IBM® DB2 Universal Database™

Application Development Guide: Building and Running Applications

Version 8

SC09-4825-00

IBM® DB2 Universal Database™

Application Development Guide: Building and Running Applications

Version 8

SC09-4825-00

Before using this information and the product it supports, be sure to read the general information under *Notices*.

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

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

Contents

About This Book

The *Application Development Guide* is a three-volume book that describes what you need to know about coding, debugging, building, and running DB2 applications:

- *Application Development Guide: Programming Client Applications* contains what you need to know to code standalone DB2 applications that run on DB2 clients. It includes information on:
  - Programming interfaces that are supported by DB2. High-level descriptions are provided for DB2 Developer's Edition, supported programming interfaces, facilities for creating Web applications, and DB2-provided programming features, such as routines and triggers.
  - The general structure that a DB2 application should follow. Recommendations are provided on how to maintain data values and relationships in the database, authorization considerations are described, and information is provided on how to test and debug your application.
  - Embedded SQL, both dynamic and static. The general considerations for embedded SQL are described, as well as the specific issues that apply to the usage of static and dynamic SQL in DB2 applications.
  - Supported host and interpreted languages, such as C/C++, COBOL, Perl, and REXX, and how to use embedded SQL in applications that are written in these languages.
  - Java (both JDBC and SQLj), and considerations for building Java applications that use WebSphere Application Servers.
  - The IBM OLE DB Provider for DB2 Servers. General information is provided about IBM OLE DB Provider support for OLE DB services, components, and properties. Specific information is also provided about Visual Basic and Visual C++ applications that use the OLE DB interface for ActiveX Data Objects (ADO).
  - National language support issues. General topics, such as collating sequences, the derivation of code pages and locales, and character conversions are described. More specific issues such as DBCS code pages, EUC character sets, and issues that apply in Japanese and Traditional Chinese EUC and UCS-2 environments are also described.
  - Transaction management. Issues that apply to applications that perform multisite updates, and to applications that perform concurrent transactions, are described.
  - Applications in partitioned database environments. Directed DSS, local bypass, buffered inserts, and troubleshooting applications in partitioned database environments are described.

  - Commonly used application techniques. Information is provided on how to use generated and identity columns, declared temporary tables, and how to use savepoints to manage transactions.
  - The SQL statements that are supported for use in embedded SQL applications.
  - Applications that access host and iSeries environments. The issues that pertain to embedded SQL applications that access host and iSeries envirionments are described.
  - The simulation of EBCDIC binary collation.
- *Application Development Guide: Programming Server Applications* contains what you need to know for server-side programming including routines, large objects, user-defined types, and triggers. It includes information on:
  - Routines (stored procedures, user-defined functions, and methods), including:
    - Routine performance, security, library management considerations, and restrictions.
    - Registering and writing routines, including the CREATE statements and debugging.
    - Procedure parameter modes and parameter handling.
    - Procedure result sets.
    - UDF features including scratchpads and scalar and table functions.
    - SQL procedures including debugging, and condition handling.
    - Parameter styles, authorizations, and binding of external routines.
    - Language-specific considerations for C, Java, and OLE automation routines.
    - Invoking routines
    - Function selection and passing distinct types and LOBs to functions.
    - Code pages and routines.
  - Large objects, including LOB usage and locators, reference variables, and CLOB data.
  - User-defined distinct types, including strong typing, defining and dropping UDTs, creating tables with structured types, using distinct types and typed tables for specific applications, manipulating distinct types and casting between them, and performing comparisons and assignments with distinct types, including UNION operations on distinctly typed columns.
  - User-defined structured types, including storing instances and instantiation, structured type hierarchies, defining structured type behavior, the dynamic dispatch of methods, the comparison, casting, and constructor functions, and mutator and observer methods for structured types.

- Typed tables, including creating, dropping, substituting, storing objects, defining system-generated object identifiers, and constraints on object identifier columns.
- Reference types, including relationships between objects in typed tables, semantic relationships with references, and referential integrity versus scoped references.
- Typed tables and typed views, including structured types as column types, transform functions and transform groups, host language program mappings, and structured type host variables.
- Triggers, including INSERT, UPDATE, and DELETE triggers, interactions with referential constraints, creation guidelines, granularity, activation time, transition variables and tables, triggered actions, multiple triggers, and synergy between triggers, constraints, and routines.

• *Application Development Guide: Building and Running Applications* contains what you need to know to build and run DB2 applications on the operating systems supported by DB2:
  - AIX
  - HP-UX
  - Linux
  - Solaris
  - Windows

  It includes information on:
  - How to set up your application development environment, including specific instructions for Java and SQL procedures, how to set up the sample database, and how to migrate your applications from previous versions of DB2.
  - DB2 supported servers and software to build applications, including supported compilers and interpreters.
  - The DB2 sample program files, makefiles, build files, and error-checking utility files.
  - How to build and run Java applets, applications, and routines.
  - How to build and run SQL procedures.
  - How to build and run C/C++ applications and routines.
  - How to build and run IBM and Micro Focus COBOL applications and routines.
  - How to build and run REXX applications on AIX and Windows.
  - How to build and run applications with ActiveX Data Objects (ADO) using Visual Basic and Visual C++ on Windows.
  - How to build and run applications with remote data objects using Visual C++ on Windows.

Part 1. The Application Development Environment

Chapter 1. DB2 Support

This volume of the Application Development Guide describes DB2 support for application development. It provides the information you need to set up your environment for developing DB2 applications, and gives 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 8 on the following platforms:

- AIX
- HP-UX
- Linux
- Solaris Operating Environment
- Windows operating systems

DB2 Application Development Client

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

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

- **Precompilers for C/C++, COBOL, and Fortran**, (providing the language is supported for that platform).
- **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 operating systems, and from various other vendors for many of the other supported platforms. For Windows 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 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)**. JDK 1.3.1 and Java Runtime Environment (JRE) 1.3.1 from IBM for AIX, IBM Developer Kit and Runtime Environment, Version 1.3.1, for Linux and for Windows, HP-UX Software Developer's Kit and Runtime Environment 1.3.1 for HP-UX, and the Java Development Kit 1.3.1 for Solaris from Sun Microsystems. Except for Solaris, the respective JDK for each operating system will be installed if any components are selected for install that require Java to run. If none are selected, the JDK can still be selected to be installed. On Solaris, JDK 1.3.1, which is shipped with DB2, must be installed by the user.
- **ActiveX Data Objects (ADO) and Object Linking and Embedding (OLE) Automation Routines (UDFs and Stored Procedures)** on Windows 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 operating systems.
- **DB2 Development Center**, a graphical application that supports the rapid development of routines (stored procedures and user-defined functions), and structured types. The Development Center provides a single development environment that supports the entire DB2 family ranging from the workstation to z/OS. You can launch the Development Center as a stand-alone application or from a DB2 Universal Database center, such as the Control Center, the Command Center, or the Task Center. The Development Center is implemented with Java, and all database connections are managed by using a Java Database Connectivity (JDBC) API. The Development Center also provides a DB2 Development Add-In for each of the following development environments:
  - Microsoft Visual C++, Version 6
  - Microsoft Visual Basic, Version 6

  - Microsoft Visual InterDev, Version 6
- **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, IBM COBOL on AIX uses this interface. Information on the set of Precompiler Services APIs is available from the PDF file, prepapi.pdf, at the anonymous FTP site:

  `ftp://ftp.software.ibm.com/ps/products/db2/info/`
- **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 UDB for z/OS and OS/390. If you migrate applications developed on a workstation to another platform, the Flagger saves you time by showing syntax incompatibilities.

Related reference:

- "PRECOMPILE" in the *Command Reference*
- "AIX Supported Software for Building and Running Applications" on page 9
- "HP-UX Supported Software for Building and Running Applications" on page 11
- "Linux Supported Software for Building and Running Applications" on page 12
- "Solaris Supported Software for Building and Running Applications" on page 14
- "Windows Supported Software for Building and Running Applications" on page 15

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. The following figure shows this relationship.

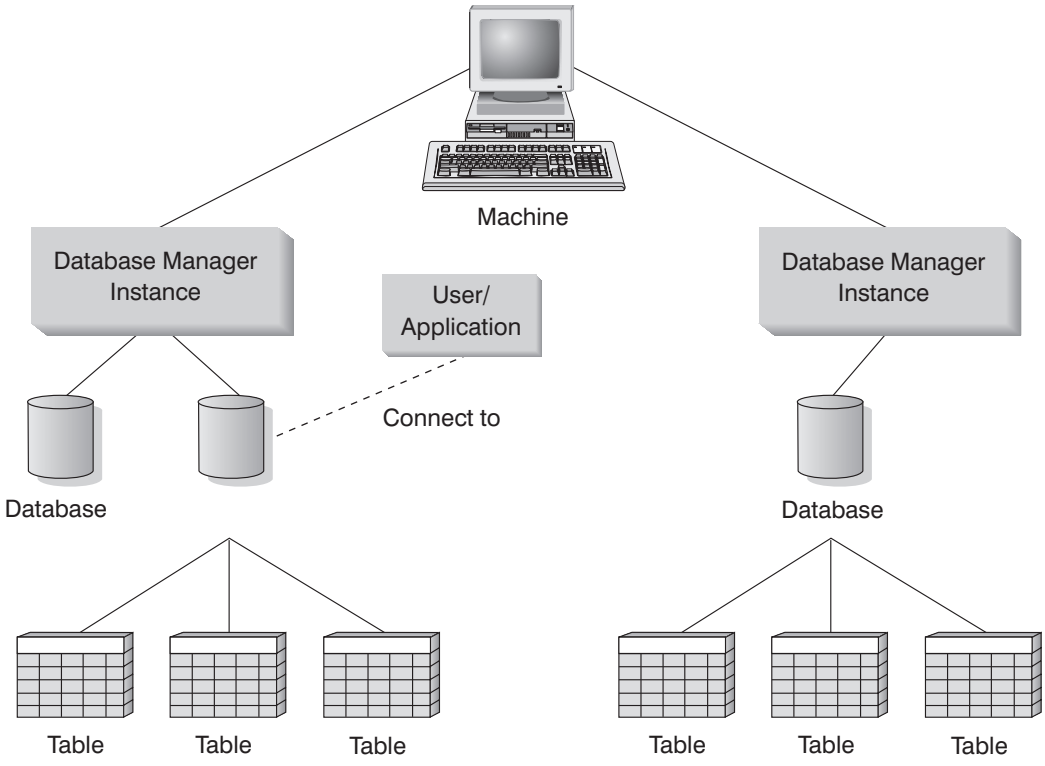

Figure 1. Database Manager Instances

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 7.1, and another running DB2 Universal Database Version 8.1. Prior to DB2 Version 8, within a version level only one release and FixPak level are supported. For example, DB2 Version 7.1 and DB2 Version 7.2 cannot coexist on a UNIX server. With DB2 Version 8, multiple FixPak levels can coexist on the same UNIX server.

With Windows® servers, you must have the same DB2 version, release, and FixPak level on each database manager instance. You cannot have one database manager instance running DB2 Universal Database Version 7.1 and another instance running DB2 Universal Database Version 8.1.

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

instance name

For UNIX platforms, this is a valid user name that you specify when you create the database manager instance.

For Windows operating systems, 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 Windows operating systems, the instance directory is `%DB2PATH%\`*instance_name*. The variable `%DB2PATH%` determines where DB2 is installed. The default installation value for `%DB2PATH%` is `\Program Files\IBM\SQLLIB`, so depending on which drive DB2 is installed, `%DB2PATH%` will point to *drive*`:\Program Files\IBM\SQLLIB`, unless the default value is changed.

The instance path on Windows servers is created based on either:

```
%DB2PATH%\%DB2INSTANCE%
```

(for example, `C:\Program Files\IBM\SQLLIB\DB2`)

or, if DB2INSTPROF is defined:

```
%DB2INSTPROF%\%DB2INSTANCE%
```

(for example, `C:\PROFILES\DB2`)

The DB2INSTPROF environment variable is used on Windows servers to support running DB2 on a network drive for which the client machine has only read access. In this case, DB2 will be set to point to *drive*`:\Program Files\IBM\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.

DB2 Supported Servers

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

- DB2 for AIX
- DB2 for HP-UX
- DB2 for Linux

- DB2 for OS/2
- DB2 for NUMA-Q
- DB2 for Solaris
- DB2 for Windows NT/2000/XP/.NET Server
- Distributed Relational Database Architecture (DRDA)-compliant application servers, such as:
  - DB2 for z/OS and OS/390
  - DB2 for AS/400 and iSeries
  - DB2 for VSE & VM (formerly SQL/DS for VM and VSE)
  - DRDA-compliant application servers from database vendors other than IBM.

Notes:

1. DB2 Version 8 HP-UX 64-bit servers do not support running DB2 Version 7 64-bit local applications.
2. DB2 for OS/2 is not available for DB2 Version 8
3. DB2 for NUMA-Q runs on the PTX operating system, and is only available for DB2 Version 7
4. DB2 Version 8 Windows 64-bit servers support connections from DB2 Version 6 and Version 7 32-bit clients only for SQL requests. Connections from Version 7 64-bit clients are not supported.

DB2 Supported Software for Building and Running Applications

DB2 Version 8 supports compilers, interpreters, and related software for the following operating systems:

- AIX
- HP-UX
- Linux
- Solaris
- Windows operating systems

DB2 supports 32-bit and 64-bit versions of each of these operating systems. There are differences for building applications in 32-bit and 64-bit environments in most cases on these operating systems. However, DB2 supports running 32-bit applications and routines (stored procedures and user-defined functions) on all supported 64-bit operating system environments except Linux IA64.

The compiler information given for each of these operating systems assumes that you are using the DB2 precompiler for that operating system, and not the precompiler support that may be built into one of the listed compilers.

For the latest DB2 compiler information and related software updates, visit the DB2 application development Web site at:

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

Note the following points about software support:

- **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.
- **Perl**. At the time of printing, Release 0.76 of the DB2 UDB driver (DBD::DB2) for the Perl Database Interface (Perl DBI) Version 0.93 or later is available for AIX, HP-UX, Linux, Solaris and Windows. The latest driver can be downloaded from:

  `http://www.ibm.com/software/data/db2/perl`
- **PHP**. PHP can be used as a method to access DB2 from web-based applications. PHP is a server-side, HTML-embedded, cross-platform scripting language. It supports DB2 access using the Unified-ODBC access method, in which the user-level PHP communicates to DB2 using ODBC calls. Unlike standard ODBC, with the Unified-ODBC method, communication is directly to the DB2 CLI layer, not through the ODBC layer. For more information about using PHP with DB2, search the DB2 support site:

  `www.ibm.com/software/data/db2/udb/winos2unix/support`

Related reference:

- "AIX Supported Software for Building and Running Applications" on page 9
- "HP-UX Supported Software for Building and Running Applications" on page 11
- "Linux Supported Software for Building and Running Applications" on page 12
- "Solaris Supported Software for Building and Running Applications" on page 14
- "Windows Supported Software for Building and Running Applications" on page 15

Supported Software by Platform

AIX Supported Software for Building and Running Applications

DB2 for AIX supports the following operating systems:

AIX/6000
Version 4.3.3 with maintenance level 9, and later

Version 5.1.0 with maintenance level 1, and later

DB2 for AIX supports the following programming languages and compilers:

C IBM C for AIX Version 5.0

C++ IBM VisualAge C++ Version 5.0

COBOL

IBM COBOL Set for AIX Version 1.1

Micro Focus COBOL Server Express Version 2.0.10

Fortran

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

Java Java Development Kit (JDK) Version 1.3.1 and Java Runtime Environment (JRE) Version 1.3.1 for AIX from IBM (installed as necessary by DB2)

Note: DB2 will install this JDK if any components are selected for install that require Java to run. If none are selected, the JDK can still be selected to be installed.

Perl Perl 5.004_04 or later, DBI 0.93 or later

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

For DB2 for AIX software support updates visit the DB2 application development Web site:

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

Related reference:

- "DB2 Supported Software for Building and Running Applications" on page 8
- "Installation requirements for partitioned DB2 servers (AIX)" in the *Quick Beginnings for DB2 Servers*
- "AIX C Application Compile and Link Options" on page 153
- "AIX C Routine Compile and Link Options" on page 158
- "AIX C++ Application Compile and Link Options" on page 164
- "AIX C++ Routine Compile and Link Options" on page 170
- "AIX IBM COBOL Application Compile and Link Options" on page 182

- "AIX IBM COBOL Routine Compile and Link Options" on page 185
- "AIX Micro Focus COBOL Application Compile and Link Options" on page 189
- "AIX Micro Focus COBOL Routine Compile and Link Options" on page 192
- "Installation requirements for DB2 servers (AIX)" in the *Quick Beginnings for DB2 Servers*

HP-UX Supported Software for Building and Running Applications

DB2 for HP-UX supports the following operating systems:

HP-UX
Versions 11 and 11i

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

C HP C Compiler version B.11.11.02

C++ HP aC++ Version A.03.31

COBOL
Micro Focus COBOL Version 4.1

Fortran
HP Fortran/9000 Version 10.0

HP-UX F77 B.11.00.01

Java Software Developer's Kit and Runtime Environment 1.3.1 for HP-UX 11.0 and 11i PA-RISC from Hewlett-Packard (installed as necessary by DB2)

Note: DB2 will install this SDK if any components are selected for install that require Java to run. If none are selected, the SDK can still be selected to be installed.

Perl Perl 5.004_04 or later, DBI 0.93 or later

For DB2 for HP-UX software support updates visit the DB2 application development Web site:

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

Related reference:

- "DB2 Supported Software for Building and Running Applications" on page 8
- "Installation requirements for partitioned DB2 servers (HP-UX)" in the *Quick Beginnings for DB2 Servers*
- "HP-UX C Application Compile and Link Options" on page 200

- "HP-UX C Routine Compile and Link Options" on page 206
- "HP-UX C++ Application Compile and Link Options" on page 211
- "HP-UX C++ Routine Compile and Link Options" on page 218
- "HP-UX Micro Focus COBOL Application Compile and Link Options" on page 225
- "HP-UX Micro Focus COBOL Routine Compile and Link Options" on page 228
- "Installation requirements for DB2 servers (HP-UX)" in the *Quick Beginnings for DB2 Servers*

Linux Supported Software for Building and Running Applications

DB2 for Linux for Intel x86 (32-bit architecture) supports the following operating system environment:

Linux kernel Version 2.4.9 or later, glibc Version 2.2.4 or later, and rpm (required to install)

DB2 for Linux on S/390 supports the following operating system environment:

One of the following:

- RedHat v7.2
- SuSE SLES-7 for Linux on S/390

DB2 for Linux for Intel x86 and S/390 supports the following programming languages and compilers:

C/C++ For Linux on Intel: GNU/Linux gcc and g++ versions 2.95.3 and 2.96

For Linux/390: GNU/Linux gcc and g++ version 2.95.3

Please visit the DB2 application development Web site for information on gcc and g++ version 3.0 support for Linux on Intel and for future version support for Linux/390:

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

Java For Linux on Intel: IBM Developer Kit and Runtime Environment for Linux, Java 2 Technology Edition, Version 1.3.1, 32-bit version (installed as necessary by DB2)

For Linux/390: IBM zSeries Developer Kit for Linux, Java 2 Technology Edition (at the Sun 1.3.1 SDK level, and installed as necessary by DB2)

Note: DB2 will install the appropriate Developer Kit if any components are selected for install that require Java to run. If none are selected, the Developer Kit can still be selected to be installed.

Perl Perl 5.004_04 or later, DBI 0.93 or later

REXX For Linux on Intel: Object REXX Interpreter for Linux Version 2.1

For Linux/390: Object REXX 2.2.0 for Linux/390

DB2 for Linux on IA64 supports the following operating system environment:

One of the following:

- Red Hat 7.2
- SuSE SLES-7 for Linux on IA64

DB2 for Linux on IA64 supports the following programming languages and compilers:

C GNU/Linux gcc version 3.0.2

C++ GNU/Linux g++ version 3.0.2

Java IBM Developer Kit and Runtime Environment for Linux, Java 2 Technology Edition, Version 1.3.1, 64-bit version (installed as necessary by DB2). To use this JDK, you must also have installed gcc 3.0.2 and the gcc3 libstdc++ runtime libraries.

Note: DB2 will install this Developer Kit if any components are selected for install that require Java to run. If none are selected, the Developer Kit can still be selected to be installed.

Perl Perl 5.6

Note: Running DB2 32-bit applications or routines (stored procedures and user-defined functions) is not supported on Linux IA64.

For DB2 for Linux software support updates visit the DB2 application development Web site:

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

Related reference:

- "DB2 Supported Software for Building and Running Applications" on page 8
- "Installation requirements for partitioned DB2 servers (Linux)" in the *Quick Beginnings for DB2 Servers*
- "Linux C Application Compile and Link Options" on page 234

- "Linux C Routine Compile and Link Options" on page 240
- "Linux C++ Application Compile and Link Options" on page 245
- "Linux C++ Routine Compile and Link Options" on page 250
- "Installation requirements for DB2 Personal Edition (Linux)" in the *Quick Beginnings for DB2 Personal Edition*
- "Installation requirements for DB2 servers (Linux)" in the *Quick Beginnings for DB2 Servers*

Solaris Supported Software for Building and Running Applications

DB2 for Solaris supports the following operating system:

Solaris
Solaris 7, Solaris 8, and Solaris 9

DB2 for Solaris supports the following programming languages and compilers:

C Forte C Versions 5.0, 6, and 6.1

Note: These compiler versions used to be called ″SPARCompiler″.

C++ Forte C++ Versions 5.0, 6, and 6.1

Note: These compiler versions used to be called ″SPARCompiler″.

COBOL
Micro Focus COBOL Server Express Version 2.0.10

Fortran
SPARCompiler Fortran Versions 4.2 and 5.0

Java Java Development Kit (JDK) Version 1.3.1 for Solaris from Sun Microsystems (shipped with DB2)

Perl Perl 5.004_04 or later, DBI 0.93 or later

For DB2 for Solaris software support updates visit the DB2 application development Web site:

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

Related reference:

- "DB2 Supported Software for Building and Running Applications" on page 8
- "Installation requirements for partitioned DB2 servers (Solaris)" in the *Quick Beginnings for DB2 Servers*
- "Solaris C Application Compile and Link Options" on page 258
- "Solaris C Routine Compile and Link Options" on page 264

- "Solaris C++ Application Compile and Link Options" on page 270
- "Solaris C++ Routine Compile and Link Options" on page 277
- "Solaris Micro Focus COBOL Application Compile and Link Options" on page 283
- "Solaris Micro Focus COBOL Routine Compile and Link Options" on page 287
- "Installation requirements for DB2 servers (Solaris)" in the *Quick Beginnings for DB2 Servers*

Windows Supported Software for Building and Running Applications

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

Microsoft Windows XP

Microsoft Windows .NET

Microsoft Windows 2000

Microsoft Windows NT
Version 4.0 with Service Pack 6a or later.

Microsoft Windows ME

Microsoft Windows 98

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

Basic Microsoft Visual Basic Version 4.2 and Version 5.0

Microsoft Visual Basic 6.0 Professional Edition

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

Intel C++ Compiler for 32-bit applications Version 5 or later

COBOL

Micro Focus COBOL Version 4.0.20

Micro Focus COBOL Net Express Version 3.1.0

IBM VisualAge COBOL Version 2.0

REXX IBM Object REXX for Windows NT/95 Version 1.1

For information on obtaining IBM Object REXX for Windows, visit:

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

Java IBM Developer Kit and Runtime Environment for Windows, Java 2 Technology Edition, Version 1.3.1, 32-bit version (installed as necessary by DB2)

Note: DB2 will install this Developer Kit if any components are selected for install that require Java to run. If none are selected, the Developer Kit can still be selected to be installed.

Java Development Kit (JDK) 1.3.1 for Win32 from Sun Microsystems

Perl Perl 5.004_04, DBI 0.93

Microsoft Windows Scripting Host
Version 5.1

DB2 for Windows 64-bit operating systems supports the following:

Microsoft Windows XP 64-bit Edition

Microsoft Windows .NET Server 64-bit Edition

DB2 for Windows 64-bit operating systems supports the following programming languages:

C/C++ Intel C++ Compiler for Itanium Version 6.0

Microsoft's C/C++ compiler for the Intel Itanium architecture

Java IBM Developer Kit and Runtime Environment for Windows, Java 2 Technology Edition, Version 1.3.1, 64-bit version (installed as necessary by DB2)

Note: DB2 will install this Developer Kit if any components are selected for install that require Java to run. If none are selected, the Developer Kit can still be selected to be installed.

Microsoft Windows Scripting Host
Version 5.1

Note: Windows .NET Server includes all of the following:

- Windows .NET Web Server
- Windows .NET Standard Server
- Windows .NET Enterprise Server
- Windows .NET Datacenter Server

For DB2 for Windows software support updates visit the DB2 application development Web site:

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

Related reference:

- "DB2 Supported Software for Building and Running Applications" on page 8

- "Installation requirements for DB2 servers (Windows)" in the *Quick Beginnings for DB2 Servers*
- "Windows C/C++ Application Compile and Link Options" on page 303
- "Windows C/C++ Routine Compile and Link Options" on page 308
- "Windows IBM COBOL Application Compile and Link Options" on page 313
- "Windows IBM COBOL Routine Compile and Link Options" on page 316
- "Windows Micro Focus COBOL Application Compile and Link Options" on page 320
- "Windows Micro Focus COBOL Routine Compile and Link Options" on page 322
- "Installation requirements for DB2 Personal Edition (Windows)" in the *Quick Beginnings for DB2 Personal Edition*
- "Installation requirements for a partitioned DB2 server (Windows)" in the *Quick Beginnings for DB2 Servers*

Chapter 2. Setup

General Setup Information

For DB2 CLI setup information, see the *CLI Guide and Reference*.

Setting Up the Application Development Environment

In order to build and run DB2 applications, you must use a compiler or interpreter for one of the supported programming languages for your operating system. You have to set up the DB2 environment and configure it for your development requirements. There are certain procedures to follow in order to migrate DB2 applications from a previous version of DB2. Also, you may want to create the DB2 sample database for testing purposes.

Prerequisites:

Ensure the environment for the DB2-supported compiler or interpreter you plan to use 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.

Install the Application Development client on the client or server workstation you are using. If you are developing applications from a remote client, ensure

your client machine can reach the machine on which the DB2 database server resides. Also ensure your client can successfully connect to the database. You can use the command line processor (CLP) or client configuration assistant (CCA) to test connectivity.

Procedure:

To set up your application development environment:

1. Unless the defaults are acceptable, Update the Database Manager Configuration File
2. If you will be programming with DB2 CLI, Java, or SQL procedures, you have to configure your environment before you perform any platform-specific changes with the instructions in the following:
   - Setting Up the CLI Environment
   - Setting Up the Java environment
   - Setting Up the SQL Procedures environment
3. Configure your operating system environment with the instructions in the following:
   - Setting Up the UNIX Environment
   - Setting Up the Windows Environment
4. Optional: Setting Up the sample Database

Related concepts:

- "Database Manager Instances" on page 5
- "Migrating Applications" on page 48

Related tasks:

- "Updating the Database Manager Configuration File" on page 21
- "Setting Up the CLI Environment" in the *CLI Guide and Reference, Volume 1*
- "Setting Up the Java Environment" on page 22
- "Setting Up the SQL Procedures Environment" on page 23
- "Setting Up the UNIX Application Development Environment" on page 25
- "Setting Up the Windows Application Development Environment" on page 35
- "Setting Up the sample Database" on page 42

Related reference:

- "DB2 Application Development Client" on page 3
- "DB2 Supported Servers" on page 7
- "DB2 Supported Software for Building and Running Applications" on page 8

- "AIX Supported Software for Building and Running Applications" on page 9
- "HP-UX Supported Software for Building and Running Applications" on page 11
- "Linux Supported Software for Building and Running Applications" on page 12
- "Solaris Supported Software for Building and Running Applications" on page 14
- "Windows Supported Software for Building and Running Applications" on page 15

Updating the Database Manager Configuration File

This file contains important settings for application development.

For routines (stored procedures and UDFs), the keyword KEEPFENCED has the default value YES. This keeps the routine process alive. If you are developing a routine, you may want to test loading the same shared library a number of times. This default setting may interfere with reloading the library. It is best to change the value of this keyword to NO while developing routines, and then change it back to YES when you are ready to load the final version of your shared library.

For threadsafe routines, the process used to run the routine remains on the instance after completion (only the thread within the process that is used to invoke the routine terminates). For this reason, when developing a new routine you should define the routine as not threadsafe. Then, if appropriate, enable the routine to run in a threadsafe mode when it is put into production.

Note: KEEPFENCED was known as KEEPDARI in previous versions of DB2.

For Java application development, you need to update the JDK_PATH keyword with the path where the Java Development Kit (JDK) is installed.

Note: JDK_PATH was known as JDK11_PATH in previous versions of DB2.

Procedure:

To change these settings enter:

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

For example, to set the keyword KEEPFENCED to NO:

```
db2 update dbm cfg using KEEPFENCED NO
```

To set the JDK_PATH keyword to the directory /home/db2inst/jdk13:

```
db2 update dbm cfg using JDK_PATH /home/db2inst/jdk13
```

To view the current settings in the database manager configuration file, enter:

```
db2 get dbm cfg
```

Note: On Windows, you need to enter these commands in a DB2 command window.

Related tasks:

- "Setting Up the Java Environment" on page 22

Related reference:

- "CREATE FUNCTION statement" in the *SQL Reference, Volume 2*
- "CREATE PROCEDURE statement" in the *SQL Reference, Volume 2*
- "GET DATABASE MANAGER CONFIGURATION" in the *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION" in the *Command Reference*

Setting Up the Java Environment

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.

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

Procedure:

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, `DbInfo.class`, you can use the following command:

```
java -native DbInfo
```

You can specify native threads as the default thread support for some Java Virtual Machines by setting the THREADS_FLAG environment variable to ″native″. This documentation 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.

When the above are installed and working, you can set up your particular operating system Java environment by following the steps in one of the following:
- Setting Up the UNIX Java Environment
- Setting Up the Windows Java Environment

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

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

Related tasks:

- “Setting Up the UNIX Java Environment” on page 26
- “Setting Up the Windows Java Environment” on page 38

Related reference:

- “db2profc - DB2 SQLj Profile Customizer” in the *Command Reference*
- “db2profp - DB2 SQLj Profile Printer” in the *Command Reference*

Setting Up the SQL Procedures Environment

Compiler configuration is done with two DB2 registry variables:

DB2_SQLROUTINE_COMPILER_PATH
: Is assigned the pathname of a script that sets the compiler environment variables.

DB2_SQLROUTINE_COMPILE_COMMAND
Is assigned the full command DB2 uses to compile the C files generated for SQL procedures.

You can either use the `db2set` command or use the SQL Stored Procedures Build Options dialog from the Development Center to set the value of these DB2 registry variables. Using the SQL Stored Procedures Build Options dialog eliminates the need to physically access the database server or to restart it in order for the changes to take effect.

The steps below demonstrate the `db2set` command.

Prerequisites:

For SQL procedures support you have to install the Application Development Client and a DB2 supported C or C++ compiler on the server.

Procedure:

To configure your SQL procedures environment, do the steps in one of the following, depending on your platform:

- Setting Up the UNIX SQL Procedures Environment
- Setting Up the Windows SQL Procedures Environment

Related concepts:

- "DB2 registry and environment variables" in the *Administration Guide: Performance*

Related tasks:

- "Setting Up the UNIX SQL Procedures Environment" on page 31
- "Setting Up the Windows SQL Procedures Environment" on page 40
- "Creating SQL Procedures" on page 133
- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on UNIX" on page 134
- "Calling SQL Procedures with Client Applications on Windows" on page 135
- "Distributing Compiled SQL Procedures" on page 140
- "Rebinding SQL Procedures" on page 141

UNIX

For UNIX DB2 CLI setup information, see the *CLI Guide and Reference*.

Setting Up the UNIX Application Development Environment

You need to set environment variables for your database instance. Each database manager instance has two files, `db2profile` and `db2cshrc`, which are scripts to set the environment variables for that instance.

Procedure:

Run the correct script for the shell you are using:

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.

If you include this command in your `.profile` or `.login` file, the command runs automatically when you log on.

If you will be using ODBC or DB2 CLI, Java, or SQL Procedures, do the steps in the following topics:

- Setting Up the UNIX ODBC Environment
- Setting Up the UNIX Java Environment
- Setting Up the UNIX SQL Procedures Environment

Related concepts:

- "UNIX Environment Variable Settings" on page 26

Related tasks:

- "Setting Up the UNIX ODBC Environment" in the *CLI Guide and Reference, Volume 1*
- "Setting Up the UNIX Java Environment" on page 26
- "Setting Up the UNIX SQL Procedures Environment" on page 31

Related reference:

- "AIX Supported Software for Building and Running Applications" on page 9
- "HP-UX Supported Software for Building and Running Applications" on page 11
- "Linux Supported Software for Building and Running Applications" on page 12
- "Solaris Supported Software for Building and Running Applications" on page 14

UNIX Environment Variable Settings

Depending on the UNIX® platform you are on, values for the following environment variables are set, either in `db2profile` (for bash or korn shell) or `db2cshrc` (for C shell), and a call to these files are put in the `.profile` (bash or korn shell) or `.login` (C shell) file.

AIX:

- PATH, includes several DB2® directories including `sqllib/bin`
- LIBPATH, includes the directory `sqllib/lib` (see note below)

HP-UX:

- PATH, includes several DB2 directories including `sqllib/bin`
- SHLIB_PATH (32-bit and 64-bit) or LD_LIBRARY_PATH (64-bit), includes the directory `sqllib/lib` (see note below)

Linux and Solaris:

- PATH, includes several DB2 directories including `sqllib/bin`
- LD_LIBRARY_PATH, includes the directory `sqllib/lib` (see note below)

Note: If you are running a local 32-bit application in a 64-bit DB2 instance, LIBPATH, SHLIB_PATH, and LD_LIBRARY_PATH should contain `sqllib/lib32` instead of `sqllib/lib`.

The blank files `sqllib/userprofile` and `sqllib/usercshrc` are created during instance creation to allow users to place their own instance environmental settings. These files will not be modified during an instance update (`db2iupdt`) in any DB2 FixPak or future version install. If you do not want the new environment settings in the `db2profile` or `db2cshrc` scripts, you can override them using the corresponding "user" script, which is called at the end of the `db2profile` or `db2cshrc` script. During an instance migration (`db2imigr`), the user scripts are copied over so that your environment modifications will still be in use. These user scripts are only available starting with DB2 Version 7.

