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 168 for details.

To call the stored procedures in the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

User-Defined Functions (UDFs)

The script file `bldudf`, 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 do not need to connect to a database, 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
# bldudf script file -- HP-UX
# Builds a C UDF library
# Usage: bldudf <prog_name>

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

# Compile the program.
cc +DAportable +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 sqllib/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 bldudf
Compile Options: **cc** The C compiler. **+DAportable** Generates code compatible across PA_RISC 1.x and 2.0 workstations. **+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.
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 `udfsrv` from the source file `udfsrv.c`, enter:

```
bldudf udfsrv
```

The script file copies the UDF to the `sqllib/function` directory.

If necessary, set the file mode for the UDF so the client application can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI udfcli program from the source file udfcli.c, in sqllib/samples/cli, using the script file bldcli. Refer to "DB2 CLI Applications" on page 162 for details.

You can build the embedded SQL udfcli program from the source file udfcli.sqc, in sqllib/samples/c, using the script file bldapp. Refer to "DB2 API and Embedded SQL Applications" on page 168 for details.

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

```
udfcli
```

The calling application calls the ScalarUDF function from the udfsrv library.

Multi-threaded Applications

Note: HP-UX provides a POSIX thread library and a DCE thread library. Multi-threaded applications using the POSIX thread library are supported by DB2.

Multi-threaded applications on HP-UX 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, bldmt, in sqllib/samples/c, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- HP-UX
# Builds a C multi-threaded embedded SQL program
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# Precompile and bind the program.
embprep $1 $2 $3 $4

# Compile the program.
cc +DAportable -Aa +e -I$DB2PATH/include -D_POSIX_C_SOURCE=199506L -c $1.c
```

```
# Link the program
cc +DAportable -o $1 $1.o -L$DB2PATH/lib -ldb2 -lpthread
```

Besides the `-D_POSIX_C_SOURCE=199506L` compile option, and the `-lpthread` link option, discussed above, and the absence of a utility file being linked in, the other compile and link options are the same as those used for the `bldapp` file. For information on these options, see "DB2 API and Embedded SQL Applications" on page 168.

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

```
bldmt 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 following topics:

- DB2 API and Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)
- Multi-threaded Applications

DB2 API and Embedded SQL Applications

The script file `bldapp`, in `sqllib/samples/cpp`, contains the commands to build a DB2 application program.

The first parameter, `$1`, specifies the name of your source file. This is the only required parameter for programs not containing embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- HP-UX
# Builds a C++ application program
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]
```

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  CC +DAportable +a1 -ext -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  CC +DAportable +a1 -ext -I$DB2PATH/include -c utilapi.C
fi

# Compile the program.
CC +DAportable +a1 -ext -I$DB2PATH/include -c $1.C

if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  CC +DAportable -o $1 $1.o utilemb.o -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  CC +DAportable -o $1 $1.o utilapi.o -L$DB2PATH/lib -ldb2
fi
```

Compile and Link Options for bldapp

Compile Options:

CC The C compiler.

+DAportable
Generates code compatible across PA_RISC 1.x and 2.0 workstations.

+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.

Compile and Link Options for bldapp
Link Options: **CC** Use the compiler as a front end for the linker. **+DAportable** Uses code compatible across PA_RISC 1.x and 2.0 workstations. **-o $1** Specify the executable. **$1.o** Specify the program object file. **utilemb.o** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.o** If a non-embedded SQL program, include the DB2 API utility object file for error checking. **-L$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 non-embedded SQL DB2 API sample program `client` from the source file `client.C`, enter:

```
bldapp client
```

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

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqC`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldapp updat
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldapp updat database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

Note: Please see the information for building C++ stored procedures in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file `bldsrv`, 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. Since the stored procedure must be build on the same instance where the database resides, you do not need parameters for user ID and password.

Only the first parameter, the source file name, is required. Database name is 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
# bldsrv script file -- HP-UX
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# Compile the program. First ensure it is coded with extern "C".
CC +DAportable +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 sqllib/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 bldsrv
Compile Options: **CC** The C++ compiler. **+DAportable** Generates code compatible across PA_RISC 1.x and 2.0 workstations. **+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.
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 `spserver` from the source file `spserver.sqC`, if connecting to the `sample` database, enter:

```
bldsrv spserver
```

If connecting to another database, also enter the database name:

```
bldsrv spserver database
```

The script file copies the shared library to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared libary, `spserver`, you can build the client application `spclient` that calls the stored procedures within the shared library.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 176 for details.

To call the stored procedures in the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The stored procedures return the output to the client application.

User-Defined Functions (UDFs)

Note: Please see the information for building C++ UDFs in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file `bldudf`, 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 do not need to connect to a database, precompile, and bind the program.

Parameter $1, specifies the name of your source file. The script file uses the source file name for the shared library name.

```
#! /bin/ksh
# bldudf script file -- HP-UX
# Builds a C or C++ UDF library
# Usage: bldudf <prog_name>

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

# Compile the program.
if [[ -f $1".c" ]]
then
  CC +DAportable -ext +a1 +z -I$DB2PATH/include -c $1.c
elif [[ -f $1".C" ]]
then
  CC +DAportable -ext +a1 +z -I$DB2PATH/include -c $1.C
fi

# Link the program.
CC +DAportable -b -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldudf	
Compile Options:	
CC	The C++ compiler.
+DAportable	Generates code compatible across PA_RISC 1.x and 2.0 workstations.
-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 bldudf
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 `udfsrv` from the source file `udfsrv.c`, enter:

```
bldudf udfsrv
```

The script file copies the UDF to the `sqllib/function` directory.

If necessary, set the file mode for the UDF so the client application can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. You can build the `udfcli` program from the `udfcli.sqC` source file in `sqllib/samples/cpp` using the script file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 176 for details.

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

```
udfcli
```

The calling application calls the `ScalarUDF` function in the `udfsrv` library.

Multi-threaded Applications

Note: HP-UX 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.

Multi-threaded applications on HP-UX 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, bldmt, in sqllib/samples/cpp, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- HP-UX
# Builds a C++ multi-threaded embedded SQL program
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# Precompile and bind the program.
embprep $1 $2 $3 $4

# Compile the program.
CC +DAportable +a1 -ext -I$DB2PATH/include \
   -D_HPUX_SOURCE -D_POSIX_C_SOURCE=199506L -c $1.C

# Link the program
CC -o $1 $1.o -L$DB2PATH/lib -ldb2 -lpthread
```

Besides the -D_HPUX_SOURCE and -D_POSIX_C_SOURCE=199506L compile options, and the -lpthread link option, discussed above, and the absence of a utility file linked in, the other compile and link options are the same as those used for the embedded SQL script file, bldapp. For information on these options, see "DB2 API and Embedded SQL Applications" on page 176.

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

```
bldmt 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 and 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, you have to add the DB2 Generic API entry points to the Micro Focus runtime module `rts32` by executing the `mkrts` command provided by Micro Focus. You also need to run `mkcheck` to update the check file. If this is not run, you will receive a 173 error in SQLGSTRT.

  Before running `mkrts` and `mkcheck`, `COBOPT` must be set in the following steps:

  1. Log in as root.
  2. From the directory $COBDIR/src/rts enter:

     ```
     COBOPT=/opt/IBMdb2/V7.1/lib/db2mkrts.args; export COBOPT
     ksh mkrts
     mv $COBDIR/rts32 $COBDIR/rts32.orig
     cp rts32 $COBDIR/rts32
     ```

  3. You must also rebuild the `check` executable which Hewlett-Packard ships with the product. If you do not rebuild the `check` executable located in your `$COBDIR` directory, attempts to compile using `cob -C SQL` will fail with a run-time system 173 error because the DB2 pre-processor calls the DB2 library. To rebuild `check`, you should change to the `src/sql` directory under your `$COBDIR` directory as root and run the `mkcheck` script. Once the script completes, you need to move the resulting `check` executable to your `$COBDIR` directory. From the directory `$COBDIR/src/sql`, enter:

     ```
     COBOPT=/opt/IBMdb2/V7.1/lib/db2mkrts.args; export COBOPT
     ksh mkcheck
     mv $COBDIR/check $COBDIR/check.orig
     cp check $COBDIR/check
     ```

  Now you can execute `mkrts` with the arguments supplied in the following directory:

  ```
  /opt/IBMdb2/V7.1/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/V7.1/include/cobol_mf
  ```

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

DB2 API and Embedded SQL Applications

The script file `bldapp`, in `sqllib/samples/cobol_mf`, contains the commands to build DB2 API and embedded SQL application programs.

The first parameter, `$1`, specifies the name of your source file. This is the only required parameter, and the only one needed for DB2 API programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- HP-UX
# Builds a Micro Focus COBOL application program
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqb" ]]
then
  embprep $1 $2 $3 $4
fi

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

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

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

# Link the program.
cob +DAportable -x $1.o checkerr.o -L$DB2PATH/lib -ldb2 -ldb2gmf
```

Compile and Link Options for bldapp
Compile Options: **cob** The Micro Focus COBOL compiler. **+DAportable** Generates code compatible across PA_RISC 1.x and 2.0 workstations. **-cx** Compile to object module.
Link options: **cob** Use the compiler as a front end for the linker. **+DAportable** Use code compatible across PA_RISC 1.x and 2.0 workstations. **-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 non-embedded SQL sample program, `client`, from the source file `client.cbl`, enter:

```
bldapp client
```

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

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqb`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, updat.

There are three ways to run this embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The script file bldsrv, 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. Since the stored procedure must be build on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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
# bldsrv script file -- HP-UX
# Builds a Micro Focus COBOL stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

# Compile the program.
cob +DAportable +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.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldsrv
Compile Options: **cob** The COBOL compiler. **+DAportable** Generates code compatible across PA_RISC 1.x and 2.0 workstations. **+z** Generate position-independent code. **-cx** Compile to object module.
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`, if connecting to the sample database, enter:

```
bldsrv outsrv
```

If connecting to another database, also enter the database name:

```
bldsrv outsrv database
```

The script file copies the stored procedure to the `sqllib/function` directory.

If necessary, set the file mode for the stored procedure so the client application can access 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, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 186 for details.

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

```
outcli database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the stored procedure library, `outsrv`, which executes the stored procedure function of the same name on the server database, and then returns the output to the client application.

Exiting the Stored Procedure

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

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

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

Chapter 8. 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.ibm.com/software/data/db2/udb/ad`

Linux C

This section includes the following topics:

- DB2 CLI Applications
- DB2 CLI Stored Procedures
- DB2 API and Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)
- Multi-threaded Applications

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.

This is the only required parameter, and the only one needed for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

If the program contains embedded SQL, indicated by the `.sqc` extension, then the `embprep` script is called to precompile the program, producing a program file with a `.c` extension.

```
#! /bin/ksh
# bldcli script file -- Linux
# Builds a CLI program with Linux C
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
cc -I$DB2PATH/include -c utilcli.c

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

# Link the program.
cc -o $1 $1.o utilcli.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcli

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	
Link options:	
cc	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
utilcli.o	Include the utility object file for error checking.
-L$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 `tbinfo` from the source file `tbinfo.c`, enter:

```
bldcli tbinfo
```

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

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldcli dbusemx
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldcli dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldcli dbusemx database userid password
   ```

The result is an executable file, `dbusemx`.

There are three ways to run this embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:

   ```
   dbusemx
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   dbusemx database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   dbusemx database userid password
   ```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions on more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs.

The script file bldapi in sqllib/samples/cli contains the commands to build a DB2 CLI program with DB2 APIs. This file compiles and links in the utilapi utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the bldcli script. Please see "DB2 CLI Applications" on page 191 for the compile and link options common to both bldapi and bldcli.

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

```
bldapi dbmconn
```

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

```
dbmconn
```

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 utilcli.c
```

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

# Link the program.
cc -o $1 $1.o utilcli.o -shared -L$DB2PATH/lib -Wl,-rpath,$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
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.
Link Options: **cc** Use the compiler as a front end for the linker. **-o $1** Specify the executable. **$1.o** Include the program object file. **utilcli.o** Include the utility object file for error-checking. **-shared** Generate a shared library. **-L$DB2PATH/lib** Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. **-Wl,-rpath,$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 `spserver` from source file `spserver.c`, enter:

```
bldclisp spserver
```

The script file copies the shared library to the `sqllib/function` directory.

Next, catalog the stored procedures by running the spcreate.db2 script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, spserver, you can build the CLI client application spclient that calls the shared library.

You can build spclient by using the script file, bldcli. Refer to "DB2 CLI Applications" on page 191 for details.

To access the shared library, run the sample client application by entering:

spclient *database userid password*

where

database
Is the name of the database to which you want to connect. The name could be sample, or its alias, or another database name.

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, spserver, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

DB2 API and Embedded SQL Applications

The script file bldapp, in sqllib/samples/c, contains the commands to build a DB2 application program.

The first parameter, $1, specifies the name of your source file. This is the only required parameter, and the only one needed for DB2 API programs that do not contain embedded SQL. Building embedded SQL programs requires a

connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind script, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- Linux
# Builds a C application program.
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  cc -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  cc -I$DB2PATH/include -c utilapi.c
fi

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

if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o.
  cc -o $1 $1.o utilemb.o -L$DB2PATH/lib \
    -Wl,-rpath,$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o.
  cc -o $1 $1.o utilapi.o -L$DB2PATH/lib \
    -Wl,-rpath,$DB2PATH/lib -ldb2
fi
```

Compile and Link Options for bldapp
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.
Link options: **cc** Use the compiler as a front end for the linker. **-o $1** Specify the executable. **$1.o** Specify the object file. **utilemb.o** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.o** If a non-embedded SQL program, include the DB2 API utility object file for error checking. **-L$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 DB2 API non-embedded SQL sample program, `client`, from the source file `client.c`, enter:

```
bldapp client
```

The result is an executable file, `client`.

To run the executable file, enter the executable name:

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqc`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldapp updat
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldapp updat database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:
   ```
   updat
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   updat database
   ```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The script file `bldsrv`, 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. Since the stored procedure must be build on the same instance where the database resides, you do not have to specify parameters for user ID and password.

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

```
#! /bin/ksh
# bldsrv script file -- Linux
# Builds a C stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

# Set DB2PATH to where DB2 will be accessed.
```

```
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Precompile and bind the program.
embprep $1 $2

# 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 -Wl,-rpath,$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 bldsrv

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.

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.

-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 `spserver` from source file `spserver.sqc`, if connecting to the `sample` database, enter:

```
bldsrv spserver
```

If connecting to another database, also enter the database name:

```
bldsrv spserver database
```

The script file copies the shared library to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the client application `spclient` that accesses it.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 196 for details.

To call the stored procedures in the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the stored procedure library, spserver, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

User-Defined Functions (UDFs)

The script file bldudf in sqllib/samples/c contains the commands to build a UDF. A UDF does not contain embedded SQL statements. So to build a UDF program, 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
# bldudf script file -- Linux
# Builds a C UDF library
# Usage: bldudf <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard 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 -Wl,-rpath,$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 bldudf

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 bldudf	
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: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-Wl,-rpath,$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.
-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 `udfsrv` from the source file `udfsrv.c`, enter:

```
bldudf udfsrv
```

The script file copies the UDF to the `sqllib/function` directory.

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

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI `udfcli` program from the source file `udfcli.c`, in `sqllib/samples/cli`, using the script file `bldcli`. Refer to "DB2 CLI Applications" on page 191 for details.

You can build the embedded SQL `udfcli` program from the source file `udfcli.sqc`, in `sqllib/samples/c`, using the script file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 196 for details.

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

```
udfcli
```

The calling application calls the ScalarUDF function from the udfsrv library.

Multi-threaded Applications

Multi-threaded applications using Linux C need to be compiled with -D_REENTRANT and linked with -lpthread.