Related tasks:

- "Setting Up the UNIX Application Development Environment" on page 25

Java

Setting Up the UNIX Java Environment

To run JDBC and SQLJ programs on UNIX with DB2 JDBC support, commands to update your 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". (on HP-UX, Linux and Solaris only)

2. CLASSPATH includes:
   - "." (the current directory)
   - the file `sqllib/java/db2java.zip`
   - the file `sqllib/java/db2jcc.jar`

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`

Procedure:

To run DB2 Java routines (stored procedures and 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 JDK_PATH /home/db2inst/jdk13
```

where */home/db2inst/jdk13* is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the `JDK_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 `JDK_PATH` field appears near the beginning of the output.

When the above are installed and working, you can set up your specific UNIX operating system environment by following the steps in one of the following:
- Setting up the AIX Java Environment
- Setting up the HP-UX Java Environment
- Setting up the Linux Java Environment
- Setting up the Solaris Java Environment

Related tasks:
- "Setting Up the AIX Java Environment" on page 28
- "Setting Up the HP-UX Java Environment" on page 28
- "Setting Up the Linux Java Environment" on page 29
- "Setting Up the Solaris Java Environment" on page 30
- "Updating the Database Manager Configuration File" on page 21

Related reference:

- "GET DATABASE MANAGER CONFIGURATION" in the *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION" in the *Command Reference*

Setting Up the AIX Java Environment

Procedure:

To build Java applications on AIX with DB2 JDBC support, you need:

1. Java Development Kit (JDK) Version 1.3.1 and Java Runtime Environment (JRE) Version 1.3.1 for AIX from IBM.
2. DB2 Java Enablement, provided on DB2 Universal Database Version 8 for AIX clients and servers.

Related concepts:

- "Java Sample Programs" on page 107
- "Java Applet Considerations" on page 109

Related tasks:

- "Setting Up the sample Database" on page 42

Related reference:

- "AIX Supported Software for Building and Running Applications" on page 9

Setting Up the HP-UX Java Environment

Procedure:

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

1. Software Developer's Kit and Runtime Environment 1.3.1 for HP-UX 11.0 and 11i PA-RISC from Hewlett-Packard.
2. DB2 Java Enablement, provided on DB2 Universal Database Version 8 for HP-UX clients and servers.

To run Java routines (stored procedures and user-defined functions), make sure the shared library path is similar to the following:

```
export SHLIB_PATH=$JAVADIR/jre/lib/PA_RISC:\
                  $JAVADIR/jre/lib/PA_RISC/classic:\
                  $HOME/sqllib/lib:\
                  /usr/lib:$SHLIB_PATH
```

where $JAVADIR is normally set to /opt/java1.3 (the default location of the Java SDK).

Note: DB2 does nor support the HotSpot Java Virtual Machine for routines.

Related concepts:

- "Java Sample Programs" on page 107
- "Java Applet Considerations" on page 109

Related tasks:

- "Setting Up the sample Database" on page 42

Related reference:

- "HP-UX Supported Software for Building and Running Applications" on page 11

Setting Up the Linux Java Environment

Procedure:

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

1. One of the following:
   - For Linux on Intel: IBM Developer Kit and Runtime Environment for Linux, Java 2 Technology Edition, Version 1.3.1, 32-bit version
   - For Linux/390: IBM zSeries Developer Kit for Linux, Java 2 Technology Edition
   - For Linux on IA64: IBM Developer Kit and Runtime Environment for Linux, Java 2 Technology Edition, Version 1.3.1, 64-bit version
2. DB2 Java Enablement, provided on DB2 Universal Database Version 8 for Linux clients and servers.

To run Java stored procedures or user-defined functions, the Linux run-time linker must be able to access certain Java shared libraries, and DB2 must be able to load these libraries and the Java virtual machine. Since the program that does this loading runs with setuid privileges, it will only look for the dependent libraries in /lib or /usr/lib.

You can add the location of the Java shared libraries to /etc/ld.so.conf, but we recommend creating symbolic links in /usr/lib to point to these libraries. For the IBM JDK 1.3, you need symbolic links to libjava.so, libjvm.so, and libhpi.so. You can create the symbolic links by running the following commands as root:

```
cd /usr/lib
ln -fs JAVAHOME/jre/bin/libjava.so .
ln -fs JAVAHOME/jre/bin/classic/libjvm.so .
ln -fs JAVAHOME/jre/bin/libhpi.so .
```

where *JAVAHOME* is the base directory for the JDK. If DB2 cannot find these libraries, you will get a -4301 error when trying to run a Java routine, and there will be messages in the administration notification log about libraries not found.

If you decide to add the location of the Java shared libraries to `/etc/ld.so.conf`, you must refresh the run-time linker cache by running the following command as root:

```
bash# ldconfig
```

Related concepts:

- "Java Sample Programs" on page 107
- "Java Applet Considerations" on page 109

Related tasks:

- "Setting Up the sample Database" on page 42

Related reference:

- "Linux Supported Software for Building and Running Applications" on page 12

Setting Up the Solaris Java Environment

Procedure:

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. Java Development Kit (JDK) Version 1.3.1 for Solaris from Sun Microsystems.
2. DB2 Java Enablement, provided on DB2 Universal Database Version 8 for Solaris clients and servers.

Related concepts:

- "Java Sample Programs" on page 107
- "Java Applet Considerations" on page 109

Related tasks:

- "Setting Up the sample Database" on page 42

Related reference:

- "Solaris Supported Software for Building and Running Applications" on page 14

SQL Procedures

Setting Up the UNIX SQL Procedures Environment

SQL procedures require the setting up of two environment variables on the server with your compiler configuration:
DB2_SQLROUTINE_COMPILER_PATH and
DB2_SQLROUTINE_COMPILE_COMMAND.

Restrictions:

The instance owner must belong to the primary group of the fenced id.

Procedure:

The first time you compile a stored procedure, DB2 will generate the executable script file `$HOME/sqllib/function/routine/sr_cpath` (which contains the default values for the compiler environment variables). If the default values are not appropriate for your compiler, do one of the following:

- edit the `sr_cpath` script file.
- set the DB2_SQLROUTINE_COMPILER_PATH DB2 registry variable to contain the full path name of another executable script that specifies the desired settings.

The installation of the Application Development Client provides a default compilation command that works for at least one of the compilers supported on each server platform:

AIX IBM C for AIX Version 5.0

HP-UX
HP aC++ Version A.03.31

Linux GNU/Linux g++

Solaris
Forte C++ Versions 5.0, 6, and 6.1

To use other compilers, or to customize the default command, you must set the `DB2_SQLROUTINE_COMPILE_COMMAND` DB2 registry variable as follows:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=<compile_command>
```

where <compile_command> is the C or C++ compilation command, including the options and parameters required to create routines. A `db2start` command must be executed for the setting to take effect.

In the compile 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. An example using both these keywords can be seen in the command to set the default values of DB2_SQLROUTINE_COMPILE_COMMAND for IBM C for AIX Version 5.0:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=xlc -I$HOME/sqllib/include \
SQLROUTINE_FILENAME.c -bE:SQLROUTINE_FILENAME.exp -e SQLROUTINE_ENTRY \
-o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -ldb2
```

Related tasks:

- "Retaining Intermediate Files for SQL Procedures" on page 137
- "Customizing Precompile and Bind Options for SQL Procedures" on page 138
- "Backing Up and Restoring SQL Procedures" on page 139
- "Creating SQL Procedures" on page 133
- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on UNIX" on page 134
- "Distributing Compiled SQL Procedures" on page 140

Related reference:

- "UNIX Default DB2 SQLROUTINE COMPILE COMMAND Values" on page 32
- "AIX Supported Software for Building and Running Applications" on page 9
- "HP-UX Supported Software for Building and Running Applications" on page 11
- "Linux Supported Software for Building and Running Applications" on page 12
- "Solaris Supported Software for Building and Running Applications" on page 14

UNIX Default DB2 SQLROUTINE COMPILE COMMAND Values

The following are the commands to set the default values for the DB2_SQLROUTINE_COMPILE_COMMAND for C or C++ compilers on supported UNIX server platforms:

AIX To use IBM C for AIX Version 5.0:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=xlc -I$HOME/sqllib/include \
SQLROUTINE_FILENAME.c -bE:SQLROUTINE_FILENAME.exp \
-e SQLROUTINE_ENTRY -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -ldb2
```

To use IBM VisualAge C++ 5.0:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=xlC -I$HOME/sqllib/include \
SQLROUTINE_FILENAME.C -bE:SQLROUTINE_FILENAME.exp \
-e SQLROUTINE_ENTRY -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -ldb2
```

This is the default compile command if the DB2_SQLROUTINE_COMPILE_COMMAND DB2 registry variable is not set.

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

To use IBM VisualAge C++ for AIX Version 5 incremental compiler:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="vacbld"
```

If you do not specify the configuration file after the `vacbld` command, DB2 will create the following default configuration file at the first attempt to create an SQL procedure:

```
$HOME/sqllib/function/routine/sqlproc.icc
```

You can specify your own configuration file when setting the DB2 registry value for `DB2_SQLROUTINE_COMPILE_COMMAND`:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="vacbld \
  $HOME/sqllib/function/sqlproc.icc"
```

HP-UX

To use HP C Compiler Version B.11.11.02:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=cc +DAportable +ul -Aa +z \
-I$HOME/sqllib/include -c SQLROUTINE_FILENAME.c; \
ld -b -o SQLROUTINE_FILENAME SQLROUTINE_FILENAME.o \
-L$HOME/sqllib/lib -ldb2
```

To use HP aC++ Version A.03.31:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=aCC +DAportable +u1 +z -ext \
-I$HOME/sqllib/include SQLROUTINE_FILENAME.C -b \
-o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -ldb2
```

This is the default compile command if the DB2_SQLROUTINE_COMPILE_COMMAND DB2 registry variable is not set.

Note: To compile 64-bit SQL procedures on HP-UX, take out the `+DAportable` option and add the `+DA2.0W` option to the above commands.

Linux To use GNU/Linux gcc:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=cc -fpic \
-I$HOME/sqllib/include SQLROUTINE_FILENAME.c \
-shared -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -ldb2
```

To use GNU/Linux g++:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=g++ -fpic \
-I$HOME/sqllib/include SQLROUTINE_FILENAME.C \
-shared -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -ldb2
```

This is the default compile command if the DB2_SQLROUTINE_COMPILE_COMMAND DB2 registry variable is not set.

Solaris

To use Forte C Version 5.0:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=cc -xarch=v8plusa -Kpic \
-I$HOME/sqllib/include SQLROUTINE_FILENAME.c \
-G -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib \
-R$HOME/sqllib/lib -ldb2
```

To use Forte C++ Version 5.0:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=CC -xarch=v8plusa -Kpic \
-I$HOME/sqllib/include SQLROUTINE_FILENAME.C \
-G -o SQLROUTINE_FILENAME -L$HOME/sqllib/lib \
-R$HOME/sqllib/lib -ldb2
```

This is the default compile command if the DB2_SQLROUTINE_COMPILE_COMMAND DB2 registry variable is not set.

Notes:

1. The compiler option `-xarch=v8plusa` has been added to the default compiler command to avoid a problem where the compiler does not produce valid executables when linking with `libdb2.so`.
2. To compile 64-bit SQL procedures on Solaris, take out the `-xarch=v8plusa` option and add the `-xarch=v9` option to the above commands.

Related tasks:

- "Setting Up the UNIX SQL Procedures Environment" on page 31

Windows

For Windows DB2 CLI setup information, see the *CLI Guide and Reference*.

Setting Up the Windows Application Development Environment

When you install the DB2 AD Client on Windows XP, Windows .Net Server, 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. When you install the DB2 AD Client on Windows 98 or Windows ME, 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. Exercise caution when changing these environment variables. Do not change the DB2PATH environment variable. 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.

Procedure:

To override the environment variable settings, use any of the following:

- The Windows XP control panel
- The Windows .NET Server control panel
- The Windows NT control panel
- The Windows 2000 control panel
- The Windows 98 or Windows ME command window
- The Windows 98 or Windows ME autoexec.bat file

When using the variable %DB2PATH% in a command, put the full path in quotes, as in set LIB="%DB2PATH%\lib";%LIB%. The default installation value for this variable is `\Program Files\IBM\SQLLIB`, which contains a space, so not using quotes may cause an error.

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%
  ```

  Note: To enable cross-developing 64-bit applications from a 32-bit environment, you must ensure that the LIB environment variable points to %DB2PATH%\lib\Win64. This means you have to change it from the default setting. By default, the environment variable points to %DB2PATH%\lib, which is for developing 32-bit applications on 32-bit environments, or 64-bit applications on 64-bit environments.
- 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 manually, use the NET START DB2NTSECSERVER command.

  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 the ″Start″ button.
  2. Click ″Settings″.
  3. Click ″Control Panel″.
  4. In the Control Panel, click ″Services″.
  5. In the Services window, highlight ″DB2 Security Server″.
  6. If it does not have the settings ″Started″ and ″Automatic″ listed, click ″Startup″.
  7. Click ″Automatic″.
  8. Click ″OK″.
  9. Reboot your machine to have the settings take effect.

  Windows 2000 and Windows .NET Server

  1. Click the ″Start″ button.
  2. Click ″Settings″.
  3. Click ″Control Panel″.
  4. Click ″Administrative Tools″.
  5. Click ″Services″.

6. In the Services window, highlight ″DB2 Security Server″.
7. If it does not have the settings ″Started″ and ″Automatic″ listed, click ″Action″ from the top menu.
8. Click ″Properties″.
9. Make sure you are in the ″General″ tab.
10. Choose ″Automatic″ from the 'Startup Type' drop-down menu.
11. Click ″OK″.
12. Reboot your machine to have the settings take effect.

Windows XP

1. Click the ″Start″ button.
2. Click ″Settings″.
3. Click ″Control Panel″.
4. Click ″Performance and Maintenance″.
5. Click ″Administrative Tools″.
6. Click ″Services″.
7. In the Services window, highlight ″DB2 Security Server″.
8. If it does not have the settings ″Started″ and ″Automatic″ listed, click ″Action″ from the top menu.
9. Click ″Properties″.
10. Make sure you are in the ″General″ tab.
11. Choose ″Automatic″ from the 'Startup Type' drop-down menu.
12. Click ″OK″.
13. Reboot your machine to have the settings take effect.

The database manager on a Windows XP, Windows .NET Server, Windows NT, or a Windows 2000 environment is implemented as a service, and hence does not return errors or warnings when the service is started, though 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 event logs or the DB2 Administration Notification log for any errors that may have occurred during the running of these commands.

If you will be using DB2 CLI, Java, or SQL procedures, proceed to the appropriate task:

- Setting Up the Windows CLI Environment
- Setting Up the Windows Java Environment
- Setting Up the Windows SQL Procedures Environment

Related tasks:

- "Setting Up the Windows CLI Environment" in the *CLI Guide and Reference, Volume 1*
- "Setting Up the Windows Java Environment" on page 38
- "Setting Up the Windows SQL Procedures Environment" on page 40

Related reference:

- "Windows Supported Software for Building and Running Applications" on page 15

Java

Setting Up the Windows Java Environment

This topic provides the information you need to build and run DB2 Java programs in a Windows environment.

Procedure:

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

1. One of the following:
   - IBM Developer Kit and Runtime Environment for Windows, Java 2 Technology Edition, Version 1.3.1
   - Java Development Kit (JDK) 1.3.1 for Win32 from Sun Microsystems
2. DB2 Java Enablement, provided on DB2 Universal Database Version 8 for Windows clients and servers.

To run DB2 Java routines (stored procedures and 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 JDK_PATH c:\jdk13
```

where *c:\jdk13* is the path where the JDK is installed.

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

```
db2 update dbm cfg using JDK_PATH 'c:\Program Files\jdk13'
```

or use the short-form of the directory name which does not have the space:

```
db2 update dbm cfg using JDK_PATH c:\progra~1\jdk13
```

You can check the DB2 database manager configuration to verify the correct value for the JDK_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 JDK_PATH field appears near the beginning of the output.

The following commands can be put into a batch file to set your Java environment for the IBM Java Development Kits. The batch file must be run in a DB2 command window. Make sure you make all necessary path changes to suit your particular environment. Similar commands can be used for other supported JDKs.

Here are the commands for an example batch file to set up the Sun JDK 1.3.1 environment:

```
set JDKPATH=D:\JAVA\SUNjdk131
set PATH=%JDKPATH%\bin;%PATH%
set CLASSPATH=%CLASSPATH%;%JDKPATH%\lib\jdbc2_0-stdext.jar
db2 update dbm cfg using JDK_PATH %JDKPATH%
db2 terminate
db2stop
db2start
```

The batch file must be run in a DB2 Command window.

Related concepts:

- "Windows Java Environment Settings" on page 39
- "Java Sample Programs" on page 107
- "Java Applet Considerations" on page 109

Related tasks:

- "Setting Up the sample Database" on page 42

Related reference:

- "Windows Supported Software for Building and Running Applications" on page 15

Windows Java Environment Settings

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 sqllib\java\db2java.zip
- the file sqllib\java\db2jcc.jar

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: The Microsoft Software Developer's Kit for Java is not supported in DB2 Version 8. It cannot be used for SQLJ customization nor for running type 2 JDBC applications.

Related tasks:

- "Setting Up the Windows Java Environment" on page 38

SQL Procedures

Setting Up the Windows SQL Procedures Environment

SQL procedures are supported on the following Windows operating systems: Windows NT, Windows 2000, Windows XP, and Windows .NET Server.

SQL Procedures require setting two environment variables, DB2_SQLROUTINE_COMPILER_PATH and DB2_SQLROUTINE_COMPILE_COMMAND, with your compiler configuration on the server. If the environment variables for your compiler are set as SYSTEM variables, no configuration is needed.

Restrictions:

A problem exists for 64-bit Windows when trying to set the variable DB2_SQLROUTINE_COMPILER_PATH because it requires the complete path to a file and will not allow arguments/switches. So if a user specified the following, it would not work:

```
db2set DB2_SQLROUTINE_COMPILER_PATH="C:\MsSdk64\SetEnv.bat /XP64 /RETAIL"
```

The workaround is to create another batch file that calls Microsoft's setup batch file with the appropriate flags, for example:

```
db2set DB2_SQLROUTINE_COMPILER_PATH="C:\MsSdk64\SetEnvXP64.bat"
```

where the contents of `C:\MsSdk64\SetEnvXP64.bat` would be:

```
call C:\MsSdk64\SetEnv.bat /XP64 /RETAIL
```

This problem does not occur on Windows 32-bit environments because `vcvars32.bat` does not require nor accept any parameters.

Procedure:

Assuming the C++ compiler is installed on the C: drive, set the DB2_SQLROUTINE_COMPILER_PATH DB2 registry variable as follows:

For Microsoft Visual C++ Version 5.0:

```
db2set DB2_SQLROUTINE_COMPILER_PATH="c:\devstudio\vc\bin\vcvars32.bat"
```

For Microsoft Visual C++ Version 6.0:

```
db2set DB2_SQLROUTINE_COMPILER_PATH="c:\Micros~1\vc98\bin\vcvars32.bat"
```

Change the drive or the path, if necessary, to reflect the location of the C++ compiler on your system.

For the compile command, use the keyword SQLROUTINE_FILENAME to replace the filename for the generated SQC, C, PDB, DEF, EXP, messages log and shared library files. To use Microsoft Visual C++ Versions 5.0 and 6.0:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="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"
```

This is the default compile command if the DB2_SQLROUTINE_COMPILE_COMMAND DB2 registry variable is not set.

Here's an example of the environment setup commands for the Microsoft Visual C++ Version 6.0 compiler. The following commands can be executed by cutting and pasting them into a batch file and running the file in a DB2 command window. Be sure to make all necessary changes, including path settings, for your particular environment:

```
@echo on
rem Setting the SQL PROCEDURE environment:
db2set DB2_SQLROUTINE_COMPILER_PATH="c:\Micros~1\vc98\bin\vcvars32.bat"
db2set DB2_SQLROUTINE_COMPILE_COMMAND="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"
@echo off
```

Related tasks:

- "Retaining Intermediate Files for SQL Procedures" on page 137
- "Customizing Precompile and Bind Options for SQL Procedures" on page 138
- "Backing Up and Restoring SQL Procedures" on page 139

- "Creating SQL Procedures" on page 133
- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on Windows" on page 135
- "Distributing Compiled SQL Procedures" on page 140

Related reference:

- "Windows Supported Software for Building and Running Applications" on page 15

Sample Database

Setting Up the sample Database

To use the sample programs shipped with DB2, you need to create the `sample` database on a server workstation. 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 utility files, to the `sample` database.

Procedure:

Here are the steps to set up the `sample` database:

1. Creating the sample Database
2. Cataloging the sample Database
3. Binding the sample Database Utilities

Related tasks:

- "Creating the sample Database" on page 42
- "Cataloging the sample Database" on page 45
- "Binding the sample Database Utilities" on page 46

Creating the sample Database

You create the `sample` database on the command line with the `db2sampl` command.

Prerequisites:

You must have System Administrator (SYSADM) or System Control (SYSCTRL) authority to create a database. SYSADM and SYSCTRL are, respectively, the first and second highest levels of authority for DB2.

Procedure:

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 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:

     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 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*, for example: "$HOME", and would be entered as:

   ```
   db2sampl path
   ```

   For example:

   ```
   db2sampl $HOME
   ```

   On Windows, this is a *drive*, for example: "C:", and would be entered as:

   ```
   db2sampl drive
   ```

For example:

```
db2sampl C:
```

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. The authentication type for the database is the same as the instance in which it is created.

Related tasks:

- "Creating the sample Database on Host or AS/400 and iSeries Servers" on page 44
- "Cataloging the sample Database" on page 45
- "Binding the sample Database Utilities" on page 46

Creating the sample Database on Host or AS/400 and iSeries Servers

If you want to run the sample programs against a Host server such as DB2 UDB for z/OS and OS/390, or an AS/400 and iSeries server, you need to create a database that contains the sample tables described in the SQL Reference.

Note: You need DB2 Connect to connect to a host server.

Restrictions:

There are some SQL syntax and DB2 command differences between DB2 on the workstation and DB2 on host systems. When accessing databases on DB2 UDB for z/OS and OS/390 or DB2 for AS/400 and iSeries, make sure your programs use SQL statements and precompile/bind options that are supported on these database systems.

Procedure:

To create the database:

1. Create the `sample` database in a DB2 workstation server instance using `db2sampl`.
2. Connect to the `sample` database.
3. Export the sample table data to a file.
4. Connect to the host database.
5. Create the sample tables.
6. Import the sample table data from the file where you exported the data on the workstation server.

Related concepts:

- "Export Overview" in the *Data Movement Utilities Guide and Reference*
- "Import Overview" in the *Data Movement Utilities Guide and Reference*

Related tasks:

- "Cataloging the sample Database" on page 45
- "Binding the sample Database Utilities" on page 46

Related samples:

- "expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

Cataloging the sample Database

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 on the server 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.

Procedure:

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

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

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

You must also catalog the remote node before you can connect to the database.

Related tasks:

- "Cataloging the TCP/IP node on the client" in the *Installation and Configuration Supplement*
- "Cataloging a database using the CLP" in the *Installation and Configuration Supplement*
- "Cataloging the NetBIOS node on the DB2 client" in the *Installation and Configuration Supplement*

- "Cataloging the Named Pipes node on the client" in the *Installation and Configuration Supplement*
- "Cataloging the APPC node on the DB2 client" in the *Installation and Configuration Supplement*
- "Binding the sample Database Utilities" on page 46

Binding the sample Database Utilities

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 utilities, 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.

Procedure:

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

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 grant public

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

   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 operating system:

1. From the Start Menu, select Programs.

2. From the Programs Menu (or from 'All Programs' on Windows XP), 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 grant public

   db2 bind "%DB2PATH%\bnd\@db2cli.lst" blocking all
   sqlerror continue messages cli.msg grant public
   ```

   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 files, `bind.msg` and `cli.msg`.

For all clients accessing host servers, specify one of the following `.lst` files instead of `db2ubind.lst`:

ddcsmvs.lst
: for DB2 for z/OS and OS/390

ddcsvm.lst
: for DB2 for VM

ddcsvse.lst
: for DB2 for VSE

ddcs400.lst
: for DB2 for AS/400 and iSeries

For example:

- If accessing a DB2 for z/OS and OS/390 server from a UNIX client, enter:

  ```
  db2 bind BNDPATH/@ddcsmvs.lst blocking all sqlerror continue \
    messages bind.msg grant public
  ```

- If accessing a DB2 for z/OS and OS/390 server from a Windows client, enter:

  ```
  db2 bind "%DB2PATH%\bnd\@ddcsmvs.lst" blocking all
     sqlerror continue messages bind.msg grant public
  ```

Related reference:

- "BIND" in the *Command Reference*

Migrating Applications

DB2 Version 8 supports the following DB2 versions for migration:

- DB2 Version 6
- DB2 Version 7.1
- DB2 Version 7.2
- DataJoiner® Version 2.1.x

When you migrate to a later version of DB2, your database and node directories are migrated automatically. To migrate from any other previous version of DB2, you must first migrate to one of the above supported versions that supports this migration, and then migrate from that version to DB2 version 8.

Points to keep in mind when migrating from the following specific environments:

Java The type 3 driver, formerly known as the "net" driver, is deprecated. DB2 Java applets should be migrated to the type 4 driver. To convert a type 3 JDBC applet to use the new type 4 driver, make the following changes:

1. The type 4 driver archive is `db2jcc.jar`. In the `.html` file associated with the applet, change the archive from `db2java.zip` to `db2jcc.jar`. Copy `db2jcc.jar` to the web server.
2. The type 4 driver class name is `com.ibm.db2.jcc.DB2Driver`. In the applet `.java` files, change the type 3 driver class name, `COM.ibm.db2.jdbc.net.DB2Driver`, to the type 4 class name. There may not be a reference to the driver class if the applet uses `javax.sql.DataSource` to obtain connections.
3. Both the type 3 and type 4 driver use a data source URL of the same form: `jdbc:db2://server:portnumber/dbname`. However, the three parts: `server`, `portnumber`, and `dbname` have a different meaning in the two drivers.

   The type 3 driver is a three tier model with a client (the browser running the applet), a JDBC Applet Server, and a DB2 server. The `server` and `portnumber` in the URL refer to the JDBC Applet Server. The `dbname is the database alias cataloged on the system running the JDBC Applet Server.`

   The type 4 driver client connects directly to the DB2 server so the `server` and `portnumber` are those of the DB2 server TCP/IP listener. The `dbname` is the database alias cataloged on the DB2 server system.

If the applet uses DriverManager.getConnection to connect to DB2, update the .java files and (if necessary) the .html file with the new URL for the type 4 driver.

4. If the applet makes use of COM.ibm.db2.jdbc.DB2DataSource, new javax.sql.DataSource objects of the class com.ibm.jcc.db2.DB2SimpleDataSource must be created. The applet must be updated to use this new class.

Note: The type 4 driver does not support JDBC 2.0 Optional Package APIs for connection pooling (javax.sql.ConnectionPoolDataSource, javax.sql.PooledConnection). If the applet makes use of these type 3 features, it cannot be migrated to the type 4 driver.

32-bit to 64-bit Environments

It may not be necessary to migrate your applications from 32-bit to 64-bit. DB2 Version 8 supports running an existing 32-bit local application in all 64-bit DB2 instances except Linux IA64. In order to do this, the user must rebind the 32-bit application and then run it with the appropriate library path. For example, to run a 32-bit application in a 64-bit DB2 instance on AIX:

- On bash or Korn shell:

```
export LIBPATH=$HOME/sqllib/lib32
my32bitapp
```

- On C shell:

```
setenv LIBPATH $HOME/sqllib/lib32
my32bitapp
```

If you want to migrate a 32-bit application to a 64-bit operating environment while still running on a 32-bit server, use the LONGERROR precompile option to prepare for porting the application. Set LONGERROR to YES in the 32-bit environment so that the precompiler returns an error whenever it encounters a host variable of the long type. Then follow these steps:

1. Prune the use of long types for host variables, unless long types are necessary. Instead, use the new portable host variables, sqlint32 or sqluint32 . For example:

```
EXEC SQL BEGIN DECLARE SECTION;
 long y;     /* this declaration generates an error on 64 bit */
 sqlint32 x; /* this declaration is acceptable for 64 bit */
EXEC SQL END DECLARE SECTION;
```

2. Precompile the application against a database on a 64-bit server. This creates a new package for the application that is being ported.
3. Compile the application in 64-bit mode.
4. Link the application with the new 64-bit DB2 libraries.

5. Bind the application to a database on a 64-bit server.

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.

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.

Here are points to keep in mind when developing your applications. They will help make your applications portable:

- On UNIX, use only the default library search path, `/usr/lib:/lib`, in your applications. On Windows® 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 `sqllib\lib` directory on Windows operating systems. This makes applications highly dependent on specific user names and environments.
- Generally, do not use the following environment variables to alter search paths in a particular environment: LIBPATH (AIX), SHLIB_PATH (HP-UX 32-bit), LD_LIBRARY_PATH (HP-UX 64-bit, Linux, and Solaris), and LIB (Windows). These variables override the search paths specified in the applications running in the environment, so applications might not be able to find the libraries or the files they need.

- In DB2 Universal Database™ Versions 6, 7, and 8, 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, Version 7, or Version 8 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, Version 7, or Version 8 function prototypes. If this occurs, use the compiler option `-DSQLOLDCHAR` to eliminate the problem.

On UNIX platforms, if you have applications from a previous DB2 release version and you want them to run in both a database instance of the previous version as well as a DB2 Version 8 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.

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 in the directory where the executable resides (which may be `/usr/bin` or your instance directory:

AIX dump -H *executable_filename*

HP-UX
chatr *executable_filename*

Linux objdump -p *executable_filename*

Solaris
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 DB2 Version 7.2 for the AIX C sample application, `dbcat`, taken in the samples sub-directory of the DB2 instance, `/home/dbinst/samples/c`:

Sample Dump Listing for C application dbcat

```
dbcat:

                        ***Loader Section***
                     Loader Header Information
VERSION#         #SYMtableENT     #RELOCent        LENidSTR
0x00000001       0x0000000f       0x00000015       0x00000047

#IMPfilID        OFFidSTR         LENstrTBL        OFFstrTBL
0x00000003       0x00000284       0x0000007f       0x000002cb


                        ***Import File Strings***
INDEX  PATH                           BASE               MEMBER
0      /home/db2inst/sqllib/lib:/usr/lib:/lib

1                                     libc.a             shr.o
2                                     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, and 2 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_07_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 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 on AIX 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
/usr/lpp/db2_07_01/lib

Version 8
/usr/opt/db2_08_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 and Solaris. You can see the library search path using the appropriate system command for your platform given in the answer to Question 1.

Conditions

Once you have the answers to the questions above, you may need to make changes to your environment. Read the conditions listed below. If one of the conditions applies to your situation, make the necessary changes.

Condition 1: If a Version 7 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_07_01/lib/libdb2.a, and the database server is DB2 Universal Database Version 8 for AIX, do one of the following:
  - Change the symbolic link to point to:
    /usr/opt/db2_08_01/lib/libdb2.a

    As root, you can change links using the "db2ln" command as follows:
    /usr/opt/db2_08_01/cfg/db2ln
  - Set the LIBPATH environment variable to point to /usr/opt/db2_08_01/lib or *INSTHOME*/sqllib/lib, where *INSTHOME* is the home directory of the Version 8 DB2 instance owner.

  - Configure a TCP/IP connection from the application (client) instance to the server instance.
- If there is a symbolic link from `/usr/lib/libdb2.a` to `/usr/opt/db2_08_01/lib/libdb2.a`, and the database server is DB2 Version 7, configure a TCP/IP connection from the application (client) instance to the server instance.

Condition 2: If a Version 7 application loads a shared library out of the $HOME path of a DB2 Version 7 instance owner (`$HOME/sqllib/lib/libdb2.a`), and the database server is DB2 Universal Database Version 8 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/opt/db2_08_01/lib` or *INSTHOME*`/sqllib/lib`, where *INSTHOME* is the home directory of the Version 8 instance owner.
- Configure a TCP/IP connection from the application (client) instance to the server instance.

Condition 3: If a Version 7 application loads a shared library out of the DB2 Version 7 install path (`/usr/lpp/db2_07_01/lib/libdb2.a`), and the database server is DB2 Universal Database Version 8 for AIX, do one of the following:

- Set the LIBPATH environment variable to point to `/usr/opt/db2_08_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.

Condition 4: If a Version 7 application loads a shared library out of the DB2 Universal Database Version 8 for AIX install path (`/usr/opt/db2_08_01/lib/libdb2.a`), and the database server is DB2 Version 7, configure a TCP/IP connection from the application (client) instance to the server instance.

Related concepts:

- "Migration recommendations" in the *Quick Beginnings for DB2 Servers*
- "JDBC 2.1 Core API Restrictions by the DB2 JDBC Type 4 Driver" in the *Application Development Guide: Programming Client Applications*
- "JDBC 2.1 Optional Package API Support by the DB2 JDBC Type 4 Driver" in the *Application Development Guide: Programming Client Applications*

Related tasks:

- "Setting Up the Application Development Environment" on page 19

- "Creating links for DB2 files" in the *Installation and Configuration Supplement*
- "Migrating databases" in the *Quick Beginnings for DB2 Servers*
- "Migrating Explain tables" in the *Quick Beginnings for DB2 Servers*
- "Migrating instances (UNIX)" in the *Quick Beginnings for DB2 Servers*
- "Taking a V6 or V7 DB2 server offline for DB2 migration" in the *Quick Beginnings for DB2 Servers*
- "Migrating DB2 (Windows)" in the *Quick Beginnings for DB2 Servers*
- "Migrating DB2 (UNIX)" in the *Quick Beginnings for DB2 Servers*

Related reference:

- "Migration restrictions" in the *Quick Beginnings for DB2 Servers*
- "Version 8 incompatibilities between releases" in the *Administration Guide: Planning*
- "Version 7 incompatibilities between releases" in the *Administration Guide: Planning*

Where to Go Next

Once your environment is set up, you are ready to build your DB2 applications. The following chapter discusses the sample programs and related files, including the build files. The chapters following this use the build files and samples to show you how to compile, link, and run your applications in your programming environment. Read the specific chapter for your particular application development needs.

Chapter 3. Sample Programs and Related Files

This chapter describes sample programs and related files for the programming languages for all platforms supported by DB2. It presents the design for the samples based on the component structure of DB2, and gives a listing of the DB2 samples with a description of each one. It also explains the uses of the build files, makefiles and error-checking utilities that come with DB2.

Sample Files

The sample programs come with the DB2® Application Development (DB2 AD) Client. Not all sample programs are available on all platforms or supported programming languages. You can use the sample programs as templates to create your own applications, and as a learning tool to understand DB2 functionality.

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.

Besides the sample program files, other sample files are provided by DB2 in the samples directories under `sqllib/samples` (UNIX) and under `sqllib\samples` (Windows). These include build files and makefiles to compile and link the sample programs, error-checking utility files which are linked in to most sample programs, and various script files to assist in application development. For example, scripts are provided to catalog and uncatalog stored procedures and UDFs in several language sub-directories. Each samples directory has a README file which describes the files contained in the directory.

HTML versions of most sample program files are provided, accessible in the online documentation. These 'samples in HTML' are linked into the documentation topics to demonstrate the functionality described by them. Keywords, such as SQL statements and DB2 APIs, are hot-linked within the samples in HTML so the user can go directly to the documentation describing them. Most samples in HTML have a link in the comment section at the top of the file to a sample output file showing typical results of running the compiled sample program. Please note that the actual output is, in many cases, machine and platform-dependent, so the output you receive from running the same program may vary.

Below is a table showing the sample directories and README files for the main supported programming languages/APIs by platform. The README files are hot-linked in the online documentation, and the samples listings within them have hotlinks to the sample file source code. You can also access the sample files in the listed samples directories. For the directory paths, the UNIX-style slashes are used, as in `samples/c`, except where the directory is for Windows only, as in `samples\VB\ADO`.

Table 1. Sample README files by Platform

Platform —> Language	**AIX**	**HP-UX**	**Linux**	**Solaris**	**Windows**
C samples/c	README	README	README	README	README
C++ samples/cpp	README	README	README	README	README
CLI samples/cli	README	README	README	README	README
JDBC samples/java	README	README	README	README	README
SQLJ samples/java	README	README	README	README	README
IBM COBOL samples/cobol	README	n/a	n/a	n/a	README
Micro Focus COBOL samples/cobol_mf	README	README	n/a	README	README
Visual Basic samples\VB\ADO	n/a	n/a	n/a	n/a	ReadMe.txt
SQL Procedures samples/sqlproc	README	README	README	README	README
CLP samples/clp	README	README	README	README	README

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 2.

Sample File Extensions by Program Group
Table 3 on page 60.

Table 2. Sample File Extensions by Language

Language	Directory	Embedded SQL Programs	Non-embedded SQL Programs
C	`samples/c` `samples/cli` (CLI programs)	`.sqc`	`.c`
C++	`samples/cpp`	`.sqC` (UNIX) `.sqx` (Windows)	`.C` (UNIX) `.cxx` (Windows)
COBOL	`samples/cobol` `samples/cobol_mf`	`.sqb`	`.cbl`

JAVA	`samples/java/jdbc`
	`samples/java/sqlj`
	`samples/java/WebSphere`
	`samples/java/plugin`
		`.sqlj`	`.java`

REXX	`samples/rexx`	`.cmd`	`.cmd`

Visual Basic	`samples\VB\ADO`
	`samples\VB\MTS`
	`samples\VB\RDO`
			`.bas .frm .vbp`

Visual C++	`samples\VC\ADO`		`.cpp .dsp .dsw`

Table 3. Sample File Extensions by Program Group

Sample Group	Directory	File Extension
CLP	`samples/clp`	`.db2`
OLE	`samples\ole\msvb` (Visual Basic) `samples\ole\msvc` (Visual C++)	`.bas` `.vbp` (Visual Basic) `.cpp` (Visual C++)
OLE DB	`samples\oledb`	`.db2`
SQL Procedures	`samples/sqlproc`	`.db2` (SQL Procedure scripts) `.c` (CLI Client Applications) `.sqc` (embedded C Client Applications) `.java` (JDBC Client Applications)
User Exit	`samples/c`	`.cadsm` (UNIX & Windows) `.cdisk` (UNIX & Windows) `.ctape` (UNIX) `.cxbsa` (UNIX & Windows)

Note:

Directory Delimiters
The directory delimiter on UNIX® is a `/`. On Windows® it is a `\`. In the tables, the UNIX delimiters are used unless the directory is only available on Windows.

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® and Windows 32-bit operating systems in the `cobol` subdirectory.

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

Java™ Samples
Are Java Database Connectivity (JDBC) applets, applications, and routines, embedded SQL for Java (SQLJ) applets, applications, and routines. Also, WebSphere samples and plugin example files for the DB2 Control Center. Java samples are available on all supported DB2 platforms.

REXX Samples
Are only supplied for AIX and Windows 32-bit 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 operating systems only.

Visual Basic Samples
Are ActiveX Data Objects, Remote Data Objects, and Microsoft Transaction Server samples, supplied on Windows operating systems only.

Visual C++ Samples
Are ActiveX Data Object samples, supplied on Windows 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 are subdirectories 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 Windows operating systems.**

  You can find the C source code for embedded SQL and DB2 API programs in `sqllib\samples\c` under the DB2 install directory; the C source code for DB2 CLI programs is in `sqllib\samples\cli`. For additional information about the sample programs in the samples tables, refer to the `README` file in the appropriate `sqllib\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.

Related concepts:

- "Build Files" on page 94
- "Makefiles" on page 97
- "Error-Checking Utilities" on page 101
- "Sample Programs: Structure and Design" on page 62

Related reference:

- "C/C++ Samples" on page 69
- "DB2 CLI Samples" on page 72
- "JDBC Samples" on page 74
- "SQLJ Samples" on page 77
- "SQL Procedure Samples" on page 84
- "Visual Basic Samples" on page 86
- "Visual C++ Samples" on page 88
- "Object Linking and Embedding (OLE) Samples" on page 89
- "Object Linking and Embedding Database (OLE DB) Table Function Samples" on page 90
- "Command Line Processor (CLP) Samples" on page 90
- "Log Management User Exit Samples" on page 93
- "COBOL Samples" on page 80
- "Java WebSphere Samples" on page 79
- "Java Plugin Samples" on page 79
- "Windows Management Instrumentation Samples" on page 88
- "REXX Samples" on page 91

Sample Programs: Structure and Design

Most of the DB2® samples in C, CLI, C++, Java, and Visual Basic ADO are organized to reflect an object-based design model of the database components. The samples are grouped in categories representing different levels of DB2. The level to which a sample belongs is indicated by a two character prefix at the beginning of the sample name (for Java, the first letter is capitalized). Not all levels are represented in the samples for each Application Programming Interface, but for the samples as a whole, the levels are represented as follows:

prefix	DB2 Level
il	Installation Image Level
cl	Client Level
in	Instance Level

db Database Level

ts Table Space Level

tb Table Level

dt Data Type Level

The levels show a hierarchical structure. The Installation image level is the top level of DB2. Below this level, a client-level application can access different instances; an instance can have one or more databases; a database has table spaces within which tables exist, and which in turn hold data of different data types.

This design does not include all DB2 samples. The purpose of some samples is to demonstrate different methods for accessing data. These methods are the main purpose of these samples so they are represented by these methods in a similar manner as the above:

prefix Programming Method

fn SQL Function

sp Stored Procedure

ud User Defined Function

Besides these categories, there are a set of tutorial samples to introduce basic concepts of database programming. These samples use some of the simpler functions represented in the samples design, and begin with the characters: ″tut″.

There are other samples not included in this design, such as the Log Management User Exit samples, samples in COBOL, Visual C++, REXX, Object Linking and Embedding (OLE) samples, CLP scripts, and SQL Procedures.

Below are the sample programs implemented in this design.

Note: Java™ program names have the first character, and sometimes other characters, in upper case, and do not have the underscores in the tutorial sample names. Visual Basic samples have some characters in upper case (but not the first character).

Table 4. Sample Program Design in DB2 Universal Database

Sample Program	Description
Tutorial Samples	

Table 4. Sample Program Design in DB2 Universal Database (continued)

Sample Program	Description
tut_mod	How to modify table data. Supported languages/APIs: C, C++, CLI, JDBC
tut_read	How to read tables. Supported languages/APIs: C, C++, CLI, JDBC
tut_use	How to use a database. Supported languages/APIs: C, C++, CLI
Installation Image Level	
ilinfo	How to get/set installation level info. Supported languages/APIs: CLI, JDBC
Client Level	
cliExeSQL	How to execute SQL statements. Supported languages/APIs: Visual Basic/ADO
cli_info	How to get/set client level information. Supported languages/APIs: C, C++, CLI, Visual Basic/ADO
clihandl	How to allocate and free handles. Supported languages/APIs: CLI
clisqlca	How to work with SQLCA data. Supported languages/APIs: CLI
clisnap	How to capture a client level snapshot. Supported languages/APIs: C, C++
Instance Level	
inattach	How to attach to/detach from an instance. Supported languages/APIs: C, C++
inauth	How to display authorities at instance level. Supported languages/APIs: C, C++
ininfo	How to get and set instance level information. Supported languages/APIs: C, C++, CLI
insnap	How to capture an instance level snapshot. Supported languages/APIs: C, C++

Table 4. Sample Program Design in DB2 Universal Database (continued)

Sample Program	Description
instart	How to stop and start the current local instance. Supported languages/APIs: C, C++
Database Level	
dbauth	How to grant/display/revoke authorities at the database level. Supported languages/APIs: C, C++, JDBC, SQLJ
dbcfg	How to configure database and database manager parameters. Supported languages/APIs: C, C++
dbconn	How to connect/disconnect from a database. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ, Visual Basic/ADO
dbcreate	How to create and drop databases. Supported languages/APIs: C, C++
dbinfo	How to get and set information at a database level. Supported languages/APIs: C, C++, CLI, JDBC, Visual Basic/ADO
dbinline	How to use inline SQL procedure language. Supported languages/APIs: C
dbCommit	How to control autocommit dynamically on the database level. Supported languages/APIs: Visual Basic/ADO
dbmcon	How to connect/disconnect from multiple databases. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
dbmconx	How to connect/disconnect from multiple databases. Supported languages/APIs: CLI
dbmigrat	How to migrate a database. Supported languages/APIs: C, C++
dbnative	How to translate a statement with an ODBC escape clause. Supported languages/APIs: CLI, JDBC
dbpkg	How to work with packages. Supported languages/APIs: C, C++
dbrecov	How to recover a database. Supported languages/APIs: C, C++

Table 4. Sample Program Design in DB2 Universal Database (continued)

Sample Program	Description
dbsample	How to create the sample database including Host and AS/400® tables and views. Supported languages/APIs: C
DbSeq	How to create, alter and drop a sequence in a database. Supported languages/APIs: JDBC
dbsnap	How to capture a database level snapshot. Supported languages/APIs: C, C++
dbthrds	How to use threads. Supported languages/APIs: C, C++
dbuse	How to use database objects. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
dbusemx	How to use database objects with embedded SQL. Supported languages/APIs: CLI
Table Space Level	
tscreate	How to create/drop bufferpools and table spaces. Supported languages/APIs: C, C++
tsinfo	How to get information at table space level. Supported languages/APIs: C, C++
Table Level	
tbconstr	How to work with table constraints. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
tbcreate	How to create, alter, and drop tables. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
TbGenCol	How to use generated columns. Supported languages/APIs: JDBC
tbident	How to use identity columns. Supported languages/APIs: C, JDBC, SQLJ
tbinfo	How to get and set information at a table level. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ

Table 4. Sample Program Design in DB2 Universal Database (continued)

Sample Program	Description
tbmod	How to modify information in a table. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
tbmove	How to move table data. Supported languages/APIs: C, C++
tbpriv	How to grant/display/revoke table level privileges. Supported languages/APIs: C, C++, JDBC, SQLJ
tbread	How to read information in a table. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
tbreorg	How to reorganize a table. Supported languages/APIs: C, C++
tbsavept	How to use external savepoints. Supported languages/APIs: C
tbtemp	How to use a declared temporary table. Supported languages/APIs: C, JDBC
tbtrig	How to use a trigger on a table. Supported languages/APIs: C, C++, JDBC, SQLJ
Data Type Level	
dtformat	How to use load and import data format extensions. Supported languages/APIs: C
dtHier	How to retrieve hierarchical data. Supported languages/APIs: Visual Basic/ADO
dtinfo	How to get information about data types. Supported languages/APIs: CLI, JDBC
dtlob	How to read and write LOB data. Supported languages/APIs: C, C++, CLI, JDBC, Visual Basic/ADO
dtstruct	How to create, use, and drop a hierarchy of structured types and typed tables. Supported languages/APIs: C++
dtudt	How to create/use/drop user-defined distinct types. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ

Table 4. Sample Program Design in DB2 Universal Database (continued)

Sample Program	Description
DB2 Functions	
fnuse	How to use SQL functions. Supported languages/APIs: C, C++
Stored Procedures	
spcall	How to call stored procedures. Supported languages/APIs: CLI, Visual Basic/ADO
spclient	Client application that calls the stored procedures in spserver. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
spserver	Stored procedure routines called by spclient. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
User-Defined Functions	
udfcli	Client application to call UDFs in udfsrv. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
udfsrv	User-defined function library called by udfcli. Supported languages/APIs: C, C++, CLI, JDBC, SQLJ
udfjcli	Client application to call Parameter Style Java user-defined functions in udfjsrv. Supported languages/APIs: JDBC, SQLJ
udfjsrv	Parameter Style Java user-defined function library called by udfjcli. Supported languages/APIs: JDBC, SQLJ
udfsqlcl	Client application to call Java user-defined functions in udfsqlsv. Supported languages/APIs: JDBC
udfsqlsv	Java user-defined function library with SQL statements called by udfsqlcl. Supported languages/APIs: JDBC
udfUse	How to create and work with UDTs and UDFs. Supported languages/APIs: Visual Basic/ADO

Related concepts:

- "Build Files" on page 94
- "Makefiles" on page 97
- "Error-Checking Utilities" on page 101

- "Sample Files" on page 57

Related reference:
- "C/C++ Samples" on page 69
- "DB2 CLI Samples" on page 72
- "JDBC Samples" on page 74
- "SQLJ Samples" on page 77
- "SQL Procedure Samples" on page 84
- "Visual Basic Samples" on page 86
- "Visual C++ Samples" on page 88
- "Object Linking and Embedding (OLE) Samples" on page 89
- "Object Linking and Embedding Database (OLE DB) Table Function Samples" on page 90
- "Command Line Processor (CLP) Samples" on page 90
- "Log Management User Exit Samples" on page 93
- "COBOL Samples" on page 80
- "Java WebSphere Samples" on page 79
- "Java Plugin Samples" on page 79
- "Windows Management Instrumentation Samples" on page 88
- "REXX Samples" on page 91

Sample Programs by Language and Application Interface

C/C++ Samples

Note: File extensions:

C .c (non-embedded SQL)

.sqc (embedded SQL)

C++ .C (non-embedded SQL)

.sqC (embedded SQL)

Table 5. C/C++ Sample Program Files

Sample Program Name	Program Description
Tutorial Samples - Programs that demonstrate basic database operations.	
`tut_mod.sqc`, `tut_mod.sqC`	How to modify table data.
`tut_read.sqc`, `tut_read.sqC`	How to read tables.

Table 5. C/C++ Sample Program Files (continued)

Sample Program Name	Program Description
tut_use.sqc, tut_use.sqC	How to use a database.
Client Level - Samples that deal with the client level of DB2.	
cli_info.c, cli_info.C	How to get and set client level information.
clisnap.c, clisnap.C	How to capture a snapshot at the client level.
Instance Level - Samples that deal with the instance level of DB2.	
inattach.c, inattach.C	How to attach to/detach from an instance.
inauth.sqc, inauth.sqC	How to display authorities at instance level.
ininfo.c, ininfo.C	How to get and set instance level information.
insnap.c, insnap.C	How to capture a snapshot at the instance level.
instart.c, instart.C	How to stop and start the current local instance.
Database Level - Samples that deal with database objects in DB2.	
dbauth.sqc, dbauth.sqC	How to grant/display/revoke authorities at the database level
dbcfg.sqc, dbcfg.sqC	How to configure database and database manager parameters.
dbconn.sqc, dbconn.sqC	How to connect and disconnect from a database.
dbcreate.c, dbcreate.C	How to create and drop databases.
dbinfo.c, dbinfo.C	How to get and set information at a database level.
dbinline.sqc	How to use inline SQL procedure language.
dbmcon.sqc, dbmcon.sqC	How to connect to and disconnect from multiple databases.
dbmcon1.h, dbmcon1.h	Header file for dbmcon1.sqc, dbmcon1.sqC
dbmcon1.sqc, dbmcon1.sqC	Support file for dbmcon.sqc, dbmcon.sqC.
dbmcon2.h, dbmcon2.h	Header file for dbmcon2.sqc, dbmcon2.sqC
dbmcon2.sqc, dbmcon2.sqC	Support file for dbmcon.sqc, dbmcon.sqC.
dbmigrat.c, dbmigrat.C	How to migrate a database.
dbpkg.sqc, dbpkg.sqC	How to work with packages.

Table 5. C/C++ Sample Program Files (continued)

Sample Program Name	Program Description
dbrecov.sqc, dbrecov.sqC	How to recover a database.
dbsample.sqc	How to create the sample database including Host and AS/400 tables and views.
dbsnap.c, dbsnap.C	How to capture a snapshot at the database level.
dbthrds.sqc, dbthrds.sqC	How to use threads.
dbuse.sqc, dbuse.sqC	How to use database objects.
Table Space Level - Samples that deal with the table space level of DB2.	
tscreate.sqc, tscreate.sqC	How to create and drop buffer pools and table spaces.
tsinfo.sqc, tsinfo.sqC	How to get information at the table space level.
Table Level - Samples that deal with table objects in DB2.	
tbconstr.sqc, tbconstr.sqC	How to work with table constraints.
tbcreate.sqc, tbcreate.sqC	How to create, alter and drop tables.
tbident.sqc	How to use identity columns.
tbinfo.sqc, tbinfo.sqC	How to get and set information at a table level.
tbmod.sqc, tbmod.sqC	How to modify information in a table.
tbmove.sqc, tbmove.sqC	How to move a table data.
tbpriv.sqc, tbpriv.sqC	How to grant/display/revoke table level privileges.
tbread.sqc, tbread.sqC	How to read information in a table.
tbreorg.sqc, tbreorg.sqC	How to reorganize a table.
tbsavept.sqc	How to use external savepoints.
tbtemp.sqc	How to use a declared temporary table.
tbtrig.sqc, tbtrig.sqC	How to use a trigger on a table.
Data Type Level - Samples that deal with data types.	
dtformat.sqc	How to use load and import data format extensions.

Table 5. C/C++ Sample Program Files (continued)

Sample Program Name	Program Description
`dtlob.sqc`, `dtlob.sqC`	How to read and write LOB data.
`dtstruct.sqC`	How to create, use, and drop a hierarchy of structured types and typed tables.
`dtudt.sqc`, `dtudt.sqC`	How to create, use, and drop user-defined distinct types.
DB2 Function Level	
`fnuse.sqc`, `fnuse.sqC`	How to use SQL functions.
Stored Procedure Level - Samples that demonstrate stored procedures.	
`spcat`	Stored procedure catalog script for the spserver program. This script calls spdrop.db2 and spcreate.db2.
`spcreate.db2`	CLP script to issue CREATE PROCEDURE statements.
`spdrop.db2`	CLP script to drop stored procedures from the catalog.
`spclient.sqc`, `spclient.sqC`	Client program used to call the server routines declared in spserver.sqc, spserver.sqC.
`spserver.sqc`, `spserver.sqC`	Stored procedure routines built and run on the server.
UDF Level - Samples that demonstrate user-defined functions.	
`udfcli.sqc`, `udfcli.sqC`	Client application which calls the user-defined function in udfsrv.c, udfsrv.C.
`udfsrv.c`, `udfsrv.C`	User-defined function ScalarUDF called by udfcli.sqc, udfcli.sqC.
`udfemcli.sqc`, `udfemcli.sqC`	Client application which calls the embedded SQL user defined function library udfemsrv.
`udfemsrv.sqc`, `udfemsrv.sqC`	Embedded SQL User-defined function library called by udfemcli.
Other	
`utilsnap.c`, `utilsnap.C`	Utilities for the snapshot monitor samples.

Related concepts:

- "Sample Files" on page 57
- "Sample Programs: Structure and Design" on page 62

DB2 CLI Samples

Table 6. Sample CLI Program Files

Sample Program Name	Program Description
Tutorial Samples - Programs that demonstrate basic database operations.	

Table 6. Sample CLI Program Files (continued)

Sample Program Name	Program Description
tut_mod.c	How to modify table data.
tut_read.c	How to read tables.
tut_use.c	How to use a database.
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).
Client Level - Samples that deal with the client level of DB2.	
cli_info.c	How to get and set client level information.
clihandl.c	How to allocate and free handles.
clisqlca.c	How to work with SQLCA data.
Instance Level - Samples that deal with the instance level of DB2.	
ininfo.c	How to get and set instance level information.
Database Level - Samples that deal with database objects in DB2.	
dbcongui.c	How to connect to a database with a Graphical User Interface (GUI).
dbconn.c	How to connect and disconnect from a database.
dbinfo.c	How to get and set information at a database level.
dbmcon.c	How to connect and disconnect from multiple databases.
dbmconx.c	How to connect and disconnect from multiple databases with embedded SQL.
dbmconx1.h	Header file for dbmconx1.sqc.
dbmconx1.sqc	Embedded SQL file for the dbmconx program.
dbmconx2.h	Header file for dbmconx2.sqc.
dbmconx2.sqc	Embedded SQL file for the dbmconx program.
dbnative.c	How to translate a statement that contains an ODBC escape clause to a data source specific format.
dbuse.c	How to use database objects.
dbusemx.sqc	How to use database objects with embedded SQL.
Table Level - Samples that deal with table objects in DB2.	
tbconstr.c	How to work with table constraints.
tbcreate.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.

Table 6. Sample CLI Program Files (continued)

Sample Program Name	Program Description
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.
Stored Procedure Level - Samples that demonstrate stored procedures.	
`spcat`	Stored procedure catalog script for the spserver program. This script calls spdrop.db2 and spcreate.db2.
`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.
`spserver.c`	Stored procedure functions built and run on the server.
`spclires.c`	Client application that demonstrates the difference between SQLMoreResults and SQLNextResults for multiple result sets.
`spcall.c`	Client program for calling any stored procedure.
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.
Common Utility Files	
`utilcli.c`	Utility functions used in CLI samples.
`utilcli.h`	Header file for utility functions used in CLI samples.

Related concepts:

- "Sample Files" on page 57
- "Sample Programs: Structure and Design" on page 62

JDBC Samples

Table 7. Sample JDBC Program Files

Sample Program Name	Program Description
Tutorial Samples - Programs that demonstrate basic database operations.	
`TutMod.java`	How to modify table data.
`TutRead.java`	How to read tables.

Installation Image Level - Samples that deal with the installation image level of DB2.

Table 7. Sample JDBC Program Files (continued)

Sample Program Name	Program Description
`IlInfo.java`	How to get and set installation level information.
Database Level - Samples that deal with database objects in DB2.	
`DbAuth.java`	How to grant/display/revoke authorities at the database level.
`DbConn.java`	How to connect and disconnect from a database.
`DbInfo.java`	How to get and set information at a database level.
`DbMCon.java`	How to connect and disconnect from multiple databases.
`DbNative.java`	How to translate a statement that contains an ODBC escape clause to a data source specific format.
`DbSeq.java`	How to create, alter and drop a sequence in a database.
`DbUse.java`	How to use database objects.
Table Level - Samples that deal with table objects in DB2.	
`TbConstr.java`	How to work with table constraints.
`TbCreate.java`	How to create, alter and drop tables.
`TbGenCol.java`	How to use generated columns.
`TbIdent.java`	How to use Identity Columns.
`TbInfo.java`	How to get and set information at a table level.
`TbMod.java`	How to modify information in a table.
`TbPriv.java`	How to grant/display/revoke table level privileges.
`TbRead.java`	How to read information in a table.
`TbTemp.java`	How to use Declared Temporary Tables.
`TbTrig.java`	How to use a trigger on a table.
Data Type Level - Samples that deal with data types.	
`DtInfo.java`	How to get information about data types.
`DtLob.java`	How to read and write LOB data.
`DtUdt.java`	How to create, use, and drop user-defined distinct types.
Applets - Samples that demonstrate applets.	
`Applt.java`	How to use applets.
Stored Procedures - Samples that demonstrate stored procedures.	
`spcat`	Stored procedure catalog script for the spserver program. This script calls SpDrop.db2 and SpCreate.db2.
`SpCreate.db2`	CLP script to issue CREATE PROCEDURE statements.
`SpDrop.db2`	CLP script to drop stored procedures from the catalog.

Table 7. Sample JDBC Program Files (continued)

Sample Program Name	Program Description
`SpClient.java`	Client program used to call the server functions declared in SpServer.java.
`SpServer.java`	Stored procedure functions built and run on the server.
UDFs - Samples that demonstrate user-defined functions.	
`UDFcli.java`	Client application which calls the user-defined function library UDFsrv.
`UDFsrv.java`	User-defined functions called by UDFcli.java.
`udfcat`	UDF catalog script for the UDFsrv program. This script calls UDFDrop.db2 and UDFCreate.db2.
`UDFDrop.db2`	CLP script to drop UDFs from the catalog.
`UDFCreate.db2`	CLP script to issue CREATE PROCEDURE statements.
`UDFjcli.java`	Client application which calls the user-defined function library UDFjsrv.
`UDFjsrv.java`	User-defined functions called by UDFjcli.java.
`udfjcat`	UDF catalog script for the UDFjsrv program. This script calls UDFjDrop.db2 and UDFjCreate.db2.
`UDFjDrop.db2`	CLP script to drop UDFs from the catalog.
`UDFjCreate.db2`	CLP script to issue CREATE PROCEDURE statements.
`UDFsCreate.db2`	How to catalog the UDFs contained in UDFsqlsv.java
`UDFsDrop.db2`	How to uncatalog the UDFs contained in UDFsqlsv.java
`UDFsqlcl.java`	Call the UDFs in UDFsqlsv.java
`UDFsqlsv.java`	User-Defined Functions with SQL statements called by UDFsqlcl.java
Java Beans - Samples that demonstrate Java Bean classes.	
`CreateEmployee.java`	How to create an employee record.
`GeneratePayroll.java`	How to generate payroll reports by department.

Related concepts:

- "Java Sample Programs" on page 107
- "Sample Files" on page 57
- "Sample Programs: Structure and Design" on page 62

Related reference:

- "SQLJ Samples" on page 77
- "Java WebSphere Samples" on page 79
- "Java Plugin Samples" on page 79

SQLJ Samples

Table 8. Sample SQLJ Program Files

Sample Program Name	Program Description
Tutorial Samples - Programs that demonstrate basic database operations.	
`TutMod.sqlj`	How to modify table data.
`TutRead.sqlj`	How to read tables.
Database Level - Samples that deal with database objects in DB2.	
`DbAuth.sqlj`	How to grant/display/revoke authorities at the database level.
`DbConn.sqlj`	How to connect and disconnect from a database.
`DbMCon.java`	How to connect and disconnect from multiple databases.
`DbMCon1.sqlj`	Support file for DbMCon.java.
`DbMCon2.sqlj`	Support file for DbMCon.java.
`DbUse.sqlj`	How to use database objects.
Table Level - Samples that deal with table objects in DB2.	
`TbConstr.sqlj`	How to work with table constraints.
`TbCreate.sqlj`	How to create, alter and drop tables.
`TbIdent.sqlj`	How to use identity columns.
`TbInfo.sqlj`	How to get and set information at a table level.
`TbMod.sqlj`	How to modify information in a table.
`TbPriv.sqlj`	How to grant/display/revoke table level privileges.
`TbRead.sqlj`	How to read information in a table.
`TbTrig.sqlj`	How to use a trigger on a table.
Data Type Level - Samples that deal with data types.	
`DtUdt.sqlj`	How to create, use, and drop user-defined distinct types.
Applet Level - Samples that demonstrate applets.	
`Applt.sqlj`	How to use applets.
Stored Procedure Level - Samples that demonstrate stored procedures.	
`spcat`	Stored procedure catalog script for the SpServer program. This script calls SpDrop.db2 and SpCreate.db2.
`SpCreate.db2`	CLP script to issue CREATE PROCEDURE statements.
`SpDrop.db2`	CLP script to drop stored procedures from the catalog.
`SpClient.sqlj`	Client program used to call the server functions declared in SpServer.sqlj.
`SpServer.sqlj`	Stored procedure functions built and run on the server.
`SpIterat.sqlj`	Iterator class file for SpServer.sqlj.

Table 8. Sample SQLJ Program Files (continued)

Sample Program Name	Program Description
UDF Level - Samples that demonstrate user-defined functions.	
`UDFcli.sqlj`	Client application which calls the user-defined function library UDFsrv.
`UDFsrv.java`	User-defined functions called by UDFcli.
`udfcat`	UDF catalog script for the UDFsrv program. This script calls UDFDrop.db2 and UDFCreate.db2.
`UDFDrop.db2`	CLP script to drop UDFs from the catalog.
`UDFCreate.db2`	CLP script to issue CREATE PROCEDURE statements.
`UDFjcli.sqlj`	Client application which calls the user-defined function library UDFjsrv.
`UDFjsrv.java`	User-defined functions called by UDFjcli.
`udfjcat`	UDF catalog script for the UDFjsrv program. This script calls UDFjDrop.db2 and UDFjCreate.db2.
`UDFjDrop.db2`	CLP script to drop UDFs from the catalog.
`UDFjCreate.db2`	CLP script to issue CREATE PROCEDURE statements.
Java Beans - Samples that demonstrate Java Bean classes.	
`CreateEmployee.sqlj`	How to create an employee record.
`GeneratePayroll.sqlj`	How to generate payroll reports by department.

Related concepts:

- "Java Sample Programs" on page 107
- "Sample Files" on page 57
- "Sample Programs: Structure and Design" on page 62

Related reference:

- "JDBC Samples" on page 74
- "Java WebSphere Samples" on page 79
- "Java Plugin Samples" on page 79

Java WebSphere Samples

Table 9. Java WebSphere Sample Files

Sample Program Name	Program Description
`AccessEmployee.ear`	This Enterprise ARchive (.EAR) file consists of four modules containing 32 different `.class`, `.JSP` and `.HTML` files. This EAR file, easily deployed using IBM WebSphere Application Server, demonstrates how Java clients can interact with Enterprise Java Beans (EJBs) to access data stored in DB2. The `AccessEmployee.ear` file is located in the `samples/java/WebSphere` directory.

Related concepts:

- "Java Sample Programs" on page 107
- "Sample Files" on page 57
- "Sample Programs: Structure and Design" on page 62

Related reference:

- "JDBC Samples" on page 74
- "SQLJ Samples" on page 77
- "Java Plugin Samples" on page 79

Java Plugin Samples

Table 10. Java Control Center Plugin Sample Files

Sample Program Name	Program Description
`Example1.java`	How to add a new toolbar button to the Control Center toolbar.
`Example2.java`	How to add new menu actions to Control Center Database objects.
`Example3.java`	How to add new objects under Database objects in the Control Center tree.
`Example3Folder.java`	How to add new objects under Database objects in the Control Center tree.
`Example4.java`	How to remove the Configure menu items for Database and Instance objects.
`Example5.java`	How to remove Alter actions from Tables.
`Example6.java`	How to disable Database Configuration default buttons.

Related concepts:

- "Introducing the plug-in architecture for the Control Center" in the *Administration Guide: Implementation*
- "Control Center plug-in performance considerations" in the *Administration Guide: Implementation*

- "Compiling and running the example plugins" in the *Administration Guide: Implementation*
- "Java Sample Programs" on page 107
- "Sample Files" on page 57
- "Sample Programs: Structure and Design" on page 62
- "Writing plugins as Control Center extensions" in the *Administration Guide: Implementation*

Related tasks:

- "Creating a plugin that adds a toolbar button" in the *Administration Guide: Implementation*
- "Setting attributes for a plugin tree object" in the *Administration Guide: Implementation*

Related reference:

- "JDBC Samples" on page 74
- "SQLJ Samples" on page 77
- "Java WebSphere Samples" on page 79

COBOL Samples

Note: The COBOL samples are not structured in the DB2 level design used for the C, CLI, C++, and Java samples.

Table 11. COBOL DB2 API Sample Programs with No Embedded SQL

Sample Program	Included APIs
`checkerr.cbl`	• sqlaintp - Get Error Message • sqlogstt - Get SQLSTATE Message
`client.cbl`	• sqleqryc - Query Client • sqlesetc - Set Client
`d_dbconf.cbl`	• sqleatin - Attach • sqledtin - Detach • sqlfddb - Get Database Configuration Defaults
`d_dbmcon.cbl`	• sqleatin - Attach • sqledtin - Detach • sqlfdsys - Get Database Manager Configuration Defaults
`db_udcs.cbl`	• sqleatin - Attach • sqlecrea - Create Database • sqledrpd - Drop Database

Table 11. COBOL DB2 API Sample Programs with No Embedded SQL (continued)

Sample Program	Included APIs
dbcat.cbl	• sqlecadb - Catalog Database • sqledcls - Close Database Directory Scan • sqledgne - Get Next Database Directory Entry • sqledosd - Open Database Directory Scan • sqleuncd - Uncatalog Database
dbcmt.cbl	• 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.cbl	• sqleatin - Attach • sqlecrea - Create Database • sqledrpd - Drop Database • sqlfrdb - Reset Database Configuration • sqlfudb - Update Database Configuration • sqlfxdb - Get Database Configuration
dbinst.cbl	• sqleatcp - Attach and Change Password • sqleatin - Attach • sqledtin - Detach • sqlegins - Get Instance
dbmconf.cbl	• sqleatin - Attach • sqledtin - Detach • sqlfrsys - Reset Database Manager Configuration • sqlfusys - Update Database Manager Configuration • sqlfxsys - Get Database Manager Configuration
dbsnap.cbl	• sqleatin - Attach • sqlmonss - Get Snapshot
dbstart.cbl	• sqlepstart - Start Database Manager
dbstop.cbl	• sqlefrce - Force Application • sqlepstp - Stop Database Manager

Table 11. COBOL DB2 API Sample Programs with No Embedded SQL (continued)

Sample Program	Included APIs
dcscat.cbl	• 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
ebcdicdb.cbl	• sqleatin - Attach • sqlecrea - Create Database • sqledrpd - Drop Database
migrate.cbl	• sqlemgdb - Migrate Database
monreset.cbl	• sqleatin - Attach • sqlmrset - Reset Monitor
monsz.cbl	• sqleatin - Attach • sqlmonss - Get Snapshot • sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer
nodecat.cbl	• sqlectnd - Catalog Node • sqlencls - Close Node Directory Scan • sqlengne - Get Next Node Directory Entry • sqlenops - Open Node Directory Scan • sqleuncn - Uncatalog Node
restart.cbl	• sqlerstd - Restart Database
setact.cbl	• sqlesact - Set Accounting String
sws.cbl	• sqleatin - Attach • sqlmon - Get/Update Monitor Switches

Table 12. COBOL DB2 API Embedded SQL Sample Programs

Sample Program	Included APIs
dbauth.sqb	• sqluadau - Get Authorizations
dbstat.sqb	• db2Reorg - Reorganize Table • db2Runstats - Run Statistics
expsamp.sqb	• sqluexpr - Export • sqluimpr - Import

Table 12. COBOL DB2 API Embedded SQL Sample Programs (continued)

Sample Program	Included APIs
impexp.sqb	• sqluexpr - Export • sqluimpr - Import
loadqry.sqb	• db2LoadQuery - Load Query
rebind.sqb	• sqlarbnd - Rebind
tabscont.sqb	• sqlbctcq - Close Tablespace Container Query • sqlbftcq - Fetch Tablespace Container Query • sqlbotcq - Open Tablespace Container Query • sqlbtcq - Tablespace Container Query • sqlefmem - Free Memory
tabspace.sqb	• 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.sqb	• sqluexpr - Export • sqluload - Load • sqluvqdp - Quiesce Tablespaces for Table
tspace.sqb	• 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

Table 13. COBOL Embedded SQL Sample programs with No DB2 APIs

Sample Program Name	Program Description
advsql.sqb	Demonstrates the use of advanced SQL expressions like CASE, CAST, and scalar full selects.
cursor.sqb	Demonstrates the use of a cursor using static SQL.
delet.sqb	Demonstrates static SQL to delete items from a database.
dynamic.sqb	Demonstrates the use of a cursor using dynamic SQL.
joinsql.sqb	Demonstrates using advanced SQL join expressions.
lobeval.sqb	Demonstrates the use of LOB locators and defers the evaluation of the actual LOB data.
lobfile.sqb	Demonstrates the use of LOB file handles.
lobloc.sqb	Demonstrates the use of LOB locators.
openftch.sqb	Demonstrates fetching, updating, and deleting rows using static SQL.
static.sqb	Demonstrates static SQL to retrieve information.
tabsql.sqb	Demonstrates the use of advanced SQL table expressions.
trigsql.sqb	Demonstrates using advanced SQL triggers and constraints.
updat.sqb	Demonstrates static SQL to update a database.
varinp.sqb	Demonstrates variable input to Embedded Dynamic SQL statement calls using parameter markers.

Related concepts:

• "Sample Files" on page 57

SQL Procedure Samples

Table 14. SQL Procedure Sample Program Files

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.

Table 14. SQL Procedure Sample Program Files (continued)

Sample Program Name	Program Description
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.
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.
nestedsp.db2	The OUT_AVERAGE, OUT_MEDIAN, and MAX_SALARY procedures return average, median and max values from the "staff" table of the sample database.
NestedSP.java	Calls the OUT_AVERAGE 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.

Table 14. SQL Procedure Sample Program Files (continued)

Sample Program Name	Program Description
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.

Related concepts:

• "Sample Files" on page 57

Visual Basic Samples

Table 15. Visual Basic ADO Sample Program Files

Sample Program Name	Program Description
Client Level	
cliExeSQL.bas	How to execute SQL statements.
cli_Info.bas	How to get/set client level information.
Database Level	
dbConn.bas	How to connect/disconnect from a database.
dbInfo.bas	How to get and set information at a database level.
dbCommit.bas	How to control autocommit dynamically on the database level.
Data Type Level	
dtHier.bas	How to retrieve hierarchical data.
dtLob.bas	How to read and write LOB data.
Stored Procedures	
spCall.bas	How to call stored procedures.

Table 15. Visual Basic ADO Sample Program Files (continued)

Sample Program Name	Program Description
User-Defined Functions	
udfUse.bas	How to create and work with UDTs and UDFs.

Table 16. Visual Basic RDO and MTS Sample Program Files

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. The files for this program are in samples\VB\RDO.
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. The files for this program are in samples\VB\MTS.
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. The files for this program are in samples\VB\MTS.

Related concepts:

- "Sample Files" on page 57
- "Sample Programs: Structure and Design" on page 62

Related reference:

- "Windows Management Instrumentation Samples" on page 88

Visual C++ Samples

Table 17. Visual C++ Sample Program Files

Sample Program Name	Program Description
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)
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.

Related concepts:

- "Sample Files" on page 57

Windows Management Instrumentation Samples

Table 18. Windows Management Instrumentation Sample Program Files.