The script file, bldmt, in sqllib/samples/c, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- Linux
# Builds a C multi-threaded embedded SQL program.
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# Precompile and bind the program.
embprep $1 $2 $3 $4

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

# Link the program.
cc -o $1 $1.o -lpthread -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
```

Besides the -D_REENTRANT and -lpthread options, discussed above, and the absence of a utility file linked in, the other compile and link options are the same as those used for the embedded SQL script file, bldapp. For information on these options, see "DB2 API and Embedded SQL Applications" on page 196.

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

```
bldmt thdsrver
```

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

```
thdsrver
```

Linux C++

This section covers the following topics:

- DB2 API and Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions
- Multi-threaded Applications

DB2 API and Embedded SQL Applications

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

The first parameter, `$1`, specifies the name of your source file. This is the only required parameter, and the only one needed for DB2 API programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

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

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  g++ -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  g++ -I$DB2PATH/include -c utilapi.C
fi

# Compile the program.
g++ -I$DB2PATH/include -c $1.C

if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  g++ -o $1 $1.o utilemb.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  g++ -o $1 $1.o utilapi.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
fi
```

Compile and Link Options for bldapp

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 bldapp	
Link Options:	
g++	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
utilemb.o	If an embedded SQL program, include the embedded SQL utility object file for error checking.
utilapi.o	If a non-embedded SQL program, include the DB2 API utility object file for error checking.
-L$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 non-embedded SQL sample program `client` from the source file `client.C`, enter:

```
bldapp client
```

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

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqC`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldapp updat
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

Note: Please see the information for building C++ stored procedures in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file `bldsrv`, 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. Since the stored procedure must be build on the same instance where the database resides, you do not need parameters for user ID and password.

Only the first parameter, the source file name, is required. Database name is 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
# bldsrv script file -- Linux
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2
```

```
# 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 -Wl,-rpath,$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 bldsrv
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.
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: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. **-Wl,-rpath,$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 `spserver` from the source file `spserver.sqC`, if connecting to the `sample` database, enter:

```
bldsrv spserver
```

If connecting to another database, also enter the database name:

```
bldsrv spserver database
```

The script file copies the shared library to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared libary, `spserver`, you can build the client application `spclient` that calls the stored procedures within the shared library.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 205 for details.

To call the stored procedures in the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The stored procedures return the output to the client application.

User-Defined Functions (UDFs)

Note: Please see the information for building C++ UDFs in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file `bldudf` in `sqllib/samples/cpp` contains the commands to build a UDF. A UDF does not contain embedded SQL statements. This means to build a UDF program, you do not need to connect to a database, precompile, and bind the program.

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

```
#! /bin/ksh
# bldudf script file -- Linux
# Builds a C++ UDF library
# Usage: bldudf <prog_name>

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

# Compile the program.
if [[ -f $1".c" ]]
then
  g++ -I$DB2PATH/include -c $1.c

elif [[ -f $1".C" ]]
then
  g++ -I$DB2PATH/include -c $1.C
fi

# Link the program.
g++ -o $1 $1.o -shared -L$DB2PATH/lib -Wl,-rpath,$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 bldudf	
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 bldudf	
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: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
-Wl,-rpath,$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-ldb2	Link with the DB2 library.
-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 `udfsrv` from the source file `udfsrv.c`, enter:

```
bldudf udfsrv
```

The script file copies the UDF to the `sqllib/function` directory.

If necessary, set the file mode for the UDF so the client application can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. You can build the `udfcli` program from the `udfcli.sqC` source file in `sqllib/samples/cpp` using the script file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 205 for details.

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

```
udfcli
```

The calling application calls the `ScalarUDF` function in the `udfsrv` library.

Multi-threaded Applications

Multi-threaded applications using Linux C++ need to be compiled with `-D_REENTRANT` and linked with `-lpthread`.

The script file, `bldmt`, in `sqllib/samples/cpp`, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- Linux
# Builds a C++ multi-threaded embedded SQL program.
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
embprep $1 $2 $3 $4

# Compile the program.
g++ -D_REENTRANT -I$DB2PATH/include -c $1.C

# Link the program.
g++ -lpthread -o $1 $1.o -L$DB2PATH/lib -Wl,-rpath,$DB2PATH/lib -ldb2
```

Besides the `-D_REENTRANT` and `-lpthread` options, discussed above, and the absence of a utility file linked in, the other compile and link options are the same as those used for the embedded SQL script file, `bldapp`. For information on these options, see "DB2 API and Embedded SQL Applications" on page 205.

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

```
bldmt thdsrver
```

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

```
thdsrver
```

Chapter 9. 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.ibm.com/software/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 CLI Applications
- DB2 CLI Stored Procedures
- DB2 API and 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 same command files are in both these directories. They contain commands to accept either a C or C++ source file, depending on the file extension.

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.

This is the only required parameter, and the only one needed for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

If the program contains embedded SQL, indicated by the `.sqc` extension, then the `embprep` command file is called to precompile the program, producing a program file with a `.c` extension.

```
@echo off
rem  bldcli command file - OS/2
rem  Builds a CLI program with IBM VisualAge C++.
rem  Usage: bldcli prog_name [ db_name [ userid password ]]

if exist "%1.sqc" call embprep %1 %2 %3 %4
if exist "%1.sqx" call embprep %1 %2 %3 %4
if "%1" == "" goto error

rem Compile the error-checking utility.
icc -C+ -O- -Ti+ utilcli.c

rem  Compile the program.
if exist "%1.sqx" goto cpp
icc -C+ -O- -Ti+ %1.c
goto link_step
:cpp
icc -C+ -O- -Ti+ %1.cxx

rem  Link the program.
:link_step
ilink /NOFREE /NOI /DEBUG /ST:64000 /PM:VIO %1.obj
   utilcli.obj,%1.exe,NUL,db2cli.lib;

goto exit
:error
echo Usage: bldcli prog_name [ db_name [ userid password ]]
:exit
@echo on
```

Compile and Link Options for bldcli
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
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. **utilcli.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 `tbinfo` from the source file `tbinfo.c`, enter:

```
bldcli tbinfo
```

The result is an executable file, `tbinfo.exe`. You can run the executable file by entering the executable name (without the extension):

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldcli dbusemx
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldcli dbusemx database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldcli dbusemx database userid password
   ```

The result is an executable file, `dbusemx.exe`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name (without the extension):

   ```
   dbusemx
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   dbusemx database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   dbusemx database userid password
   ```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions on more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs.

The command file `bldapi` in `%DB2PATH%\samples\cli` contains the commands to build a DB2 CLI program with DB2 APIs. This file compiles and links in the `utilapi` utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the `bldcli` command file. Please see "DB2 CLI Applications" on page 216 for the compile and link options common to both `bldapi` and `bldcli`.

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

```
bldapi dbmconn
```

The result is an executable file, `dbmconn.exe`. You can run the executable file by entering the executable name (without the extension):

```
dbmconn
```

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 utilcli.c
rem Compile the program.
if exist "%1.cxx" goto cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c
goto link_step
:cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.cxx

:link_step
rem Link the program and produce a DLL.
ilink /NOFREE /MAP /NOI /DEBUG /ST:64000 %1.obj
   utilcli.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

Compile Options:

icc	The IBM VisualAge C++ compiler.
-C+	Perform compile only; no link. The command file has separate compile and link 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
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 sample program `spserver` from the source file `spserver.c`, enter:

```
bldclisp spserver
```

The script file copies the shared library to the server in the path `%DB2PATH%\function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the CLI client application `spclient` that calls the stored procedures in the shared library.

You can build `spclient` by using the command file, `bldcli`. Refer to "DB2 CLI Applications" on page 216 for details.

To access the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

DB2 API and Embedded SQL Applications

The command file `bldapp.cmd`, in `%DB2PATH%\samples\c`, and in `%DB2PATH%\samples\cpp`, contains the commands to build a DB2 application program.

The first parameter, `%1`, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind command file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
@echo off
rem bldapp command file -- OS/2
rem Builds a VisualAge C++ application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

if exist "%1.sqx" goto embedded
if exist "%1.sqc" goto embedded
goto non_embedded

:embedded
rem Precompile and bind the program.
call embprep %1 %2 %3 %4
rem Compile the program.
if exist "%1.cxx" goto cpp_embedded
```

```
icc -c utilemb.c
icc -C+ -O- -Ti+ %1.c
goto link_embedded
:cpp_embedded
icc -c utilemb.cxx
icc -C+ -O- -Ti+ %1.cxx
goto link_embedded

:non_embedded
rem Compile the program.
if exist "%1.cxx" goto cpp
icc -c utilapi.c
icc -C+ -O- -Ti+ %1.c
goto link_non_embedded
:cpp
icc -c utilapi.cxx
icc -C+ -O- -Ti+ %1.cxx
goto link_non_embedded

rem Link the program.
:link_embedded
ilink /NOFREE /NOI /DEBUG /ST:64000 /PM:VIO %1.obj utilemb.obj,,,db2api;
goto exit
:link_non_embedded
ilink /NOFREE /NOI /DEBUG /ST:64000 /PM:VIO %1.obj utilapi.obj,,,db2api;
:exit
@echo on
```

Compile and Link Options for bldapp

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 bldapp
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. **utilemb.obj** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.obj** If not an embedded SQL program, include the DB2 API utility object file for error checking. **db2api** Link with the DB2 library. Refer to your compiler documentation for additional compiler options.

To build the DB2 API non-embedded SQL sample program, `client`, from the source file `client.c`, enter:

```
bldapp client
```

The result is an executable file, `client.exe`.

To run the executable file, enter the executable name (without the extension):

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqc`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat.exe`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name (without the extension):

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The command file `bldsrv`, 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. Since the stored procedure must be build on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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 bldsrv command file -- OS/2
rem Builds a VisualAge C++ stored procedure
rem Usage: bldsrv <prog_name> [ <db_name> ]

rem Precompile and bind the program.
call embprep %1 %2

rem Compile the program.
if exist "%1.cxx" goto cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c
goto link_step
:cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.cxx

:link_step
rem Link the program.
ilink /NOFREE /NOI /DEBUG /ST:64000 %1.obj,%1.dll,,db2api,%1.def;
```

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

Compile and Link Options for bldsrv
Compile Options: **icc** The IBM VisualAge C++ compiler. **-C+** Perform compile only; no link. The command file has separate compile and link 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.
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 sample program `spserver` from source file `spserver.sqc`, if connecting to the `sample` database, enter:

```
bldsrv spserver
```

If connecting to another database, also enter the database name:

```
bldsrv spserver database
```

The command file copies the shared library to the server in the path `%DB2PATH%\function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the client application `spclient` that accesses the shared library.

You can build `spclient` by using the command file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 221 for details.

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

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared libary, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

User-Defined Functions (UDFs)

The command file `bldudf`, 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 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  bldudf command file -- OS/2
rem  Builds a VisualAge C++ user-defined function (UDF)
rem  Usage: bldudf <prog_name>

if "%1" == "" goto error

rem Compile the program.
if exist "%1.cxx" goto cpp
icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.c
goto link_step
:cpp
rem icc -C+ -Ti+ -Ge- -Gm+ -W2 %1.cxx

:link_step
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

goto exit
:error
echo Usage: bldudf prog_name
:exit
@echo on
```

Compile and Link Options for bldudf

Compile Options:

icc	The IBM VisualAge C++ compiler.
-C+	Perform compile only; no link. The command file has separate compile and link 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 bldudf
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 program `udfsrv` from the source file `udfsrv.c`, enter:

```
bldudf udfsrv
```

The script file copies the UDF to the server in the path `%DB2PATH%\function`.

If necessary, set the file mode for the UDF so the client application can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI `udfcli` program from the `udfcli.c` source file in `%DB2PATH%\samples\cli` using the command file `bldcli.cmd`. Refer to "DB2 CLI Applications" on page 216 for details.

You can build the embedded SQL `udfcli` program from the source file `udfcli.sqc`, in `%DB2PATH%\samples\c`, using the command file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 221 for details.

To call the UDF, run the sample calling application by entering the executable name (without the extension):

```
udfcli
```

The calling application calls the `ScalarUDF` function from the `udfsrv` library.

IBM VisualAge C++ for OS/2 Version 4.0

Application building information for the VisualAge C++ version 4 compiler is common to AIX, OS/2 and Windows 32-bit operating systems. See "VisualAge C++ Version 4.0" on page 129 for this information.

IBM VisualAge COBOL for OS/2

This section contains the following topics:

- Using the Compiler
- DB2 API and 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 makes OS/2 allow more file handles on the machine performing 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.

- If you are using the "System/390 host data type support" feature of the IBM VisualAge COBOL compiler, the DB2 include files for your applications are in the following directory:

  ```
  %DB2PATH%\include\cobol_i
  ```

  If you are building DB2 sample programs using the command files provided, the include file path specified in the command 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 VisualAge COBOL compiler, or you are using an earlier version of this compiler, then the DB2 include files for your applications are in the following directory:

  ```
  %DB2PATH%\include\cobol_a
  ```

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

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

Embedded SQL Applications

The command file `bldapp.cmd`, in `%DB2PATH%\samples\cobol`, contains the commands to build a DB2 application program.

The first parameter, `%1`, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind command file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
@echo off
rem bldapp command file -- OS/2
rem Builds a VisualAge COBOL application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

rem If an embedded SQL program, precompile and bind it.
if exist "%1.sqb" goto prepbind
goto compile_step
:prepbind
call embprep %1 %2 %3 %4

:compile_step
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
@echo on
```

Compile and Link Options for bldapp

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`.

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 non-embedded SQL sample program `client` from the source file `client.cbl`, enter:

```
bldapp 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 file extension):

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqb`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat.exe`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name (without the file extension):

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The command file `bldsrv`, 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. Since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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 bldsrv command file -- OS/2
rem Builds a VisualAge COBOL stored procedure
rem Usage: bldsrv <prog_name> [ <db_name> ]
```

```
rem Precompile and bind the program.
call embprep %1 %2

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

Compile and Link Options for bldsrv

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`.

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 sample program `outsrv` from the source file `outsrv.sqb`, connecting to the sample database, enter:

```
bldsrv outsrv
```

If connecting to another database, also include the database name:

```
bldsrv outsrv database
```

The command file uses the module definition file, `outsrv.def`, contained in the same directory as the sample programs, to build the DLL. The command file copies the stored procedure DLL, `outsrv.dll`, on the server in the path `%DB2PATH%\function`.

If necessary, set the file mode for the DLL so the client application can access it.

Once you build the DLL, `outsrv`, you can build the client application `outcli` that accesses the DLL. You can build `outcli` using the command file, `bldapp`. Refer to "Embedded SQL Applications" on page 230 for details.

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

```
outcli database userid password
```

where

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 accesses the DLL, `outsrv`, and executes the stored procedure function of the same name on the server database, and then returns the output to the client application.

Micro Focus COBOL

This section contains the following topics:

- Using the Compiler
- DB2 API and 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 and Embedded SQL Applications

The command file `bldapp`, in `%DB2PATH%\samples\cobol_mf`, contains the commands to build a DB2 application program.

The first parameter, `%1`, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind command file, `embprep`. If no database name is supplied,

the default sample database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
@echo off
rem bldapp command file -- OS/2
rem Builds a Micro Focus COBOL application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

rem If an embedded SQL program, precompile and bind it.
if exist "%1.sqb" goto prepbind
goto compile_step
:prepbind
call embprep %1 %2 %3 %4

:compile_step
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 bldapp
Compile Option: **cobol** The Micro Focus COBOL compiler.
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 non-embedded SQL sample program, client, from the source file client.cbl, enter:

```
bldapp 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 file extension):

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqb`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldapp updat
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldapp updat database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat.exe`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name (without the file extension):
   ```
   updat
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   updat database
   ```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The command file `bldsrv`, 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. Since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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 bldsrv command file -- OS/2
rem Builds a Micro Focus COBOL stored procedure
rem Usage: bldsrv <prog_name> [ <db_name> ]

rem Precompile and bind the program.
call embprep %1 %2

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 the stored procedure to the %DB2PATH%\function directory.
copy %1.dll %DB2PATH%\function
@echo on
```

Compile and Link Options for bldsrv
Compile Option: **cobol** The Micro Focus COBOL compiler.
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 sample program `outsrv` from the source file `outsrv.sqb`, if connecting to the sample database, enter:

```
bldsrv outsrv
```

If connecting to another database, also enter the database name:

```
bldsrv outsrv database
```

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`.