Sample File Name	File Description
backupdb.vbs	How to backup a database
createdb.vbs	How to create and drop a database.
listsvr.vbs	How to enumerate server instances and start/stop a DB2 instance.
perfmon.mof	MOF file for perfmon.vbs.
perfmon.vbs	How to obtain a DB2 performance counter. Note: you must run "mofcomp perfmon.mof" first.
regvar.mof	MOF file for regvar.vbs.
regvar.vbs	How to obtain a DB2 registry variable. Note: you must run "mofcomp regvar.mof" first.
restoredb.vbs	How to restore a database.
rollfwddb.vbs	How to rollforward a database.
updatedbcfg.vbs	How to get and update the database configuration.
updatedbmcfg.vbs	How to get and update the database manager configuration.

Related concepts:

- "Sample Files" on page 57
- "Windows Management Instrumentation (WMI)" on page 291

Related reference:

- "Visual Basic Samples" on page 86

Object Linking and Embedding (OLE) Samples

Table 19. 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++ DB2 CLI sample that calls the Visual Basic stored procedure, `salarysrv`.
`salarycltvb`	A Visual Basic DB2 CLI sample that calls the Visual Basic stored procedure, `salarysrv`.
`salsvado`	An OLE automation stored procedure, implemented in 32-bit Visual Basic and ADO, that demonstrates output parameters by calculating the median salary in newly-created table, STAFF2, and demonstrates result sets by retrieving salaries from the table.
`salclado`	A Visual Basic client that calls the Visual Basic stored procedure, `salsvado`.
`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++).

Related concepts:

- "Sample Files" on page 57

Related reference:

- "Object Linking and Embedding Database (OLE DB) Table Function Samples" on page 90

Object Linking and Embedding Database (OLE DB) Table Function Samples

Table 20. Object Linking and Embedding Database (OLE DB) Table Functions

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

Related concepts:

- "Sample Files" on page 57

Related reference:

- "Object Linking and Embedding (OLE) Samples" on page 89

Command Line Processor (CLP) Samples

Table 21. Command Line Processor (CLP) Sample Scripts.

Sample File Name	File Description
const.db2	Creates a table with a CHECK CONSTRAINT clause.
cte.db2	Demonstrates a common table expression.
flt.db2	Demonstrates a recursive query.
join.db2	Demonstrates an outer join of tables.
stock.db2	Demonstrates the use of triggers.
testdata.db2	Uses DB2 built-in functions such as RAND() and TRANSLATE() to populate a table with randomly generated test data.

Related concepts:

- "Sample Files" on page 57

REXX Samples

Table 22. REXX Sample Program Files.

Sample File Name	File Description
`blobfile.cmd`	Demonstrates Binary Large OBject (BLOB) manipulation.
`chgisl.cmd`	Demonstrates the CHANGE ISOLATION LEVEL API.
`client.cmd`	Demonstrates the SET CLIENT and QUERY CLIENT APIs.
`d_dbconf`	Demonstrates the API: GET DATABASE CONFIGURATION DEFAULTS
`d_dbmcon`	Demonstrates the API: GET DATABASE MANAGER CONFIGURATION DEFAULTS
`db_udcs`	Demonstrates the CREATE DATABASE and DROP DATABASE APIs to simulate the collating behavior of a DB2 for MVS/ESA CCSID 500 (EBCDIC International) collating sequence
`dbauth`	Demonstrates the GET AUTHORIZATIONS API

`dbcat`	Demonstrates the following APIs:
	`CATALOG DATABASE`
	`CLOSE DATABASE DIRECTORY SCAN`
	`GET NEXT DATABASE DIRECTORY ENTRY`
	`OPEN DATABASE DIRECTORY SCAN`
	`UNCATALOG DATABASE`

`dbcmt`	Demonstrates the following APIs:
	`CHANGE DATABASE COMMENT`
	`GET ERROR MESSAGE`
	`INSTALL SIGNAL HANDLER`

`dbconf`	Demonstrates the following APIs:
	`CREATE DATABASE`
	`DROP DATABASE`
	`GET DATABASE CONFIGURATION`
	`RESET DATABASE CONFIGURATION`
	`UPDATE DATABASE CONFIGURATION`

`dbinst`	Demonstrates the following APIs:
	`ATTACH TO INSTANCE`
	`DETACH FROM INSTANCE`
	`GET INSTANCE`

`dbmconf`	Demonstrates the following APIs:
	`GET DATABASE MANAGER CONFIGURATION`
	`RESET DATABASE MANAGER CONFIGURATION`
	`UPDATE DATABASE MANAGER CONFIGURATION`

`dbstart`	Demonstrates the START DATABASE MANAGER API

`dbstat`	Demonstrates the following APIs:
	`REORGANIZE TABLE`
	`RUN STATISTICS`

Table 22. REXX Sample Program Files. (continued)

Sample File Name	File Description

dbstop	Demonstrates the following APIs:
	`FORCE USERS`
	`STOP DATABASE MANAGER`

dcscat	Demonstrates the following APIs:
	`ADD DCS DIRECTORY ENTRY`
	`CLOSE DCS DIRECTORY SCAN`
	`GET DCS DIRECTORY ENTRY FOR DATABASE`
	`GET DCS DIRECTORY ENTRIES`
	`OPEN DCS DIRECTORY SCAN`
	`UNCATALOG DCS DIRECTORY ENTRY`

dynamic	Demonstrates the use of a ″CURSOR″ using dynamic SQL
ebcdicdb	Demonstrates the CREATE DATABASE and DROP DATABASE APIs to simulate the collating behavior of a DB2 for MVS/ESA CCSID 037 (EBCDIC US English) collating sequence
impexp	Demonstrates the EXPORT and IMPORT APIs
lobeval	Demonstrates deferring the evaluation of a LOB within a database
lobfile	Demonstrates the use of LOB file handles
lobloc	Demonstrates the use of LOB locators
lobval	Demonstrates the use of LOBs
migrate	Demonstrates the MIGRATE DATABASE API

nodecat	Demonstrates the following APIs:
	`CATALOG NODE`
	`CLOSE NODE DIRECTORY SCAN`
	`GET NEXT NODE DIRECTORY ENTRY`
	`OPEN NODE DIRECTORY SCAN`
	`UNCATALOG NODE`

quitab	Demonstrates the API: QUIESCE TABLESPACES FOR TABLE

rechist	Demonstrates the following APIs:
	`CLOSE RECOVERY HISTORY FILE SCAN`
	`GET NEXT RECOVERY HISTORY FILE ENTRY`
	`OPEN RECOVER HISTORY FILE SCAN`
	`PRUNE RECOVERY HISTORY FILE ENTRY`
	`UPDATE RECOVERY HISTORY FILE ENTRY`

restart	Demonstrates the RESTART DATABASE API
sqlecsrx	An example of a collating sequence
updat	Uses dynamic SQL to update a database

Related concepts:

- "Sample Files" on page 57

Related tasks:

- "Building REXX Applications on AIX" on page 194
- "Building Object REXX Applications on Windows" on page 323

Log Management User Exit Samples

Note: Instructions for compiling the Log Management User Exit programs are given at the top of each of the source files listed in the following table.

Table 23. Log Management User Exit Sample Program Files.

Sample File Name	File Description
`db2uext2.cadsm`	This is a sample User Exit utilizing Tivoli Storage Manager (TSM) 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 a 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 operating systems. **Note:** Applications on AIX using the TSM API Client must be built with the xlc_r or xlC_r compiler invocations, not with xlc or xlC, even if the applications are single-threaded. This ensures that the libraries are thread-safe. If you have an application that is compiled with a non-thread-safe library, you can apply fixtest IC21925E or contact your application provider. The fixtest is available on the index.storsys.ibm.com anonymous ftp server.
`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 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.

Table 23. Log Management User Exit Sample Program Files. (continued)

Sample File Name	File Description
db2uext2.cxbsa	This is a sample User Exit utilizing XBSA APIs to Archive and Retrieve database log files. This 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 operating systems.

Related concepts:

- "Sample Files" on page 57

Related reference:

- "Tivoli Storage Manager" in the *Data Recovery and High Availability Guide and Reference*

Build Files, Makefiles, and Error-Checking Utilities

Build Files

The files used to demonstrate building sample programs are known as script files on UNIX and batch files on Windows. We refer to them, generically, as build files. They contain the recommended compile and link commands for supported platform compilers.

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. The following table lists the different types of build files for building different types of programs. These build files, unless otherwise indicated, are for supported languages on all supported platforms. The build files have the .bat (batch) extension on Windows, which is not included in the table. There is no extension for UNIX platforms.

Table 24. DB2 Build Files

Build File	Types of Programs Built
bldapp	Application programs
bldrtn	Routines (stored procedures and UDFs)
bldsqlj	Java SQLJ applications
bldsqljs	Java SQLJ routines (stored procedures and UDFs)
bldmc	C/C++ multi-connection applications
bldmt	UNIX C/C++ Multi-Threaded applications

Table 24. DB2 Build Files (continued)

Build File	Types of Programs Built
bldcli	CLI client applications for SQL procedures in the sqlproc samples sub-directory.
bldevm	AIX and Windows event monitor program, `evm` (C language only)

Note: The `bldcli` file is the same as the `bldapp` file in the `samples/cli` directory. It was given a different name because the embedded C `bldapp` file is also included in the `samples/sqlproc` directory.

The following table lists the build files by platform and programming language, and the directories where they are located. In the online documentation, the build file names are hot-linked to the source files in HTML. The user can also access the text files in the appropriate samples directories.

Table 25. Build Files by language and Platform

Platform —> Language	AIX	HP-UX	Linux	Solaris	Windows

C
samples/c
	`bldapp`
	`bldrtn`
	`bldmt`
	`bldmc`
	`bldevm`
		`bldapp`
		`bldrtn`
		`bldmt`
		`bldmc`
			`bldapp`
			`bldrtn`
			`bldmt`
			`bldmc`
				`bldapp`
				`bldrtn`
				`bldmt`
				`bldmc`
					`bldapp.bat`
					`bldrtn.bat`
					`bldmc.bat`
					`bldevm.bat`

C++
samples/cpp
	`bldapp`
	`bldrtn`
	`bldmt`
	`bldmc`
		`bldapp`
		`bldrtn`
		`bldmt`
		`bldmc`
			`bldapp`
			`bldrtn`
			`bldmt`
			`bldmc`
				`bldapp`
				`bldrtn`
				`bldmt`
				`bldmc`
					`bldapp.bat`
					`bldrtn.bat`
					`bldmc.bat`

CLI
samples/cli
	`bldapp`
	`bldrtn`
	`bldmc`
		`bldapp`
		`bldrtn`
		`bldmc`
			`bldapp`
			`bldrtn`
			`bldmc`
				`bldapp`
				`bldrtn`
				`bldmc`
					`bldapp.bat`
					`bldrtn.bat`
					`bldmc.bat`

SQLJ samples/java/sqlj	`bldsqlj` `bldsqljs`	`bldsqlj` `bldsqljs`	`bldsqlj` `bldsqljs`	`bldsqlj` `bldsqljs`	`bldsqlj.bat` `bldsqljs.bat`
IBM COBOL samples/cobol	`bldapp` `bldrtn`	`n/a`	`n/a`	`n/a`	`bldapp.bat` `bldrtn.bat`
Micro Focus COBOL samples/cobol_mf	`bldapp` `bldrtn`	`bldapp` `bldrtn`	`n/a`	`bldapp` `bldrtn`	`bldapp.bat` `bldrtn.bat`

The build files are used in the documentation for building applications and routines because they demonstrate very clearly the compile and link options that DB2 recommends for 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 makefiles access the build files for compiling and linking the programs they make. The build files use the $1 variable on UNIX and the %1 variable on Windows operating systems to substitute internally for the program name. Incremented numbers in these variable names 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 stand-alone applications, routines (stored procedures and UDFs) or more specialized program types such as multi-connection or multi-threaded programs. 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 overwritten each time a program is built, even if the source file is not modified. This is not the case when using 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 overwrite 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 for user ID and password, depending on where the embedded SQL program is being built.

Except for SQLJ, 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 will not need to 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 SQLJ build files require user ID and password for the `db2profc` customizer, even for accessing a local database. This follows the conventions of the type 4 driver.

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.

Related concepts:

- "Makefiles" on page 97
- "Error-Checking Utilities" on page 101
- "Sample Files" on page 57

Related reference:

- "AIX Supported Software for Building and Running Applications" on page 9
- "HP-UX Supported Software for Building and Running Applications" on page 11
- "Linux Supported Software for Building and Running Applications" on page 12
- "Solaris Supported Software for Building and Running Applications" on page 14
- "Windows Supported Software for Building and Running Applications" on page 15

Makefiles

Each samples directory for a supported compiler includes a makefile for building most of the supplied sample programs within the directory. The makefile calls the build files to compile and link each program. The syntax for the makefiles and the output from their commands differ in some important respects from the build files. However, by using the makefile as the 'front-end' for the build files, the user is able to exploit the makefile's simple and powerful commands:

`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.

Java™ does not usually use makefiles, and make executables are not shipped with Java Development Kits (JDKs). However, DB2® provides makefiles as an option for the Java samples, in case the user wants the convenience of the make commands. To use the Java makefiles, you must have a make executable available that is normally used with another language compiler.

Here are the makefiles by platform provided by DB2 for the main programming languages/APIs, and the sample directories where they are placed. These are hot-linked in the online documentation, and the sample programs that they build are linked within them. These files can also be accessed in the sample directories.

Table 26. Sample Makefiles by Platform

Platform —> Language	**AIX**	**HP-UX**	**Linux**	**Solaris**	**Windows**
C samples/c	makefile	makefile	makefile	makefile	makefile
C++ samples/cpp	makefile	makefile	makefile	makefile	makefile
CLI samples/cli	makefile	makefile	makefile	makefile	makefile
JDBC samples/java/jdbc	makefile	makefile	makefile	makefile	makefile
SQLJ samples/java/sqlj	makefile	makefile	makefile	makefile	makefile
IBM COBOL samples/cobol	makefile	n/a	n/a	n/a	makefile
Micro Focus COBOL samples/cobol_mf	makefile	makefile	n/a	makefile	makefile
SQL Procedures samples/sqlproc	makefile	makefile	makefile	makefile	makefile

Unlike the build files, the makefiles will not overwrite 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. Because they require hardcoding the program name within the file itself, you may consider the makefiles 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.

The makefiles organize the programs they call under several client and server program categories represented by variables (see the makefiles for details). If you are adding a program to a makefile, make sure you add it to be accessed by the correct variables. For example, a program that can run on any client (local to the server or remote) is placed under the `client_run` variable.

You also need to specify the program name under the `cleanall` variable to be sure that the executable produced can be deleted by the `make cleanall` command. Also, if it is an embedded SQL program, specify the non-embedded SQL file created as a result of precompilation under the `clean` variable so that the `make clean` command (as well as the `make cleanall` command which calls it) will delete the non-embedded SQL file.

In addition, you need to specify the new file with the correct syntax to call the appropriate build file to compile and link the program.

To appreciate where a new file needs to be added to one of the sample makefiles, here are all the places where the embedded SQL program, `dbauth`, is located in the AIX® C makefile:

```
#*****************************************************************************
#                   2f - make client_run
#*****************************************************************************

client_run : \
       cli_info clisnap \
       dbauth dbconn dbcreate dbinfo dbmcon \
. . .
#*****************************************************************************
#                   2g - make clean
#*****************************************************************************

clean :
       $(ERASE) *.o
       $(ERASE) *.DEL *.TXT *.MSG
       $(ERASE) dbauth.c dbcfg.c dbconn.c dbmcon.c dbmcon1.c dbmcon2.c
. . .
#*****************************************************************************
#                   2h - make cleanall
#*****************************************************************************

cleanall : \
```

```
        clean
        $(ERASE) *.bnd
        $(ERASE) cli_info clisnap
        $(ERASE) dbauth dbcfg dbconn dbcreate dbinfo dbmcon dbmcon1 dbmcon2
. . .
#******************************************************************************
#                   3b -  regular samples, embedded SQL
#******************************************************************************

dbauth :
        $(BLDAPP) dbauth $(ALIAS) $(UID) $(PWD)
```

The three variables following the program name in the last line above, `ALIAS`, `UID`, and `PWD`, represent, respectively, the database alias name, the user ID, and the password for the database. These variables are passed to the build file (in this case, the `bldapp` build file represented by the `BLDAPP` variable). If the program uses embedded SQL, `ALIAS`, `UID`, and `PWD` are in turn passed to the `embprep` precompile and bind script, which the build file calls. Before using the makefile, you may need to change the values for these variables. By default, `ALIAS` is set to the `sample` database, and `UID` and `PWD` have no value set.

`UID` and `PWD`, are optional parameters that do not need to be set 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 will need to modify the makefile to give the correct values to the `UID` and `PWD` variables in order to access the database.

For multi-connection programs, the C, CLI, and C++ makefiles also have a second database alias, `ALIAS2`, which by default is set to the `sample2` database. The corresponding user ID and password variables, `UID2` and `PWD2` have no value set. As with the `UID` and `PWD` variables, they do not need a value if the second database is accessed locally.

The makefiles also define an `ERASE` variable to delete files when the `make clean` and `make cleanall` commands are called. On UNIX, this is set to `rm -f`; on Windows® it is set to `del`.

Related concepts:

- "Build Files" on page 94
- "Error-Checking Utilities" on page 101
- "Sample Files" on page 57

Related reference:

- "AIX Supported Software for Building and Running Applications" on page 9

- "HP-UX Supported Software for Building and Running Applications" on page 11
- "Linux Supported Software for Building and Running Applications" on page 12
- "Solaris Supported Software for Building and Running Applications" on page 14
- "Windows Supported Software for Building and Running Applications" on page 15

Error-Checking Utilities

The DB2® AD Client provides several utility files. These files have functions for error-checking and printing out error information. 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` (sqlogstt) and `GETERROR MESSAGE` (sqlaintp) 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 routines (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 IBM COBOL programs

checkerr.cbl
: For Micro Focus COBOL programs

utilcli.c
: For CLI programs

utilcli.h
: Header file for utilcli.c

utilapi.c
: For C non-embedded SQL programs

utilapi.h
: Header file for utilapi.c

utilemb.sqc
For C embedded SQL programs

utilemb.h
Header file for utilemb.sqc

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

utilapi.h
Header file for utilapi.C

utilemb.sqC
For C++ embedded SQL programs

utilemb.h
Header file for utilemb.sqC

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 `dbuse` calls the utility functions by using the macro, supplying the value `"Delete with host variables -- Execute"` for the `MSG_STR` parameter of the `SqlInfoPrint()` function:

```
EXEC SQL DELETE FROM org
  WHERE deptnumb = :hostVar1 AND
        division = :hostVar2;
EMB_SQL_CHECK("Delete with host variables -- Execute");
```

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

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

Related concepts:

- "Build Files" on page 94
- "Makefiles" on page 97
- "Sample Files" on page 57

Part 2. Building and Running Platform-Independent Applications

Chapter 4. Java

This chapter provides detailed information for building Java applets and applications. For the latest DB2 Java application development updates, visit the Web page at:

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

Java Sample Programs

DB2® provides sample programs to demonstrate building and running JDBC programs that exclusively use dynamic SQL, and SQLJ programs that use static SQL. There are separate directories for JDBC and SQLJ samples under the `java` samples directory. Here is the Java samples directory structure on UNIX® and Windows® operating systems:

- On UNIX:

  `sqllib/samples/java`
  Contains a README file for Java sample programs in all sub-directories.

  `sqllib/samples/java/jdbc`
  Contains JDBC sample program files.

  `sqllib/samples/java/sqlj`
  Contains SQLJ sample programs.

  `sqllib/samples/java/Websphere`
  Contains Websphere sample programs.

  `sqllib/samples/java/plugin`
  Contains plugin example files for the DB2 Control Center.

sqllib/samples/java/plugin/doc
Contains javadoc files for the plugin interfaces.

- On Windows:

sqllib\samples\java
Contains a README file for Java sample programs in all sub-directories.

sqllib\samples\java\jdbc
Contains JDBC sample programs.

sqllib\samples\java\sqlj
Contains SQLJ sample programs.

sqllib\samples\java\Websphere
Contains Websphere sample programs.

sqllib\samples\java\plugin
Contains plugin example files for the DB2 Control Center.

sqllib\samples\java\plugin\doc
Contains javadoc files for the plugin interfaces.

The SQLJ samples directory contains build files (scripts on UNIX, batch files on Windows) to build the embedded SQL for Java programs. The JDBC directory does not contain build files because building JDBC programs on the command line using `javac` is so simple that build files are not needed.

Both the JDBC and SQLJ samples directories also contain optional makefiles. Makefiles are not widely used with Java, and the Java Development Kits (JDKs) do not ship with make executable files. DB2 supplies Java sample makefiles in case the user wants the convenience they provide. Each Java `makefile` builds all the supplied sample programs in either the JDBC or SQLJ samples directory. You can use a make program, such as gnumake, that's used with another language compiler.

There are two SQLJ build files provided: `bldsqlj` on UNIX, or `bldsqlj.bat` on Windows, which builds SQLJ applets and applications, and `bldsqljs` on UNIX, or `bldsqljs.bat` on Windows, which builds SQLJ routines (stored procedures and user-defined functions).

Related tasks:

- "Setting Up the Java Environment" on page 22
- "Building JDBC Applets" on page 111
- "Building JDBC Applications" on page 112
- "Building JDBC Routines" on page 113
- "Building SQLJ Applets" on page 117
- "Building SQLJ Applications" on page 119

- "Building SQLJ Routines" on page 125
- "Building SQLJ Programs" on page 116

Related reference:
- "JDBC Samples" on page 74
- "SQLJ Samples" on page 77
- "Java WebSphere Samples" on page 79
- "Java Plugin Samples" on page 79

Java Applet Considerations

DB2® databases can be accessed by using Java™ applets. Please keep the following points in mind when using them:

1. If you are using the now deprecated type 3 driver (also known as the "net" driver), it is essential that the `db2java.zip` file used by the Java applet be at the same FixPak level as the JDBC applet server. Under normal circumstances, `db2java.zip` is loaded from the Web Server where the JDBC applet server is running. This ensures a match. If, however, your configuration has the Java applet loading `db2java.zip` from a different location, a mismatch can occur. Matching FixPak levels between the two files is strictly enforced at connection time. If a mismatch is detected, the connection is rejected, and the client receives one of the following exceptions:
   - If db2java.zip is at DB2 Version 7 FixPak 2 or later:

     ```
     COM.ibm.db2.jdbc.DB2Exception: [IBM][JDBC Driver]
     CLI0621E  Unsupported JDBC server configuration.
     ```

   - If db2java.zip is prior to FixPak 2:

     ```
     COM.ibm.db2.jdbc.DB2Exception: [IBM][JDBC Driver]
     CLI0601E  Invalid statement handle or statement is closed.
     SQLSTATE=S1000
     ```

   If a mismatch occurs, the JDBC applet server logs one of the following messages in the `jdbcerr.log` file:
   - If the JDBC applet server is at DB2 Version 7 FixPak 2 or later:

     ```
     jdbcFSQLConnect: JDBC Applet Server and client (db2java.zip)
     versions do not match. Unable to proceed with connection., einfo= -111
     ```

   - If the JDBC applet server is prior to FixPak 2:

     ```
     jdbcServiceConnection(): Invalid Request Received., einfo= 0
     ```

To test your JDBC environment, you can use the sample file, db2JDBCVersion.java, in sqllib\samples\java (Windows), or in sqllib/samples/java (UNIX). The db2JDBCVersion program checks which version of the DB2 JDBC driver is currently in use, and whether the JDBC environment is correctly set up for it.

Users are strongly recommended to migrate their applets to the type 4 driver.

2. 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 1.3 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 supplied sample applet Applt in those browsers:

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

   As a workaround, 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.

3. 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 sqllib\java directory on Windows® 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 1.3 documentation.
4. 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 by using `kill -9` (for Unix) or the Task Manager (for Windows).

5. For information on running DB2 Java applets on a Web server, specifically the Domino™ Go Webserver, visit:

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

Related tasks:

- "Setting Up the Java Environment" on page 22
- "Building JDBC Applets" on page 111
- "Building SQLJ Applets" on page 117

JDBC

Building JDBC Applets

`Applt` demonstrates a dynamic SQL Java applet to access a DB2 database.

Procedure:

You can use the, now deprecated, type 3 driver (also known as the "net" driver), or the type 4 driver. Sections for connecting with both these drivers are presented below. It is strongly recommended that you migrate your applets to the type 4 driver.

To build and run the JDBC applet, `Applt`, by commands entered at the command line, either ensure that a web server is installed and running on your DB2 machine (server or client), or 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
```

Connecting with the Type 3 ("net") Driver

To connect with the type 3 driver, first modify the `Applt.html` file according to the instructions in the file. Then, 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
```

Make sure the JDBC port number in the connection string is the recommended default, "6789". Only change this if you are sure the number does not conflict with another port number. Do not use the database port number, "50000".

Connecting with the Type 4 Driver

To connect with the type 4 driver, modify the `Applt.html` file according to the instructions in the file, except the TCP/IP port number specified should be the database port number, "50000".

Building the Applet

1. Compile `Applt.java` to produce the file `Applt.class` with this command:
   ```
   javac Applt.java
   ```
2. Ensure that your working directory is accessible by your web browser. If it is not, copy `Applt.class` and `Applt.html` into a directory that is accessible.
3. If using a type 3 driver, copy `sqllib\java\db2java.zip` on Windows, or `sqllib/java/db2java.zip` on UNIX, into the same directory as `Applt.class` and `Applt.html`.

   If using a type 4 driver, copy `sqllib\java\db2jcc.jar` on Windows or `sqllib/java/db2jcc.jar` on UNIX, into the same directory as `Applt.class` and `Applt.html`.
4. On your client machine, start your web browser (which must support Java 1.3) and load `Applt.html`.

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

Related concepts:

- "Java Applet Considerations" on page 109

Related tasks:

- "Building JDBC Applications" on page 112
- "Building JDBC Routines" on page 113
- "Building SQLJ Applets" on page 117

Related reference:

- "JDBC Samples" on page 74

Related samples:

- "Applt.java -- A Java applet that use JDBC applet driver to access a database (JDBC)"

Building JDBC Applications

`DbInfo` demonstrates a dynamic SQL Java application accessing a DB2 database.

Procedure:

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

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

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

Note: If you are running a Java application on Unix in a 64-bit DB2 instance but the JDK is 32-bit, you must change the DB2 library path before running the application. For example on AIX:

- If using bash or Korn shell:
  ```
  export LIBPATH=$HOME/sqllib/lib32
  ```
- If using C shell:
  ```
  setenv LIBPATH $HOME/sqllib/lib32
  ```

Related tasks:

- "Building JDBC Applets" on page 111
- "Building JDBC Routines" on page 113
- "Building SQLJ Applications" on page 119

Related reference:

- "JDBC Samples" on page 74

Related samples:

- "DbInfo.java -- How to get/set info in a database (JDBC)"

Building JDBC Routines

DB2 provides sample programs demonstrating JDBC routines (stored procedures and user-defined functions) in the `samples/java/jdbc` directory on UNIX, and the `samples\java\jdbc` directory on Windows. Routines 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.

Procedure:

The following examples show you how to build routines comprising:

- stored procedures
- user-defined functions without SQL statements
- user-defined functions with SQL statements

Stored Procedures

Spserver demonstrates dynamic SQL PARAMETER STYLE JAVA stored procedures.

To build and run this program on the server from 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 sqllib\function directory on Windows operating systems, or to the sqllib/function directory on UNIX.
3. Next, catalog the routines by running the spcat script on the server. Enter:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling Spdrop.db2, then catalogs them by calling Spcreate.db2, and finally disconnects from the database. You can also run the Spdrop.db2 and Spcreate.db2 scripts individually.
4. Then, stop and restart the database to allow the new class file to be recognized. If necessary, set the file mode for the class file to "read" so it is readable by the fenced user.
5. Compile and run the Spclient client application to access the stored procedure class.

User-Defined Functions without SQL Statements

UDFsrv is a user-defined function library that does not contain SQL statements. DB2 provides both a JDBC client application, UDFcli, and an SQLJ client application, UDFcli, that can access the UDFsrv library.

To build and run the UDF program on the server from 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 sqllib\function directory on Windows 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. Both versions of the client program contain the CREATE FUNCTION SQL statement that you use to register the UDFs contained in UDFsrv with the database, and also contain SQL statements that make use of the UDFs, once they have been registered.

User-Defined Functions with SQL Statements

UDFsqlsv is a user-defined function library that contains SQL statements. DB2 provides a JDBC client application, UDFsqlcl, to access the UDFsqlsv library.

To build and run the UDF program on the server from the command line:

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

   ```
   javac UDFsqlsv.java
   ```

2. Copy `UDFsqlsv.class` to the `sqllib\function` directory on Windows operating systems, or to the `sqllib/function` directory on UNIX.
3. To access the `UDFsqlsv` library, use the client program, `UDFsqlcl`, which contains the `CREATE FUNCTION` SQL statement that you use to register the UDFs contained in `UDFsqlsv` with the database. The client program also contains SQL statements that make use of the UDFs, once they have been registered.

You can also use the Java makefile to build the above programs.

Related tasks:

- "Building JDBC Applets" on page 111
- "Building JDBC Applications" on page 112
- "Building SQLJ Routines" on page 125

Related reference:

- "JDBC Samples" on page 74

Related samples:

- "spcat -- To catalog SQLj stored procedures on UNIX"
- "SpClient.java -- Call a variety of types of stored procedures from SpServer.java (JDBC)"
- "SpCreate.db2 -- How to catalog the stored procedures contained in SpServer.java "
- "SpDrop.db2 -- How to uncatalog the stored procedures contained in SpServer.java"
- "SpServer.java -- Provide a variety of types of stored procedures to be called from (JDBC)"
- "UDFcli.java -- Call the UDFs in UDFsrv.java (JDBC)"
- "UDFCreate.db2 -- How to catalog the Java UDFs contained in UDFsrv.java "
- "UDFDrop.db2 -- How to uncatalog the Java UDFs contained in UDFsrv.java "
- "UDFsCreate.db2 -- How to catalog the UDFs contained in UDFsqlsv.java "
- "UDFsDrop.db2 -- How to uncatalog the UDFs contained in UDFsqlsv.java "
- "UDFsqlcl.java -- Call the UDFs in UDFsqlsv.java (JDBC)"

- "UDFsqlsv.java -- Provide UDFs to be called by UDFsqlcl.java (JDBC)"
- "UDFsrv.java -- Provide UDFs to be called by UDFcli.java (JDBC)"

SQLJ

Building SQLJ Programs

DB2 supplies build files to build the SQLJ sample programs. For applets and applications, you can use the `bldsqlj` script on UNIX or the `bldsqlj.bat` batch file on Windows. For routines (stored procedures and user-defined functions), you can use the `bldsqljs` script on UNIX, or the `bldsqljs.bat` batch file on Windows.

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

Note: In previous versions of DB2, the `db2profc` command used a URL of the form `-url=jdbc:db2:dbname` where `dbname` was the locally cataloged database alias. The new format follows the conventions for the DB2 type 4 JDBC driver: `-url=jdbc:db2://hostname:portnumber/dbname` where `hostname` is the name of the DB2 server, `portnumber` is the TCP/IP listener port number of the DB2 server and `dbname` is the database alias cataloged on the DB2 Server. This means the DB2 Server must be configured for TCP/IP connections.

Procedure:

To build and run SQLJ programs with the IBM Java Development Kit for UNIX and Windows operating systems, you must turn off the JDK's just-in-time compiler with the following command for your operating system:

UNIX: For bash or Korn shell:

```
export JAVA_COMPILER=NONE
```

For C shell:

```
setenv JAVA_COMPILER NONE
```

Windows:

```
SET JAVA_COMPILER=NONE
```

To build the different types of DB2 SQLJ programs, see the following:

- Building SQLJ Applets
- Building SQLJ Applications
- Building SQLJ Routines

Related concepts:

- "Java Sample Programs" on page 107
- "Java Applet Considerations" on page 109

Related tasks:

- "Building SQLJ Applets" on page 117
- "Building SQLJ Applications" on page 119
- "Building SQLJ Routines" on page 125

Related reference:

- "SQLJ Samples" on page 77

Building SQLJ Applets

The following steps show how to build the `Applt` sample that demonstrates an SQLJ applet accessing a DB2 database. These steps use the build file, `bldsqlj` (UNIX), or `bldsqlj.bat` (Windows), which contains commands to build either an SQLJ applet or application.

The build file takes up to six parameters: $1, $2, $3, $4, $5, and $6 on UNIX, and %1, %2, %3, %4, %5, and %6 on Windows. The first parameter specifies the name of your program. The second parameter specifies the user ID for the database instance, the third parameter specifies the password. The fourth parameter specifies the server name. The fifth parameter specifies the port number. And the sixth parameter specifies the database name. For all but the first parameter, program name, default values can be used. See the build file for details about using default parameter values.

Procedure:

You can use the, now deprecated, type 3 driver (also known as the "net" driver), or the type 4 driver. Sections for connecting with both these drivers are presented below. It is strongly recommended that you migrate your applets to the type 4 driver.

To run this applet, either ensure that a web server is installed and running on your DB2 machine (server or client), or 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
```

Connecting with the Type 3 ("net") Driver

To connect with the type 3 driver, first modify the `Applt.html` file according to the instructions in the file. Then, 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
```

Make sure the JDBC port number in the connection string is the recommended default, "6789". Only change this if you are sure the number does not conflict with another port number. Do not use the database port number, "50000".

Connecting with the Type 4 Driver

To connect with the type 4 driver, modify the Applt.html file according to the instructions in the file, except the TCP/IP port number specified should be the database port number, "50000".

Building the Applet

1. Build the applet with this command:

   ```
   bldsqlj Applt <userid> <password> <server_name> <port_number> <db_name>
   ```

   where all parameters except the program name can have default values, as explained in the build file.
2. Ensure that your working directory is accessible by your web browser, or by appletviewer if you are using it. If your directory is not accessible, 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
   ```
3. If using a type 3 driver, copy sqllib\java\runtime.zip and sqllib\java\db2java.zip on Windows, or sqllib/java/runtime.zip and sqllib/java/db2java.zip on UNIX, into the same directory as Applt.class and Applt.html.

   If using a type 4 driver, copy sqllib\java\runtime.zip and sqllib\java\db2jcc.jar on Windows or sqllib/java/runtime.zip and sqllib/java/db2jcc.jar on UNIX, into the same directory as Applt.class and Applt.html.
4. On your client machine, start your web browser (which must support JDK 1.3), or appletviewer, and load Applt.html.

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

Related concepts:

- "Java Applet Considerations" on page 109

Related tasks:

- "Building JDBC Applets" on page 111
- "Building SQLJ Applications" on page 119
- "Building SQLJ Routines" on page 125

Related reference:

- "SQLJ Application Options for UNIX" on page 121
- "SQLJ Application Options for Windows" on page 124
- "SQLJ Samples" on page 77

Related samples:

- "Applt.sqlj -- An SQLJ applet that uses a JDBC applet driver to access a database (SQLj)"
- "bldsqlj.bat -- Builds a Java embedded SQL (SQLJ) application or applet on Windows"
- "bldsqlj -- Builds Java embedded SQL (SQLJ) applications and applets on UNIX"

Building SQLJ Applications

The following steps show how to build the `TbMod` sample that demonstrates an SQLJ application accessing a DB2 database. These steps use the build file, `bldsqlj` (UNIX), or `bldsqlj.bat` (Windows), which contains commands to build either an SQLJ applet or application.

The build file takes up to six parameters: $1, $2, $3, $4, $5, and $6 on UNIX, and %1, %2, %3, %4, %5, and %6 on Windows. The first parameter specifies the name of your program. The second parameter specifies the user ID for the database instance, the third parameter specifies the password. The fourth parameter specifies the server name. The fifth parameter specifies the port number. And the sixth parameter specifies the database name. For all but the first parameter, program name, default values can be used. See the build file for details about using default parameter values.

Procedure:

To build `TbMod` with the build file, `bldsqlj` (UNIX) or `bldsqlj.bat` (Windows), enter this command:

```
bldsqlj TbMod <userid> <password> <server_name> <port_number> <db_name>
```

where all parameters except the program name can have default values, as explained in the build file.

Run the Java interpreter on the application with this command:

```
java TbMod
```

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

Note: If you are running a Java application on Unix in a 64-bit DB2 instance but the JDK is 32-bit, you must change the DB2 library path before running the application. For example on AIX:

- If using bash or Korn shell:

```
export LIBPATH=$HOME/sqllib/lib32
```

- If using C shell:

```
setenv LIBPATH $HOME/sqllib/lib32
```

Related tasks:

- "Building JDBC Applications" on page 112
- "Building SQLJ Applets" on page 117
- "Building SQLJ Routines" on page 125

Related reference:

- "SQLJ Application Options for UNIX" on page 121
- "SQLJ Application Options for Windows" on page 124
- "SQLJ Samples" on page 77

Related samples:

- "bldsqlj.bat -- Builds a Java embedded SQL (SQLJ) application or applet on Windows"
- "bldsqlj -- Builds Java embedded SQL (SQLJ) applications and applets on UNIX"
- "TbMod.sqlj -- How to modify table data (SQLj)"

UNIX Build Script for SQLJ Applications and Applets

```
#! /bin/sh
# SCRIPT: bldsqlj
# Builds UNIX Java embedded SQL (SQLJ) applications
# Usage: bldsqlj prog_name (requires hardcoding user ID and password)
#        bldsqlj prog_name userid password
#        bldsqlj prog_name userid password server_name
#        bldsqlj prog_name userid password server_name port_number
#        bldsqlj prog_name userid password server_name port_number db_name
#
#        Defaults:
#          userid      = $USER variable
#          password    = $PSWD variable
#          server_name = local hostname
#          port_number = 50000
#          db_name     = sample

# To add defaults for user ID (USER) and password (PSWD)
# Uncomment the following and add the values in the quotes
```

```
# USER=""
# PSWD=""


# Translate and compile the SQLJ source file
# and bind the package to the database.
 if ( (( $# == 1 )) && [ $USER != "" && $PSWD != "" ] ) ||
    (( $# >= 3 && $# <= 6 ))
 then
    sqlj $1.sqlj

    if (($# == 1))
    then
       db2profc -url=jdbc:db2://$(hostname):50000/sample \
         -user=$USER -password=$PSWD \
         -prepoptions="package using $1" $1_SJProfile0
    elif (($# == 3))
    then
       db2profc -url=jdbc:db2://$(hostname):50000/sample -user=$2 \
         -password=$3 -prepoptions="package using $1" $1_SJProfile0
    elif (($# == 4))
    then
       db2profc -url=jdbc:db2://$4:50000/sample -user=$2 -password=$3 \
         -prepoptions="package using $1" $1_SJProfile0
    elif (($# == 5))
    then
       db2profc -url=jdbc:db2://$4:$5/sample -user=$2 -password=$3 \
         -prepoptions="package using $1" $1_SJProfile0
    else
       db2profc -url=jdbc:db2://$4:$5/$6 -user=$2 -password=$3 \
         -prepoptions="package using $1" $1_SJProfile0
    fi
 else
    echo 'Usage: bldsqlj prog_name (requires hardcoding user ID and password)'
    echo '       bldsqlj prog_name userid password'
    echo '       bldsqlj prog_name userid password server_name'
    echo '       bldsqlj prog_name userid password server_name port_number'
    echo '       bldsqlj prog_name userid password server_name port_number
      db_name'
    echo ''
    echo '       Defaults:'
    echo '         userid      = '$USER
    echo '         password    = '$PSWD
    echo '         server_name = '$(hostname)
    echo '         port_number = 50000'
    echo '         db_name     = sample'
 fi
```

SQLJ Application Options for UNIX

The following table contains the SQLJ translator and precompile options used in the `bldsqlj` build script on UNIX. These are the options DB2 recommends that you use to build SQLJ applications and applets on UNIX platforms.

Translator and Precompile Options for **bldsqlj**	
sqlj	The SQLJ translator (also compiles the program).
$1.sqlj	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://servername:50000/sample`.
-user	Specifies a user ID.
-password	Specifies a password.
-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.

Related tasks:

- "Building SQLJ Applets" on page 117
- "Building SQLJ Applications" on page 119

Related reference:

- "SQLJ Stored Procedure Options for UNIX" on page 128

Related samples:

- "bldsqlj -- Builds Java embedded SQL (SQLJ) applications and applets on UNIX"

Windows Batch File for SQLJ Applications and Applets

```
@echo off
rem BATCH FILE: bldsqlj.bat
rem Builds Windows Java embedded SQL (SQLJ) applications and applets
rem To add defaults for user ID (USER) and password (PSWD)
rem Uncomment the following and add the appropriate values
rem set USER=
rem set PSWD=

goto start
:usage
echo Usage: bldsqlj prog_name (requires hardcoding user ID and password)
echo        bldsqlj prog_name userid password
echo        bldsqlj prog_name userid password server_name
echo        bldsqlj prog_name userid password server_name port_number
```

```
echo        bldsqlj prog_name userid password server_name port_number db_name
echo.
echo        Defaults:
echo          userid      = %USER%
echo          password    = %PSWD%
echo          server_name = %COMPUTERNAME%
echo          port_number = 50000
echo          db_name     = sample
goto exit

:start
rem Translate and compile the SQLJ source file
rem and bind the package to the database.
if "%1" == "" goto usage
if "%2" == "" goto case1
if "%3" == "" goto usage
if "%4" == "" goto case3
if "%5" == "" goto case4
if "%6" == "" goto case5
if "%7" == "" goto case6
goto usage

:case1
   if "%USER%" == "" goto usage
   if "%PSWD%" == "" goto usage
   if "%COMPUTERNAME%" == "" goto nohostname
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%COMPUTERNAME%:50000/sample -user=%USER%
     -password=%PSWD% -prepoptions="package using %1" %1_SJProfile0
   goto continue

:case3
   if "%COMPUTERNAME%" == "" goto nohostname
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%COMPUTERNAME%:50000/sample -user=%2
     -password=%3 -prepoptions="package using %1" %1_SJProfile0
   goto continue

:case4
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%4:50000/sample -user=%2 -password=%3
     -prepoptions="package using %1" %1_SJProfile0
   goto continue

:case5
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%4:%5/sample -user=%2 -password=%3
     -prepoptions="package using %1" %1_SJProfile0
   goto continue

:case6
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%4:%5/%6 -user=%2 -password=%3
     -prepoptions="package using %1" %1_SJProfile0
   goto continue
```

```
:continue
rem Put any post building steps here
goto exit

:nohostname
echo Local server name (hostname) could not be determined.
echo.
goto usage

:exit
@echo on
```

SQLJ Application Options for Windows

The following table contains the SQLJ translator and precompile options used in the `bldsqlj.bat` batch file on Windows operating systems. These are the options DB2 recommends that you use to build SQLJ routines (stored procedures and user-defined functions).

Translator and Precompile Options for **bldsqlj.bat**	
sqlj	The SQLJ translator (also compiles the program).
%1.sqlj	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://servername:50000/sample`.
-user	Specifies a user ID.
-password	Specifies a password.
-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.

Related tasks:

- "Building SQLJ Applets" on page 117
- "Building SQLJ Applications" on page 119

Related reference:

- "SQLJ Stored Procedure Options for Windows" on page 130

Related samples:

- "bldsqlj.bat -- Builds a Java embedded SQL (SQLJ) application or applet on Windows"

Building SQLJ Routines

DB2 provides sample programs demonstrating SQLJ routines (stored procedures and user-defined functions) in the `samples/java/sqlj` directory on UNIX, and the `samples\java\sqlj` directory on Windows. Routines 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.

In the same directory, DB2 also supplies the build file, `bldsqljs` (UNIX), or `bldsqljs.bat` (Windows), which contains commands to build routines.

The build file takes up to six parameters: $1, $2, $3, $4, $5, and $6 on UNIX, and %1, %2, %3, %4, %5, and %6 on Windows. The first parameter specifies the name of your program. The second parameter specifies the user ID for the database instance, the third parameter specifies the password. The fourth parameter specifies the server name. The fifth parameter specifies the port number. And the sixth parameter specifies the database name. For all but the first parameter, program name, default values can be used. See the build file for details about using default parameter values.

Procedure:

The following example shows you how to build a class file with stored procedures.

`Spserver` demonstrates PARAMETER STYLE JAVA stored procedures using the JDBC application driver to access a DB2 database.

To build this stored procedure class with the build file, `bldsqljs` (UNIX) or `bldsqljs.bat` (Windows):

1. Enter the following command:

   ```
   bldsqljs Spserver <userid> <password> <server_name> \
     <port_number> <db_name>
   ```

   where all parameters except the program name can have default values, as explained in the build file.
2. Next, catalog the routines by running the `spcat` script on the server. Enter:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling `Spdrop.db2`, then catalogs them by calling `Spcreate.db2`, and finally disconnects from the database. You can also run the `Spdrop.db2` and `Spcreate.db2` scripts individually.

3. Then, stop and restart the database to allow the new class file to be recognized. If necessary, set the file mode for the class file to ″read″ so it is readable by the fenced user.
4. Build and run the `Spclient` client application to call the stored procedures. You can build `Spclient` with the application build file, `bldsqlj` (UNIX) or `bldsqlj.bat` (Windows).

You can also use the Java `makefile` to build the above programs.

Related tasks:

- “Building JDBC Routines” on page 113
- “Building SQLJ Applets” on page 117
- “Building SQLJ Applications” on page 119

Related reference:

- “SQLJ Stored Procedure Options for UNIX” on page 128
- “SQLJ Stored Procedure Options for Windows” on page 130
- “SQLJ Samples” on page 77

Related samples:

- “bldsqljs.bat -- Builds a Java embedded SQL (SQLJ) stored procedure on Windows”
- “bldsqljs -- Builds Java embedded SQL (SQLJ) stored procedures on UNIX”
- “spcat -- To catalog SQLj stored procedures on UNIX”
- “SpClient.sqlj -- Call a variety of types of stored procedures from SpServer.sqlj (SQLj)”
- “SpCreate.db2 -- How to catalog the stored procedures contained in SpServer.sqlj ”
- “SpDrop.db2 -- How to uncatalog the stored procedures contained in SpServer.sqlj”
- “SpIterat.sqlj -- Iterator class file for SpServer.sqlj (SQLj)”
- “SpServer.sqlj -- Provide a variety of types of stored procedures to be called from (SQLj)”

UNIX Build Script for SQLJ Routines

```
#! /bin/sh
# SCRIPT: bldsqljs
# Builds UNIX Java embedded SQL (SQLJ) routines
# Usage: bldsqljs prog_name  (requires hardcoding user ID and password)
#        bldsqljs prog_name userid password
#        bldsqljs prog_name userid password server_name
#        bldsqljs prog_name userid password server_name port_number
#        bldsqljs prog_name userid password server_name port_number db_name
#
```

```
#        Defaults:
#          userid      = $USER variable
#          password    = $PSWD variable
#          server_name = local hostname
#          port_number = 50000
#          db_name     = sample

# To add defaults for user ID (USER) and password (PSWD)
# Uncomment the following and add the values in the quotes
# USER=""
# PSWD=""

# 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 ( (( $# == 1 )) && [ $USER != "" && $PSWD != "" ] ) ||
    (( $# >= 3 && $# <= 6 ))
 then
    sqlj $1.sqlj

    if (($# == 1))
    then
       db2profc -url=jdbc:db2://$(hostname):50000/sample \
         -user=$USER -password=$PSWD \
         -prepoptions="package using $1" $1_SJProfile0
    elif (($# == 3))
    then
       db2profc -url=jdbc:db2://$(hostname):50000/sample -user=$2 \
         -password=$3 -prepoptions="package using $1" $1_SJProfile0
    elif (($# == 4))
    then
       db2profc -url=jdbc:db2://$4:50000/sample -user=$2 -password=$3 \
         -prepoptions="package using $1" $1_SJProfile0
    elif (($# == 5))
    then
       db2profc -url=jdbc:db2://$4:$5/sample -user=$2 -password=$3 \
         -prepoptions="package using $1" $1_SJProfile0
    else
       db2profc -url=jdbc:db2://$4:$5/$6 -user=$2 -password=$3 \
         -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
 else
    echo 'Usage: bldsqljs prog_name (requires hardcoding user ID and password)'
    echo '       bldsqljs prog_name userid password'
    echo '       bldsqljs prog_name userid password server_name'
```

```
    echo '       bldsqljs prog_name userid password server_name port_number'
    echo '       bldsqljs prog_name userid password server_name port_number
      db_name'
    echo ''
    echo '       Defaults:'
    echo '         userid      = '$USER
    echo '         password    = '$PSWD
    echo '         server_name = '$(hostname)
    echo '         port_number = 50000'
    echo '         db_name     = sample'
fi
```

SQLJ Stored Procedure Options for UNIX

The following table contains the SQLJ translator and precompile options used in the `bldsqljs` build script on UNIX. These are the options DB2 recommends that you use to build SQLJ routines (stored procedures and user-defined functions) on UNIX platforms.

Translator and Precompile Options for `bldsqljs`
`sqlj` The SQLJ translator (also compiles the program).
`$1.sqlj` 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://servername:50000/sample`.
`-user` Specifies a user ID.
`-password` Specifies a password.
`-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.

Related tasks:

- "Building SQLJ Routines" on page 125

Related reference:

- "SQLJ Application Options for UNIX" on page 121

Related samples:

- "bldsqljs -- Builds Java embedded SQL (SQLJ) stored procedures on UNIX"

Windows Batch File for SQLJ Routines

```
@echo off
rem BATCH FILE: bldsqljs.bat
rem Builds Windows Java embedded SQL (SQLJ) routines

rem To add defaults for user ID (USER) and password (PSWD)
rem Uncomment the following and add the appropriate values
rem set USER=
rem set PSWD=

goto start
:usage
echo Usage: bldsqljs prog_name (requires hardcoding user ID and password)
echo        bldsqljs prog_name userid password
echo        bldsqljs prog_name userid password server_name
echo        bldsqljs prog_name userid password server_name port_number
echo        bldsqljs prog_name userid password server_name port_number db_name
echo.
echo        Defaults:
echo          userid      = %USER%
echo          password    = %PSWD%
echo          server_name = %COMPUTERNAME%
echo          port_number = 50000
echo          db_name     = sample
goto exit

:start
rem Translate and compile the SQLJ source file
rem and bind the package to the database.
if "%DB2PATH%" == "" goto nodb2cmd
if "%1" == "" goto usage
if "%2" == "" goto case1
if "%3" == "" goto usage
if "%4" == "" goto case3
if "%5" == "" goto case4
if "%6" == "" goto case5
if "%7" == "" goto case6
goto usage

:case1
   if "%USER%" == "" goto usage
   if "%PSWD%" == "" goto usage
   if "%COMPUTERNAME%" == "" goto nohostname
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%COMPUTERNAME%:50000/sample -user=%USER%
     -password=%PSWD% -prepoptions="package using %1" %1_SJProfile0
   goto continue

:case3
   if "%COMPUTERNAME%" == "" goto nohostname
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%COMPUTERNAME%:50000/sample -user=%2
     -password=%3 -prepoptions="package using %1" %1_SJProfile0
   goto continue
```

```
:case4
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%4:50000/sample -user=%2 -password=%3
     -prepoptions="package using %1" %1_SJProfile0
   goto continue

:case5
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%4:%5/sample -user=%2 -password=%3
     -prepoptions="package using %1" %1_SJProfile0
   goto continue

:case6
   sqlj %1.sqlj
   db2profc -url=jdbc:db2://%4:%5/%6 -user=%2 -password=%3
     -prepoptions="package using %1" %1_SJProfile0
   goto continue

:continue
rem Copy the *.class and *.ser files to the 'function' directory.
copy %1*.class %DB2PATH%\function\
copy %1*.ser %DB2PATH%\function\
goto exit

:nodb2cmd
echo DB2 command line environment not initialized.  Please run db2cmd
  and try again.
goto exit

:nohostname
echo Local server name (hostname) could not be determined.
echo.
goto usage

:exit
@echo on
```

SQLJ Stored Procedure Options for Windows

The following table contains the SQLJ translator and precompile options used in the `bldsqljs.bat` batch file on Windows operating systems. These are the options DB2 recommends that you use to build SQLJ routines (stored procedures and user-defined functions).

Translator and Precompile Options for `bldsqljs.bat`
`sqlj` The SQLJ translator (also compiles the program).
`%1.sqlj` 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://servername:50000/sample`.
`-user` Specifies a user ID.
`-password` Specifies a password.
`-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.

Related tasks:

- "Building SQLJ Routines" on page 125

Related reference:

- "SQLJ Application Options for Windows" on page 124

Related samples:

- "bldsqljs.bat -- Builds a Java embedded SQL (SQLJ) stored procedure on Windows"

Chapter 5. SQL Procedures

Creating SQL Procedures

The DB2 Command Line Processor scripts (those ending with the `.db2` extension), in the `sqllib/samples/sqlproc` directory on UNIX and the `sqllib\samples\sqlproc` directory on Windows, execute CREATE PROCEDURE statements to create stored procedures on the server. Each CLP script has a corresponding client application file of the same name, with an extension denoting its language and application interface: `.sqc` (for C embedded SQL), `.c` (for DB2 CLI), or `.java` (for JDBC).

Procedure:

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
```

Now, the SQL procedure is ready to be called.

Related tasks:

- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on UNIX" on page 134
- "Calling SQL Procedures with Client Applications on Windows" on page 135
- "Distributing Compiled SQL Procedures" on page 140
- "Rebinding SQL Procedures" on page 141

Related samples:

- "rsultset.db2 -- To create the MEDIAN_RESULT_SET SQL procedure "

Calling SQL Procedures with Client Applications on UNIX

You can call SQL procedures on UNIX by building and running client applications. DB2 supplies sample client programs in the `sqllib/samples/sqlproc` directory to call the SQL procedures that can be created from the sample command line processor scripts that are also supplied. There are client source files for DB2 CLI, C embedded SQL, and JDBC.

Build scripts are also supplied (except for JDBC) to create the client programs for these application interfaces:

- `bldcli` contains the commands to build a DB2 CLI client application for SQL procedures. This is the same file as the `bldapp` script in the `sqllib/samples/cli` directory.
- `bldapp` contains the commands to build a C embedded SQL client application for SQL procedures. This is the same file as the `bldapp` script in the `sqllib/samples/c` directory.

Procedure:

Depending on the application interface you are using, you can build and run a sample client program to call SQL procedures by following these examples:

DB2 CLI

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*

C embedded SQL

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*

JDBC To build the JDBC client application, `NestedSP`, from the source file `NestedSP.java`, compile the source file:

```
javac NestedSP.java
```

The result is the class file, `NestedSP.class`.

To call the stored procedure, run the java interpreter on the application:

```
java NestedSP userid password database
```

Related tasks:

- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on Windows" on page 135
- "Distributing Compiled SQL Procedures" on page 140
- "Rebinding SQL Procedures" on page 141

Related samples:

- "basecase.sqc -- To call the UPDATE_SALARY SQL procedure"
- "rsultset.c -- To call the MEDIAN_RESULT_SET SQL procedure"

Calling SQL Procedures with Client Applications on Windows

You can call SQL procedures on Windows by building and running client applications. DB2 supplies sample client programs in the `sqllib\samples\sqlproc` directory to call the SQL procedures that can be created from the sample command line processor scripts that are also supplied. There are client source files for DB2 CLI, C embedded SQL, and JDBC.

Batch files are provided (except for JDBC) to create the client programs for these application interfaces:

- `bldcli.bat` contains the commands to build a DB2 CLI client application for SQL procedures. This is the same file as `bldapp.bat` in the `sqllib\samples\cli` directory.

- `bldapp.bat` contains the commands to build a C embedded SQL client application for SQL procedures. This is the same file as `bldapp.bat` in the `sqllib\samples\c` directory.

Procedure:

Depending on the application interface you are using, you can build and run a sample client program to call SQL procedures by following these examples:

DB2 CLI

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

```
bldcli rsultset
```

This command creates the executable file, `rsultset.exe`.

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*

C embedded SQL

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

```
bldapp basecase database userid password
```

The result is an executable file, `basecase.exe`.

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

`basecase` *database userid password*

JDBC

To build the JDBC client application, `NestedSP`, from the source file `NestedSP.java`, compile the source file:

```
javac NestedSP.java
```

The result is the class file, `NestedSP.class`.

To call the stored procedure, run the java interpreter on the application:

```
java NestedSP userid password database
```

Related tasks:

- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on UNIX" on page 134
- "Distributing Compiled SQL Procedures" on page 140
- "Rebinding SQL Procedures" on page 141

Related samples:

- "basecase.sqc -- To call the UPDATE_SALARY SQL procedure"
- "rsultset.c -- To call the MEDIAN_RESULT_SET SQL procedure"

Retaining Intermediate Files for SQL Procedures

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 retain the files so you can examine the SQC, C and message log files created by DB2.

Procedure:

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 on the server to ″1″, ″y″ or ″yes″, as in the following command:

```
db2set DB2_SQLROUTINE_KEEP_FILES=1
```

Then stop and restart DB2 for the change to take affect.

You will then have to manually delete intermediate files that may be left when an SQL procedure is not created successfully. These files are in the following directories:

UNIX

```
$HOME/sqllib/function/routine/sqlproc/<db_name>/<schema_name>/tmp
```

Windows

```
sqllib\function\routine\sqlproc\<db_name>\<schema_name>\tmp
```

where <*db_name*> and <*schema_name*> are the database and schema used to create the SQL procedures.

Related tasks:

- "Customizing Precompile and Bind Options for SQL Procedures" on page 138

- "Backing Up and Restoring SQL Procedures" on page 139
- "Creating SQL Procedures" on page 133
- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on UNIX" on page 134
- "Calling SQL Procedures with Client Applications on Windows" on page 135
- "Distributing Compiled SQL Procedures" on page 140
- "Rebinding SQL Procedures" on page 141

Customizing Precompile and Bind Options for SQL Procedures

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.

Procedure:

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}
FEDERATED {NO | YES}
INSERT {DEF | BUF}
ISOLATION {CS |RR |UR |RS |NC}
QUERYOPT optimization-level
SYNCPOINT {ONEPHASE | TWOPHASE | NONE}
```

Example. To have the package use the ISO format for dates and Repeatable Read as the isolation level, specify the options as follows:

```
db2set DB2_SQLROUTINE_PREPOPTS="DATETIME ISO ISOLATION RR"
```

Then stop and restart DB2 for the change to take affect.

Related tasks:

- "Retaining Intermediate Files for SQL Procedures" on page 137

- "Backing Up and Restoring SQL Procedures" on page 139
- "Creating SQL Procedures" on page 133
- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on UNIX" on page 134
- "Calling SQL Procedures with Client Applications on Windows" on page 135
- "Distributing Compiled SQL Procedures" on page 140
- "Rebinding SQL Procedures" on page 141

Backing Up and Restoring SQL Procedures

When an SQL procedure is created, the generated shared dynamic linked library (DLL) is kept in the database catalog, along with the source text, package, and related files. Therefore, all this information is saved when you perform a database backup.

Procedure:

At database recovery time, all SQL procedure executables on the filesystem that belong to the database being recovered will be removed. If the index creation configuration parameter, `indexrec`, is set to RESTART, all SQL procedure executables will be extracted from the catalog table and put back on the filesystem at next connect time. Otherwise, the SQL executables will be extracted on first execution of the SQL procedures.

The executables will be put back in the following directory:

UNIX $HOME/sqllib/function/routine/sqlproc/<*database_name*>

Windows
sqllib\function\routine\sqlproc\<*database_name*>

where <*database_name*> represents the database with which the SQL procedures were created.

If the first attempt to connect to a database after a restore operation returns:

```
SQL2048N An error occurred while accessing object "SQL PROCEDURE FILES".
Reason code: "7".
```

Stop DB2 with `db2stop`, and restart with `db2start`.

Related tasks:

- "Retaining Intermediate Files for SQL Procedures" on page 137

- "Customizing Precompile and Bind Options for SQL Procedures" on page 138
- "Creating SQL Procedures" on page 133
- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on UNIX" on page 134
- "Calling SQL Procedures with Client Applications on Windows" on page 135
- "Distributing Compiled SQL Procedures" on page 140
- "Rebinding SQL Procedures" on page 141

Distributing Compiled SQL Procedures

When you define an SQL procedure, it is converted to a C program, precompiled and bound against the target database, and compiled and linked to create a shared library. The compile and link steps require a C or C++ compiler to be available on the database server machine. However, once you define an SQL procedure, you can distribute it in compiled form to DB2 database server machines that use the same operating system and the same version of DB2, but do not necessarily have access to a C or C++ compiler. Under these conditions, DB2 allows the user to extract SQL procedures in compiled form from one database and install SQL procedures in compiled form on another database on a different server machine.

DB2 provides both a command line interface and a programming interface for extracting and installing. The command line interface consists of two CLP commands: `GET ROUTINE` and `PUT ROUTINE`. The programming interface consists of two built-in stored procedures: `GET_ROUTINE_SAR` and `PUT_ROUTINE_SAR`.

Procedure:

To distribute a compiled SQL procedure from one database server to another database server, perform the following steps:

1. Build the application, including defining the SQL procedures that are part of the application.
2. After testing the procedures, extract the compiled version of each procedure into a different file, either by issuing the `GET ROUTINE` command or by invoking the `GET_ROUTINE_SAR` stored procedure. Copy the files to your distribution media (if necessary).

   Example. Assume `SCHEMA1.MYPROC` is an SQL Procedure that exists on the source server. The following command creates a file named `myproc.sar` that contains a distributable representation of the procedure:

   ```
   GET ROUTINE INTO myproc.sar FROM PROCEDURE SCHEMA1.MYPROC
   ```

3. Install the compiled version of each procedure on each server, either by issuing the PUT ROUTINE command, or by invoking the PUT_ROUTINE_SAR stored procedure, using the files created in the previous step.

   Example. The following command installs the procedure from the previous example on the target server:

   ```
   PUT ROUTINE FROM myproc.sar
   ```

   Each database server must have the same operating system and DB2 level.

Note: If a GET ROUTINE or a PUT ROUTINE operation (or their corresponding procedure) fails to execute successfully, it will always return an error, along with diagnostic text providing information about the cause of the failure. For example, if the procedure name provided to GET ROUTINE does not identify an SQL procedure, diagnostic ″-204, 42704″ text will be returned, where ″-204″ and ″42704″ are the SQLCODE and SQLSTATE, respectively, that identify the cause of the problem. The SQLCODE and SQLSTATE in this example indicate that the procedure name provided in the GET ROUTINE command is undefined.

Related tasks:

- “Setting Up the SQL Procedures Environment” on page 23
- “Creating SQL Procedures” on page 133
- “Calling Stored Procedures with the CALL Statement” on page 143
- “Calling SQL Procedures with Client Applications on UNIX” on page 134
- “Calling SQL Procedures with Client Applications on Windows” on page 135
- “Rebinding SQL Procedures” on page 141

Related reference:

- “GET_ROUTINE_SAR procedure” in the *SQL Reference, Volume 1*
- “PUT_ROUTINE_SAR procedure” in the *SQL Reference, Volume 1*
- “GET ROUTINE” in the *Command Reference*
- “PUT ROUTINE” in the *Command Reference*

Rebinding SQL Procedures

Procedure:

To rebind the package corresponding to an SQL procedure, call the SYSPROC.REBIND_ROUTINE_PACKAGE built-in stored procedure.

For example, if an SQL procedure named MYSCHEMA.MYPROC exists in the database, its package can be rebound from the command line processor (CLP) by issuing the following command:

```
CALL SYSPROC.REBIND_ROUTINE_PACKAGE('P', 'MYSCHEMA.MYPROC', 'CONSERVATIVE')
```

where 'P' indicates that 'MYSCHEMA.MYPROC' is a procedure name. A value of 'SP' for the first parameter would indicate that 'MYSCHEMA.MYPROC' is a specific procedure name. 'CONSERVATIVE' indicates that conservative rebinding semantics should be applied. See the REBIND command in the related links below for more details on conservative rebinding.

Related tasks:

- "Setting Up the SQL Procedures Environment" on page 23
- "Retaining Intermediate Files for SQL Procedures" on page 137
- "Customizing Precompile and Bind Options for SQL Procedures" on page 138
- "Backing Up and Restoring SQL Procedures" on page 139
- "Creating SQL Procedures" on page 133
- "Calling Stored Procedures with the CALL Statement" on page 143
- "Calling SQL Procedures with Client Applications on UNIX" on page 134
- "Calling SQL Procedures with Client Applications on Windows" on page 135
- "Distributing Compiled SQL Procedures" on page 140

Related reference:

- "REBIND" in the *Command Reference*

Chapter 6. Calling Stored Procedures

This chapter provides detailed information for calling DB2 stored procedures on the command line with the CALL statement.

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

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

Calling Stored Procedures with the CALL Statement

You can call stored procedures by using the `call` SQL Statement from the DB2 command line processor interface. The stored procedure being called must be defined in the catalog.

Procedure:

To call the stored procedure, 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.

To use the `call` statement, enter the stored procedure name plus any IN or INOUT parameters, as well as '?' as a place-holder for each OUT parameter.

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 SQL procedure 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`, and a question mark, '?', for the OUT parameter. The DEPT_MEDIAN procedure accesses the STAFF table of the sample database. The variable `deptNumber` is assigned to the DEPT column of the STAFF table, so you can obtain a valid value from the DEPT column: for example, the value "51".

Now, you can enter the `call` statement with the procedure name and the value for the IN parameter, and a question mark, '?', for the value of the OUT parameter. The procedure's parameters must be enclosed in parentheses as follows:

```
db2 call dept_median (51, ?)
```

Note: On UNIX platforms the parentheses have special meaning to the command shell, so they must be preceded with a "\" character or surrounded with quotes, as follows:

```
db2 "call dept_median (51, ?)"
```

You do not use quotes if you are using the interactive mode of the command line processor.

After running the above command, you should receive this result:

```
Value of output parameters
--------------------------
Parameter Name  : MEDIANSALARY
Parameter Value : +1.76545000000000E+004

DB20000I  The SQL command completed successfully.
```

Related tasks:

- "Calling SQL Procedures with Client Applications on UNIX" on page 134
- "Calling SQL Procedures with Client Applications on Windows" on page 135
- "Distributing Compiled SQL Procedures" on page 140

Related samples:

- "whiles.db2 -- To create the DEPT_MEDIAN SQL procedure "
- "whiles.sqc -- To call the DEPT_MEDIAN SQL procedure"

Part 3. Building and Running Platform-Specific Applications

Chapter 7. AIX

This chapter provides detailed information for building applications on AIX. For the latest DB2 application development updates for AIX, visit the Web page at:

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

Important Considerations

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

AIX Export Files for Routines

External routines are compiled on the server, and stored and executed in shared libraries on the server. These shared libraries are created when you compile the routines.

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 routines in the library. Other UNIX platforms simply export all global functions in the library. This is an example of an AIX export file:

```
#! spserver export file
outlanguage
```

The export file `spserver.exp` lists the stored procedure `outlanguage`. The linker uses `spserver.exp` to create the shared library `spserver` that contains the `outlanguage` stored procedure.

The AIX linker documentation has additional information on export files.

Related concepts:

• "AIX Routines and the CREATE Statement" on page 148

AIX Routines and the CREATE Statement

The following explains the relationship between compiling and linking your routine and the information you provide in the EXTERNAL NAME clause of the CREATE statement.

When you compile and link your program, you can identify external functions by using an export file specified with the `-bE:` option.

Suppose that the library `myrtns` contains three routines: `modify`, `remove`, and `add`. You identify `modify` as the default entry point, by putting it as the first entry in the export file that is linked in in the link step. The `remove` and `add` functions are indicated as additional exportable functions by also including them in an export file.

In the link step, you specify:

```
-bE:myrtns.exp
```

which identifies the export file `myrtns.exp`.

The export file looks like this:

```
modify
remove
add
```

Finally, your EXTERNAL NAME clauses for the routines, which are implemented with the `modify`, `remove`, and `add` functions, are coded as follows:

```
EXTERNAL NAME '/u/mydir/routines/myrtns!modify'
```

and

```
EXTERNAL NAME '/u/mydir/routines/myrtns!remove'
```

and

```
EXTERNAL NAME '/u/mydir/routines/myrtns!add'
```

Note: The default path used will be `sqllib/function`. This means if the EXTERNAL NAME clause is specified as follows:

```
EXTERNAL NAME 'myrtns!modify'
```

DB2 will attempt to load `myrtns` from `sqllib/function`.

Related concepts:

- "AIX Export Files for Routines" on page 148

Replacing an AIX Shared Library

Procedure:

After a shared library is built, it is typically copied into a directory from which DB2 will access it. When attempting to replace a routine 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.

Considerations for Installing COBOL on AIX

Because of the way AIX loads routines 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 (routine) at run time.

When a routine 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 routine load will fail, and you will receive SQLCODE -444.

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 routine directory), but this only works for one database instance; `/usr/lib` works for everyone on the machine.

Related tasks:

- "Setting Up the UNIX Application Development Environment" on page 25
- "Configuring the IBM COBOL Compiler on AIX" on page 179
- "Configuring the Micro Focus COBOL Compiler on AIX" on page 186

IBM C

Building information for DB2 CLI applications and routines is in the *CLI Guide and Reference*.

Building C Applications on AIX

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The build file, `bldapp`, 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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

To build the DB2 API non-embedded SQL sample program, `cli_info`, from the source file `cli_info.c`, enter:

```
bldapp cli_info
```

The result is an executable file, `cli_info`.

To run the executable file, enter the executable name:

```
cli_info
```

Building and Running Embedded SQL Applications

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

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

The result is an executable file, `tbmod`.

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

Related concepts:

• "Build Files" on page 94

Related tasks:

• "Building C Routines on AIX" on page 154

Related reference:

- "AIX C Application Compile and Link Options" on page 153
- "C/C++ Samples" on page 69

Related samples:

- "bldapp -- Builds AIX C application programs"
- "cli_info.c -- Set and get information at the client level (C)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "tbmod.sqc -- How to modify table data (C)"

Build Script for C Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds AIX C application programs
# 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
  EXTRA_CFLAG=-q64
else
  EXTRA_CFLAG=
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 $EXTRA_CFLAG -I$DB2PATH/include -c utilemb.c
else
  # Compile the utilapi.c error-checking utility.
  xlc $EXTRA_CFLAG -I$DB2PATH/include -c utilapi.c
fi

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

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

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

AIX C Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building C embedded SQL and DB2 API applications with the AIX IBM C compiler, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp
Compile Options:
xlc The IBM C compiler.
$EXTRA_CFLAG Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-I$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.
$EXTRA_CFLAG Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-o $1 Specify the executable program.
$1.o Specify the program object file.
utilemb.o If an embedded SQL program, include the embedded SQL utility object file for error checking.
utilapi.o If not an embedded SQL program, include the DB2 API utility object file for error checking.
-ldb2 Link to the DB2 library.
-L$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`.
Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building C Applications on AIX" on page 150

Related reference:

- "AIX C Routine Compile and Link Options" on page 158

Related samples:

- "bldapp -- Builds AIX C application programs"

Building C Routines on AIX

DB2 provides build scripts for compiling and linking C programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, contains the commands to build routines (stored procedures and user-defined functions). The script compiles the routines into a shared library that can be loaded by the database manager and called by a client application.

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

The database parameter is optional. If no database name is supplied, the program uses the default `sample` database. And since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Procedure:

The following examples show you how to build routine shared libraries with:

- stored procedures
- non-embedded SQL user-defined functions (UDFs)
- embedded SQL user-defined functions (UDFs)

Stored Procedure Shared Library

To build the sample program `spserver` from the source file `spserver.sqc`:

1. If connecting to the `sample` database, enter the build script name and program name:

   ```
   bldrtn spserver
   ```

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

   ```
   bldrtn spserver database
   ```

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

2. Next, catalog the routines by running the spcat script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling spdrop.db2, then catalogs them by calling spcreate.db2, and finally disconnects from the database. You can also call the spdrop.db2 and spcreate.db2 scripts individually.
3. Then, stop and restart the database to allow the new shared library to be recognized.

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, bldapp.

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 for the user ID.

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.

Non-embedded SQL UDF Shared Library

To build the user-defined function program, udfsrv, from the source file udfsrv.c, enter the build script name and program name:

```
bldrtn udfsrv
```

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

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 client program from the source file udfcli.c, in sqllib/samples/cli, using the script, bldapp.

You can build the embedded SQL udfcli client program from the source file udfcli.sqc, in sqllib/samples/c, using the script, bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfcli *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 for the user ID.

The client application accesses the shared library, udfsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Embedded SQL UDF Shared Library

To build the embedded SQL user-defined function program, udfemsrv, from the source file udfemsrv.sqc, if connecting to the sample database, enter the build script name and program name:

```
bldrtn udfemsrv
```

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

```
bldrtn udfemsrv database
```

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

Once you build udfemsrv, you can build the client application, udfemcli, that calls it. You can build the udfemcli client program from the source file udfemcli.sqc, in sqllib/samples/c, using the script, bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfemcli *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 for the user ID.

The client application accesses the shared library, `udfemsrv`, and executes the user-defined functions on the server database. The output is returned to the client application.

Related concepts:

- "Build Files" on page 94

Related tasks:

- "Building C Applications on AIX" on page 150

Related reference:

- "AIX C Routine Compile and Link Options" on page 158
- "C/C++ Samples" on page 69

Related samples:

- "bldrtn -- Builds AIX C routines (stored procedures and UDFs)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "spcat -- To catalog C stored procedures on UNIX"
- "spclient.sqc -- Call various stored procedures (C)"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "
- "spserver.sqc -- A variety of types of stored procedures (C)"
- "udfcli.sqc -- Call a variety of types of user-defined functions (C)"
- "udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfemsrv.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfsrv.c -- Call a variety of types of user-defined functions (C)"

Build Script for C Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds AIX C routines (stored procedures and UDFs)
# Usage: bldrtn <prog_name> [ <db_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
  EXTRA_CFLAG=-q64
else
  EXTRA_CFLAG=
fi

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

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

# Link the program using the export file $1.exp,
xlc_r $EXTRA_CFLAG -qmkshrobj -o $1 $1.o -ldb2 -L$DB2PATH/lib -bE:$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
```

AIX C Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building C routines (stored procedures and user-defined functions) with the AIX IBM C compiler, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn
Compile Options: **xlc_r** Use the multi-threaded version of the IBM C compiler, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED). **$EXTRA_CFLAG** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-I$DB2PATH/include** Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`. **-c** Perform compile only; no link. Compile and link are separate steps.
Link options: **xlc_r** Use the multi-threaded version of the compiler as a front end for the linker. **$EXTRA_CFLAG** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-qmkshrobj** Create the shared library. **-o $1** Specify the output file name. **$1.o** Specify the object file. **-ldb2** Link with the DB2 library. **-L$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`. **-bE:$1.exp** Specify an export file. The export file contains a list of the routines. Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building C Routines on AIX" on page 154

Related reference:

- "AIX C Application Compile and Link Options" on page 153

Related samples:

- "bldrtn -- Builds AIX C routines (stored procedures and UDFs)"

Building C Multi-Threaded Applications on AIX

C multi-threaded applications on AIX need to be compiled and linked with the `xlc_r` compiler instead of the `xlc` compiler or, for C++, with the `xlC_r` compiler instead of the `xlC` compiler. The `_r` versions 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 your compiler documentation.