If necessary, set the file mode for the DLL so the client application can access it.

Once you build the DLL, `outsrv`, you can build the client application `outcli` that accesses it. You can build `outcli` using the command file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 235 for details.

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

```
outcli database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the DLL, `outsrv`, and executes the stored procedure function of the same name on the server database. The output is then returned 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 10. Building PTX Applications

This chapter provides detailed information for building applications on PTX with DB2 for NUMA-Q. 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 PTX, visit the Web page at:

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

ptx/C

This section covers the following topics:

- DB2 CLI Applications
- DB2 CLI Applications with DB2 APIs
- DB2 CLI Stored Procedures
- DB2 API and Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-defined Functions (UDFs)
- Multi-threaded Applications

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.

This is the only required parameter, and the only one needed for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

If the program contains embedded SQL, indicated by the `.sqc` extension, then the `embprep` script is called to precompile the program, producing a program file with a `.c` extension.

```
#! /bin/ksh
# bldcli script file -- PTX
# Builds a DB2 CLI program
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
cc -I$DB2PATH/include -c utilcli.c

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

# Link the program.
cc -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcli

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. Compile and link are separate steps.

Compile and Link Options for bldcli	
Link Options:	
cc	Use the compiler as a front end for the linker.
-o $1	Specify the executable program.
$1.o	Include the program object file.
utilcli.o	Include the utility object file for error checking.
-L$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.
Refer to your compiler documentation for additional compiler options.	

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

```
bldcli tbinfo
```

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

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldcli dbusemx
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldcli dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldcli dbusemx database userid password
   ```

The result is an executable file, `dbusemx`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:
   ```
   dbusemx
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   dbusemx database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   dbusemx database userid password
   ```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions on more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs.

The script file `bldapi` in `sqllib/samples/cli` contains the commands to build a DB2 CLI program with DB2 APIs. This file compiles and links in the `utilapi` utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the `bldcli` script. Please see "DB2 CLI Applications" on page 241 for the compile and link options common to both `bldapi` and `bldcli`.

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

```
bldapi dbmconn
```

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

```
dbmconn
```

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 -- PTX
# 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 utilcli.c

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

# Link the program.
cc -G -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldclisp
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. Compile and link are separate steps.

Compile and Link Options for bldclisp	
Link Options:	
cc	Use the compiler as a front end for the linker.
-G	Generate a shared library.
-o $1	Specify the executable.
$1.o	Include the program object file.
utilcli.o	Include the utility object file for error-checking.
-L$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.
-ldb2	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

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

```
bldclisp spserver
```

The script file copies the shared library to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the CLI client application `spclient` that accesses the shared library.

You can build `spclient` by using the script file, `bldcli`. Refer to "DB2 CLI Applications" on page 241 for details.

To access the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, which executes a number of stored procedure functions on the server database, and then returns the output to the client application.

DB2 API and Embedded SQL Applications

The build file, `bldapp`, in `sqllib/samples/c`, contains the commands to build DB2 API and embedded SQL programs.

The first parameter, `$1`, specifies the name of your source file. This is the only required parameter, and the only one needed for DB2 API programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- PTX
# Builds a C application program.
# Usage:  bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# if an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
```

```
  cc -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  cc -I$DB2PATH/include -c utilapi.c
fi

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

if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  cc -o $1 $1.o utilemb.o -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  cc -o $1 $1.o utilapi.o -L$DB2PATH/lib -ldb2
fi
```

Compile and Link Options for bldapp
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. Compile and link are separate steps.
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. **-ldb2** Link with the DB2 library. Refer to your compiler documentation for additional compiler options.

To build the non-embedded SQL sample program, `client`, from the source file `client.c`, enter:

```
bldapp client
```

The result is an executable file, `client`.

To run the executable file, enter the executable name:

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqc`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The script file, `bldsrv`, 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. Since the stored procedure must be build on the same instance where the database resides, you do not have to specify parameters for user ID and password.

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

```
#! /bin/ksh
# bldsrv script file -- PTX
# Builds a C stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# 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 -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldsrv
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. Compile and link are separate steps.
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. **-ldb2** Link with the DB2 library. Refer to your compiler documentation for additional compiler options.

To build the sample program `spserver` from the source file `spserver.sqc`, if connecting to the `sample` database, enter:

```
bldsrv spserver
```

If connecting to another database, also enter the database name:

```
bldsrv spserver database
```

The script file copies the shared library to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the client application `spclient` that accesses it.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 247 for details.

To call the stored procedures in the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, spserver, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

User-Defined Functions (UDFs)

The script file, bldudf, 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 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 for the shared library name.

```
#! /bin/ksh
# bldudf script file -- PTX
# Builds a C user-defined function library
# Usage: bldudf <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 -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldudf	
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. Compile and link are separate steps.

Compile and Link Options for bldudf	
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.
-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 `udfsrv` from the source file `udfsrv.c`, enter:

```
bldudf udfsrv
```

The script file copies the UDF to the `sqllib/function` directory.

If necessary, set the file mode for the UDF so the client application can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI `udfcli` program from the source file `udfcli.c`, in `sqllib/samples/cli`, using the script file `bldcli`. Refer to "DB2 CLI Applications" on page 241 for details.

You can build the embedded SQL `udfcli` program from the source file `udfcli.sqc`, in `sqllib/samples/c`, using the script file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 247 for details.

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

```
udfcli
```

The calling application calls the `ScalarUDF` function from the `udfsrv` library.

Multi-threaded Applications

Multi-threaded applications using ptx/C need to be compiled and linked with `-Kthread`.

The script file, `bldmt`, in `sqllib/samples/c`, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- PTX
# Builds a C multi-threaded embedded SQL program.
# Usage:  bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# Precompile and bind the program.
embprep $1 $2 $3 $4

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

# Link the program.
cc -Kthread -o $1 $1.o -L$DB2PATH/lib -ldb2
```

Besides the `-Kthread` option, discussed above, and the absence of a utility file linked in, the other compile and link options are the same as those used for the embedded SQL script file, `bldapp`. For information on these options, see "DB2 API and Embedded SQL Applications" on page 247.

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

```
bldmt thdsrver
```

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

```
thdsrver
```

ptx/C++

This section covers the following topics:

- DB2 API and Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-defined Functions (UDFs)
- Multi-threaded Applications

DB2 API and Embedded SQL Applications

The build file, `bldapp`, in `sqllib/samples/cpp`, contains the commands to build DB2 API and embedded SQL programs.

The first parameter, `$1`, specifies the name of your source file. This is the only required parameter, and the only one needed for DB2 API programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- PTX
# Builds a C++ application program
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# if an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  c++ -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  c++ -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c utilapi.C
fi

# Compile the program.
c++ -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c $1.C

if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  c++ -o $1 $1.o utilemb.o -L$DB2PATH/lib -ldb2 -lseq
else
  # Link the program with utilapi.o
  c++ -o $1 $1.o utilapi.o -L$DB2PATH/lib -ldb2 -lseq
fi
```

Compile and Link Options for bldapp

Compile Options:

c++ The C++ compiler.

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

-D_RWSTD_COMPILE_INSTANTIATE=0
Do not instantiate the rogue wave classes.

-c Perform compile only, no link. Compile and link are separate steps.

Compile and Link Options for bldapp	
Link Options:	
c++	Use the compiler as a front end for the linker.
-o $1	Specify the executable.
$1.o	Include the program object file.
utilemb.o	If an embedded SQL program, include the embedded SQL utility object file for error checking.
utilapi.o	If a non-embedded SQL program, include the DB2 API utility object file for error checking.
-L$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.
-ldb2	Link with the DB2 library.
-lseq	Link with the Sequent library.
Refer to your compiler documentation for additional compiler options.	

To build the DB2 API non-embedded SQL sample program, `client`, from the source file `client.C`, enter:

```
bldapp client
```

The result is an executable file, `client`.

To run the executable file, enter the executable name:

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqC`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, add the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, add the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, updat.

There are three ways to run this embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

Note: Please see the information for building C++ stored procedures in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file, bldsrv, 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. This is the only required parameter. Building embedded SQL programs requires a connection to the database so an additional optional parameter, $2, specifies the name of the database to which you want to connect. If no database name is supplied, the default sample database is used. Since the stored procedure must be built on the same instance where the database resides, no additional parameters for user ID and password are needed. The script file, bldsrv, passes the parameters to the precompile and bind file, embprep.

The source file name, $1, is used for the shared library name.

```
#! /bin/ksh
# bldsrv script file -- PTX
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# Compile the program. First ensure it is coded with extern "C".
c++ -KPIC -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c $1.C
```

```
# Link the program and create a shared library.
c++ -G -o $1 $1.o -L$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1.so $DB2PATH/function/$1
```

Compile and Link Options for bldsrv
Compile Options: **c++** 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` **-D_RWSTD_COMPILE_INSTANTIATE=0** Do not instantiate the rogue wave classes. **-c** Perform compile only, no link. This book assumes that compile and link are separate steps.
Link Options: **c++** 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. **-ldb2** Link with the DB2 library. Refer to your compiler documentation for additional compiler options.

To build the sample program `spserver` from the source file `spserver.sqC`, if connecting to the `sample` database, enter:

```
bldsrv spserver
```

If connecting to another database, also enter the database name:

```
bldsrv spserver database
```

The script file copies the shared library to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared libary, `spserver`, you can build the client application `spclient` that calls the stored procedures within the shared library.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 255 for details.

To call the stored procedures in the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The stored procedures return the output to the client application.

User-Defined Functions (UDFs)

Note: Please see the information for building C++ UDFs in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file, bldudf, 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 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 for the shared library name.

```
#! /bin/ksh
# bldudf script file -- PTX
# Builds a C++ user-defined function library
# Usage: bldudf <prog_name>

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

# Compile the program.
c++ -KPIC -I$DB2PATH/include  -D_RWSTD_COMPILE_INSTANTIATE=0 -c $1.c

# Link the program and create a shared library.
c++ -G -o $1 $1.o -L$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1.so $DB2PATH/function/$1
```

Compile and Link Options for bldudf

Compile Options:

c++ 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.

-D_RWSTD_COMPILE_INSTANTIATE=0
Do not instantiate the rogue wave classes.

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

Compile and Link Options for bldudf	
Link Options:	
c++	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.
-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 `udfsrv` from the source file `udfsrv.c`, enter:

```
bldudf udfsrv
```

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 client application can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. You can build the `udfcli` program from the `udfcli.sqC` source file in `sqllib/samples/cpp` using the script file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 255 for details.

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

```
udfcli
```

The calling application calls the `ScalarUDF` function in the `udfsrv` library.

Multi-threaded Applications

Multi-threaded applications using ptx/C++ need to be compiled and linked in with `-Kthread`.

The script file, `bldmt`, in `sqllib/samples/cpp`, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- PTX
# Builds a C++ multi-threaded embedded SQL program
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# Precompile and bind the program.
embprep $1 $2 $3 $4

# Compile the program.
c++ -Kthread -I$DB2PATH/include -D_RWSTD_COMPILE_INSTANTIATE=0 -c $1.C

# Link the program.
c++ -Kthread -o $1 $1.o -L$DB2PATH/lib -ldb2 -lseq
```

Besides the -Kthread option, discussed above, and the absence of a utility file linked in, the other compile and link options are the same as those used for the embedded SQL script file, bldapp. For information on these options, see "DB2 API and Embedded SQL Applications" on page 255.

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

```
bldmt thdsrver
```

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

```
thdsrver
```

Chapter 11. Building Silicon Graphics IRIX Applications

This chapter provides detailed information for building DB2 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.ibm.com/software/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 42 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 `64` (the 64 object type). DB2 does not yet support `64`, but does support the `o32` and `n32` object types.

The operating system 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 is 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.

Note: To build `n32` object type applications with a build file documented in this chapter, uncomment the indicated command.

MIPSpro C

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

- DB2 CLI
- DB2 APIs
- Embedded SQL

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.

This is the only required parameter, and the only one needed for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

If the program contains embedded SQL, indicated by the .sqc extension, then the embprep script is called to precompile the program, producing a program file with a .c extension.

```
#! /bin/ksh
# bldcli script file -- Silicon Graphics IRIX
# Builds a CLI program with MIPSpro C.
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
fi

# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32

if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
fi

# Compile the error-checking utility.
cc $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilcli.c

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

# Link the program.
cc $IRIX_OBJECT_MODE -o $1 $1.o utilcli.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH
   -lm -ldb2
```

Compile and Link Options for bldcli
Compile Options: **cc** Use the C compiler. **$IRIX_OBJECT_MODE** Contains "-n32" if 'IRIX_OBJECT_MODE=-n32' is uncommented; otherwise, it contains no value. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include` **-c** Perform compile only; no link. This book assumes that compile and link are separate steps.
Link Options: **cc** Use the compiler as a front end for the linker. **$IRIX_OBJECT_MODE** Contains "-n32" if 'IRIX_OBJECT_MODE=-n32' is uncommented; otherwise, it contains no value. **-o $1** Specify the executable. **$1.o** Include the program object file. **utilcli.o** Include the utility object file for error checking. **-L$DB2_LIBPATH** Specify the location of the DB2 static and shared libraries at link-time. For o32 object type, it points to: `$DB2PATH/lib`; For n32 object type, it points to: `$DB2PATH/lib32`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. **-rpath $DB2_LIBPATH** Specify the location of the DB2 shared libraries at run-time. For o32 object type, it points to: `$DB2PATH/lib`; For n32 object type, it points to: `$DB2PATH/lib32`. **-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 `tbinfo` from the source file `tbinfo.c`, enter:

```
bldcli tbinfo
```

The result is an executable file `tbinfo`. You can run the executable file by entering the executable name, database name, and user ID and password for the instance where the database is located:

```
tbinfo database userid password
```

Building and Running Embedded SQL Applications

To build `dbusemx` from the source file `dbusemx.sqc`, include parameters for the database, and the user ID and password for the instance where the database is located:

```
bldcli dbusemx database userid password
```

The result is an executable file, `dbusemx`. To run an embedded SQL application, enter the executable name, database name, and user ID and password for the instance where the database is located:

```
dbusemx database userid password
```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions on more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs.

The script file `bldapi` in `sqllib/samples/cli` contains the commands to build a DB2 CLI program with DB2 APIs. This file compiles and links in the `utilapi` utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the `bldcli` script. Please see "DB2 CLI Applications" on page 266 for the compile and link options common to both `bldapi` and `bldcli`.

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

```
bldapi dbmconn
```

The result is an executable file `dbmconn`. You can run the executable file by entering the executable name, database name, and user ID and password for the instance where the database is located:

```
dbmconn database userid 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 spserver 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 spserver, you can build the client application that calls the stored procedure, spclient, from the source file spclient.c, by using the script file bldcli. Refer to “DB2 CLI Applications” on page 266 for details.

You can call the stored procedure by entering the executable name, database name, and user ID and password for the instance where the database is located:

```
spclient database userid password
```

The client application accesses the shared library, spserver, and executes a number of stored procedure functions on the server database. The output is returned 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, udfsrv, 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 udfsrv, you can build the DB2 CLI client application, udfcli, that calls it, from the udfcli.c source file in sqllib/samples/cli, using the DB2 CLI script file bldcli. Refer to “DB2 CLI Applications” on page 266 for details.

To call the UDF program, run the calling application by entering the executable name, database name, and user ID and password for the instance where the database is located:

```
udfcli database userid password
```

The calling application calls the ScalarUDF function from the udfsrv library.

DB2 API and Embedded SQL Applications

The script file bldapp, in sqllib/samples/c, contains the commands to build a DB2 application program. The first parameter, $1, specifies the name of your source file. This is the only required parameter, and the only one needed for DB2 API programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, bldapp passes the parameters to the precompile and bind file, embprep. If no database name is supplied, the default sample database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- Silicon Graphics IRIX
# Builds a C application program.
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32

if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
fi

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  cc $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  cc $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilapi.c
fi

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

if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  cc $IRIX_OBJECT_MODE -o $1 $1.o utilemb.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH
     -lm -ldb2
else
  # Link the program with utilapi.o
  cc $IRIX_OBJECT_MODE -o $1 $1.o utilapi.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH
     -lm -ldb2
fi
```

Compile and Link Options for bldapp
Compile Options: **cc** Use the C compiler. **$IRIX_OBJECT_MODE** Contains "-n32" if 'IRIX_OBJECT_MODE=-n32' is uncommented; otherwise, it contains no value. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include` **-c** Perform compile only; no link. This book assumes that compile and link are separate steps.
Link Options: **cc** Use the compiler as a front end for the linker. **$IRIX_OBJECT_MODE** Contains "-n32" if 'IRIX_OBJECT_MODE=-n32' is uncommented; otherwise, it contains no value. **-o $1** Specify the executable. **$1.o** Include the program object file. **utilemb.o** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.o** If a non-embedded SQL program, include the DB2 API utility object file for error checking. **-L$DB2_LIBPATH** Specify the location of the DB2 static and shared libraries at link-time. For o32 object type, it points to: `$DB2PATH/lib`; For n32 object type, it points to: `$DB2PATH/lib32`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. **-rpath $DB2_LIBPATH** Specify the location of the DB2 shared libraries at run-time. For o32 object type, it points to: `$DB2PATH/lib`; For n32 object type, it points to: `$DB2PATH/lib32`. **-lm** Link with the math library. **-ldb2** Link with the DB2 library. Refer to your compiler documentation for additional compiler options.

To build the DB2 API non-embedded SQL sample program, `client`, from the source file `client.c`, enter:

```
bldapp client
```

The result is an executable file, `client`.

To run the executable file, enter the executable name, database name, and user ID and password for the instance where the database is located:

```
client database userid password
```

Building and Running Embedded SQL Applications

To build the sample program `updat` from the source file `updat.sqc`, include parameters for the database, and the user ID and password for the instance where the database is located:

```
bldapp updat database userid password
```

The result is an executable file, `updat`. To run the executable file against the `sample` database, enter the executable name, database name, and user ID and password for the instance where the database is located:

```
updat database userid 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 `spserver` 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 `spserver`, you can build the client application that calls the stored procedure. You can build `spclient` from the source file `spclient.sqc`, by using the script file `bldapp`. Refer to “DB2 API and Embedded SQL Applications” on page 270 for details.

To call the stored procedure, run the client application by entering the executable name, database name, and user ID and password for the instance where the database is located:

```
spclient database userid password
```

The client application accesses the stored procedure library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned 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, `udfsrv`, 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 `udfsrv`, you can build the embedded SQL client application, `udfcli`, that calls it, from the `udfcli.sqc` source file in `sqllib/samples/c` using the script file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 270 for details.

To call the UDF program, run the calling application by entering the executable name, database name, and user ID and password for the instance where the database is located:

```
udfcli database userid password
```

The calling application calls the ScalarUDF function from the `udfsrv` 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` and `-lpthread` link options.

The script file `bldmt`, in `sqllib/samples/c`, contains the commands to build an embedded SQL multi-threaded 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.

```
#! /bin/ksh
# bldmt script file -- Silicon Graphics IRIX
# Builds a C multi-threaded embedded SQL program
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32

if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
fi
```

```
# Precompile and bind the program.
embprep $1 $2 $3 $4

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

# Link the program.
cc $IRIX_OBJECT_MODE -o $1 $1.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH
   -lm -ldb2_th -lpthread
```

Besides the `-ldb2_th` and `-lpthread` link options, discussed above, and the absence of a utility file being linked in, the other compile and link options are the same as those used for the embedded SQL script file, `bldapp`. For information on these options, see "DB2 API and Embedded SQL Applications" on page 270.

To build the sample program, `thdsrver`, from the source file `thdsrver.sqc`, include parameters for the database, and the user ID and password for the instance where the database is located:

```
bldmt thdsrver database userid password
```

The result is an executable file, `thdsrver`.

To run the executable file, enter the executable name, database name, and user ID and password for the instance where the database is located:

```
thdsrver database userid password
```

MIPSpro C++

This section includes the following topics:

- DB2 API and Embedded SQL Applications
- Multi-threaded Applications

DB2 API and Embedded SQL Applications

The script file `bldapp`, in `sqllib/samples/cpp`, contains the commands to build a DB2 application program.

The first parameter, `$1`, specifies the name of your source file. This is the only required parameter for non-embedded SQL applications. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`.

```
#! /bin/ksh
# bldapp script file -- Silicon Graphics IRIX
# Builds a C++ application program
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32

if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
fi

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  CC $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.c error-checking utility.
  CC $IRIX_OBJECT_MODE -I$DB2PATH/include -c utilapi.C
fi

# Compile the program.
CC $IRIX_OBJECT_MODE -I$DB2PATH/include -c $1.C

if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  CC $IRIX_OBJECT_MODE -o $1 $1.o utilemb.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH
     -lm -ldb2
else
  # Link the program with utilapi.o
  CC $IRIX_OBJECT_MODE -o $1 $1.o utilapi.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH
     -lm -ldb2
fi
```

Compile and Link Options for bldapp
Compile Options: **CC** Use the C++ compiler. **$IRIX_OBJECT_MODE** Contains "-n32" if 'IRIX_OBJECT_MODE=-n32' is uncommented; otherwise, it contains no value. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include` **-c** Perform compile only; no link. This script has separate compile and link steps.
Link Options: **CC** Use the compiler as a front end for the linker. **$IRIX_OBJECT_MODE** Contains "-n32" if 'IRIX_OBJECT_MODE=-n32' is uncommented; otherwise, it contains no value. **-o $1** Specify the executable. **$1.o** Include the program object file. **utilemb.o** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.o** If a non-embedded SQL program, include the DB2 API utility object file for error checking. **-L$DB2_LIBPATH** Specify the location of the DB2 static and shared libraries at link-time. For o32 object type, it points to: `$DB2PATH/lib`; For n32 object type, it points to: `$DB2PATH/lib32`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed. **-rpath $DB2_LIBPATH** Specify the location of the DB2 shared libraries at run-time. For o32 object type, it points to: `$DB2PATH/lib`; For n32 object type, it points to: `$DB2PATH/lib32`. **-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, include parameters for the database, and the user ID and password for the instance where the database is located:

```
bldapp updat database userid password
```

The result is an executable file, updat, To run the executable file, enter the executable name, database name, and user ID and password for the instance where the database is located:

```
updat database userid 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 spserver 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 spserver, you can build the client application that calls the stored procedure. You can build spclient from the source file spclient.sqC, by using the script file bldapp. Refer to "DB2 API and Embedded SQL Applications" on page 275 for details.

To call the stored procedure, run the client application by entering the executable name, database name, and user ID and password for the instance where the database is located:

```
spclient database userid password
```

The client application accesses the shared library, spserver, and executes a number of stored procedure functions on the server database. The stored procedures return the output 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, udfsrv, 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 udfsrv, you can build the embedded SQL client application, udfcli, that calls it, from the udfcli.sqC source file in sqllib/samples/cpp using the script file bldapp. Refer to "DB2 API and Embedded SQL Applications" on page 275 for details.

To call the UDF program, run the calling application by entering the executable name, database name, and user ID and password for the instance where the database is located:

```
udfcli database userid password
```

The calling application calls the ScalarUDF function from the udfsrv 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 and -lpthread link options.

The script file bldmt, in sqllib/samples/cpp, contains the commands to build an embedded SQL multi-threaded 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.

```
#! /bin/ksh
# bldmt script file -- Silicon Graphics IRIX
# Builds a C++ multi-threaded embedded SQL program
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# To compile with n32 object support, uncomment the following line.
# IRIX_OBJECT_MODE=-n32

if [ "$IRIX_OBJECT_MODE" = "-n32" ] ; then
  # Link with db2 n32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib32
else
  # Link with db2 o32 object type libraries.
  DB2_LIBPATH=$DB2PATH/lib
fi

# Precompile and bind the program.
embprep $1 $2 $3 $4

# Compile the program.
CC $IRIX_OBJECT_MODE -I$DB2PATH/include -c $1.C

# Link the program.
CC $IRIX_OBJECT_MODE -o $1 $1.o -L$DB2_LIBPATH -rpath $DB2_LIBPATH
   -lm -ldb2_th -lpthread
```

Besides the -ldb2_th and -lpthread link options, discussed above, and the absence of a utility file linked in, the other compile and link options are the

same as those used for the embedded SQL script file, bldapp. For information on these options, see "DB2 API and Embedded SQL Applications" on page 275.

To build the sample program, thdsrver, from the source file thdsrver.sqC, include parameters for the database, and the user ID and password for the instance where the database is located:

```
bldmt thdsrver database userid password
```

The result is an executable file, thdsrver.

To run the executable file against the sample database, enter the executable name, database name, and user ID and password for the instance where the database is located:

```
thdsrver database userid password
```

Chapter 12. Building Solaris Applications

This chapter provides detailed information for building applications in the Solaris operating environment. 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 the Solaris operating environment, visit the Web page at:

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

Notes:

1. The `-mt` multi-threaded option is used in the link steps of the DB2 build files and makefiles due to the way threads are implemented in the Solaris operating environment. This comes with a slight performance cost. If optimum performance is a consideration, you could try to link your applications without this option, and with the non-threaded `libdb2.so` library. However, 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
   ```

   Or, the application may hang and not give any error message.
2. To build 64-bit applications with the build files documented in this chapter, uncomment the indicated command.

SPARCompiler C

This section includes the following topics:

- DB2 CLI Applications
- DB2 CLI Applications with DB2 APIs
- DB2 CLI Stored Procedures
- DB2 API and Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)
- Multi-threaded Applications

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.

This is the only required parameter for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

If the program contains embedded SQL, indicated by the `.sqc` extension, then the `embprep` script is called to precompile the program, producing a program file with a `.c` extension.

```
#! /bin/ksh
# bldcli script file -- Solaris
# Builds a DB2 CLI program.
# Usage: bldcli <prog_name> [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
fi

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
```

```
fi

# Compile the error-checking utility.
cc $CFLAGS_64 -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc $CFLAGS_64 -I$DB2PATH/include  -c $1.c

# Link the program.
cc $CFLAGS_64 -o $1 $1.o utilcli.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
```

Compile and Link Options for bldcli

Compile Options:

cc Use the C compiler.

$CFLAGS_64
Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.

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

-c Perform compile only; no link. This script has separate compile and link steps.

Link Options:

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

$CFLAGS_64
Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.

-o $1 Specify the executable program.

$1.o Include the program object file.

utilcli.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 tbinfo from the source file tbinfo.c, enter:

```
bldcli tbinfo
```

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

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, dbusemx, from the source file dbusemx.sqc:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldcli dbusemx
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldcli dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldcli dbusemx database userid password
   ```

The result is an executable file, dbusemx.

There are three ways to run the embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:
   ```
   dbusemx
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   dbusemx database
   ```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   dbusemx database userid password
   ```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions on more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs.

The script file bldapi in sqllib/samples/cli contains the commands to build a DB2 CLI program with DB2 APIs. This file compiles and links in the utilapi utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the bldcli script.

Please see "DB2 CLI Applications" on page 282 for the compile and link options common to both `bldapi` and `bldcli`.

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

```
bldapi dbmconn
```

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

```
dbmconn
```

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

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi

# Compile the error-checking utility.
cc $CFLAGS_64 -Kpic -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.c

# Link the program.
cc $CFLAGS_64 -G -o $1 $1.o utilcli.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldclisp
Compile options: **cc** The C compiler. **$CFLAGS_64** Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-Kpic** Generate position-independent code for shared libraries. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`. **-c** Perform compile only; no link. This book assumes that compile and link are separate steps.
Link Options: **cc** Use the compiler as a front end for the linker. **$CFLAGS_64** Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-o $1** Specify the executable. **$1.o** Include the program object file. **utilcli.o** Include the utility object file for error-checking. **-L$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 `spserver` from source file `spserver.c`, enter:

```
bldclisp spserver
```

The script file copies the stored procedure to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the spcreate.db2 script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure spserver, you can build the CLI client application spclient that calls the stored procedure.

You can build spclient by using the script file, bldcli. Refer to "DB2 CLI Applications" on page 282 for details.

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

spclient *database userid password*

where

database
Is the name of the database to which you want to connect. The name could be sample, or its alias, or another database name.

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the stored procedure library, spserver, which executes a number of stored procedure functions on the server database. The output is returned to the client application.

DB2 API and Embedded SQL Applications

The script file, bldapp, in sqllib/samples/c, contains the commands to build a DB2 application program.

The first parameter, $1, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three

optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- Solaris
# Builds a C application program.
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqc" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.c error-checking utility.
  cc $CFLAGS_64 -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  cc $CFLAGS_64 -I$DB2PATH/include -c utilapi.c
fi

# Compile the program.
cc $CFLAGS_64 -I$DB2PATH/include -c $1.c

if [[ -f $1".sqc" ]]
then
  # Link the program with utilemb.o
  cc $CFLAGS_64 -o $1 $1.o utilemb.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  cc $CFLAGS_64 -o $1 $1.o utilapi.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2
fi
```

Compile and Link Options for bldapp
Compile Options: **cc** The C compiler. **$CFLAGS_64** Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include` **-c** Perform compile only; no link. This script has separate compile and link steps.
Link Options: **cc** Use the compiler as a front end for the linker. **$CFLAGS_64** Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-o $1** Specify the executable. **$1.o** Include the program object file. **utilemb.o** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.o** If not an embedded SQL program, include the DB2 API utility object file for error checking. **-L$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 DB2 API non-embedded SQL sample program, `client`, from the source file `client.c`, enter:

```
bldapp client
```

The result is an executable file, `client`.

To run the executable file, enter the executable name:

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqc`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The script file, `bldsrv`, 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. Since the stored procedure must be build on the same instance where the database resides, there are no parameters for user ID and password.

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

```
#! /bin/ksh
# bldsrv script file -- Solaris
# Builds a C stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi

# Compile the program.
cc $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.c

# Link the program and create a shared library
cc $CFLAGS_64 -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldsrv

Compile Options:

cc The C compiler.

$CFLAGS_64
Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.

-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 script has separate compile and link steps.

Compile and Link Options for bldsrv	
Link Options:	
cc	Use the compiler as a front end for the linker.
$CFLAGS_64	Contains ″-xarch=v9″ value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-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 `spserver` from the source file `spserver.sqc`, if connecting to the `sample` database, enter:

```
bldsrv spserver
```

If connecting to another database, also enter the database name:

```
bldsrv spserver database
```

The script file copies the stored procedure to the `sqllib/function` directory.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure `spserver`, you can build the client application `spclient` that calls the stored procedure.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 287 for details.

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

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the stored procedure library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

User-Defined Functions (UDFs)

The script file, `bldudf`, 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
# bldudf script file -- Solaris
# Builds a C UDF library
# Usage: bldudf <prog_name>

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

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
```

```
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi

# Compile the program.
cc $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.c

# Link the program and create a shared library.
cc $CFLAGS_64 -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldudf
Compile Options: **cc** The C compiler. **$CFLAGS_64** Contains ″-xarch=v9″ value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-Kpic** Generate position-independent code for shared libraries. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`. **-c** Perform compile only; no link. This script has separate compile and link steps.