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

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 for the user ID. 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.

Besides the `xlc_r` compiler 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`.

Procedure:

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

```
bldmt dbthrds
```

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

```
dbthrds
```

Related concepts:

- "Build Files" on page 94

Related tasks:

- "Building C Applications on AIX" on page 150

Related reference:

- "AIX C Application Compile and Link Options" on page 153
- "C/C++ Samples" on page 69

Related samples:

- "bldmt -- Builds AIX C multi-threaded applications"
- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C Multi-threaded Applications

```
#! /bin/sh
# SCRIPT: bldmt
# Builds AIX C multi-threaded applications
# 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 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

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

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

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

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

VisualAge C++

Building C++ Applications on AIX

DB2 provides build scripts for compiling and linking C++ embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The build file, bldapp 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 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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

To build the non-embedded SQL sample program cli_info from the source file cli_info.C, enter:

```
bldapp cli_info
```

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

```
cli_info
```

Building and Running Embedded SQL Applications

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

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

   ```
   bldapp tbmod
   ```

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

   ```
   bldapp tbmod database
   ```

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

   ```
   bldapp tbmod database userid password
   ```

The result is an executable file, tbmod.

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:

   ```
   tbmod
   ```

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

   ```
   tbmod database
   ```

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

   ```
   tbmod database userid password
   ```

Related concepts:

- "Build Files" on page 94

Related tasks:

- "Building C++ Routines on AIX" on page 165

Related reference:

- "AIX C++ Application Compile and Link Options" on page 164
- "C/C++ Samples" on page 69

Related samples:

- "bldapp -- Builds AIX C++ applications"
- "cli_info.C -- Set and get information at the client level (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C++ Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds AIX C++ applications
# 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
  EXTRA_CFLAG=-q64
else
  EXTRA_CFLAG=
```

```
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 $EXTRA_CFLAG -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  xlC $EXTRA_CFLAG -I$DB2PATH/include -c utilapi.C
fi

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

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

AIX C++ Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building C++ embedded SQL and DB2 API applications with the AIX IBM VisualAge C++ compiler, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp
Compile Options:
xlC The VisualAge C++ compiler.
EXTRA_CFLAG 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 bldapp
Link options: **xlC** Use the compiler as a front end for the linker. **EXTRA_CFLAG** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-o $1** Specify the executable program. **$1.o** Specify the program object file. **utilapi.o** Include the API utility object file for non-embedded SQL programs. **utilemb.o** Include the embedded SQL utility object file for embedded SQL programs. **-ldb2** Link with the DB2 library. **-L$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`. Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building C++ Applications on AIX" on page 161
- "Building C++ Embedded SQL Applications with Configuration Files" on page 175
- "Building C++ DB2 API Applications with Configuration Files" on page 174

Related reference:

- "AIX C++ Routine Compile and Link Options" on page 170

Related samples:

- "bldapp -- Builds AIX C++ applications"

Building C++ Routines on AIX

DB2 provides build scripts for compiling and linking C++ programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The script file `bldrtn` contains the commands to build routines. The script file compiles the routines into a shared library that can be loaded by the database manager and called by a client application.

The first parameter, $1, specifies the name of your source file. The second parameter, $2, specifies the name of the database to which you want to connect.

The database parameter is optional. If no database name is supplied, the program uses the default sample database. And since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Procedure:

The following examples show you how to build routine shared libraries with:

- stored procedures
- non-embedded SQL user-defined functions (UDFs)
- embedded SQL user-defined functions (UDFs)

Stored Procedure Shared Library

To build the sample program spserver from the source file spserver.sqC:

1. If connecting to the sample database, enter the build script name and program name:

   ```
   bldrtn spserver
   ```

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

   ```
   bldrtn spserver database
   ```

   The script file copies the shared library to the server in the path sqllib/function.
2. Next, catalog the routines by running the spcat script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling spdrop.db2, then catalogs them by calling spcreate.db2, and finally disconnects from the database. You can also call the spdrop.db2 and spcreate.db2 scripts individually.
3. Then, stop and restart the database to allow the new shared library to be recognized.

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.

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 for the user ID.

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.

Non-embedded SQL UDF Shared Library

To build the user-defined function program, `udfsrv`, from the source file `udfsrv.C`, enter the build script name and program name:

```
bldrtn udfsrv
```

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

If necessary, set the file mode for the UDF so the database manager can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. You can build `udfcli` from the source file `udfcli.sqC` using the script file, `bldapp`.

To call the UDFs in the shared library, run the client application by entering:

`udfcli` *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 for the user ID.

The client application accesses the shared library, udfsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Embedded SQL UDF Shared Library

To build the embedded SQL user-defined function program, udfemsrv from the source file udfemsrv.sqC, if connecting to the sample database, enter the build script name and program name:

```
bldrtn udfemsrv
```

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

```
bldrtn udfemsrv database
```

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

Once you build udfemsrv, you can build the client application, udfemcli, that calls it. You can build udfemcli from the source file udfemcli.sqC using the script file bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfemcli *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 for the user ID.

The client application accesses the shared library, udfemsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Related concepts:

- "Build Files" on page 94

Related tasks:

- "Building C++ Applications on AIX" on page 161

Related reference:

- "AIX C++ Routine Compile and Link Options" on page 170
- "C/C++ Samples" on page 69

Related samples:

- "bldrtn -- Builds AIX C++ routines (stored procedures and UDFs)"
- "spclient.sqC -- Call various stored procedures (C++)"
- "spserver.sqC -- A variety of types of stored procedures (C++)"
- "udfcli.sqC -- Call a variety of types of user-defined functions (C++)"
- "udfemcli.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "udfemsrv.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "udfsrv.C -- Call a variety of types of user-defined functions (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "spcat -- To catalog C stored procedures on UNIX"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "

Build Script for C++ Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds AIX C++ routines (stored procedures and UDFs)
# Usage: bldrtn <prog_name> [ <db_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
  EXTRA_CFLAG=-q64
else
  EXTRA_CFLAG=
fi

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

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

# Link using export file $1.exp, creating shared library $1
xlC_r $EXTRA_CFLAG -qmkshrobj -o $1 $1.o -L$DB2PATH/lib -ldb2 -bE $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
```

AIX C++ Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building C++ routines (stored procedures and user-defined functions) with the AIX VisualAge C++ compiler, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn	
Compile Options:	
xlC_r	The multi-threaded version of the IBM VisualAge C++ compiler, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).
$EXTRA_CFLAG	Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value.
-I$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 bldrtn
Link options: **xlC_r** Use the multi-threaded version of the compiler as a front-end for the linker. **$EXTRA_CFLAG** Contains "-q64" value if 'BUILD_64BIT=true' is uncommented; otherwise, it contains no value. **-qmkshrobj** Create a shared library. **-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 DB2 library. **-bE $1.exp** Specify an export file. The export file contains a list of the routines. Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building C Routines on AIX" on page 154
- "Building C++ Stored Procedures with Configuration Files" on page 176
- "Building C++ User-defined Functions with Configuration Files" on page 177

Related reference:

- "AIX C++ Application Compile and Link Options" on page 164

Related samples:

- "bldrtn -- Builds AIX C++ routines (stored procedures and UDFs)"

Building C++ Multi-Threaded Applications on AIX

C++ multi-threaded applications on AIX need to be compiled and linked with the `xlC_r` compiler instead of the `xlC` compiler or, for C, with the `xlc_r` compiler instead of the `xlc` compiler. The `_r` versions 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 your compiler documentation.

DB2 provides build scripts for compiling and linking C++ embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The script, `bldmt`, contains the commands to build multi-threaded applications.

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 for the user ID. 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.

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`.

Procedure:

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

```
bldmt dbthrds
```

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

```
dbthrds
```

Related concepts:

- "Build Files" on page 94

Related tasks:

- "Building C++ Applications on AIX" on page 161

Related reference:

- "AIX C++ Application Compile and Link Options" on page 164
- "C/C++ Samples" on page 69

Related samples:

- "bldmt -- Builds AIX C++ multi-threaded applications"

- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C++ Multi-threaded Applications

```
#! /bin/sh
# SCRIPT: bldmt
# Builds AIX C++ multi-threaded applications
# 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
  EXTRA_CFLAG=-q64
else
  EXTRA_CFLAG=
fi

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

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

VisualAge C++ Configuration Files

Note: Building information for CLI applications and routines is in the *CLI Guide and Reference*.

Building VisualAge C++ Programs with Configuration Files

VisualAge C++ Version 5.0 has both an incremental compiler and a batch mode compiler. While the batch mode compiler uses makefiles and build files, the incremental compiler uses configuration files instead. See the documentation that comes with VisualAge C++ Version 5.0 to learn more about this.

DB2 provides configuration files for the different types of DB2 programs you can build with the VisualAge C++ compiler.

Procedure:

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 topics describing how you can use the configuration files provided by DB2 to compile different types of programs:

- Building C++ Embedded SQL Applications with Configuration Files
- Building C++ DB2 API Applications with Configuration Files
- Building C++ Stored Procedures with Configuration Files
- Building C++ User-defined Functions with Configuration Files

Related tasks:

- "Building C++ Embedded SQL Applications with Configuration Files" on page 175
- "Building C++ DB2 API Applications with Configuration Files" on page 174
- "Building C++ Stored Procedures with Configuration Files" on page 176
- "Building C++ User-defined Functions with Configuration Files" on page 177
- "Building C Applications on AIX" on page 150
- "Building C Routines on AIX" on page 154
- "Building C++ Applications on AIX" on page 161
- "Building C++ Routines on AIX" on page 165

Building C++ DB2 API Applications with Configuration Files

The configuration file, `api.icc`, in `sqllib/samples/c` and in `sqllib/samples/cpp`, allows you to build DB2 API programs in C or C++ on AIX.

Procedure:

To use the configuration file to build the DB2 API sample program `cli_info` from the source file `cli_info.c`, do the following:

1. Set the API environment variable to the program name by entering:
   - For bash or Korn shell:
     ```
     export API=cli_info
     ```
   - For C shell:
     ```
     setenv API cli_info
     ```
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++.

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

```
cli_info
```

Related tasks:

- "Building C++ Embedded SQL Applications with Configuration Files" on page 175
- "Building C++ Stored Procedures with Configuration Files" on page 176
- "Building C++ User-defined Functions with Configuration Files" on page 177

Building C++ Embedded SQL Applications with Configuration Files

The configuration file, emb.icc, in sqllib/samples/c and sqllib/samples/cpp, allows you to build DB2 embedded SQL applications in C and C++ on AIX.

Procedure:

To use the configuration file to build the embedded SQL application tbmod from the source file tbmod.sqc, do the following:

1. Set the EMB environment variable to the program name by entering:
   - For bash or Korn shell:

     ```
     export EMB=tbmod
     ```

   - For C shell:

     ```
     setenv EMB tbmod
     ```

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++.

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

```
tbmod
```

Related tasks:

- "Building C++ DB2 API Applications with Configuration Files" on page 174
- "Building C++ Stored Procedures with Configuration Files" on page 176
- "Building C++ User-defined Functions with Configuration Files" on page 177

Building C++ Stored Procedures with Configuration Files

The configuration file, stp.icc, in sqllib/samples/c and sqllib/samples/cpp, allows you to build DB2 embedded SQL stored procedures in C and C++ on AIX.

Procedure:

To use the configuration file to build the embedded SQL stored procedure shared library spserver from the source file spserver.sqc, do the following:

1. Set the STP environment variable to the program name by entering:
   - For bash or Korn shell:
     ```
     export STP=spserver
     ```
   - For C shell:
     ```
     setenv 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++.

The stored procedure shared library is copied to the server in the path sqllib/function.

Next, catalog the stored procedures in the shared library by running the spcat script on the server:

```
spcat
```

This script connects to the sample database, uncatalogs the stored procedures if they were previously cataloged by calling `spdrop.db2`, then catalogs them by calling `spcreate.db2`, and finally disconnects from the database. You can also call the `spdrop.db2` and `spcreate.db2` scripts individually.

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 shared library, `spserver`, you can build the client application, `spclient`, that calls the stored procedures in it. You can build `spclient` using the configuration file, `emb.icc`.

To call the stored procedures, 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.

Related tasks:

- "Building C++ Embedded SQL Applications with Configuration Files" on page 175
- "Building C++ DB2 API Applications with Configuration Files" on page 174
- "Building C++ User-defined Functions with Configuration Files" on page 177

Building C++ User-defined Functions with Configuration Files

The configuration file, `udf.icc`, in `sqllib/samples/c` and `sqllib/samples/cpp`, allows you to build user-defined functions in C and C++ on AIX.

Procedure:

To use the configuration file to build the user-defined function program udfsrv from the source file udfsrv.c, do the following:

1. Set the UDF environment variable to the program name by entering:
   - For bash or Korn shell:
     ```
     export UDF=udfsrv
     ```
   - For C shell:
     ```
     setenv 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++.

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, by using the configuration file cli.icc.

You can build the embedded SQL udfcli program from the source file udfcli.sqc, in sqllib/samples/c, by using the configuration file emb.icc.

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.

Related tasks:

- "Building C++ Embedded SQL Applications with Configuration Files" on page 175
- "Building C++ DB2 API Applications with Configuration Files" on page 174

- "Building C++ Stored Procedures with Configuration Files" on page 176

IBM COBOL Set for AIX

Configuring the IBM COBOL Compiler on AIX

The following are steps you need to take if you develop applications that contain embedded SQL and DB2 API calls, and you are using the IBM COBOL Set for AIX compiler.

Procedure:

- 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".
  ```

Related concepts:

- "Considerations for Installing COBOL on AIX" on page 149

Related tasks:

- "Setting Up the UNIX Application Development Environment" on page 25
- "Building IBM COBOL Applications on AIX" on page 180

• "Building IBM COBOL Routines on AIX" on page 183

Building IBM COBOL Applications on AIX

DB2 provides build scripts for compiling and linking COBOL embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cobol` directory, along with sample programs that can be built with these files.

The build file, `bldapp` 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 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.

Procedure:

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
   ```

Related concepts:

- "Build Files" on page 94

Related tasks:

- "Building IBM COBOL Routines on AIX" on page 183

Related reference:

- "AIX IBM COBOL Application Compile and Link Options" on page 182
- "COBOL Samples" on page 80

Related samples:

- "bldapp -- Builds AIX COBOL applications"
- "client.cbl -- How to set and query a client (IBM COBOL)"
- "embprep -- To prep and bind a COBOL embedded SQL sample on AIX"
- "updat.sqb -- How to update, delete and insert table data (IBM COBOL)"

Build Script for IBM COBOL Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds AIX COBOL applications
# 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 $1.o checkerr.o -L$DB2PATH/lib -ldb2
```

AIX IBM COBOL Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL embedded SQL and DB2 API applications with the AIX IBM COBOL Set compiler, as demonstrated in the `bldapp` build script.

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. **$1.o** Specify the program object file. **checkerr.o** Include the utility object file for error-checking. **-L$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.

Related tasks:

- "Building IBM COBOL Applications on AIX" on page 180

Related reference:

- "AIX IBM COBOL Routine Compile and Link Options" on page 185

Related samples:

- "bldapp -- Builds AIX COBOL applications"

Building IBM COBOL Routines on AIX

DB2 provides build scripts for compiling and linking COBOL embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cobol` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, in `sqllib/samples/cobol`, contains the commands to build routines (stored procedures). The script compiles the routines 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 shared library 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. The script uses the source file name, `$1`, for the shared library name. Database name is optional. If no database name is supplied, the program uses the default `sample` database.

Procedure:

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

```
bldrtn outsrv
```

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

```
bldrtn outsrv database
```

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

Once you build the routine shared library, `outsrv`, you can build the client application, `outcli`, that calls the routine within the library. You can build `outcli` using the script file `bldapp`.

To call the routine, 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 for the user ID.

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

Related concepts:

- "Build Files" on page 94

Related tasks:

- "Building IBM COBOL Applications on AIX" on page 180

Related reference:

- "AIX IBM COBOL Routine Compile and Link Options" on page 185
- "COBOL Samples" on page 80

Related samples:

- "bldrtn -- Builds AIX COBOL routines (stored procedures)"
- "embprep -- To prep and bind a COBOL embedded SQL sample on AIX"
- "outcli.sqb -- Call stored procedures using the SQLDA structure (IBM COBOL)"
- "outsrv.sqb -- Demonstrates stored procedures using the SQLDA structure (IBM COBOL)"

Build Script for IBM COBOL Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds AIX COBOL routines (stored procedures)
# Usage:  bldrtn <program_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 creating shared library $1 with export file $1.exp
cob2 -o $1 $1.o checkerr.o -bnoentry -bE:$1.exp \
     -L$DB2PATH/lib -ldb2

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

AIX IBM COBOL Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL routines (stored procedures) with the IBM COBOL Set compiler on AIX, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn
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`.

Compile and Link Options for bldrtn	
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.
-bnoentry	Do not 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.	

Related tasks:

- "Building IBM COBOL Routines on AIX" on page 183

Related reference:

- "AIX IBM COBOL Application Compile and Link Options" on page 182

Related samples:

- "bldrtn -- Builds AIX COBOL routines (stored procedures)"

Micro Focus COBOL

Configuring the Micro Focus COBOL Compiler on AIX

Do the following if you develop applications that contain embedded SQL and DB2 API calls with the Micro Focus COBOL compiler.

Procedure:

- When you precompile your application using the command line processor command `db2 prep`, use the `target mfcob` option.
- 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:

– On bash or Korn shell:

```
export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
```

– On C shell:

```
setenv COBCPY $COBCPY:$HOME/sqllib/include/cobol_mf
```

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

Related concepts:

- "Considerations for Installing COBOL on AIX" on page 149

Related tasks:

- "Setting Up the UNIX Application Development Environment" on page 25
- "Building Micro Focus COBOL Applications on AIX" on page 187
- "Building Micro Focus COBOL Routines on AIX" on page 190

Building Micro Focus COBOL Applications on AIX

DB2 provides build scripts for compiling and linking Micro Focus COBOL embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cobol_mf` directory, along with sample programs that can be built with these files.

The build file, `bldapp` 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 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.

Procedure:

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
   ```

Related concepts:

- "Build Files" on page 94

Related tasks:

- "Building Micro Focus COBOL Routines on AIX" on page 190

Related reference:

- "AIX Micro Focus COBOL Application Compile and Link Options" on page 189
- "COBOL Samples" on page 80

Related samples:

- "bldapp -- Builds AIX Micro Focus COBOL applications"
- "client.cbl -- How to set and query a client (MF COBOL)"
- "updat.sqb -- How to update, delete and insert table data (MF COBOL)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for Micro Focus COBOL Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds AIX Micro Focus COBOL applications
# 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.
COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

# 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 $1.o checkerr.o -L$DB2PATH/lib -ldb2 -ldb2gmf
```

AIX Micro Focus COBOL Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL embedded SQL and DB2 API applications with the Micro Focus COBOL compiler on AIX, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp	
Compile Options:	
cob	The COBOL compiler.
-c	Perform compile only; no link.
-x	When used with `-c`, produces an object file.
Link Options:	
cob	Use the compiler as a front end for the linker.
-x	Produces an executable program.
-o $1	Specify the executable program.
$1.o	Specify the program object file.
-L$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 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.	

Related tasks:

- "Building Micro Focus COBOL Applications on AIX" on page 187

Related reference:

- "AIX Micro Focus COBOL Routine Compile and Link Options" on page 192

Related samples:

- "bldapp -- Builds AIX Micro Focus COBOL applications"

Building Micro Focus COBOL Routines on AIX

DB2 provides build scripts for compiling and linking Micro Focus COBOL embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cobol_mf` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, contains the commands to build routines (stored procedures). The script compiles the routine source file into a shared library that can be called by a client application.

The first parameter, $1, specifies the name of your source file. The script uses the source file name for the shared library name. The second parameter, $2, specifies the name of the database to which you want to connect. Since the shared library must be built in 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.

Procedure:

To build the sample program `outsrv` from the source file `outsrv.sqb`, if connecting to the sample database, enter:

```
bldrtn outsrv
```

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

```
bldrtn outsrv database
```

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

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`.

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 for the user ID.

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.

Related concepts:

- "Build Files" on page 94

Related tasks:

- "Building Micro Focus COBOL Applications on AIX" on page 187

Related reference:

- "AIX Micro Focus COBOL Routine Compile and Link Options" on page 192
- "COBOL Samples" on page 80

Related samples:

- "bldrtn -- Builds AIX Micro Focus COBOL routines (stored procedures)"
- "outcli.sqb -- Call stored procedures using the SQLDA structure (MF COBOL)"
- "outsrv.sqb -- Demonstrates stored procedures using the SQLDA structure (MF COBOL)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for Micro Focus COBOL Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds AIX Micro Focus COBOL routines (stored procedures)
# Usage: bldrtn <program_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.
COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

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

# Link the program.
cob -x -o $1 $1.o -Q -bnoentry \
  -Q -bI:$DB2PATH/lib/db2g.imp -L$DB2PATH/lib -ldb2 -ldb2gmf

# 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
```

AIX Micro Focus COBOL Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL routines (stored procedures) with the Micro Focus COBOL compiler on AIX, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn	
Compile Options:	
cob	The COBOL compiler.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.
-x	Compile to an object module when used with the `-c` option.
Link Options:	
cob	Use the compiler as a front-end for the linker.
-x	Produce a shared library.
-o $1	Specify the executable program.
$1.o	Specify the program object file.
-Q -bnoentry	Do not specify the default entry point to the shared library.
-Q -bI:$DB2PATH/lib/db2g.imp	Provides a list of entry points to the DB2 application library.
-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.	

Related tasks:

- "Building Micro Focus COBOL Routines on AIX" on page 190

Related reference:

- "AIX Micro Focus COBOL Application Compile and Link Options" on page 189

Related samples:

- "bldrtn -- Builds AIX Micro Focus COBOL routines (stored procedures)"

Building REXX Applications on AIX

The following shows you how to build REXX applications on AIX. DB2 for AIX supports classic REXX as well as 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.

Procedure:

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

For bash or Korn shell, enter:

```
export LIBPATH=$LIBPATH:/lib:/usr/lib:/usr/opt/db2_08_01/lib
```

For C shell, enter:

```
setenv LIBPATH $LIBPATH:/lib:/usr/lib:/usr/opt/db2_08_01/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 (including an extension, if one exists).
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`, enter:

```
updat.cmd
```

Related tasks:

- "Setting Up the UNIX Application Development Environment" on page 25

Related reference:

- "REXX Samples" on page 91

Chapter 8. HP-UX

This chapter provides detailed information for building DB2 applications on HP-UX. 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`

HP-UX C

Building information for DB2 CLI Applications and routines is in the *CLI Guide and Reference*.

Building C Applications on HP-UX

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The build script, `bldapp`, 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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

To build the DB2 API non-embedded SQL sample program, cli_info, from the source file cli_info.c, enter:

```
bldapp cli_info
```

The result is an executable file, cli_info.

To run the executable file, enter the executable name:

```
cli_info
```

Building and Running Embedded SQL Applications

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

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

   ```
   bldapp tbmod
   ```

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

   ```
   bldapp tbmod database
   ```

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

   ```
   bldapp tbmod database userid password
   ```

The result is an executable file, tbmod.

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:

```
tbmod
```

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

   ```
   tbmod database
   ```

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

   ```
   tbmod database userid password
   ```

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "HP-UX C Application Compile and Link Options" on page 200

Related samples:

- "bldapp -- Builds HP-UX C applications"
- "cli_info.c -- Set and get information at the client level (C)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "tbmod.sqc -- How to modify table data (C)"

Build Script for C Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds HP-UX C applications
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

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

if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG="+DA2.0W"
else
  EXTRA_CFLAG="+DAportable"
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  $EXTRA_CFLAG  -Ae -I$DB2PATH/include -c utilemb.c
```

```
else
  # Compile the utilapi.c error-checking utility.
  cc  $EXTRA_CFLAG  -Ae -I$DB2PATH/include -c utilapi.c
fi

# Compile the program.
cc  $EXTRA_CFLAG  -Ae -I$DB2PATH/include -c $1.c

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

HP-UX C Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building C embedded SQL and DB2 API applications with the HP-UX C compiler, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp
Compile Options: **cc** The C compiler. **$EXTRA_CFLAG** Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64-bit, the value is `+DA2.0W`. **+DAportable (32-bit only)** Generates code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information. **+DA2.0W (64-bit only)** Must be used to generate 64-bit code. **-Ae** Enables HP ANSI extended mode. **-I$DB2PATH/include** Specifies the location of the DB2 include files. **-c** Perform compile only; no link. Compile and link are separate steps.

Compile and Link Options for bldapp
Link Options: **cc** Use the compiler as a front end to the linker. **$EXTRA_CFLAG** Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64-bit, the value is `+DA2.0W`. **+DAportable (32-bit only)** Use code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information. **+DA2.0W (64-bit only)** Must be used to generate 64-bit code. **-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.

Related tasks:

- "Building C Applications on HP-UX" on page 197

Related samples:

- "bldapp -- Builds HP-UX C applications"

Building C Routines on HP-UX

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The script file `bldrtn` contains the commands to build routines (stored procedures and user-defined functions). The script file compiles the routines into a shared library that can be loaded by the database manager and called by a client application.

The first parameter, $1, specifies the name of your source file. The second parameter, $2, specifies the name of the database to which you want to connect.

The database parameter is optional. If no database name is supplied, the program uses the default sample database. And since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Procedure:

The following examples show you how to build routine shared libraries with:

- stored procedures
- non-embedded SQL user-defined functions (UDFs)
- embedded SQL user-defined functions (UDFs)

Stored Procedure Shared Library

To build the sample program spserver from the source file spserver.sqc:

1. If connecting to the sample database, enter the build file name and the program name:

   ```
   bldrtn spserver
   ```

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

   ```
   bldrtn spserver database
   ```

   The script file copies the shared library to the server in the path sqllib/function.
2. Next, catalog the routines by running the spcat script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling spdrop.db2, then catalogs them by calling spcreate.db2, and finally disconnects from the database. You can also call the spdrop.db2 and spcreate.db2 scripts individually.
3. Then, stop and restart the database to allow the new shared library to be recognized.

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.

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.

Non-embedded SQL UDF Shared Library

To build the user-defined function program, `udfsrv`, from the source file `udfsrv.c`, enter the build script name and the program name:

```
bldrtn udfsrv
```

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

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` client program from the source file `udfcli.c`, in `sqllib/samples/cli`, using the script file `bldapp`.

You can build the embedded SQL `udfcli` client program from the source file `udfcli.sqc`, in `sqllib/samples/c`, using the script file `bldapp`.

To call the UDFs in the shared library, run the client application by entering:

`udfcli` *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, udfsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Embedded SQL UDF Shared Library

To build the embedded SQL user-defined function program, udfemsrv, from the source file udfemsrv.sqc, if connecting to the sample database, enter the build script name and program name:

```
bldrtn udfemsrv
```

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

```
bldrtn udfemsrv database
```

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

Once you build udfemsrv, you can build the client application, udfemcli, that calls it. You can build the udfemcli client program from the source file udfemcli.sqc, in sqllib/samples/c, using the script file bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfemcli *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, udfemsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69

- "HP-UX C Routine Compile and Link Options" on page 206

Related samples:

- "bldrtn -- Builds HP-UX C routines (stored procedures and UDFs)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "spcat -- To catalog C stored procedures on UNIX"
- "spclient.sqc -- Call various stored procedures (C)"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "
- "spserver.sqc -- A variety of types of stored procedures (C)"
- "udfcli.sqc -- Call a variety of types of user-defined functions (C)"
- "udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfemsrv.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfsrv.c -- Call a variety of types of user-defined functions (C)"

Build Script for C Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds HP-UX C routines (stored procedures and UDFs)
# Usage: bldrtn <prog_name> [ <db_name> ]

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

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

if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG="+DA2.0W"
else
  EXTRA_CFLAG="+DAportable"
fi

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

# Compile the program.
cc $EXTRA_CFLAG  +u1 +z -Ae -I$DB2PATH/include \
```

```
  -D_POSIX_C_SOURCE=199506L -c $1.c

# Link the program to create a shared library
ld -b -o $1 $1.o -L$DB2PATH/lib -ldb2 -lpthread

# 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
```

HP-UX C Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building C routines (stored procedures and user-defined functions) with the HP-UX C compiler, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn
Compile Options:
cc The C compiler.
$EXTRA_CFLAG Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64-bit, the value is `+DA2.0W`.
+DAportable (32-bit only) Generates code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information.
+DA2.0W (64-bit only) Must be used to generate 64-bit code.
+u1 Allow unaligned data access. Use only if your application uses unaligned data.
+z Generate position-independent code.
-Ae Enables HP ANSI extended mode.
-I$DB2PATH/include Specify the location of the DB2 include files. For example: `-I$DB2PATH/include`.
-D_POSIX_C_SOURCE=199506L POSIX thread library option that ensures _REENTRANT is defined, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).
-c Perform compile only; no link. This book assumes that compile and link are separate steps.

Compile and Link Options for bldrtn
Link Options: **ld** Use the linker to link. **-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. **-lpthread** Link with the POSIX thread library. Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building C Routines on HP-UX" on page 201

Related samples:

- "bldrtn -- Builds HP-UX C routines (stored procedures and UDFs)"

Building C Multi-Threaded Applications on HP-UX

HP-UX provides a POSIX thread library and a DCE thread library. Only 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`.

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The script file, `bldmt` contains the commands to build multi-threaded applications. It takes up to four parameters, represented by the variables `$1`, `$2`, `$3`, and `$4`.

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.

Procedure:

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

```
bldmt dbthrds
```

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

```
dbthrds
```

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69

Related samples:

- "bldmt -- Builds HP-UX C multi-threaded applications"
- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C Multi-threaded Applications

```
#! /bin/sh
# SCRIPT: bldmt
# Builds HP-UX C multi-threaded applications
# Usage: bldmt <prog_name> [ <db_name> [ <userid> <password> ]]

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

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

if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG="+DA2.0W"
else
  EXTRA_CFLAG="+DAportable"
fi

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

```
fi

# Compile the program.
cc $EXTRA_CFLAG  -Ae -I$DB2PATH/include -D_POSIX_C_SOURCE=199506L -c $1.c

# Link the program
cc $EXTRA_CFLAG  -o $1 $1.o -L$DB2PATH/lib -ldb2 -lpthread
```

HP-UX C++

Building C++ Applications on HP-UX

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The build script, `bldapp`, contains the commands to build DB2 API and embedded SQL applications. The script takes up to four parameters.

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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

To build the non-embedded SQL sample program `cli_info` from the source file `cli_info.C`, enter:

```
bldapp cli_info
```

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

```
cli_info
```

Building and Running Embedded SQL Applications

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

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

The result is an executable file, `tbmod`.

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

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "HP-UX C++ Application Compile and Link Options" on page 211

Related samples:

- "bldapp -- Builds HP-UX C++ applications"
- "cli_info.C -- Set and get information at the client level (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C++ Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds HP-UX C++ applications
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

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

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

if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG="+DA2.0W"
else
  EXTRA_CFLAG="+DAportable"
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.
  aCC $EXTRA_CFLAG -ext -I$DB2PATH/include -c utilemb.C
else
  # Compile the utilapi.C error-checking utility.
  aCC $EXTRA_CFLAG -ext -I$DB2PATH/include -c utilapi.C
fi

# Compile the program.
aCC $EXTRA_CFLAG -ext -I$DB2PATH/include -c $1.C

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

HP-UX C++ Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building C++ embedded SQL and DB2 API applications with the HP-UX C++ compiler, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp
Compile Options: **aCC** The HP aC++ compiler. **$EXTRA_CFLAG** Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64-bit, the value is `+DA2.0W`. **+DAportable (32-bit only)** Generates code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information. **+DA2.0W (64-bit only)** Must be used to generate 64-bit code. **-ext** Allow various C++ extensions including ″long long″ support. **-I$DB2PATH/include** Specifies 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: **aCC** Use the HP aC++ compiler as a front end for the linker. **$EXTRA_CFLAG** Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64-bit, the value is `+DA2.0W`. **+DAportable (32-bit only)** Use code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information. **+DA2.0W (64-bit only)** Must be used to generate 64-bit code. **-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.

Related tasks:

- "Building C++ Applications on HP-UX" on page 209

Related samples:

- "bldapp -- Builds HP-UX C++ applications"

Building C++ Routines on HP-UX

DB2 provides build scripts for compiling and linking C++ embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, contains the commands to build routines (stored procedures or user-defined functions). The script compiles the routines into a shared library that can be called by a client application.

The first parameter, $1, specifies the name of your source file. The second parameter, $2, specifies the name of the database to which you want to connect.

The database parameter is optional. If no database name is supplied, the program uses the default sample database. And since the shared library must be built on the same instance where the database resides, there are no parameters for user ID and password.