Compile and Link Options for bldudf	
Link Options:	
cc	Use the compiler as a front end for the linker.
$CFLAGS_64	Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-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 `udfsrv` from the source file `udfsrv.c`, enter:

```
bldudf udfsrv
```

The script file copies the UDF to the `sqllib/function` directory.

If necessary, set the file mode for the UDF so the client application can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. DB2 CLI and embedded SQL versions of this program are provided.

You can build the DB2 CLI `udfcli` program from the source file `udfcli.c`, in `sqllib/samples/cli`, using the script file `bldcli`. Refer to "DB2 CLI Applications" on page 282 for details.

You can build the embedded SQL `udfcli` program from the source file `udfcli.sqc`, in `sqllib/samples/c`, using the script file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 287 for details.

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

```
udfcli
```

The calling application calls the ScalarUDF function from the udfsrv 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, bldmt, in sqllib/samples/c, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- Solaris
# Builds a C multi-threaded embedded SQL program.
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# Precompile and bind the program.
embprep $1 $2 $3 $4

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi

# Compile the program.
cc $CFLAGS_64 -mt -D_POSIX_PTHREAD_SEMANTICS -I$DB2PATH/include -c $1.c

# Link the program.
cc $CFLAGS_64  -mt -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -lpthread
```

Besides the -mt, -D_POSIX_PTHREAD_SEMANTICS, and -lpthread options, discussed above, and the absence of a utility file linked in, the other compile and link options are the same as those used for the embedded SQL script file, bldapp. For information on these options, see "DB2 API and Embedded SQL Applications" on page 287.

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

```
bldmt 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 and Embedded SQL Applications
- Embedded SQL Stored Procedures
- User-Defined Functions (UDFs)
- Multi-threaded Applications

DB2 API and Embedded SQL Applications

The script file, bldapp, in sqllib/samples/cpp, contains the commands to build a DB2 application program.

The first parameter, $1, specifies the name of your source file. This is the only required parameter for programs not containing embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, bldapp passes the parameters to the precompile and bind file, embprep. If no database name is supplied, the default sample database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

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

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

```
# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqC" ]]
then
  embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  CC $CFLAGS_64 -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  CC $CFLAGS_64 -I$DB2PATH/include -c utilapi.C
fi

# Compile the program.
CC $CFLAGS_64 -I$DB2PATH/include -c $1.C

if [[ -f $1".sqC" ]]
then
  # Link the program with utilemb.o
  CC $CFLAGS_64 -o $1 $1.o utilemb.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -mt
else
  # Link the program with utilapi.o
  CC $CFLAGS_64 -o $1 $1.o utilapi.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -mt
fi
```

Compile and Link Options for bldapp

Compile Options:

CC The C++ compiler.

$CFLAGS_64
Contains ″-xarch=v9″ value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.

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

-c Perform compile only; no link. This script has separate compile and link steps.

Compile and Link Options for bldapp	
Link options:	
CC	Use the compiler as a front end for the linker.
$CFLAGS_64	Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-o $1	Specify the executable.
$1.o	Include the program object file.
utilemb.o	If an embedded SQL program, include the embedded SQL utility object file for error checking.
utilapi.o	If a non-embedded SQL program, include the DB2 API utility object file for error checking.
-L$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 non-embedded SQL DB2 API sample program `client` from the source file `client.C`, enter:

```
bldapp client
```

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

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqC`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

Note: Please see the information for building C++ stored procedures in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file `bldsrv`, 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. Since the stored procedure must be build on the same instance where the database resides, you do not need parameters for user ID and password.

Only the first parameter, the source file name, is required. Database name is 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
# bldsrv script file -- Solaris
# Builds a C++ stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

# Set DB2PATH to where DB2 will be accessed.
```

```
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Precompile and bind the program.
embprep $1 $2

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi

# Compile the program.
CC $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.C

# Link the program and create a shared library
CC $CFLAGS_64 -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -mt

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldsrv

Compile Options:

CC The C++ compiler.

$CFLAGS_64
Contains ″-xarch=v9″ value if ′BUILD_64BIT=true′ is uncommented; otherwise, it contains no value.

-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 script has separate compile and link steps.

Compile and Link Options for bldsrv	
Link Options:	
CC	Use the compiler as a front end for the linker.
$CFLAGS_64	Contains ″-xarch=v9″ value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-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 `spserver` from the source file `spserver.sqC`, if connecting to the `sample` database, enter:

```
bldsrv spserver
```

If connecting to another database, also enter the database name:

```
bldsrv spserver database
```

The script file copies the shared library to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared libary, `spserver`, you can build the client application `spclient` that calls the stored procedures within the shared library.

You can build `spclient` by using the script file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 297 for details.

To call the stored procedures in the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The stored procedures return the output to the client application.

User-Defined Functions (UDFs)

Note: Please see the information for building C++ UDFs in "C++ Considerations for UDFs and Stored Procedures" on page 60.

The script file, `bldudf`, 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
# bldudf script file -- Solaris
# Builds a C++ UDF library
# Usage: bldudf <prog_name>

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

```
DB2PATH=$HOME/sqllib

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi

# Compile the program.
if [[ -f $1".c" ]]
then
  CC $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.c
elif [[ -f $1".C" ]]
then
  CC $CFLAGS_64 -Kpic -I$DB2PATH/include -c $1.C
fi

# Link the program and create a shared library.
CC $CFLAGS_64 -G -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -ldb2apie

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Compile and Link Options for bldudf

Compile Options:

CC The C++ compiler.

$CFLAGS_64
Contains ″-xarch=v9″ value if ’BUILD_64BIT=true’ is uncommented; otherwise, it contains no value.

-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 script has separate compile and link steps.

Compile and Link Options for bldudf	
Link Options:	
CC	Use the compiler as a front end for the linker.
$CFLAGS_64	Contains "-xarch=v9" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-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 `udfsrv` from the source file `udfsrv.c`, enter:

```
bldudf udfsrv
```

The script file copies the UDF to the `sqllib/function` directory.

If necessary, set the file mode for the UDF so the client application can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. You can build the `udfcli` program from the `udfcli.sqC` source file in `sqllib/samples/cpp` using the script file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 297 for details.

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

```
udfcli
```

The calling application calls the `ScalarUDF` function in the `udfsrv` 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, `bldmt`, in `sqllib/samples/cpp`, contains the commands to build an embedded SQL multi-threaded 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
# bldmt script file -- Solaris
# Builds a C++ multi-threaded embedded SQL program
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

# Precompile and bind the program.
embprep $1 $2 $3 $4

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAGS_64=-xarch=v9
else
  CFLAGS_64=
fi

# Compile the program.
CC $CFLAGS_64 -mt -D_POSIX_PTHREAD_SEMANTICS -I$DB2PATH/include -c $1.C

# Link the program.
CC $CFLAGS_64 -mt -o $1 $1.o -L$DB2PATH/lib -R$DB2PATH/lib -ldb2 -lpthread
```

Besides the `-mt`, `-D_POSIX_PTHREAD_SEMANTICS`, and `-lpthread` options, discussed above, and the absence of a utility file linked in, the other compile and link options are the same as those used for the embedded SQL script file, `bldapp`. For information on these options, see "DB2 API and Embedded SQL Applications" on page 297.

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

```
bldmt 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 and DB2 Embedded 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, 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/V7.1/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 and Embedded SQL Applications

The script file bldapp, in sqllib/samples/cobol_mf, contains the commands to build a DB2 application program.

The first parameter, $1, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building

embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldapp` passes the parameters to the precompile and bind file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
#! /bin/ksh
# bldapp script file -- Solaris
# Builds a Micro Focus COBOL application program
# Usage: bldapp [ <db_name> [ <userid> <password> ]]

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

# If an embedded SQL program, precompile and bind it.
if [[ -f $1".sqb" ]]
then
  embprep $1 $2 $3 $4
fi

# 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 bldapp	
Compile Options:	
cob	The Micro Focus COBOL compiler.
-cx	Compile to object module.

Compile and Link Options for bldapp	
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 non-embedded SQL sample program, `client`, from the source file `client.cbl`, enter:

```
bldapp client
```

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

```
client
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqb`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldapp updat
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldapp updat database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

```
updat
```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

Notes:

1. Before building Micro Focus stored procedures on Solaris, run the following commands:

   ```
   db2stop
   db2set DB2LIBPATH=$LD_LIBRARY_PATH
   db2set DB2ENVLIST="COBDIR LD_LIBRARY_PATH"
   db2set
   db2start
   ```

   Ensure that `db2stop` stops the database. The last `db2set` command is issued to check your settings: make sure `DB2LIBPATH` and `DB2ENVLIST` are set correctly.

2. Some of the more recent versions of the Micro Focus COBOL compiler used on Solaris cannot be used to create a statically-linked stored procedure. As such, the makefile and script file, `bldsrv`, 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 `bldsrv`. 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 `bldsrv`, in `sqllib/samples/cobol_mf`, 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. Since the stored procedure must be build on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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
# bldsrv script file -- Solaris
# Builds a Micro Focus COBOL stored procedure
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Precompile and bind the program.
embprep $1 $2

# Set COBCPY to include the DB2 COPY files directory.
export COBCPY=$DB2PATH/include/cobol_mf:$COBCPY

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

# Create the wrapper program for the stored procedure.
wrapsrv $1

# Link the program creating shared library $1 with main entry point ${1}_wrap
cob -x -o $1 ${1}_wrap.c $1.o -Q -G -L$DB2PATH/lib -ldb2 -ldb2gmf

# Copy the shared library to the sqllib/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 bldsrv	
Compile Options:	
cob	The COBOL compiler.
-cx	Compile to object module.

Compile and Link Options for bldsrv
Link Options: **cob** Use the compiler to link edit. **-x** Produce an executable program. **-o $1** Specify the executable program. **${1}_wrap.c** Specify the wrapper program. **$1.o** Specify the program object file. **-Q** **-G** **-L$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 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.

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

```
#! /bin/ksh
# wrapsrv script file
# Creates the wrapper program for Micro Focus COBOL stored procedures
# Usage: wrapsrv <stored_proc>

# Note: The client program calls "<stored_proc>_wrap" not "<stored_proc>"

# 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
```

To build the sample program `outsrv` from the source file `outsrv.sqb`, if connecting to the sample database, enter:

```
bldsrv outsrv
```

If connecting to another database, also enter the database name:

```
bldsrv outsrv database
```

The script file copies the stored procedure to the server in the path `sqllib/function`.

If necessary, set the file mode for the stored procedure so the client application can access 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, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 307 for details.

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

```
outcli database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the stored procedure library, `outsrv`, which executes the stored procedure function of the same name on the server database, and then returns the output 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.

Chapter 13. 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.ibm.com/software/data/db2/udb/ad
```

Notes:

1. All applications on Windows 32-bit operating systems, both embedded SQL and non-embedded SQL, must be built in a DB2 command window, and not from an operating system command prompt.
2. Any path names used in your programs that include the variable `%DB2PATH%` should be enclosed in quotes, as in: `"%DB2PATH%\function"`, as

the default installation for DB2 on Windows 32-bit operating systems for Version 7.1 is `\Program Files\sqllib`, which contains a space. If quotes are not used, you may get an error such as: "the syntax of the command is incorrect". In this chapter, such a path name is only enclosed in quotes if given as part of a command or code example.

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 AD Client 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, FIRSTNME 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 and Stored Procedures

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 and stored procedures. 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 and stored procedures in Microsoft Visual Basic, located in the directory `%DB2PATH%\samples\ole\msvb`. For information on building and running the OLE automation UDF and stored procedure samples, please see the `README` 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 CLI Applications
- DB2 CLI Stored Procedures
- DB2 API and 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 same batch files have been placed in these two directories. They contain commands to accept either a C or C++ source file, depending on the file extension. Batch files are used in this section to demonstrate building programs, except in the first two topics, ″ActiveX Data Objects (ADO)″ and ″Object Linking and Embedding (OLE) Automation″.

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 and Stored Procedures

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 implemented 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 334. For information on building Visual C++ stored procedures with DB2 CLI, see "DB2 CLI Stored Procedures" on page 326. For information on building Visual C++ embedded SQL stored procedures for DB2, see "Embedded SQL Stored Procedures" on page 331.

DB2 provides self-containing samples of OLE automation UDFs and stored procedures in Microsoft Visual C++, located in the directory %DB2PATH%\samples\ole\msvc. For information on building and running the OLE automation UDF and stored procedure samples, please see the README file in %DB2PATH%\samples\ole.

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.

This is the only required parameter, and the only one needed for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

If the program contains embedded SQL, indicated by the .sqc or .sqx extension, then the embprep batch file is called to precompile the program, producing a program file with either a .c or a .cxx extension, respectively.

```
@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 [ db_name [ userid password ]]

if exist "%1.sqc" call embprep %1 %2 %3 %4
if exist "%1.sqx" call embprep %1 %2 %3 %4

rem Compile the error-checking utility.
cl -Z7 -Od -c -W1 -D_X86=1 -DWIN32 utilcli.c

rem  Compile the program.
if exist "%1.sqx" goto cpp
cl -Z7 -Od -c -W1 -D_X86=1 -DWIN32 %1.c
goto link_step
:cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx

rem  Link the program.
:link_step
link -debug:full -debugtype:cv -OUT:%1.exe %1.obj utilcli.obj db2cli.lib
@echo on
```

Compile and Link Options for bldmcli
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.
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. **utilcli.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 `tbinfo` from the source file `tbinfo.c`, enter:

```
bldmcli tbinfo
```

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

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, dbusemx, from the source file dbusemx.sqc:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldmcli dbusemx
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldmcli dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldmcli dbusemx database userid password
   ```

The result is an executable file, dbusemx.

There are three ways to run this embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:
   ```
   dbusemx
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   dbusemx database
   ```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   dbusemx database userid password
   ```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions on more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs.

The script file bldmapi in sqllib/samples/cli contains the commands to build a DB2 CLI program with DB2 APIs. This file compiles and links in the utilapi utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the bldmcli batch file. Please see "DB2 CLI Applications" on page 322 for the compile and link options common to both bldmapi and bldmcli.

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

```
bldmapi dbmconn
```

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

```
dbmconn
```

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.
if exist "%1.cxx" goto cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c  utilcli.c
goto link_step
:cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx  utilcli.c

rem Link the program.
:link_step
link -debug:full -debugtype:cv -dll -out:%1.dll %1.obj utilcli.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
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.
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. **utilcli.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 `spserver` stored procedure from the source file `spserver.c`, enter:

```
bldmclis spserver
```

The batch file uses the module definition file `spserver.def`, contained in the same directory as the CLI sample programs, to build the stored procedure. The batch file copies the stored procedure DLL, `spserver.dll`, to the server in the path `%DB2PATH%\function`.

Next, catalog the stored procedures by running the spcreate.db2 script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure spserver, you can build the CLI client application spclient that calls the stored procedure.

You can build spclient by using the script file, bldmcli. Refer to "DB2 CLI Applications" on page 322 for details.

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

spclient *database userid password*

where

database
Is the name of the database to which you want to connect. The name could be sample, or its alias, or another database name.

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the stored procedure library, spserver, which executes a number of stored procedure functions on the server database. The output is returned to the client application.

DB2 API and Embedded SQL Applications

The batch file bldmapp.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. This is the only required parameter for programs that do not contain embedded SQL. Building

embedded SQL programs requires a connection to the database so three additional, optional, parameters are also provided: 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.