Procedure:

The following examples show you how to build routine shared libraries with:

- stored procedures
- non-embedded SQL user-defined functions (UDFs)
- embedded SQL user-defined functions (UDFs)

Stored Procedure Shared Library

To build the sample program spserver from the source file spserver.sqC:

1. If connecting to the sample database, enter the build file name and program name:

   ```
   bldrtn spserver
   ```

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

   ```
   bldrtn spserver database
   ```

   The script copies the stored procedure to the server in the path sqllib/function.
2. Next, catalog the routines by running the spcat script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling spdrop.db2, then catalogs them by calling spcreate.db2, and finally disconnects from the database. You can also call the spdrop.db2 and spcreate.db2 scripts individually.
3. Then, stop and restart the database to allow the new shared library to be recognized.

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.

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 for the user ID.

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.

Non-embedded SQL UDF Shared Library

To build the user-defined function program, `udfsrv`, from the source file `udfsrv.C`, enter the build script name and program name:

```
bldrtn udfsrv
```

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

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. You can build `udfcli` from the source file `udfcli.sqC` using the script, `bldapp`.

To call the UDFs in the shared library, run the client application by entering:

`udfcli` *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 for the user ID.

The client application accesses the shared library, `udfsrv`, and executes the user-defined functions on the server database. The output is returned to the client application.

Embedded SQL UDF Shared Library

To build the embedded SQL user-defined function program, udfemsrv, from the source file udfemsrv.sqC, if connecting to the sample database, enter the build script name and program name:

```
bldrtn udfemsrv
```

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

```
bldrtn udfemsrv database
```

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

Once you build udfemsrv, you can build the client application, udfemcli, that calls it. You can build udfemcli from the source file, udfemcli.sqC, using the script, bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfemcli *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 for the user ID.

The client application accesses the shared library, udfemsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "HP-UX C++ Routine Compile and Link Options" on page 218

Related samples:

- "bldrtn -- Builds HP-UX C++ routines (stored procedures and UDFs)"
- "spclient.sqC -- Call various stored procedures (C++)"
- "spserver.sqC -- A variety of types of stored procedures (C++)"
- "udfcli.sqC -- Call a variety of types of user-defined functions (C++)"

- "udfemcli.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "udfemsrv.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "udfsrv.C -- Call a variety of types of user-defined functions (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "spcat -- To catalog C stored procedures on UNIX"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "

Build Script for C++ Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds HP-UX C++ routines (stored procedures and UDFs)
# Usage: bldsrv <prog_name> [ <db_name> ]

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

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

if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG="+DA2.0W"
else
  EXTRA_CFLAG="+DAportable"
fi

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

# Compile the program. First ensure it is coded with extern "C".
aCC  $EXTRA_CFLAG +u1 +z -ext -mt -I$DB2PATH/include -c $1.C

# Link the program to create a shared library.
aCC  $EXTRA_CFLAG -mt -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
```

HP-UX C++ Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building C++ routines (stored procedures and user-defined functions) with the HP-UX C++ compiler, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn
Compile Options: **aCC** The HP aC++ compiler. **$EXTRA_CFLAG** Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64-bit, the value is `+DA2.0W`. **+DAportable (32-bit only)** Generates code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information. **+DA2.0W (64-bit only)** Must be used to generate 64-bit code. **+u1** Allows unaligned data access. **+z** Generate position-independent code. **-ext** Allow various C++ extensions including ″long long″ support. **-mt** Allows threads support for the HP aC++ compiler, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED). **-I$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 bldrtn
Link Options: **aCC** Use the HP aC++ compiler as a front end for the linker. **$EXTRA_CFLAG** Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64-bit, the value is `+DA2.0W`. **+DAportable (32-bit only)** Use code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information. **+DA2.0W (64-bit only)** Must be used to generate 64-bit code. **-mt** Allows threads support for the HP aC++ compiler, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED). **-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.

Related tasks:

- "Building C++ Routines on HP-UX" on page 213

Related samples:

- "bldrtn -- Builds HP-UX C++ routines (stored procedures and UDFs)"

Building C++ Multi-Threaded Applications on HP-UX

HP-UX provides a POSIX thread library and a DCE thread library. Only multi-threaded applications using the POSIX thread library are supported by DB2 on HP-UX.

For the HP-UX C++ compiler, `-mt` must be used for multi-threaded applications in both the compile and link steps.

DB2 provides build scripts for compiling and linking C++ embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The script, bldmt, contains the commands to build multi-threaded applications. It takes up to four parameters, represented by the variables $1, $2, $3, and $4.

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.

Procedure:

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

```
bldmt dbthrds
```

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

```
dbthrds
```

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69

Related samples:

- "bldmt -- Builds HP-UX C++ multi-threaded applications"
- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C++ Multi-threaded Applications

```
#! /bin/sh
# SCRIPT: bldmt
# Builds HP-UX C++ multi-threaded applications
# 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 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG="+DA2.0W"
```

```
else
  EXTRA_CFLAG="+DAportable"
fi

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

# Compile the program.
aCC $EXTRA_CFLAG -ext -mt -I$DB2PATH/include -c $1.C

# Link the program
aCC $EXTRA_CFLAG -mt -o $1 $1.o -L$DB2PATH/lib -ldb2
```

Micro Focus COBOL

Configuring the Micro Focus COBOL Compiler on HP-UX

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

Procedure:

- When you precompile your application using the command line processor command `db2 prep`, use the `target mfcob` option.
- 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.

  You have to run `mkrts` and `mkcheck` to set up your environment before building your COBOL applications. 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/IBM/db2/V8.1/lib/db2mkrts.args; export COBOPT
     ksh mkrts
     mv $COBDIR/rts32 $COBDIR/rts32.orig
     cp rts32 $COBDIR/rts32
     ```

     where `$COBDIR` is the directory where the COBOL compiler is installed, normally `/opt/cobol/cobdir`.
  3. You must also rebuild the `check` executable which Hewlett-Packard ships with the product. If you do not rebuild the `check` executable located in your `$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/IBM/db2/V8.1/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/IBM/db2/V8.1/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,

  – on bash or Korn shell, enter:

  ```
  export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
  ```

  – on C shell, enter:

  ```
  setenv COBCPY ${COBCPY}:${HOME}/sqllib/include/cobol_mf
  ```

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

Related tasks:

- "Building Micro Focus COBOL Routines on HP-UX" on page 226
- "Building Micro Focus COBOL Applications on HP-UX" on page 222

Building Micro Focus COBOL Applications on HP-UX

DB2 provides build scripts for compiling and linking Micro Focus COBOL embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cobol_mf` directory, along with sample programs that can be built with these files.

The build script, `bldapp`, 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.

Procedure:

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*

Related concepts:

- "Build Files" on page 94

Related reference:

- "HP-UX Micro Focus COBOL Application Compile and Link Options" on page 225
- "COBOL Samples" on page 80

Related samples:

- "bldapp -- Builds HP-UX Micro Focus COBOL applications"
- "client.cbl -- How to set and query a client (MF COBOL)"
- "updat.sqb -- How to update, delete and insert table data (MF COBOL)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for Micro Focus COBOL Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds HP-UX Micro Focus COBOL applications
# 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.
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
```

HP-UX Micro Focus COBOL Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL embedded SQL and DB2 API applications with the Micro Focus COBOL compiler on HP-UX, as demonstrated in the `bldapp` build script.

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 and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information. **-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 and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information. **-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. **-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.

Related tasks:

- "Building Micro Focus COBOL Applications on HP-UX" on page 222

Related samples:

- "bldapp -- Builds HP-UX Micro Focus COBOL applications"

Building Micro Focus COBOL Routines on HP-UX

DB2 provides build scripts for compiling and linking Micro Focus COBOL embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cobol_mf` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, contains the commands to build routines (stored procedures). The script compiles the routines into a shared library on the server that can be called by a client application. It takes one or two parameters, represented inside `bldrtn` by the variables `$1` and `$2`.

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

Procedure:

To build the sample program `outsrv` from the source file `outsrv.sqb`, if connecting to the sample database, enter:

```
bldrtn outsrv
```

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

```
bldrtn outsrv database
```

The script copies the shared library to the `sqllib/function` directory.

Once you build the shared library, `outsrv`, you can build the client application `outcli` that calls the routine within it. You can build `outcli` using the script file, `bldapp`.

To call the routine, 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 alias, or another name.

userid Is a valid user ID.

password
Is a valid password for the user ID.

The client application accesses the shared library, `outsrv`, which executes the routine of the same name on the server database, and then returns the output to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "HP-UX Micro Focus COBOL Routine Compile and Link Options" on page 228
- "COBOL Samples" on page 80

Related samples:

- "bldrtn -- Builds HP-UX Micro Focus COBOL routines (stored procedures)"
- "outcli.sqb -- Call stored procedures using the SQLDA structure (MF COBOL)"
- "outsrv.sqb -- Demonstrates stored procedures using the SQLDA structure (MF COBOL)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for Micro Focus COBOL Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds HP-UX Micro Focus COBOL routines (stored procedures)
# Usage: bldrtn <prog_name> [ <db_name> ]

# 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
fi

# Set COBCPY to include the DB2 COPY files directory.
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
```

HP-UX Micro Focus COBOL Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL routines (stored procedures) with the Micro Focus COBOL compiler on HP-UX, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn
Compile Options: **cob** The COBOL compiler. **+DAportable** Generates code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information. **+z** Generate position-independent code. **-cx** Compile to object module.
Link Options: **ld** Use the linker to link. **-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. **-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.

Related tasks:

- "Building Micro Focus COBOL Routines on HP-UX" on page 226

Related samples:

- "bldrtn -- Builds HP-UX Micro Focus COBOL routines (stored procedures)"

Chapter 9. Linux

This chapter provides detailed information for building applications on Linux. 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

Building information for DB2 CLI Applications and routines is in the *CLI Guide and Reference*.

Building C Applications on Linux

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The build file, `bldapp` 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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

To build the DB2 API non-embedded SQL sample program, cli_info, from the source file cli_info.c, enter:

```
bldapp cli_info
```

The result is an executable file, cli_info.

To run the executable file, enter the executable name:

```
cli_info
```

Building and Running Embedded SQL Applications

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

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

   ```
   bldapp tbmod
   ```

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

   ```
   bldapp tbmod database
   ```

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

   ```
   bldapp tbmod database userid password
   ```

The result is an executable file, tbmod.

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:

   ```
   tbmod
   ```

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

   ```
   tbmod database
   ```

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

   ```
   tbmod database userid password
   ```

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "Linux C Application Compile and Link Options" on page 234

Related samples:

- "bldapp -- Builds Linux C applications"
- "cli_info.c -- Set and get information at the client level (C)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "tbmod.sqc -- How to modify table data (C)"

Build Script for C Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds Linux C applications
# 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 specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-Wl,-rpath$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

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

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

if [ -f $1".sqc" ]
```

```
then
  # Link the program with utilemb.o.
  gcc -o $1 $1.o utilemb.o $EXTRA_LFLAG \
    -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o.
  gcc -o $1 $1.o utilapi.o $EXTRA_LFLAG \
    -L$DB2PATH/lib -ldb2
fi
```

Linux C Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building C embedded SQL and DB2 API applications with the Linux C compiler, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp	
Compile Options:	
gcc	The GNU/Linux C compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files.
-c	Perform compile only; no link. This script file has separate compile and link steps.

Compile and Link Options for bldapp	
Link options:	
gcc	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.
$EXTRA_LFLAG	If 'RUNTIME=true' is uncommented, contains value "-Wl,-rpath,$DB2PATH/lib"; otherwise, contains no value.
-Wl,-rpath,$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-L$DB2PATH/lib	Specify the location of the DB2 shared libraries at link-time. For example: `$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.	

Related tasks:

- "Building C Applications on Linux" on page 231

Related samples:

- "bldapp -- Builds Linux C applications"

Building C Routines on Linux

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, contains the commands to build routines (stored procedures or user-defined functions). The script file compiles the routines into a shared library that can be called by a client application.

The first parameter, $1, specifies the name of your source file. The second parameter, $2, specifies the name of the database to which you want to connect.

The database parameter is optional. If no database name is supplied, the program uses the default sample database. And since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Procedure:

The following examples show you how to build routine shared libraries with:

- stored procedures
- non-embedded SQL user-defined functions (UDFs)
- embedded SQL user-defined functions (UDFs)

Stored Procedure Shared Library

To build the sample program spserver from the source file spserver.sqc:

1. If connecting to the sample database, enter the build script name and program name:

   ```
   bldrtn spserver
   ```

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

   ```
   bldrtn spserver database
   ```

   The script copies the stored procedure to the server in the path sqllib/function.
2. Next, catalog the routines by running the spcat script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling spdrop.db2, then catalogs them by calling spcreate.db2, and finally disconnects from the database. You can also call the spdrop.db2 and spcreate.db2 scripts individually.
3. Then, stop and restart the database to allow the new shared library to be recognized.

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, bldapp.

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 for the user ID.

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.

Non-embedded SQL UDF Shared Library

To build the user-defined function program, `udfsrv`, from the source file `udfsrv.c`, enter the build script name and program name:

```
bldrtn udfsrv
```

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

If necessary, set the file mode for the UDF so the database manager can access it.

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. You can build `udfcli` from the source file `udfcli.sqc` using the script file `bldapp`.

To call the UDFs in the shared library, run the client application by entering:

`udfcli` *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 for the user ID.

The client application accesses the shared library, udfsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Embedded SQL UDF Shared Library

To build the embedded SQL user-defined function program, udfemsrv, from the source file udfemsrv.sqc, if connecting to the sample database, enter the build script name and program name:

```
bldrtn udfemsrv
```

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

```
bldrtn udfemsrv database
```

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

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

Once you build udfemsrv, you can build the client application, udfemcli, that calls it. You can build the udfemcli client program from the source file udfemcli.sqc, in sqllib/samples/c, using the script file bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfemcli *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, udfemsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69

- "Linux C Routine Compile and Link Options" on page 240

Related samples:

- "bldrtn -- Builds Linux C routines (stored procedures or UDFs)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "spcat -- To catalog C stored procedures on UNIX"
- "spclient.sqc -- Call various stored procedures (C)"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "
- "spserver.sqc -- A variety of types of stored procedures (C)"
- "udfcli.sqc -- Call a variety of types of user-defined functions (C)"
- "udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfemsrv.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfsrv.c -- Call a variety of types of user-defined functions (C)"

Build Script for C Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds Linux C routines (stored procedures or UDFs)
# Usage: bldrtn <prog_name> [ <db_name> ]

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

# Set the runtime path since routines run as setuid.
EXTRA_LFLAG="-Wl,-rpath$DB2PATH/lib"

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

# Compile the program.
gcc -fpic -I$DB2PATH/include -c $1.c -D_REENTRANT

# Link the program and create a shared library
gcc -shared -o $1 $1.o $EXTRA_LFLAG -L$DB2PATH/lib -ldb2 -lpthread
```

```
# 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
```

Linux C Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building C routines (stored procedures and user-defined functions) with the Linux C compiler, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn
Compile Options: **gcc** The GNU/Linux C compiler. **-fpic** Generate position independent code. **-I$DB2PATH/include** Specify the location of the DB2 include files. **-c** Perform compile only; no link. This script file has separate compile and link steps. **-D_REENTRANT** Defines _REENTRANT, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).
Link Options: **gcc** Use the compiler as a front end for the linker. **-shared** Generate a shared library. **-o $1** Specify the executable. **$1.o** Include the program object file. **$EXTRA_LFLAG** Contains the value "-Wl,-rpath,$DB2PATH/lib" to set the runtime path since routines run as setuid. **-Wl,-rpath,$DB2PATH/lib** Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`. **-L$DB2PATH/lib** Specify the location of the DB2 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. **-lpthread** Link with the POSIX thread library. Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building C Routines on Linux" on page 235

Related samples:

- "bldrtn -- Builds Linux C routines (stored procedures or UDFs)"

Building C Multi-Threaded Applications on Linux

Multi-threaded applications using Linux C need to be compiled with `-D_REENTRANT` and linked with `-lpthread`.

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The script, `bldmt`, contains the commands to build multi-threaded applications.

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.

Procedure:

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

```
bldmt dbthrds
```

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

```
dbthrds
```

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69

Related samples:

- "bldmt -- Builds Linux C multi-threaded applications"
- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"

- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C Multi-threaded Applications

```
#! /bin/sh
# SCRIPT: bldmt
# Builds Linux C multi-threaded applications
# 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 specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-Wl,-rpath$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

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

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

# Link the program.
gcc -o $1 $1.o $EXTRA_LFLAG -L$DB2PATH/lib -ldb2 -lpthread
```

Linux C++

Building C++ Applications on Linux

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The build script, `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 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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

To build the non-embedded SQL sample program cli_info from the source file cli_info.C, enter:

```
bldapp cli_info
```

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

```
cli_info
```

Building and Running Embedded SQL Applications

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

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

The result is an executable file, tbmod.

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

```
tbmod database
```

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

```
tbmod database userid password
```

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "Linux C++ Application Compile and Link Options" on page 245

Related samples:

- "bldapp -- Builds Linux C++ applications"
- "cli_info.C -- Set and get information at the client level (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C++ Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds Linux C++ applications
# 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 specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-Wl,-rpath$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

# If an embedded SQL program, precompile and bind it.
if [ -f $1".sqC" ]
then
  ./embprep $1 $2 $3 $4
  # Compile the utilemb.C error-checking utility.
  g++ -I$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 $EXTRA_LFLAG -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  g++ -o $1 $1.o utilapi.o $EXTRA_LFLAG -L$DB2PATH/lib -ldb2
fi
```

Linux C++ Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building C++ embedded SQL and DB2 API applications with the Linux C++ compiler, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp	
Compile Options:	
g++	The GNU/Linux C++ compiler.
-I$DB2PATH/include	Specify the location of the DB2 include files.
-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.
$EXTRA_LFLAG	If 'RUNTIME=true' is uncommented, contains value "-Wl,-rpath,$DB2PATH/lib"; otherwise, contains no value.
-Wl,-rpath,$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-L$DB2PATH/lib	Specify the location of the DB2 shared libraries at link-time. For example: `$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.	

Related tasks:

- "Building C++ Applications on Linux" on page 242

Related samples:

- "bldapp -- Builds Linux C++ applications"

Building C++ Routines on Linux

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, contains the commands to build routines (stored procedures and user-defined functions). The script compiles the routines into a shared library that can be called by a client application. It takes one or two parameters.

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 database parameter is optional. If no database name is supplied, the program uses the default sample database.

Procedure:

The following examples show you how to build routine shared libraries with:

- stored procedures
- non-embedded SQL user-defined functions (UDFs)
- embedded SQL user-defined functions (UDFs)

Stored Procedure Shared Library

To build the sample program spserver from the source file spserver.sqC:

1. If connecting to the sample database, enter the build file name and program name:

   ```
   bldrtn spserver
   ```

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

   ```
   bldrtn spserver database
   ```

   The script file copies the stored procedure to the server in the path sqllib/function.
2. Next, catalog the routines by running the spcat script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling spdrop.db2, then catalogs them by calling spcreate.db2, and finally disconnects from the database. You can also call the spdrop.db2 and spcreate.db2 scripts individually.
3. Then, stop and restart the database to allow the new shared library to be recognized.

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, bldapp.

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 for the user ID.

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.

Non-embedded SQL UDF Shared Library

To build the user-defined function program, `udfsrv`, from the source file `udfsrv.C`, enter the build script name and program name:

```
bldrtn udfsrv
```

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

Once you build `udfsrv`, you can build the client application, `udfcli`, that calls it. You can build `udfcli` from the source file `udfcli.sqC` using the script, `bldapp`.

To call the UDFs in the shared library, run the client application by entering:

`udfcli` *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 for the user ID.

The client application accesses the shared library, `udfsrv`, and executes the user-defined functions on the server database. The output is returned to the client application.

Embedded SQL UDF Shared Library

To build the embedded SQL user-defined function program, udfemsrv, from the source file, udfemsrv.sqC, if connecting to the sample database, enter the build script name and program name:

```
bldrtn udfemsrv
```

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

```
bldrtn udfemsrv database
```

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

Once you build udfemsrv, you can build the client application, udfemcli, that calls it. You can build udfemcli from the source file udfemcli.sqC using the script, bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfemcli *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 for the user ID.

The client application accesses the shared library, udfemsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "Linux C++ Routine Compile and Link Options" on page 250

Related samples:

- "bldrtn -- Builds Linux C++ routines (stored procedures and UDFs)"
- "spclient.sqC -- Call various stored procedures (C++)"
- "spserver.sqC -- A variety of types of stored procedures (C++)"
- "udfcli.sqC -- Call a variety of types of user-defined functions (C++)"

- “udfemcli.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)”
- “udfemsrv.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)”
- “udfsrv.C -- Call a variety of types of user-defined functions (C++)”
- “embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs”
- “spcat -- To catalog C stored procedures on UNIX”
- “spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc ”
- “spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc ”

Build Script for C++ Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds Linux C++ routines (stored procedures and UDFs)
# Usage: bldsrv <prog_name> [ <db_name> ]

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

# Set the runtime path since routines run as setuid.
EXTRA_LFLAG="-Wl,-rpath$DB2PATH/lib"

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

# Compile the program.
g++ -fpic -I$DB2PATH/include -c $1.C -D_REENTRANT

# Link the program and create a shared library.
g++ -shared -o $1 $1.o $EXTRA_LFLAG -L$DB2PATH/lib -ldb2 -lpthread

# 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
```

Linux C++ Routine Compile and Link Options

These are the compile and link options recommended by DB2 for building C++ routines (stored procedures and user-defined functions) with the Linux C++ compiler, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn
Compile Options: **g++** The GNU/Linux C++ compiler. **-fpic** Generate position independent code. **-I$DB2PATH/include** Specify the location of the DB2 include files. **-c** Perform compile only; no link. This script file has separate compile and link steps. **-D_REENTRANT** Defines _REENTRANT, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).
Link Options: **g++** Use the compiler as a front end for the linker. **-shared** Generate a shared library. **-o $1** Specify the executable. **$1.o** Include the program object file. **$EXTRA_LFLAG** Contains the value ″-Wl,-rpath,$DB2PATH/lib″ to set the runtime path since routines run as setuid. **-Wl,-rpath,$DB2PATH/lib** Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`. **-L$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. **-lpthread** Link with the POSIX thread library. Refer to your compiler documentation for additional compiler options.

Related tasks:

- “Building C++ Routines on Linux” on page 246

Related samples:

- “bldrtn -- Builds Linux C++ routines (stored procedures and UDFs)”

Building C++ Multi-Threaded Applications on Linux

Multi-threaded applications using Linux C++ need to be compiled with `-D_REENTRANT` and linked with `-lpthread`.

DB2 provides build scripts for compiling and linking C++ embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The script script, `bldmt`, contains the commands to build an embedded SQL multi-threaded program. It takes up to four parameters.

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.

Procedure:

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

```
bldmt dbthrds
```

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

```
dbthrds
```

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69

Related samples:

- "bldmt -- Builds Linux C++ multi-threaded applications"
- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C++ Multi-threaded Applications

```
#! /bin/sh
# SCRIPT: bldmt
# Builds Linux C++ multi-threaded applications
# 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 specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-Wl,-rpath$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

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

# Compile the program.
g++ -I$DB2PATH/include -c $1.C -D_REENTRANT

# Link the program.
g++ -o $1 $1.o $EXTRA_LFLAG -L$DB2PATH/lib -ldb2 -lpthread
```

Chapter 10. Solaris

This chapter provides detailed information for building applications in the Solaris operating environment. For the latest DB2 application development updates for Solaris, visit the Web page at:

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

Forte C

Note: Forte C was formerly called ″SPARCompiler C″.

Building information for DB2 CLI Applications and routines is in the *CLI Guide and Reference*.

Building C Applications on Solaris

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The build script, `bldapp`, contains the commands to build DB2 application programs. It takes up to four parameters.

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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

To build the DB2 API non-embedded SQL sample program, cli_info, from the source file cli_info.c, enter:

```
bldapp cli_info
```

The result is an executable file, cli_info.

To run the executable file, enter the executable name:

```
cli_info
```

Building and Running Embedded SQL Applications

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

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

   ```
   bldapp tbmod
   ```

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

   ```
   bldapp tbmod database
   ```

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

   ```
   bldapp tbmod database userid password
   ```

The result is an executable file, tbmod.

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:

   ```
   tbmod
   ```

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

   ```
   tbmod database
   ```

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

   ```
   tbmod database userid password
   ```

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "Solaris C Application Compile and Link Options" on page 258

Related samples:

- "bldapp -- Builds Solaris C applications"
- "cli_info.c -- Set and get information at the client level (C)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "tbmod.sqc -- How to modify table data (C)"

Build Script for C Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds Solaris C applications
# 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
  CFLAG_ARCH=v9
else
  CFLAG_ARCH=v8plusa
fi

# To specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true
```

```
if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-R$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

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

# Compile the program.
cc -xarch=$CFLAG_ARCH -I$DB2PATH/include -c $1.c

if [ -f $1".sqc" ]
then
  # Link the program with utilemb.o
  cc -xarch=$CFLAG_ARCH -mt -o $1 $1.o utilemb.o $EXTRA_LFLAG \
    -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  cc -xarch=$CFLAG_ARCH -mt -o $1 $1.o utilapi.o $EXTRA_LFLAG \
    -L$DB2PATH/lib -ldb2
fi
```

Solaris C Application Compile and Link Options

These are the compile and link options recommended by DB2 for building C embedded SQL and DB2 API applications with the Forte C compiler, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp
Compile Options:
cc The C compiler.
-xarch=$CFLAG_ARCH This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either "v8plusa" for 32-bit, or "v9" for 64-bit.
-I$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. **-xarch=$CFLAG_ARCH** This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either ″v8plusa″ for 32-bit, or ″v9″ for 64-bit. **-mt** Link in multi-thread support. Needed for linking with libdb2. **Note:** If POSIX threads are used, DB2 applications also have to link with `-lpthread`, whether or not they are threaded. **-o $1** Specify the executable. **$1.o** Include the program object file. **utilemb.o** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.o** If not an embedded SQL program, include the DB2 API utility object file for error checking. **$EXTRA_LFLAG** If 'RUNTIME=true' is uncommented, contains value ″-R$DB2PATH/lib″; otherwise, contains no value. **-R$DB2PATH/lib** Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`. **-L$DB2PATH/lib** Specify the location of the DB2 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.

Related tasks:

- "Building C Applications on Solaris" on page 255

Related samples:

- "bldapp -- Builds Solaris C applications"

Building C Routines on Solaris

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, contains the commands to build routines (stored procedures and user-defined functions). The script file compiles the routines into a shared library that can be loaded by the database manager and called by a client application.

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

The database parameter is optional. If no database name is supplied, the program uses the default `sample` database. And since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Procedure:

The following examples show you how to build routine shared libraries with:

- stored procedures
- non-embedded SQL user-defined functions (UDFs)
- embedded SQL user-defined functions (UDFs)

Stored Procedure Shared Library

To build the sample program `spserver` from the source file `spserver.sqc`:

1. If connecting to the `sample` database, enter the build script name and the program name:

   ```
   bldrtn spserver
   ```

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

   ```
   bldrtn spserver database
   ```

   The script file copies the shared library to the `sqllib/function` directory on the server.
2. Next, catalog the routines by running the `spcat` script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling `spdrop.db2`, then catalogs them by

calling spcreate.db2, and finally disconnects from the database. You can also call the spdrop.db2 and spcreate.db2 scripts individually.

3. Then, stop and restart the database to allow the new shared library to be recognized.

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.

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 for the user ID.

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.

Non-embedded SQL UDF Shared Library

To build the user-defined function program, udfsrv, from the source file udfsrv.c, enter the build script name and the program name:

```
bldrtn udfsrv
```

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

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 client program from the source file udfcli.c, in sqllib/samples/cli, using the script, bldapp.

You can build the embedded SQL udfcli client program from the source file udfcli.sqc, in sqllib/samples/c, using the script file, bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfcli *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 for the user ID.

The client application accesses the shared library, udfsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Embedded SQL UDF Shared Library

To build the embedded SQL user-defined function program, udfemsrv, from the source file udfemsrv.sqc, if connecting to the sample database, enter the build script name and program name:

```
bldrtn udfemsrv
```

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

```
bldrtn udfemsrv database
```

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

Once you build udfemsrv, you can build the client application, udfemcli, that calls it. You can build the udfemcli client program from the source file, udfemcli.sqc, in sqllib/samples/c, using the script file bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfemcli *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 for the user ID.

The client application accesses the shared library, udfemsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "Solaris C Routine Compile and Link Options" on page 264

Related samples:

- "bldrtn -- Builds Solaris C routines (stored procedures or UDFs)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "spcat -- To catalog C stored procedures on UNIX"
- "spclient.sqc -- Call various stored procedures (C)"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "
- "spserver.sqc -- A variety of types of stored procedures (C)"
- "udfcli.sqc -- Call a variety of types of user-defined functions (C)"
- "udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfemsrv.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfsrv.c -- Call a variety of types of user-defined functions (C)"

Build Script for C Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds Solaris C routines (stored procedures or UDFs)
# Usage: bldrtn <prog_name> [ <db_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
  CFLAG_ARCH=v9
```

```
else
  CFLAG_ARCH=v8plusa
fi

# Set the runtime path since routines run as setuid.
EXTRA_LFLAG="-R$DB2PATH/lib"

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

# Compile the program.
cc -xarch=$CFLAG_ARCH -mt -DUSE_UI_THREADS -Kpic \
  -I$DB2PATH/include -c $1.c

# Link the program and create a shared library
cc -xarch=$CFLAG_ARCH -mt -G -o $1 $1.o $EXTRA_LFLAG \
  -L$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
```

Solaris C Routine Compile and Link Options

These are the compile and link options recommended by DB2 for building C routines (stored procedures and user-defined functions) with the Forte C compiler, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn
Compile Options:
cc The C compiler.
-xarch=$CFLAG_ARCH This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either "v8plusa" for 32-bit, or "v9" for 64-bit.
-mt Allow multi-threaded support, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).
-DUSE_UI_THREADS Allows Sun's "UNIX International" threads APIs.
-Kpic Generate position-independent code for shared libraries.
-I$DB2PATH/include Specify the location of the DB2 include files.
-c Perform compile only; no link. This script has separate compile and link steps.

Compile and Link Options for bldrtn	
Link Options:	
cc	Use the compiler as a front end for the linker.
-xarch=$CFLAG_ARCH	This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either "v8plusa" for 32-bit, or "v9" for 64-bit.
-mt	This is required because the DB2 library is linked with `-mt`.
-G	Generate a shared library.
-o $1	Specify the executable.
$1.o	Include the program object file.
$EXTRA_LFLAG	Contains the value "-R$DB2PATH/lib" to set the runtime path since routines run as setuid.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-L$DB2PATH/lib	Specify the location of the DB2 shared libraries at link-time. For example: `$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.	

Related tasks:

- "Building C Routines on Linux" on page 235

Related samples:

- "bldrtn -- Builds Linux C routines (stored procedures or UDFs)"

Building C Multi-Threaded Applications on Solaris

Multi-threaded applications using SUN and POSIX thread libraries are supported by DB2. The default is Sun threads. Multi-threaded applications using Forte C on Solaris need to be compiled and linked with `-mt`. This will pass `-D_REENTRANT` to the preprocessor, and `-lthread` to the linker. You also need to specify the compile define `-DUSE_UI_THREADS`, to use Sun's "Unix International" threads APIs.

Note: If you want to use POSIX threads, you have to add the compiler option `-D_POSIX_PTHREAD_SEMANTICS`, which allows POSIX variants of functions such as `getpwnam_r()`, and also add the link option `-lpthread`. If you are using the `bldmt` script provided, you also have to delete the `-DUSE_UI_THREADS` define.

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The script, `bldmt` contains the commands to build a multi-threaded application.

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

Procedure:

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

```
bldmt dbthrds
```

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

```
dbthrds
```

Note: For multi-threaded programs with a fair number of connections, the kernel parameters `semsys:seminfo_semume` and `shmsys:shminfo_shmseg` may have to be set beyond their default values. Please see the related link below on the db2osconf utility to obtain recommendations on the values to set for these parameters.

Related concepts:

- "Build Files" on page 94
- "db2osconf - Utility for Kernel Parameter Values" in the *Command Reference*

Related reference:

- "C/C++ Samples" on page 69

Related samples:

- "bldmt -- Builds Solaris C multi-threaded applications"

- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C Multi-threaded Applications

```
#! /bin/sh
# SCRIPT: bldmt
# Builds Solaris C multi-threaded applications
# 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 64-bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAG_ARCH=v9
else
  CFLAG_ARCH=v8plusa
fi

# To specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-R$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

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

# Compile the program.
cc -xarch=$CFLAG_ARCH -mt -DUSE_UI_THREADS -I$DB2PATH/include -c $1.c

# Link the program.
cc -xarch=$CFLAG_ARCH  -mt -o $1 $1.o $EXTRA_LFLAG -L$DB2PATH/lib -ldb2
```

Forte C++

Note: Forte C++ was formerly called SPARCompiler C++

Building C++ Applications on Solaris

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The build script, `bldapp`, contains the commands to build DB2 API and embedded SQL applications. It takes up to four parameters.

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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

To build the non-embedded SQL sample program `cli_info` from the source file `cli_info.C`, enter:

```
bldapp cli_info
```

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

```
cli_info
```

Building and Running Embedded SQL Applications

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

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

   ```
   bldapp tbmod
   ```

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

   ```
   bldapp tbmod database
   ```

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

   ```
   bldapp tbmod database userid password
   ```

The result is an executable file, `tbmod`.

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:

   ```
   tbmod
   ```

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

   ```
   tbmod database
   ```

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

   ```
   tbmod database userid password
   ```

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "Solaris C++ Application Compile and Link Options" on page 270

Related samples:

- "bldapp -- Builds Solaris C++ applications"
- "cli_info.C -- Set and get information at the client level (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C++ Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds Solaris C++ applications
# 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
  CFLAG_ARCH=v9
else
  CFLAG_ARCH=v8plusa
fi

# To specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-R$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

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

# Compile the program.
CC -xarch=$CFLAG_ARCH -I$DB2PATH/include -c $1.C

if [ -f $1".sqC" ]
then
  # Link the program with utilemb.o
  CC -xarch=$CFLAG_ARCH -mt -o $1 $1.o utilemb.o $EXTRA_LFLAG \
    -L$DB2PATH/lib -ldb2
else
  # Link the program with utilapi.o
  CC -xarch=$CFLAG_ARCH -mt -o $1 $1.o utilapi.o $EXTRA_LFLAG \
    -L$DB2PATH/lib -ldb2
fi
```

Solaris C++ Application Compile and Link Options

These are the compile and link options recommended by DB2 for building C++ embedded SQL and DB2 API applications with the Forte C++ compiler, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp
Compile Options: **CC** The C++ compiler. **-xarch=$CFLAG_ARCH** This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either ″v8plusa″ for 32-bit, or ″v9″ for 64-bit. **-I$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. **-xarch=$CFLAG_ARCH** This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either ″v8plusa″ for 32-bit, or ″v9″ for 64-bit. **-mt** Link in multi-thread support. Needed for linking with libdb2. **Note:** If POSIX threads are used, DB2 applications also have to link with `-lpthread`, whether or not they are threaded. **-o $1** Specify the executable. **$1.o** Include the program object file. **utilemb.o** If an embedded SQL program, include the embedded SQL utility object file for error checking. **utilapi.o** If a non-embedded SQL program, include the DB2 API utility object file for error checking. **$EXTRA_LFLAG** If 'RUNTIME=true' is uncommented, contains value ″-R$DB2PATH/lib″; otherwise, contains no value. **-R$DB2PATH/lib** Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`. **-L$DB2PATH/lib** Specify the location of the DB2 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.

Related tasks:

- "Building C++ Applications on Solaris" on page 268

Related samples:

- "bldapp -- Builds Solaris C++ applications"

Building C++ Routines on Solaris

DB2 provides build scripts for compiling and linking C embedded SQL and DB2 API programs. These are located in the `sqllib/samples/c` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, contains the commands to build routines (stored procedures or user-defined functions). The script compiles the routines into a shared library that can be called by a client application.