For an embedded SQL program, bldmapp passes the parameters to the precompile and bind file, embprep. If no database name is supplied, the default sample database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
@echo off
rem bldmapp.bat -- Windows 32-bit operating systems
rem Builds a Microsoft Visual C++ application program
rem Usage: bldmapp prog_name [ db_name [ userid password ]]

if exist "%1.sqx" goto embedded
if exist "%1.sqc" goto embedded
goto non_embedded

:embedded
rem Precompile and bind the program.
call embprep %1 %2 %3 %4
rem  Compile the program.
if exist "%1.cxx" goto cpp_emb
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c utilemb.c
goto link_embedded
:cpp_emb
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx utilemb.cxx
rem Link the program.
:link_embedded
link -debug:full -debugtype:cv -out:%1.exe %1.obj utilemb.obj db2api.lib
goto exit

:non_embedded
rem  Compile the program.
if exist "%1.cxx" goto cpp_non
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c utilapi.c
goto link_non_embedded
:cpp_non
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx utilapi.cxx
rem Link the program.
:link_non_embedded
link -debug:full -debugtype:cv -out:%1.exe %1.obj utilapi.obj db2api.lib
:exit
@echo on
```

Compile and Link Options for bldmapp
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.
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 **utilemb.obj** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.obj** If not an embedded SQL program, include the DB2 API utility object file for error checking. **db2api.lib** Link with the DB2 library. Refer to your compiler documentation for additional compiler options.

To build the DB2 API non-embedded SQL 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:

```
bldmapp 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
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the C source file `updat.sqc` in `%DB2PATH%\samples\c`, or from the C++ source file `updat.sqx` in `%DB2PATH%\samples\cpp`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldmapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldmapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldmapp updat database userid password
   ```

The result is an executable file `updat.exe`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The batch file `bldmsrv.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. Since the stored procedure must be build on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, the source file name, is required. Database name is 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 bldmsrv.bat -- Windows 32-bit operating systems
rem Builds a Microsoft Visual C++ stored procedure
rem Usage: bldmsrv prog_name [ db_name ]

rem Precompile and bind the program.
call embprep %1 %2

rem Compile the program.
if exist "%1.cxx" goto cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c
goto link_step
:cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx

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

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

Compile and Link Options for bldmsrv

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 bldmsrv
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 spserver stored procedure DLL from either the C source file, spserver.sqc, or the C++ source file, spserver.sqx, enter:

```
bldmsrv spserver
```

If connecting to another database, also enter the database name:

```
bldmsrv spserver database
```

The batch file uses the module definition file spserver.def, contained in the same directory as the sample programs, to build the DLL. The batch file copies the DLL, spserver.dll, to the server in the path %DB2PATH%\function.

Next, catalog the stored procedures by running the spcreate.db2 script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure DLL, `spserver`, you can build the client application `spclient` that calls it.

You can build `spclient` by using the script file, `bldmapp`. Refer to "DB2 API and Embedded SQL Applications" on page 328 for details.

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

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the stored procedure DLL, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

User-Defined Functions (UDFs)

The batch file `bldmudf`, 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 bldmudf.bat -- Windows 32-bit operating systems
rem Builds a Microsoft Visual C++ user-defined function (UDF).
rem Usage: bldmudf udf_prog_name

if "%1" == "" goto error

rem Compile the program.
if exist "%1.cxx" goto cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.c
```

```
goto link_step
:cpp
cl -Z7 -Od -c -W2 -D_X86_=1 -DWIN32 %1.cxx

:link_step
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"

goto exit
:error
echo Usage: bldmudf prog_name
:exit
@echo on
```

Compile and Link Options for bldmudf	
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 bldmudf
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 `udfsrv` from the source file `udfsrv.c`, enter:

```
bldmudf udfsrv
```

The batch file uses the module definition file `udfsrv.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, `udfsrv.dll`, to the server in the path `%DB2PATH%\function`.

Once you build `udfsrv`, you can build the client application, `udfcli`, 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 `udfcli` program from the `udfcli.c` source file in `%DB2PATH%\samples\cli` using the batch file `bldmcli`. Refer to "DB2 CLI Applications" on page 322 for details.

You can build the embedded SQL C `udfcli` program from the `udfcli.sqc` source file in `%DB2PATH%\samples\c` using the batch file `bldmapp`. Refer to "DB2 API and Embedded SQL Applications" on page 328 for details.

You can build the embedded SQL C++ udfcli program from the udfcli.sqx source file in %DB2PATH%\samples\cpp using the batch file bldmapp. Refer to "DB2 API and Embedded SQL Applications" on page 328 for details.

To run the UDF, enter:

```
udfcli
```

The calling application calls the ScalarUDF function from the udfsrv DLL.

IBM VisualAge C++ Version 3.5

This section contains the following topics:

- DB2 CLI Applications
- DB2 CLI Stored Procedures
- DB2 API and 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 same batch files have been placed in both these directories. They contain commands to accept either a C or C++ source file, depending on the file extension.

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.

This is the only required parameter for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

If the program contains embedded SQL, indicated by the .sqc or the .sqx extension, then the embprep batch file is called to precompile the program, producing a program file with either a .c or .cxx extension, respectively.

```
@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 exist "%1.sqc" call embprep %1 %2 %3 %4
if exist "%1.sqx" call embprep %1 %2 %3 %4

rem Compile the error-checking utility.
icc -c -Ti -W1 /I"%DB2PATH%\include" utilcli.c

rem  Compile the program.
if exist "%1.sqx" goto cpp
icc -c -Ti -W1 /I"%DB2PATH%\include" %1.c
goto link_step
:cpp
icc -c -Ti -W1 /I"%DB2PATH%\include" %1.cxx

rem  Link the program.
:link_step
ilink /MAP /DEBUG /ST:32000 /PM:VIO %1.obj utilcli.obj db2cli.lib
@echo on
```

Compile and Link Options for bldvcli

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.

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.

utilcli.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 `tbinfo` from the source file `tbinfo.c`, enter:

```
bldvcli tbinfo
```

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

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldvcli dbusemx
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldvcli dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldvcli dbusemx database userid password
   ```

The result is an executable file, `dbusemx`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:
   ```
   dbusemx
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   dbusemx database
   ```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   dbusemx database userid password
   ```

DB2 CLI Applications with DB2 APIs

DB2 includes CLI sample programs that use DB2 APIs to create and drop a database in order to demonstrate using CLI functions on more than one database. The descriptions of the CLI sample programs in Table 7 on page 22 indicates the samples that use DB2 APIs.

The script file `bldvapi` in `sqllib/samples/cli` contains the commands to build a DB2 CLI program with DB2 APIs. This file compiles and links in the `utilapi` utility file, which contains the DB2 APIs to create and drop a database. This is the only difference between this file and the `bldvcli` batch file. Please see "DB2 CLI Applications" on page 337 for the compile and link options common to both `bldvapi` and `bldvcli`.

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

```
bldvapi dbmconn
```

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

```
dbmconn
```

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.
if exist "%1.cxx" goto cpp
icc -c+ -Ti -Ge- -Gm+ -W1 %1.c  utilcli.c
goto link_step
:cpp
icc -c+ -Ti -Ge- -Gm+ -W1 %1.cxx  utilcli.c

:link_step
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 spserverva.def, for example, the IBM VisualAge C++
rem compiler requires "EXPORTS _outlanguage@16" and not "EXPORTS outlanguage".
ilib /GI %1va.def

rem Link the program and produce a DLL.
ilink /ST:64000 /PM:VIO /MAP /DLL %1.obj utilcli.obj %1va.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	
Compile Options:	
icc	The IBM VisualAge C++ compiler.
-c+	Perform compile only; no link. This batch file has separate compile and link 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.
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 `spserver` stored procedure from the source file `spserver.c`, enter:

```
bldvclis spserver
```

The batch file uses the module definition file, `spserverva.def`, contained in the same directory as the CLI sample programs, to build the stored procedure. The batch file copies the stored procedure DLL, `spserver.dll`, to the server in the path `%DB2PATH%\function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure `spserver`, you can build the CLI client application `spclient` that calls the stored procedure.

You can build `spclient` by using the batch file, `bldvcli`. Refer to "DB2 CLI Applications" on page 337 for details.

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

`spclient` *database userid password*

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the stored procedure library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

DB2 API and Embedded SQL Applications

The batch file `bldvapp.bat`, in `%DB2PATH%\samples\c`, and in `%DB2PATH%\samples\cpp`, contains the commands to build a DB2 application program.

The first parameter, `%1`, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, `bldvapp` passes the parameters to the precompile and bind batch file, `embprep`. If no database name is supplied, the default `sample` database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
@echo off
rem bldvapp.bat -- Windows 32-bit operating systems
rem Builds a VisualAge C++ application program
rem Usage: bldvapp prog_name [ db_name [  userid password ]]

if exist "%1.sqx" goto embedded
if exist "%1.sqc" goto embedded
goto non_embedded

:embedded
rem Precompile and bind the program.
call embprep %1 %2 %3 %4
rem  Compile the program.
if exist "%1.cxx" goto cpp_emb
icc -c -Ti -W1 %1.c utilemb.c
goto link_embedded
:cpp_emb
icc -c -Ti -W1 %1.cxx utilemb.cxx
rem Link the program.
:link_embedded
ilink /MAP /DEBUG /ST:32000 /PM:VIO %1.obj utilemb.obj db2api.lib
goto exit

:non_embedded
rem  Compile the program.
if exist "%1.cxx" goto cpp_non
icc -c -Ti -W1 %1.c utilapi.c
goto link_non_embedded
:cpp_non
icc -c -Ti -W1 %1.cxx utilapi.cxx
rem Link the program.
:link_non_embedded
ilink /MAP /DEBUG /ST:32000 /PM:VIO %1.obj utilapi.obj db2api.lib
:exit
@echo on
```

Compile and Link Options for bldvapp	
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 bldvapp
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. **utilemb.obj** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.obj** If not an embedded SQL program, include the DB2 API utility object file for error checking. **db2api.lib** Link with the DB2 library. Refer to your compiler documentation for additional compiler options.

To build the DB2 API non-embedded SQL 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:

```
bldvapp 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
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the C source file `updat.sqc` in `%DB2PATH%\samples\c`, or from the C++ source file `updat.sqx` in `%DB2PATH%\samples\cpp`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldvapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldvapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldvapp updat database userid password
   ```

The result is an executable file `updat.exe`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The batch file `bldvsrv.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. Since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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 bldvsrv.bat -- Windows 32-bit operating systems
rem Builds a VisualAge C++ stored procedure
rem Usage: bldvsrv prog_name [ db_name ]

rem Precompile and bind the program.
call embprep %1 %2

rem Compile the program.
if exist "%1.cxx" goto cpp
icc -c+ -Ti -Ge- -Gm+ -W1 %1.c
goto link_step
:cpp
icc -c+ -Ti -Ge- -Gm+ -W1 %1.cxx
```

```
:link_step
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 spserverva.def, the IBM VisualAge C++ compiler requires
rem "EXPORTS _outlanguage@16" and not "EXPORTS outlanguage".
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"
@echo on
```

Compile and Link Options for bldvsrv

Compile Options:

icc The IBM VisualAge C++ compiler.

-c+ Perform compile only; no link. This batch file has separate compile and link 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.

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 spserver stored procedure DLL from either the C source file, spserver.sqc, or the C++ source file, spserver.sqx, enter:

```
bldvsrv spserver
```

If connecting to another database, also enter the database name:

```
bldmsrv spserver database
```

The batch file uses the module definition file spserverva.def, contained in the same directory as the sample programs, to build the stored procedure. The batch file copies the stored procedure DLL, spserver.dll, to the server in the path %DB2PATH%\function.

Next, catalog the stored procedures by running the spcreate.db2 script on the server. First, connect to the database:

```
db2 connect to sample
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure DLL, spserver, you can build the client application spclient that calls it.

You can build spclient by using the script file, bldvapp. Refer to "DB2 API and Embedded SQL Applications" on page 342 for details.

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

spclient *database userid password*

where

database
Is the name of the database to which you want to connect. The name could be sample, or its alias, or another database name.

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the stored procedure DLL, spserver, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

User-Defined Functions (UDFs)

The batch file bldvudf, 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 connect to a database, or 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 bldvudf.bat -- Windows 32-bit operating systems
rem Builds a VisualAge C++ user-defined function (UDF)
rem Usage: bldvudf program_name

if "%1" == "" goto error

rem Compile the program.
if exist "%1.cxx" goto cpp
icc -Ti -c+ -Ge- -Gm+ -W1 %1.c
goto link_step
:cpp
icc -Ti -c+ -Ge- -Gm+ -W1 %1.cxx

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

rem Copy the UDF DLL to the 'function' directory.
copy %1.dll "%DB2PATH%\function"

goto exit
```

```
:error
echo Usage: bldvudf prog_name
:exit
@echo on
```

Compile and Link Options for bldvudf
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.
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 `udfsrv` from the source file `udf.c`, enter:

```
bldvudf udfsrv
```

The batch file uses the module definition file udfsrv.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, udfsrv.dll, to the server in the path %DB2PATH%\function.

Once you build udfsrv, you can build the client application, udfcli, 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 udfcli program from the udfcli.c source file in %DB2PATH%\samples\cli using the batch file bldvcli. Refer to "DB2 CLI Applications" on page 337 for details.

You can build the embedded SQL C udfcli program from the udfcli.sqc source file in %DB2PATH%\samples\c using the batch file bldvapp. Refer to "DB2 API and Embedded SQL Applications" on page 342 for details.

You can build the embedded SQL C++ udfcli program from the udfcli.sqx source file in %DB2PATH%\samples\cpp using the batch file bldvapp. Refer to "DB2 API and Embedded SQL Applications" on page 342 for details.

To run the UDF, enter:

```
udfcli
```

The calling application calls the ScalarUDF function from the udfsrv DLL.

IBM VisualAge C++ Version 4.0

Application building information for the VisualAge C++ version 4 compiler is common to AIX, OS/2 and Windows 32-bit operating systems. See "VisualAge C++ Version 4.0" on page 129 for this information.

IBM VisualAge COBOL

This section contains the following topics:

- Using the Compiler
- DB2 API and 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.
- If you are using the "System/390 host data type support" feature of the IBM VisualAge COBOL compiler, the DB2 include files for your applications are in the following directory:

  ```
  %DB2PATH%\include\cobol_i
  ```

  If you are building DB2 sample programs using the batch files provided, the include file path specified in the batch 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 VisualAge COBOL compiler, or you are using an earlier version of this compiler, then the DB2 include files for your applications are in the following directory:

  ```
  %DB2PATH%\include\cobol_a
  ```

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

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

DB2 API and Embedded SQL Applications

The batch file bldapp.bat, in %DB2PATH%\samples\cobol, contains the commands to build a DB2 application program.

The first parameter, %1, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, bldapp passes the parameters to the precompile and bind file, embprep. If no database name is supplied, the default sample database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
@echo off
rem bldapp.bat  -- Windows 32-bit operating systems
rem Builds a VisualAge COBOL application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

rem If an embedded SQL program, precompile and bind it.
```

```
if not exist "%1.sqb" goto compile_step
call embprep %1 %2 %3 %4

:compile_step
rem Compile the error-checking utility.
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
@echo on
```

Compile and Link Options for bldapp

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.

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 non-embedded SQL sample program `client` from the source file `client.cbl`, enter:

```
bldapp 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
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqb`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

   ```
   bldapp updat database
   ```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

   ```
   bldapp updat database userid password
   ```

The result is an executable file, `updat`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:

   ```
   updat
   ```

2. If accessing another database on the same instance, enter the executable name and the database name:

   ```
   updat database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

   ```
   updat database userid password
   ```

Embedded SQL Stored Procedures

The batch file `bldsrv.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. Since the stored procedure must be build on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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 bldsrv.bat  -- Windows 32-bit operating systems
rem Builds a VisualAge COBOL stored procedure
rem Usage: bldsrv <prog_name> [ <db_name> ]
```

```
rem Precompile and bind the program.
call embprep %1 %2

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.
ilib /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.
copy %1.dll "%DB2PATH%\function"
@echo on
```

Compile and Link Options for bldsrv

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 batch file has separate compile and link 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"`.

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 sample program `outsrv` from the source file `outsrv.sqb`, connecting to the sample database, enter:

```
bldsrv outsrv
```

If connecting to another database, also include the database name:

```
bldsrv outsrv database
```

The script file copies the stored procedure to the server in the path `sqllib/function`.

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 batch file `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 351 for details.

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

```
outcli database userid password
```

where

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 accesses the stored procedure library, `outsrv`, which executes the stored procedure function of the same name on the server database, and then returns the output to the client application.

Micro Focus COBOL

This section includes the following topics:

- Using the Compiler
- DB2 API and 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 and Embedded SQL Applications

The batch file bldapp, in %DB2PATH%\samples\cobol_mf, contains the commands to build a DB2 application program.

The first parameter, %1, specifies the name of your source file. This is the only required parameter for programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: 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.

For an embedded SQL program, bldapp passes the parameters to the precompile and bind batch file, embprep. If no database name is supplied, the default sample database is used. The user ID and password parameters are only needed if the instance where the program is built is different from the instance where the database is located.

```
@echo off
rem bldapp.bat  -- Windows 32-bit operating systems
rem Builds a Micro Focus Cobol application program
rem Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

rem If an embedded SQL program, precompile and bind it.
if not exist "%1.sqb" goto compile_step
```

```
call embprep %1 %2 %3 %4

:compile_step
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
@echo on
```

Compile and Link Options for bldapp
Compile Option: **cobol** The Micro Focus COBOL compiler.
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 non-embedded SQL sample program, `client`, from the source file `client.cbl`, enter:

```
bldapp 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
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `updat`, from the source file `updat.sqb`:

1. If connecting to the sample database on the same instance, enter:

   ```
   bldapp updat
   ```

2. If connecting to another database on the same instance, also enter the database name:

```
bldapp updat database
```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

```
bldapp updat database userid password
```

The result is an executable file, `updat.exe`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name (without the extension):

```
updat
```

2. If accessing another database on the same instance, enter the executable name and the database name:

```
updat database
```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

```
updat database userid password
```

Embedded SQL Stored Procedures

The batch file `bldsrv`, 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. Since the stored procedure must be build on the same instance where the database resides, there are no parameters for user ID and password.

Only the first parameter, source file name, is required. Database name is 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 bldsrv.bat  -- Windows 32-bit operating systems
rem Builds a Micro Focus Cobol stored procedure
rem Usage: bldsrv <prog_name> [ <db_name> ]

rem Precompile and bind the program.
call embprep %1 %2

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"
@echo on
```

Compile and Link Options for bldsrv
Compile Options:
cobol The Micro Focus COBOL compiler.
/case Prevent external symbols being converted to upper case.
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 sample program `outsrv` from the source file `outsrv.sqb`, if connecting to the sample database, enter:

```
bldsrv outsrv
```

If connecting to another database, also enter the database name:

```
bldsrv outsrv database
```

The script file copies the DLL to the server in the path `sqllib/function`.

If necessary, set the file mode for the DLL so the client program can access it.

Once you build the DLL `outsrv`, you can build the client application `outcli` that calls it. You can build `outcli` using the batch file, `bldapp`. Refer to "DB2 API and Embedded SQL Applications" on page 356 for details.

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

```
outcli database userid password
```

where

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

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the DLL, `outsrv`, and executes the stored procedure function of the same name on the server database. The output is then returned 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.

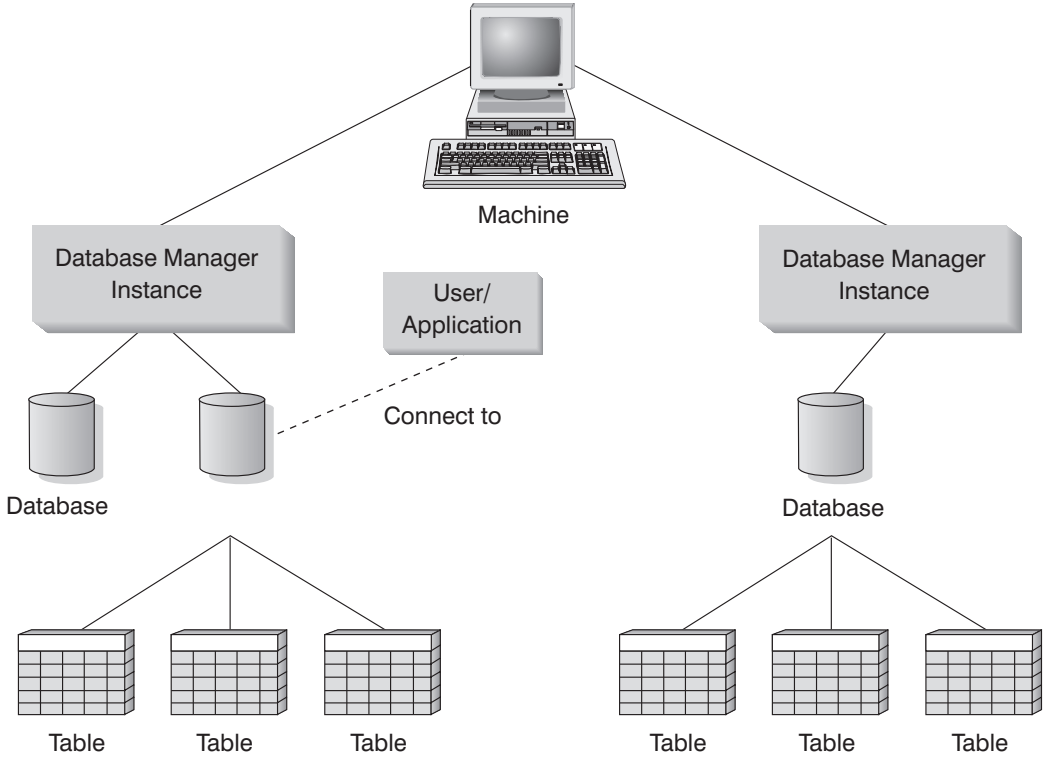

Figure 1. Database Manager Instances

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 Universal Database Version 6.1, and another running DB2

Universal Database Version 7.1. However, within a version level, only one release and modification level are supported. For example, DB2 Version 5.0 and DB2 Version 5.2 cannot coexist on a UNIX server.

With OS/2, Windows NT and Windows 2000 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 Universal Database Version 6.1, and another instance running DB2 Universal Database Version 7.1.

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, Windows NT and Windows 2000, this is an alphanumeric string of up to eight characters. An instance named "DB2" 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, Windows NT and Windows 2000, 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, Windows NT and Windows 2000 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 variable is used on OS/2, Windows NT and Windows 2000 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 7.1 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 7.1. 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).
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, DB2 Version 5, or DB2 Version 6.1, and you want them to run in both a database instance of the previous version as well as a DB2 Version 7.1 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 "Migrating

from Previous Versions of DB2″ section in the 'Planning for Installation' chapter of the *DB2 for UNIX Quick Beginnings* book.

Questions

Question 1: How was the application on the previous DB2 version 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_filename*

HP-UX
/usr/bin/chatr *executable_filename*

Linux /usr/bin/objdump -p *executable_filename*

PTX /usr/bin/dump -Lv *executable_filename*

Silicon Graphics IRIX
/bin/elfdump -Lv *executable_filename*

Solaris
/usr/bin/dump -Lv *executable_filename*

where *executable_filename* is the name of the executable file for the application.

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, 6.1 or 7.1 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.

The install paths for the different DB2 versions are as follows:

Version 1
`/usr/lpp/db2_01_01_0000/lib`

Version 2
`/usr/lpp/db2_02_01/lib`

Version 5
`/usr/lpp/db2_05_00/lib`

Version 6.1
`/usr/lpp/db2_06_01/lib`

Version 7.1
`/usr/lpp/db2_07_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 Linux, PTX, Silicon Graphics IRIX and Solaris.

Note: For n32 object type applications on Silicon Graphics IRIX, use the LD_LIBRARYN32_PATH environment variable.

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 6.1 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_06_01/lib/libdb2.a`, and the database server is DB2 Universal Database Version 7.1 for AIX, do one of the following:
  - Change the symbolic link to point to:

    ```
    /usr/lpp/db2_07_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_07_01/cfg/db2ln
    ```

  - Set the LIBPATH environment variable to point to `/usr/lpp/db2_07_01/lib` or `INSTHOME/sqllib/lib`, where INSTHOME is the home directory of the Version 7.1 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_07_01/lib/libdb2.a`, and the database server is DB2 Version 6.1, 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 6.1 application loads a shared library out of the $HOME path of a DB2 Version 6.1 instance owner (`$HOME/sqllib/lib/libdb2.a`), and the database server is DB2 Universal Database Version 7.1 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_07_01/lib` or `INSTHOME/sqllib/lib`, where `INSTHOME` is the home directory of the Version 7.1 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 6.1 application loads a shared library out of the DB2 Version 6.1 install path (`/usr/lpp/db2_06_01/lib/libdb2.a`), and the database server is DB2 Universal Database Version 7.1 for AIX, do one of the following:

- Set the LIBPATH environment variable to point to `/usr/lpp/db2_07_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 6.1 application loads a shared library out of the DB2 Universal Database Version 7.1 for AIX install path (`/usr/lpp/db2_07_01/lib/libdb2.a`), and the database server is DB2 Version 6.1, 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 will help make your applications 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 variable 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 5, code /usr/lpp/db2_05_00/lib. Ordinarily, you do not need to do this.
- When you are building an application for production, rather than internal development, 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 Versions 6.1 and 7.1, 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 6.1 or Version 7.1 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 6.1 or Version 7.1 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 7.1 and previous versions of DB2. Refer to the *Administrative API Reference* for a list of API incompatibilities between DB2 Universal Database Version 7.1 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. Using the DB2 Library

The DB2 Universal Database library consists of online help, books (PDF and HTML), and sample programs in HTML format. This section describes the information that is provided, and how you can access it.

To access product information online, you can use the Information Center. For more information, see "Accessing Information with the Information Center" on page 385. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web.

DB2 PDF Files and Printed Books

DB2 Information

The following table divides the DB2 books into four categories:

DB2 Guide and Reference Information
These books contain the common DB2 information for all platforms.

DB2 Installation and Configuration Information
These books are for DB2 on a specific platform. For example, there are separate *Quick Beginnings* books for DB2 on OS/2, Windows, and UNIX-based platforms.

Cross-platform sample programs in HTML
These samples are the HTML version of the sample programs that are installed with the Application Development Client. The samples are for informational purposes and do not replace the actual programs.

Release notes
These files contain late-breaking information that could not be included in the DB2 books.

The installation manuals, release notes, and tutorials are viewable in HTML directly from the product CD-ROM. Most books are available in HTML on the product CD-ROM for viewing and in Adobe Acrobat (PDF) format on the DB2 publications CD-ROM for viewing and printing. You can also order a printed copy from IBM; see "Ordering the Printed Books" on page 381. The following table lists books that can be ordered.

On OS/2 and Windows platforms, you can install the HTML files under the sqllib\doc\html directory. DB2 information is translated into different

languages; however, all the information is not translated into every language. Whenever information is not available in a specific language, the English information is provided

On UNIX platforms, you can install multiple language versions of the HTML files under the doc/%L/html directories, where %L represents the locale. For more information, refer to the appropriate *Quick Beginnings* book.

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

- "Viewing Information Online" on page 384
- "Searching Information Online" on page 388
- "Ordering the Printed Books" on page 381
- "Printing the PDF Books" on page 380

Table 18. DB2 Information

Name	Description	Form Number PDF File Name	HTML Directory
	DB2 Guide and Reference Information		
Administration Guide	*Administration Guide: Planning* provides an overview of database concepts, information about design issues (such as logical and physical database design), and a discussion of high availability.	SC09-2946 db2d1x70	db2d0
	Administration Guide: Implementation provides information on implementation issues such as implementing your design, accessing databases, auditing, backup and recovery.	SC09-2944 db2d2x70	
	Administration Guide: Performance provides information on database environment and application performance evaluation and tuning.	SC09-2945 db2d3x70	
	You can order the three volumes of the *Administration Guide* in the English language in North America using the form number SBOF-8934.		
Administrative API Reference	Describes the DB2 application programming interfaces (APIs) and data structures that you can use to manage your databases. This book also explains how to call APIs from your applications.	SC09-2947 db2b0x70	db2b0

Table 18. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
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.	SC09-2948 db2axx70	db2ax
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. Available in HTML format only.	No form number db2apx70	db2ap
Application Development Guide	Explains how to develop applications that access DB2 databases using embedded SQL or Java (JDBC and SQLJ). Discussion topics include writing stored procedures, writing user-defined functions, creating user-defined types, using triggers, and developing applications in partitioned environments or with federated systems.	SC09-2949 db2a0x70	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.	SC09-2950 db2l0x70	db2l0
Command Reference	Explains how to use the Command Line Processor and describes the DB2 commands that you can use to manage your database.	SC09-2951 db2n0x70	db2n0
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. This book also details how to use DRDA application servers with DB2 Connect application requesters. Available in HTML and PDF only.	No form number db2h1x70	db2h1

Table 18. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Data Movement Utilities Guide and Reference	Explains how to use DB2 utilities, such as import, export, load, AutoLoader, and DPROP, that facilitate the movement of data.	SC09-2955 db2dmx70	db2dm
Data Warehouse Center Administration Guide	Provides information on how to build and maintain a data warehouse using the Data Warehouse Center.	SC26-9993 db2ddx70	db2dd
Data Warehouse Center Application Integration Guide	Provides information to help programmers integrate applications with the Data Warehouse Center and with the Information Catalog Manager.	SC26-9994 db2adx70	db2ad
DB2 Connect User's Guide	Provides concepts, programming, and general usage information for the DB2 Connect products.	SC09-2954 db2c0x70	db2c0
DB2 Query Patroller Administration Guide	Provides an operational overview of the DB2 Query Patroller system, specific operational and administrative information, and task information for the administrative graphical user interface utilities.	SC09-2958 db2dwx70	db2dw
DB2 Query Patroller User's Guide	Describes how to use the tools and functions of the DB2 Query Patroller.	SC09-2960 db2wwx70	db2ww
Glossary	Provides definitions for terms used in DB2 and its components. Available in HTML format and in the *SQL Reference*.	No form number db2t0x70	db2t0
Image, Audio, and Video Extenders Administration and Programming	Provides general information about DB2 extenders, and information on the administration and configuration of the image, audio, and video (IAV) extenders and on programming using the IAV extenders. It includes reference information, diagnostic information (with messages), and samples.	SC26-9929 dmbu7x70	dmbu7
Information Catalog Manager Administration Guide	Provides guidance on managing information catalogs.	SC26-9995 db2dix70	db2di

Table 18. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Information Catalog Manager Programming Guide and Reference	Provides definitions for the architected interfaces for the Information Catalog Manager.	SC26-9997 db2bix70	db2bi
Information Catalog Manager User's Guide	Provides information on using the Information Catalog Manager user interface.	SC26-9996 db2aix70	db2ai
Installation and Configuration Supplement	Guides you through the planning, installation, and setup of platform-specific DB2 clients. This supplement also contains information on binding, setting up client and server communications, DB2 GUI tools, DRDA AS, distributed installation, the configuration of distributed requests, and accessing heterogeneous data sources.	GC09-2957 db2iyx70	db2iy
Message Reference	Lists messages and codes issued by DB2, the Information Catalog Manager, and the Data Warehouse Center, and describes the actions you should take. You can order both volumes of the Message Reference in the English language in North America with the form number SBOF-8932.	Volume 1 GC09-2978 db2m1x70 Volume 2 GC09-2979 db2m2x70	db2m0
OLAP Integration Server Administration Guide	Explains how to use the Administration Manager component of the OLAP Integration Server.	SC27-0787 db2dpx70	n/a
OLAP Integration Server Metaoutline User's Guide	Explains how to create and populate OLAP metaoutlines using the standard OLAP Metaoutline interface (not by using the Metaoutline Assistant).	SC27-0784 db2upx70	n/a
OLAP Integration Server Model User's Guide	Explains how to create OLAP models using the standard OLAP Model Interface (not by using the Model Assistant).	SC27-0783 db2lpx70	n/a
OLAP Setup and User's Guide	Provides configuration and setup information for the OLAP Starter Kit.	SC27-0702 db2ipx70	db2ip
OLAP Spreadsheet Add-in User's Guide for Excel	Describes how to use the Excel spreadsheet program to analyze OLAP data.	SC27-0786 db2epx70	db2ep

Table 18. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3	Describes how to use the Lotus 1-2-3 spreadsheet program to analyze OLAP data.	SC27-0785 db2tpx70	db2tp
Replication Guide and Reference	Provides planning, configuration, administration, and usage information for the IBM Replication tools supplied with DB2.	SC26-9920 db2e0x70	db2e0
Spatial Extender User's Guide and Reference	Provides information about installing, configuring, administering, programming, and troubleshooting the Spatial Extender. Also provides significant descriptions of spatial data concepts and provides reference information (messages and SQL) specific to the Spatial Extender.	SC27-0701 db2sbx70	db2sb
SQL Getting Started	Introduces SQL concepts and provides examples for many constructs and tasks.	SC09-2973 db2y0x70	db2y0
SQL Reference, Volume 1 and Volume 2	Describes SQL syntax, semantics, and the rules of the language. This book 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-8933.	Volume 1 SC09-2974 db2s1x70 Volume 2 SC09-2975 db2s2x70	db2s0
System Monitor Guide and Reference	Describes how to collect different kinds of information about databases and the database manager. This book explains how to use the information to understand database activity, improve performance, and determine the cause of problems.	SC09-2956 db2f0x70	db2f0
Text Extender Administration and Programming	Provides general information about DB2 extenders and information on the administration and configuring of the text extender and on programming using the text extenders. It includes reference information, diagnostic information (with messages) and samples.	SC26-9930 desu9x70	desu9

Table 18. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	GC09-2850 db2p0x70	db2p0
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database, Version 7.	SC09-2976 db2q0x70	db2q0
DB2 Installation and Configuration Information			
DB2 Connect Enterprise Edition for OS/2 and Windows Quick Beginnings	Provides planning, migration, installation, and configuration information for DB2 Connect Enterprise Edition on the OS/2 and Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2953 db2c6x70	db2c6
DB2 Connect Enterprise Edition for UNIX Quick Beginnings	Provides planning, migration, installation, configuration, and task information for DB2 Connect Enterprise Edition on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2952 db2cyx70	db2cy
DB2 Connect Personal Edition Quick Beginnings	Provides planning, migration, installation, configuration, and task information for DB2 Connect Personal Edition on the OS/2 and Windows 32-bit operating systems. This book also contains installation and setup information for all supported clients.	GC09-2967 db2c1x70	db2c1
DB2 Connect Personal Edition Quick Beginnings for Linux	Provides planning, installation, migration, and configuration information for DB2 Connect Personal Edition on all supported Linux distributions.	GC09-2962 db2c4x70	db2c4
DB2 Data Links Manager Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for AIX and Windows 32-bit operating systems.	GC09-2966 db2z6x70	db2z6

Table 18. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
DB2 Enterprise - Extended Edition for UNIX Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2964 db2v3x70	db2v3
DB2 Enterprise - Extended Edition for Windows Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2963 db2v6x70	db2v6
DB2 for OS/2 Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the OS/2 operating system. This book also contains installation and setup information for many supported clients.	GC09-2968 db2i2x70	db2i2
DB2 for UNIX Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on UNIX-based platforms. This book also contains installation and setup information for many supported clients.	GC09-2970 db2ixx70	db2ix
DB2 for Windows Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on Windows 32-bit operating systems. This book also contains installation and setup information for many supported clients.	GC09-2971 db2i6x70	db2i6
DB2 Personal Edition Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on the OS/2 and Windows 32-bit operating systems.	GC09-2969 db2i1x70	db2i1
DB2 Personal Edition Quick Beginnings for Linux	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on all supported Linux distributions.	GC09-2972 db2i4x70	db2i4

Table 18. DB2 Information (continued)

Name	Description	Form Number PDF File Name	HTML Directory
DB2 Query Patroller Installation Guide	Provides installation information about DB2 Query Patroller.	GC09-2959 db2iwx70	db2iw
DB2 Warehouse Manager Installation Guide	Provides installation information for warehouse agents, warehouse transformers, and the Information Catalog Manager.	GC26-9998 db2idx70	db2id
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. The sample programs are provided for informational purposes only. Not all samples are available in all programming languages. The HTML samples are only available when the DB2 Application Development Client is installed. For more information on the programs, refer to the *Application Building Guide*.	No form number	db2hs
Release Notes			
DB2 Connect Release Notes	Provides late-breaking information that could not be included in the DB2 Connect books.	See note #2.	db2cr
DB2 Installation Notes	Provides late-breaking installation-specific information that could not be included in the DB2 books.	Available on product CD-ROM only.	
DB2 Release Notes	Provides late-breaking information about all DB2 products and features that could not be included in the DB2 books.	See note #2.	db2ir

Notes:

1. The character *x* in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d0e70` identifies the English version of the *Administration Guide* and the file name `db2d0f70` identifies the French version of the same book. The following letters are used in the sixth position of the file name to indicate the language version:

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. Late breaking information that could not be included in the DB2 books is available in the Release Notes in HTML format and as an ASCII file. The HTML version is available from the Information Center and on the product CD-ROMs. To view the ASCII file:
   - On UNIX-based platforms, see the Release.Notes file. This file is located in the DB2DIR/Readme/*%L* directory, where *%L* represents the locale name and `DB2DIR` represents:
     - /usr/lpp/db2_07_01 on AIX
     - /opt/IBMdb2/V7.1 on HP-UX, PTX, Solaris, and Silicon Graphics IRIX
     - /usr/IBMdb2/V7.1 on Linux.
   - On other platforms, see the RELEASE.TXT file. This file is located in the directory where the product is installed. On OS/2 platforms, you can also double-click the **IBM DB2** folder and then double-click the **Release Notes** icon.

Printing the PDF Books

If you prefer to have printed copies of the books, you can print the PDF files found on the DB2 publications CD-ROM. Using the Adobe Acrobat Reader, you can print either the entire book or a specific range of pages. For the file name of each book in the library, see Table 18 on page 372.

You can obtain the latest version of the Adobe Acrobat Reader from the Adobe Web site at http://www.adobe.com.

The PDF files are included on the DB2 publications CD-ROM with a file extension of PDF. To access the PDF files:

1. Insert the DB2 publications CD-ROM. On UNIX-based platforms, mount the DB2 publications CD-ROM. Refer to your *Quick Beginnings* book for the mounting procedures.
2. Start the Acrobat Reader.
3. Open the desired PDF file from one of the following locations:
   - On OS/2 and Windows platforms:

     x:\doc*language* directory, where *x* represents the CD-ROM drive and *language* represent the two-character country code that represents your language (for example, EN for English).
   - On UNIX-based platforms:

     /cdrom/doc/*%L* directory on the CD-ROM, where */cdrom* represents the mount point of the CD-ROM and *%L* represents the name of the desired locale.

You can also copy the PDF files from the CD-ROM to a local or network drive and read them from there.

Ordering the Printed Books

You can order the printed DB2 books either individually or as a set (in North America only) by using a sold bill of forms (SBOF) number. To order books, 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. You can also order the books from the Publications Web page at http://www.elink.ibmlink.ibm.com/pbl/pbl.

Two sets of books are available. SBOF-8935 provides reference and usage information for the DB2 Warehouse Manager. SBOF-8931 provides reference and usage information for all other DB2 Universal Database products and features. The contents of each SBOF are listed in the following table:

Table 19. Ordering the printed books

SBOF Number	Books Included	
SBOF-8931	• Administration Guide: Planning • Administration Guide: Implementation • Administration Guide: Performance • Administrative API Reference • Application Building Guide • Application Development Guide • CLI Guide and Reference • Command Reference • Data Movement Utilities Guide and Reference • Data Warehouse Center Administration Guide • Data Warehouse Center Application Integration Guide • DB2 Connect User's Guide • Installation and Configuration Supplement • Image, Audio, and Video Extenders Administration and Programming • Message Reference, Volumes 1 and 2	• OLAP Integration Server Administration Guide • OLAP Integration Server Metaoutline User's Guide • OLAP Integration Server Model User's Guide • OLAP Integration Server User's Guide • OLAP Setup and User's Guide • OLAP Spreadsheet Add-in User's Guide for Excel • OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3 • Replication Guide and Reference • Spatial Extender Administration and Programming Guide • SQL Getting Started • SQL Reference, Volumes 1 and 2 • System Monitor Guide and Reference • Text Extender Administration and Programming • Troubleshooting Guide • What's New
SBOF-8935	• Information Catalog Manager Administration Guide • Information Catalog Manager User's Guide • Information Catalog Manager Programming Guide and Reference	• Query Patroller Administration Guide • Query Patroller User's Guide

DB2 Online Documentation

Accessing Online Help

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

Type of Help	Contents	How to Access...
Command Help	Explains the syntax of commands in the command line processor.	From the command line processor in interactive mode, enter: `? command` where *command* represents 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.
Client Configuration Assistant Help Command Center Help Control Center Help Data Warehouse Center Help Event Analyzer Help Information Catalog Manager Help Satellite Administration Center Help Script Center Help	Explains the tasks you can perform in a window or notebook. The help includes overview and prerequisite information you need to know, and it describes how to use the window or notebook controls.	From a window or notebook, click the **Help** push button or press the **F1** key.

Type of Help	Contents	How to Access...

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* represents 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* represents 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* represents an SQL statement. For example, help SELECT displays help about the SELECT statement. **Note:** SQL help is not available on UNIX-based platforms.
SQLSTATE Help	Explains SQL states and class codes.	From the command line processor in interactive mode, enter: `? sqlstate or ? class code` where *sqlstate* represents a valid five-digit SQL state and *class code* represents 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.

Viewing Information Online

The books 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:

- If you are running DB2 administration tools, use the Information Center.
- From a browser, click **File —>Open Page**. The page you open contains descriptions of and links to DB2 information:
  - On UNIX-based platforms, open the following page:

    INSTHOME/sqllib/doc/*%L*/html/index.htm

    where *%L* represents 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 the **DB2 Information** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

Installing the Netscape Browser

If you do not already have a Web browser installed, you can install Netscape from the Netscape CD-ROM found in the product boxes. For detailed instructions on how to install it, perform the following:

1. Insert the Netscape CD-ROM.
2. On UNIX-based platforms only, mount the CD-ROM. Refer to your *Quick Beginnings* book for the mounting procedures.
3. For installation instructions, refer to the CDNAV*nn*.txt file, where *nn* represents your two character language identifier. The file is located at the root directory of the CD-ROM.

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.

You can open the Information Center by double-clicking the Information Center icon. Depending on the system you are using, the icon is in the Information folder in the main product folder or the Windows **Start** menu.

You can also access the Information Center by using the toolbar and the **Help** menu on the DB2 Windows platform.

The Information Center provides six types of information. Click the appropriate tab to look at the topics provided for that type.

Tasks Key tasks you can perform using DB2.

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

Books DB2 books.

Troubleshooting
Categories of error messages and their recovery actions.

Sample Programs
Sample programs that come with the DB2 Application Development Client. If you did not install the DB2 Application Development Client, this tab is not displayed.

Web 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 a find feature, so you can look for a specific topic without browsing the lists.

For a full text search, follow the hypertext link in the Information Center to the **Search DB2 Online Information** search form.

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 using one of the following methods:

On Windows
Click **Start** and select **Programs —> IBM DB2 —> Information —> Start HTML Search Server**.

On OS/2
Double-click the **DB2 for OS/2** folder, and then double-click the **Start HTML Search Server** icon.

Refer to the release notes if you experience any other problems when searching the HTML information.

Note: The Search function is not available in the Linux, PTX, and Silicon Graphics IRIX environments.

Using DB2 Wizards

Wizards help you complete specific administration tasks by taking you through each task one step at a time. Wizards are available through the Control Center and the Client Configuration Assistant. The following table lists the wizards and describes their purpose.

Note: The Create Database, Create Index, Configure Multisite Update, and Performance Configuration wizards are available for the partitioned database environment.

Wizard	Helps You to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click **Add**.
Backup Database	Determine, create, and schedule a backup plan.	From the Control Center, right-click the database you want to back up and select **Backup —> Database Using Wizard**.
Configure Multisite Update	Configure a multisite update, a distributed transaction, or a two-phase commit.	From the Control Center, right-click the **Databases** folder and select **Multisite Update**.
Create Database	Create a database, and perform some basic configuration tasks.	From the Control Center, right-click the **Databases** folder and select **Create —> Database Using Wizard**.
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, right-click the **Tables** icon and select **Create —> Table Using Wizard**.
Create Table Space	Create a new table space.	From the Control Center, right-click the **Table Spaces** icon and select **Create —> Table Space Using Wizard**.
Create Index	Advise which indexes to create and drop for all your queries.	From the Control Center, right-click the **Index** icon and select **Create —> Index Using Wizard**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, right-click the database you want to tune and select **Configure Performance Using Wizard**. For the partitioned database environment, from the Database Partitions view, right-click the first database partition you want to tune and select **Configure Performance Using Wizard**.

Wizard	Helps You to...	How to Access...
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, right-click the database you want to restore and select **Restore —> Database Using Wizard**.

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, perform the following steps:

1. Copy all files and subdirectories from \sqllib\doc\html on your local system to a Web server. Each book has its own subdirectory that contains 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, refer to the NetQuestion Appendix in the *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. When you are able to view the book files, you can bookmark commonly viewed topics. 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 how you can serve the DB2 Universal Database online documentation files from a central machine, refer to the NetQuestion Appendix in the *Installation and Configuration Supplement*.

Searching Information Online

To find information in the HTML files, use one of the following methods:

- Click **Search** in the top frame. Use the search form to find a specific topic. This function is not available in the Linux, PTX, or Silicon Graphics IRIX environments.
- Click **Index** in the top frame. Use the index to find a specific topic in the book.
- Display the table of contents or index of the help or 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 385 for details.

Appendix E. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make

improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

IBM 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.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Trademarks

The following terms, which may be denoted by an asterisk(*), are trademarks of International Business Machines Corporation in the United States, other countries, or both.

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

The following terms are trademarks or registered trademarks of other companies:

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

Java or all Java-based trademarks and logos, and Solaris are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, other countries, or both.

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

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

Index

C

D

E

F

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Contacting IBM

If you have a technical problem, please review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. This guide suggests information that you can gather to help DB2 Customer Support 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.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:

- 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.

http://www.ibm.com/software/data/
The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more.

http://www.ibm.com/software/data/db2/library/
The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information.

Note: This information may be in English only.

http://www.elink.ibmlink.ibm.com/pbl/pbl/
The International Publications ordering Web site provides information on how to order books.

http://www.ibm.com/education/certify/
The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products, including DB2.

ftp.software.ibm.com
Log on as anonymous. In the directory /ps/products/db2, you can find demos, fixes, information, and tools relating to DB2 and many other products.

comp.databases.ibm-db2, bit.listserv.db2-l
These Internet newsgroups are available for users to discuss their experiences with DB2 products.

On Compuserve: GO IBMDB2
Enter this command to access the IBM DB2 Family forums. All DB2 products are supported through these forums.

For information on how to contact IBM outside of the United States, refer to Appendix A of the IBM Software Support Handbook. To access this document, go to the following Web page: http://www.ibm.com/support/, and then select the IBM Software Support Handbook link near the bottom of the page.

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

Part Number: CT7XXNA

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

CT7XXNA

SC09-2948-00