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

The database parameter is optional. If no database name is supplied, the program uses the default `sample` database. And since the stored procedure must be built on the same instance where the database resides, there are no parameters for user ID and password.

Procedure:

The following examples show you how to build routine shared libraries with:

- stored procedures
- non-embedded SQL user-defined functions (UDFs)
- embedded SQL user-defined functions (UDFs)

Stored Procedure Shared Library

To build the sample program `spserver` from the source file `spserver.sqC`:

1. If connecting to the `sample` database, enter the build file name and program name:

   ```
   bldrtn spserver
   ```

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

   ```
   bldrtn spserver database
   ```

   The script file copies the shared library to the `sqllib/function` directory on the server.
2. Next, catalog the routines by running the `spcat` script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling `spdrop.db2`, then catalogs them by

calling spcreate.db2, and finally disconnects from the database. You can also call the spdrop.db2 and spcreate.db2 scripts individually.

3. Then, stop and restart the database to allow the new shared library to be recognized.

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, bldapp.

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 for the user ID.

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.

Non-embedded SQL UDF Shared Library

To build the user-defined function program, udfsrv, from the source file udfsrv.C, enter the build script name and program name:

```
bldrtn udfsrv
```

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

Once you build udfsrv, you can build the client application, udfcli, that calls it. You can build udfcli from the source file udfcli.sqC using the script file bldapp.

To call the UDFs in the shared library, run the client application by entering:

udfcli *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 for the user ID.

The client application accesses the shared library, `udfsrv`, and executes the user-defined functions on the server database. The output is returned to the client application.

Embedded SQL UDF Shared Library

To build the embedded SQL user-defined function program, `udfemsrv`, from the source file, `udfemsrv.sqC`, if connecting to the `sample` database, enter the build script name and program name:

```
bldrtn udfemsrv
```

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

```
bldrtn udfemsrv database
```

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

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

Once you build `udfemsrv`, you can build the client application, `udfemcli`, that calls it. You can build `udfemcli` from the source file `udfemcli.sqC` using the script file `bldapp`.

To call the UDFs in the shared library, run the client application by entering:

`udfemcli` *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 for the user ID.

The client application accesses the shared library, udfemsrv, and executes the user-defined functions on the server database. The output is returned to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "C/C++ Samples" on page 69
- "Solaris C++ Routine Compile and Link Options" on page 277

Related samples:

- "bldrtn -- Builds Solaris C++ routines (stored procedures or UDFs)"
- "spclient.sqC -- Call various stored procedures (C++)"
- "spserver.sqC -- A variety of types of stored procedures (C++)"
- "udfcli.sqC -- Call a variety of types of user-defined functions (C++)"
- "udfsrv.C -- Call a variety of types of user-defined functions (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "

Build Script for C++ Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds Solaris C++ routines (stored procedures or UDFs)
# Usage: bldrtn <prog_name> [ <db_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
  CFLAG_ARCH=v9
else
  CFLAG_ARCH=v8plusa
fi

# Set the runtime path since routines run as setuid.
EXTRA_LFLAG="-R$DB2PATH/lib"
```

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

# Compile the program.
CC -xarch=$CFLAG_ARCH -mt -DUSE_UI_THREADS -Kpic \
  -I$DB2PATH/include -c $1.C

# Link the program and create a shared library
CC -xarch=$CFLAG_ARCH -mt -G -o $1 $1.o $EXTRA_LFLAG \
  -L$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
```

Solaris C++ Routine Compile and Link Options

These are the compile and link options recommended by DB2 for building C++ routines (stored procedures and user-defined functions) with the Forte C++ compiler, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn
Compile Options:
CC The C++ compiler.
-xarch=$CFLAG_ARCH This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either "v8plusa" for 32-bit, or "v9" for 64-bit.
-mt Allow multi-threaded support, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).
-DUSE_UI_THREADS Allows Sun's "UNIX International" threads APIs.
-Kpic Generate position-independent code for shared libraries.
-I$DB2PATH/include Specify the location of the DB2 include files.
-c Perform compile only; no link. This script has separate compile and link steps.

Compile and Link Options for bldrtn	
Link Options:	
CC	Use the compiler as a front end for the linker.
-xarch=$CFLAG_ARCH	This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either ″v8plusa″ for 32-bit, or ″v9″ for 64-bit.
-mt	This is required because the DB2 library is linked with `-mt`.
-G	Generate a shared library.
-o $1	Specify the executable.
$1.o	Include the program object file.
$EXTRA_LFLAG	Contains value ″-R$DB2PATH/lib″ to set the runtime path since routines run as setuid.
-R$DB2PATH/lib	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
-L$DB2PATH/lib	Specify the location of the DB2 shared libraries at link-time. For example: `$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.	

Related tasks:

- “Building C++ Routines on Solaris” on page 273

Related samples:

- “bldrtn -- Builds Solaris C++ routines (stored procedures or UDFs)”

Building C++ Multi-Threaded Applications on Solaris

Multi-threaded applications using SUN and POSIX thread libraries are supported by DB2. The default is Sun threads. Multi-threaded applications using Forte C++ on Solaris need to be compiled and linked with `-mt`. This will pass `-D_REENTRANT` to the preprocessor, and `-lthread` to the linker. You also need to specify the compile define `-DUSE_UI_THREADS`, to use Sun’s ″Unix International″ threads APIs.

Note: If you want to use POSIX threads, you have to add the compiler option `-D_POSIX_PTHREAD_SEMANTICS`, which allows POSIX variants of functions such as `getpwnam_r()`, and also add the link option `-lpthread`. If you are using the `bldmt` script provided, you also have to delete the `-DUSE_UI_THREADS` define.

DB2 provides build scripts for compiling and linking C++ embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cpp` directory, along with sample programs that can be built with these files.

The script, `bldmt`, contains the commands to build a multi-threaded application. It takes up to four parameters.

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.

Procedure:

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

```
bldmt dbthrds
```

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

```
dbthrds
```

Note: For multi-threaded programs with a fair number of connections, the kernel parameters `semsys:seminfo_semume` and `shmsys:shminfo_shmseg` may have to be set beyond their default values. Please see the related link below on the db2osconf utility to obtain recommendations on the values to set for these parameters.

Related concepts:

- "Build Files" on page 94
- "db2osconf - Utility for Kernel Parameter Values" in the *Command Reference*

Related reference:

- "C/C++ Samples" on page 69

Related samples:

- "bldmt -- Builds Solaris C++ multi-threaded applications"

- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for C++ Multi-threaded Applications

```
#! /bin/sh
# SCRIPT: bldmt
# Builds Solaris C++ multi-threaded applications
# 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 64-bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAG_ARCH=v9
else
  CFLAG_ARCH=v8plusa
fi

# To specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-R$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

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

# Compile the program.
CC -xarch=$CFLAG_ARCH -mt -DUSE_UI_THREADS -I$DB2PATH/include -c $1.C

# Link the program.
CC -xarch=$CFLAG_ARCH -mt -o $1 $1.o $EXTRA_LFLAG -L$DB2PATH/lib -ldb2
```

Micro Focus COBOL

Configuring the Micro Focus COBOL Compiler on Solaris

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

Procedure:

- When you precompile your application using the command line processor command `db2 prep`, use the `target mfcob` option.
- 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:

  – On bash or Korn shells:

  ```
  export COBCPY=$COBCPY:$HOME/sqllib/include/cobol_mf
  ```

  – On C shell:

  ```
  setenv COBCPY $COBCPY:$HOME/sqllib/include/cobol_mf
  ```

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

Related tasks:

- "Building Micro Focus COBOL Routines on Solaris" on page 284
- "Building Micro Focus COBOL Applications on Solaris" on page 281

Building Micro Focus COBOL Applications on Solaris

DB2 provides build scripts for compiling and linking Micro Focus COBOL embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cobol_mf` directory, along with sample programs that can be built with these files.

The build script, `bldapp`, contains the commands to build a DB2 application program. It takes up to four parameters.

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 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.

Procedure:

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
   ```

Related concepts:

- "Build Files" on page 94

Related reference:

- "Solaris Micro Focus COBOL Application Compile and Link Options" on page 283
- "COBOL Samples" on page 80

Related samples:

- "bldapp -- Builds Solaris Micro Focus COBOL applications"
- "client.cbl -- How to set and query a client (MF COBOL)"
- "updat.sqb -- How to update, delete and insert table data (MF COBOL)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for Micro Focus COBOL Applications

```
#! /bin/sh
# SCRIPT: bldapp
# Builds Solaris Micro Focus COBOL applications
# 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.
COBCPY=$COBCPY:$DB2PATH/include/cobol_mf

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

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

# Link the program.
cob -x $1.o checkerr.o -L$DB2PATH/lib -ldb2 -ldb2gmf
```

Solaris Micro Focus COBOL Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL embedded SQL and DB2 API applications with the Micro Focus COBOL compiler on Solaris, as demonstrated in the `bldapp` build script.

Compile and Link Options for bldapp	
Compile Options:	
cob	The Micro Focus COBOL compiler.
-cx	Compile to object module.
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.	

Related tasks:

- "Building Micro Focus COBOL Applications on Solaris" on page 281

Related samples:

- "bldapp -- Builds Solaris Micro Focus COBOL applications"

Building Micro Focus COBOL Routines on Solaris

DB2 provides build scripts for compiling and linking Micro Focus COBOL embedded SQL and DB2 API programs. These are located in the `sqllib/samples/cobol_mf` directory, along with sample programs that can be built with these files.

The script, `bldrtn`, contains the commands to build routines (stored procedures). The script file compiles the routines into a shared library that can be called by a client application.

The first parameter, `$1`, specifies the name of your source file. The script file uses the source file name, `$1`, for the shared library name. The second parameter, `$2`, specifies the name of the database to which you want to connect. Since the routine 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.

Procedure:

Before building Micro Focus routines 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.

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`.

Once you build the shared library, `outsrv`, you can build the client application, `outcli`, that calls the routine of the same name within it. You can build `outcli` using the script file, `bldapp`.

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 for the user ID.

The client application accesses the shared library, `outsrv`, which executes the routine of the same name on the server database, and then returns the output to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "Solaris Micro Focus COBOL Routine Compile and Link Options" on page 287
- "COBOL Samples" on page 80

Related samples:

- "bldrtn -- Builds Solaris Micro Focus COBOL routines (stored procedures)"
- "outcli.sqb -- Call stored procedures using the SQLDA structure (MF COBOL)"
- "outsrv.sqb -- Demonstrates stored procedures using the SQLDA structure (MF COBOL)"
- "embprep -- To prep and bind C/C++ and Micro Focus COBOL embedded SQL programs"

Build Script for Micro Focus COBOL Routines

```
#! /bin/sh
# SCRIPT: bldrtn
# Builds Solaris Micro Focus COBOL routines (stored procedures)
# Usage: bldrtn <prog_name> [ <db_name> ]

# 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
fi

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

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

# Link the program.
cob -x -o $1 $1.o -Q -G -L$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
```

Solaris Micro Focus COBOL Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL routines (stored procedures) with the Micro Focus COBOL compiler on Solaris, as demonstrated in the `bldrtn` build script.

Compile and Link Options for bldrtn	
Compile Options:	
cob	The COBOL compiler.
-cx	Compile to object module.
Link Options:	
cob	Use the compiler as a front-end for the linker.
-x	Produces a shared library when used with the `-G` option.
-o $1	Specify the executable program.
$1.o	Specify the program object file.
-Q -G	Generate a shared library.
-L$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`.
-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.	

Related tasks:

- "Building Micro Focus COBOL Routines on Solaris" on page 284

Related samples:

- "bldrtn -- Builds Solaris Micro Focus COBOL routines (stored procedures)"

Chapter 11. Windows Operating Systems

This chapter provides detailed information for building applications on Windows operating systems. For the latest DB2 application development updates for Windows, visit the Web page at:

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

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.

For DB2® for Windows® 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 default option for WCHARTYPE is NOCONVERT. 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 routines (stored procedures and user-defined functions).

Related reference:

- "PRECOMPILE" in the *Command Reference*

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 `sqllib\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`.

These commands must be done in a DB2 Command Window.

Related reference:

- "Object Linking and Embedding Database (OLE DB) Table Function Samples" on page 90

Windows Management Instrumentation (WMI)

Windows® Management Instrumentation (WMI) is a key component of Microsoft's Windows management services. WMI provides a consistent and richly descriptive model of the configuration, status, and operational aspects of applications and system.

The DB2® WMI provider allows WMI applications to monitor DB2 server services, enumerate and create databases, configure operational settings and perform database backup, restore, and roll-forward operations.

DB2 provides WMI sample files for the Visual Basic Scripting language located in the `sqllib\samples\wmi` directory. Before running the sample programs, ensure that the DB2 WMI Provider is registered by running the following commands:

```
mofcomp  %DB2PATH%\bin\db2wmi.mof
regsvr32 %DB2PATH%\bin\db2wmi.dll
```

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

Use the `cscript` command to run the Visual Basic Script samples. For example, to run the `listsvr` sample script, enter:

```
cscript listsvr.vbs
```

Related reference:

- "Windows Management Instrumentation Samples" on page 88

Microsoft Visual Basic

Building ADO Applications with Visual Basic

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.

Visual Basic ADO sample programs are located in the `sqllib\samples\VB\ADO` directory.

Note: To run the DB2 ADO samples, these versions or later of the following components are recommended:

1. Visual Basic 6.0 Professional Edition
2. Microsoft Data Access 2.7 SDK (optionally installed with DB2 Version 8)
3. Visual Basic Service pack 5 from http://msdn.microsoft.com/vstudio/sp/vs6sp5/vbfixes.asp.
4. The latest Visual Studio Service Pack from http://msdn.microsoft.com/vstudio/.

Procedure:

You can use either of two ODBC-compliant providers:

- IBM OLE DB provider for DB2
- Microsoft OLE DB provider for ODBC

Using the IBM OLE DB provider for DB2

DB2 Version 8.1 Clients on Windows operating systems will optionally install IBMDADB2, the IBM OLE DB 2.0-compliant provider for DB2. The provider exposes interfaces for consumers who want to access data in a DB2 database. The IBM OLE DB provider for DB2 supports the following ADO application types:

- Microsoft Active Server Pages (ASP)
- Microsoft Visual Studio C++ and Visual Basic applications
- Microsoft Visual Interdev

For details on these types of applications, refer to the ADO documentation.

To access a DB2 server using the IBM OLE DB provider for DB2, the Visual Basic application should specify the PROVIDER keyword in the ADO connection string as follows:

```
Dim c1 As ADODB.Connection
Dim c1str As String
c1str = "Provider=ibmdadb2; DSN=db2alias; UID=userid; PWD=password"
c1.Open c1str
...
```

where `db2alias` is the alias for the DB2 database which is cataloged in the DB2 database directory.

Note: When using the IBM OLE DB provider for DB2, you do not need to perform the ODBC catalog step for the datasource. This step is required when you are using the OLE DB provider for ODBC.

Using the Microsoft OLE DB provider for ODBC

To use ADO with the Microsoft OLE DB provider and 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.

Establish a connection:

```
Dim db As Connection
Set db = New Connection
```

Set client-side cursors supplied by the local cursor library:

```
db.CursorLocation = adUseClient
```

and set the provider so ADO will use the Microsoft ODBC Driver.

Accessing the sample database with ADO

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, after you have connected to the database with either the IBM OLE DB provider or the Microsoft OLE DB provider, as discussed above.

Open the `sample` database without specifying a user ID or 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
```

Related concepts:

- "Purpose of the IBM OLE DB Provider for DB2" in the *Application Development Guide: Programming Client Applications*
- "Application Types Supported by the IBM OLE DB Provider for DB2" in the *Application Development Guide: Programming Client Applications*
- "Connections to Data Sources with Visual Basic ADO Applications" in the *Application Development Guide: Programming Client Applications*
- "OLE DB Services Automatically Enabled by IBM OLE DB Provider" in the *Application Development Guide: Programming Client Applications*
- "Large Object Manipulation with the IBM OLE DB Provider" in the *Application Development Guide: Programming Client Applications*
- "MTS and COM+ Distributed Transaction Support and the IBM OLE DB Provider" in the *Application Development Guide: Programming Client Applications*
- "IBM OLE DB Provider Restrictions" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "IBM OLE DB Provider Support for OLE DB Components and Interfaces" in the *Application Development Guide: Programming Client Applications*
- "IBM OLE DB Provider Support for OLE DB Properties" in the *Application Development Guide: Programming Client Applications*
- "IBM OLE DB Provider Support for ADO Methods and Properties" in the *Application Development Guide: Programming Client Applications*
- "Visual Basic Samples" on page 86

Building RDO Applications with Visual Basic

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.

DB2 provides Visual Basic RDO sample programs in the `sqllib\samples\VB` directory.

Procedure:

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 in 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
```

Related reference:

• "Visual Basic Samples" on page 86

Object Linking and Embedding (OLE) Automation with Visual Basic

You can implement OLE automation UDFs and stored procedures in any language, as OLE is language independent. You do this 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.

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. For further information on creating and building OLE servers, refer to the Microsoft Visual Basic manual, *Creating OLE Servers, Microsoft Corporation, 1995*, and to the OLE samples provided by Microsoft Visual Basic.

DB2 provides self-containing samples of OLE automation UDFs and stored procedures in Microsoft Visual Basic, located in the directory `sqllib\samples\ole\msvb`. For information on building and running the OLE automation UDF and stored procedure samples, please see the `README` file in `sqllib\samples\ole`.

Related reference:

- "Object Linking and Embedding (OLE) Samples" on page 89

Microsoft Visual C++

This section discusses building applications with ActiveX Data Objects (ADO), Object Linking and Embedding (OLE), as well as embedded SQL and DB2 APIs.

Building information for DB2 CLI applications and routines is in the *CLI Guide and Reference*.

Building ADO Applications with Visual C++

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.

DB2 provides Visual C++ ADO sample programs in the `sqllib\samples\VC` directory.

Procedure:

You can use either of two ODBC-compliant providers:

- IBM OLE DB provider for DB2
- Microsoft OLE DB provider for ODBC

Using the IBM OLE DB provider for DB2

DB2 Version 8.1 Clients on Windows operating systems will optionally install IBMDADB2, the IBM OLE DB 2.0-compliant provider for DB2. The provider exposes interfaces for consumers who want to access data in a DB2 database. The IBM OLE DB provider for DB2 supports the following ADO application types:

- Microsoft Active Server Pages (ASP)
- Microsoft Visual Studio C++ and Visual Basic applications
- Microsoft Visual Interdev

For details on these types of applications, refer to the ADO documentation.

Using the Microsoft OLE DB provider for ODBC

DB2 ADO programs using the Microsoft OLE DB provider and 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.

Related concepts:

- "Purpose of the IBM OLE DB Provider for DB2" in the *Application Development Guide: Programming Client Applications*
- "Application Types Supported by the IBM OLE DB Provider for DB2" in the *Application Development Guide: Programming Client Applications*
- "Compilation and Linking of C/C++ Applications and the IBM OLE DB Provider" in the *Application Development Guide: Programming Client Applications*
- "Connections to Data Sources in C/C++ Applications using the IBM OLE DB Provider" in the *Application Development Guide: Programming Client Applications*
- "OLE DB Services Automatically Enabled by IBM OLE DB Provider" in the *Application Development Guide: Programming Client Applications*

- "Large Object Manipulation with the IBM OLE DB Provider" in the *Application Development Guide: Programming Client Applications*
- "IBM OLE DB Provider Restrictions" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "Data Type Mappings between DB2 and OLE DB" in the *Application Development Guide: Programming Client Applications*
- "Data Conversion for Setting Data from OLE DB Types to DB2 Types" in the *Application Development Guide: Programming Client Applications*
- "Data Conversion for Setting Data from DB2 Types to OLE DB Types" in the *Application Development Guide: Programming Client Applications*
- "IBM OLE DB Provider Support for OLE DB Components and Interfaces" in the *Application Development Guide: Programming Client Applications*
- "IBM OLE DB Provider Support for OLE DB Properties" in the *Application Development Guide: Programming Client Applications*
- "IBM OLE DB Provider Support for ADO Methods and Properties" in the *Application Development Guide: Programming Client Applications*
- "Visual C++ Samples" on page 88

Object Linking and Embedding (OLE) Automation with Visual C++

You can implement OLE automation UDFs and stored procedures in any language, as OLE is language independent. You do this 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.

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.

DB2® provides self-containing samples of OLE automation UDFs and stored procedures in Microsoft Visual C++, located in the directory `sqllib\samples\ole\msvc`. For information on building and running the OLE automation UDF and stored procedure samples, please see the `README` file in `sqllib\samples\ole`.

Related reference:

- "Object Linking and Embedding (OLE) Samples" on page 89

Building C/C++ Applications on Windows

DB2 provides batch files for compiling and linking DB2 API and embedded SQL C/C++ programs. These are located in the `sqllib\samples\c` and `sqllib\samples\cpp` directories, along with sample programs that can be built with these files.

The batch file, `bldapp.bat`, contains the commands to build DB2 API and embedded SQL programs. It takes up to four parameters, represented inside the batch file by the variables `%1`, `%2`, `%3`, and `%4`.

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 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.bat`. 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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

To build the DB2 API non-embedded SQL sample program, `cli_info`, from either the source file `cli_info.c`, in `sqllib\samples\c`, or from the source file `cli_info.cxx`, in `sqllib\samples\cpp`, enter:

```
bldapp cli_info
```

The result is an executable file, `cli_info.exe`. You can run the executable file by entering the executable name (without the extension) on the command line:

```
cli_info
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `tbmod`, from the C source file `tbmod.sqc` in `sqllib\samples\c`, or from the C++ source file `tbmod.sqx` in `sqllib\samples\cpp`:

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

The result is an executable file `tbmod.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:
   ```
   tbmod
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   tbmod database
   ```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   tbmod database userid password
   ```

Related concepts:

- "Build Files" on page 94

Related reference:

- "Windows C/C++ Application Compile and Link Options" on page 303
- "C/C++ Samples" on page 69

Related samples:

- "bldapp.bat -- Builds C applications on Windows"
- "cli_info.c -- Set and get information at the client level (C)"
- "embprep.bat -- Prep and binds a C/C++ or Micro Focus COBOL embedded SQL program on Windows"
- "tbmod.sqc -- How to modify table data (C)"
- "bldapp.bat -- Builds C++ applications on Windows"

- "cli_info.C -- Set and get information at the client level (C++)"
- "tbmod.sqC -- How to modify table data (C++)"

Batch File for C/C++ Applications

```
@echo off
rem BATCH FILE: bldapp.bat
rem Builds Windows Microsoft Visual C++ applications
rem Usage: bldapp prog_name [ db_name [ userid password ]]

rem Default compiler is set to Microsoft Visual C++
rem To use a different compiler, comment out "set BLDCOMP=cl"
rem and uncomment "set BLDCOMP=icl" or "set BLDCOMP=ecl"
rem Microsoft C/C++ Compiler
set BLDCOMP=cl

rem Intel C++ Compiler for 32-bit applications
rem set BLDCOMP=icl

rem Intel C++ Compiler for Itanium 64-bit applications
rem set BLDCOMP=ecl

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
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.c utilemb.c
goto link_embedded
:cpp_emb
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.cxx utilemb.cxx
rem Link the program.
:link_embedded
link -debug -out:%1.exe %1.obj utilemb.obj db2api.lib
goto exit

:non_embedded
rem  Compile the program.
if exist "%1.cxx" goto cpp_non
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.c utilapi.c
goto link_non_embedded
:cpp_non
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.cxx utilapi.cxx
rem Link the program.
:link_non_embedded
link -debug -out:%1.exe %1.obj utilapi.obj db2api.lib
:exit
@echo on
```

Windows C/C++ Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building C/C++ embedded SQL and DB2 API applications on Windows with the Microsoft Visual C++ compiler, as demonstrated in the `bldapp.bat` batch file.

Compile and Link Options for bldapp
Compile Options: **%BLDCOMP%** Variable for the compiler. The default is `cl`, the Microsoft Visual C++ compiler. It can be also set to `icl`, the Intel C++ Compiler for 32-bit applications, or `ecl`, the Intel C++ Compiler for Itanium 64-bit applications. **-Zi** Enable debugging information **-Od** Disable optimizations. It is easier to use a debugger with optimization off. **-c** Perform compile only; no link. The batch file has separate compile and link steps. **-W2** Output warning, error, and severe and unrecoverable error messages. **-DWIN32** Compiler option necessary for Windows operating systems.
Link Options: **link** Use the linker to link. **-debug** Include debugging information. **-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.

Related tasks:

- "Building C/C++ Applications on Windows" on page 300

Related samples:

- "bldapp.bat -- Builds C applications on Windows"
- "bldapp.bat -- Builds C++ applications on Windows"

Building C/C++ Routines on Windows

DB2 provides batch files for compiling and linking DB2 API and embedded SQL programs in C and C++. These are located in the `sqllib\samples\c` and `sqllib\samples\cpp` directories, along with sample programs that can be built with these files.

The batch file `bldrtn.bat` contains the commands to build embedded SQL routines (stored procedures and user-defined functions). The batch file builds a DLL on the server. It takes two parameters, represented inside the batch file by the variables `%1` and `%2`.

The first parameter, `%1`, specifies the name of your source file. The batch file uses the source file name for the DLL name. The second parameter, `%2`, specifies the name of the database to which you want to connect. Since the DLL 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. Database name is optional. If no database name is supplied, the program uses the default `sample` database.

Procedure:

The following examples show you how to build routine DLLs with:

- stored procedures
- non-embedded SQL user-defined functions (UDFs)
- embedded SQL user-defined functions (UDFs)

Stored Procedure DLL

To build the `spserver` DLL from either the C source file, `spserver.sqc`, or the C++ source file, `spserver.sqx`:

1. Enter the batch file name and program name:

   ```
   bldrtn spserver
   ```

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

   ```
   bldrtn 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 `sqllib\function`.

2. Next, catalog the routines by running the `spcat` script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling `spdrop.db2`, then catalogs them by calling `spcreate.db2`, and finally disconnects from the database. You can also call the `spdrop.db2` and `spcreate.db2` scripts individually.
3. Then, stop and restart the database to allow the new DLL to be recognized. If necessary, set the file mode for the DLL so the DB2 instance can access it.

Once you build the DLL, `spserver`, you can build the client application `spclient` that calls it.

You can build `spclient` by using the batch file, `bldapp.bat`.

To call the DLL, 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 for the user ID.

The client application accesses the DLL, `spserver`, and executes a number of routines on the server database. The output is returned to the client application.

Non-embedded SQL UDF DLL

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

```
bldrtn udfsrv
```

The batch file uses the module definition file, udfsrv.def, contained in the same directory as the sample program files, to build the user-defined function DLL. The batch file copies the user-defined function DLL, udfsrv.dll, to the server in the path sqllib\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 sqllib\samples\cli using the batch file bldapp.

You can build the embedded SQL C udfcli program from the udfcli.sqc source file in sqllib\samples\c using the batch file bldapp.

You can build the embedded SQL C++ udfcli program from the udfcli.sqx source file in sqllib\samples\cpp using the batch file bldapp.

To run the UDF, enter:

```
udfcli
```

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

Embedded SQL UDF DLL

To build the embedded SQL user-defined function library udfemsrv from the C source file udfemsrv.sqc in sqllib\samples\c, or from the C++ source file udfemsrv.sqx in sqllib\samples\cpp, enter:

```
bldrtn udfemsrv
```

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

```
bldrtn udfemsrv database
```

The batch file uses the module definition file, udfemsrv.def, contained in the same directory as the sample programs, to build the user-defined function DLL. The batch file copies the user-defined function DLL, udfemsrv.dll, to the server in the path sqllib\function.

Once you build udfemsrv, you can build the client application, udfemcli, that calls it. You can build udfemcli from the C source file udfemcli.sqc in sqllib\samples\c, or from the C++ source file udfemcli.sqx in sqllib\samples\cpp using the batch file bldapp.

To run the UDF, enter:

```
udfemcli
```

The calling application calls the UDFs in the `udfemsrv` DLL.

Related concepts:

- "Build Files" on page 94

Related reference:

- "Windows C/C++ Routine Compile and Link Options" on page 308
- "C/C++ Samples" on page 69

Related samples:

- "bldrtn.bat -- Builds C routines (stored procedures and UDFs) on Windows"
- "embprep.bat -- Prep and binds a C/C++ or Micro Focus COBOL embedded SQL program on Windows"
- "spcat -- To catalog C stored procedures on UNIX"
- "spclient.sqc -- Call various stored procedures (C)"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "
- "spserver.sqc -- A variety of types of stored procedures (C)"
- "udfcli.sqc -- Call a variety of types of user-defined functions (C)"
- "udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfemsrv.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfsrv.c -- Call a variety of types of user-defined functions (C)"
- "bldrtn.bat -- Builds C++ routines (stored procedures and UDFs) on Windows"
- "spclient.sqC -- Call various stored procedures (C++)"
- "spserver.sqC -- A variety of types of stored procedures (C++)"
- "udfcli.sqC -- Call a variety of types of user-defined functions (C++)"
- "udfemcli.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "udfemsrv.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "udfsrv.C -- Call a variety of types of user-defined functions (C++)"

Batch File for C/C++ Routines

```
@echo off
rem BATCH FILE: bldrtn.bat
rem Builds Windows Microsoft Visual C++ routines (stored procedures and UDFs)
```

```
rem Usage: bldrtn prog_name [ db_name ]

rem Default compiler is set to Microsoft Visual C++
rem To use a different compiler, comment out "set BLDCOMP=cl"
rem and uncomment "set BLDCOMP=icl" or "set BLDCOMP=ecl"
rem Microsoft C/C++ Compiler
set BLDCOMP=cl

rem Intel C++ Compiler for 32-bit applications
rem set BLDCOMP=icl

rem Intel C++ Compiler for Itanium 64-bit applications
rem set BLDCOMP=ecl

if exist "%1.sqc" goto embedded
if exist "%1.sqx" goto embedded
goto compile

:embedded
rem Precompile and bind the program.
call embprep %1 %2

:compile
rem Compile the program.
if exist "%1.cxx" goto cpp
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.c
goto link_step
:cpp
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.cxx

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

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

Windows C/C++ Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building C/C++ routines (stored procedures and user-defined functions) on Windows with the Microsoft Visual C++ compiler, as demonstrated in the `bldrtn.bat` batch file.

Compile and Link Options for bldrtn
Compile Options: **%BLDCOMP%** Variable for the compiler. The default is `cl`, the Microsoft Visual C++ compiler. It can be also set to `icl`, the Intel C++ Compiler for 32-bit applications, or `ecl`, the Intel C++ Compiler for Itanium 64-bit applications. **-Zi** Enable debugging information **-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. **-DWIN32** Compiler option necessary for Windows operating systems.
Link Options: **link** Use the linker to link. **-debug** Include debugging information. **-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.

Related tasks:

- "Building C/C++ Routines on Windows" on page 304

Related samples:

- "bldrtn.bat -- Builds C routines (stored procedures and UDFs) on Windows"
- "bldrtn.bat -- Builds C++ routines (stored procedures and UDFs) on Windows"

IBM VisualAge COBOL

Configuring the IBM COBOL Compiler on Windows

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the IBM VisualAge COBOL compiler, there are several points to keep in mind.

Procedure:

- 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.
- 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
  ```

  The `cobol_a` directory is the default.
- Specify COPY file names to include the `.cbl` extension as follows:

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

Related tasks:

- "Building IBM COBOL Applications on Windows" on page 311
- "Building IBM COBOL Routines on Windows" on page 314

Building IBM COBOL Applications on Windows

DB2 provides batch files for compiling and linking DB2 API and embedded SQL programs. These are located in the `sqllib\samples\cobol` directory, along with sample programs that can be built with these files.

The batch file, `bldapp.bat`, contains the commands to build a DB2 application program. It takes up to four parameters, represented inside the batch file by the variables `%1`, `%2`, `%3`, and `%4`.

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.bat`. 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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

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
```

Related concepts:

- "Build Files" on page 94

Related reference:

- "Windows IBM COBOL Application Compile and Link Options" on page 313
- "COBOL Samples" on page 80

Related samples:

- "bldapp.bat -- Builds Windows VisualAge COBOL applications"
- "client.cbl -- How to set and query a client (IBM COBOL)"
- "embprep.bat -- To prep and bind a COBOL embedded SQL program on Windows"
- "updat.sqb -- How to update, delete and insert table data (IBM COBOL)"

Batch File for IBM COBOL Applications

```
@echo off
rem BATCH FILE: bldapp.bat
rem Builds Windows VisualAge COBOL applications
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
```

Windows IBM COBOL Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL embedded SQL and DB2 API applications on Windows with the IBM VisualAge COBOL compiler, as demonstrated in the `bldapp.bat` batch file.

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 as a front-end for the linker **%1.obj** Include the program object file. **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.

Related tasks:

- "Building IBM COBOL Applications on Windows" on page 311

Related samples:

- "bldapp.bat -- Builds Windows VisualAge COBOL applications"

Building IBM COBOL Routines on Windows

DB2 provides batch files for compiling and linking DB2 API and embedded SQL programs in IBM COBOL. These are located in the `sqllib\samples\cobol` directory, along with sample programs that can be built with these files.

The batch file, `bldrtn.bat`, contains the commands to build embedded SQL routines (stored procedures). The batch file compiles the routines into a DLL on the server. It takes two parameters, represented inside the batch file by the variables `%1` and `%2`.

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

Procedure:

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

```
bldrtn outsrv
```

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

```
bldrtn outsrv database
```

The batch file copies the DLL to the server in the path `sqllib/function`.

Once you build the DLL `outsrv`, you can build the client application `outcli` that calls the routine within the DLL (which has the same name as the DLL). You can build `outcli` using the batch file `bldapp.bat`.

To call the `outsrv` routine, 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 for the user ID.

The client application accesses the DLL, `outsrv`, and executes the routine of the same name on the server database, and then returns the output to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "Windows IBM COBOL Routine Compile and Link Options" on page 316
- "COBOL Samples" on page 80

Related samples:

- "bldrtn.bat -- Builds Windows VisualAge COBOL routines (stored procedures)"
- "embprep.bat -- To prep and bind a COBOL embedded SQL program on Windows"
- "outcli.sqb -- Call stored procedures using the SQLDA structure (IBM COBOL)"
- "outsrv.sqb -- Demonstrates stored procedures using the SQLDA structure (IBM COBOL)"

Batch File for IBM COBOL Routines

```
@echo off
rem BATCH FILE: bldrtn.bat
rem Builds Windows VisualAge COBOL routines (stored procedures)
rem Usage: bldrtn <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
```

Windows IBM COBOL Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL routines (stored procedures and user-defined functions) on Windows with the IBM VisualAge COBOL compiler, as demonstrated in the `bldrtn.bat` batch file.

Compile and Link Options for bldrtn
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.

Related tasks:

- "Building IBM COBOL Routines on Windows" on page 314

Related samples:

- "bldrtn.bat -- Builds Windows VisualAge COBOL routines (stored procedures)"

Micro Focus COBOL

Configuring the Micro Focus COBOL Compiler on Windows

If you develop applications that contain embedded SQL and DB2 API calls, and you are using the Micro Focus compiler, there are several points to keep in mind.

Procedure:

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

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

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

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

You must make calls to all DB2 application programming interfaces 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.

Related tasks:

- “Building Micro Focus COBOL Applications on Windows” on page 317
- “Building Micro Focus COBOL Routines on Windows” on page 320

Building Micro Focus COBOL Applications on Windows

DB2 provides batch files for compiling and linking DB2 API and embedded SQL programs. These are located in the `sqllib\samples\cobol_mf` directory, along with sample programs that can be built with these files.

The batch file `bldapp.bat` contains the commands to build a DB2 application program. It takes up to four parameters, represented inside the batch file by the variables `%1`, `%2`, `%3`, and `%4`.

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.bat. 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.

Procedure:

The following examples show you how to build and run DB2 API and embedded SQL applications.

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
   ```

Related concepts:

- "Build Files" on page 94

Related reference:

- "Windows Micro Focus COBOL Application Compile and Link Options" on page 320
- "COBOL Samples" on page 80

Related samples:

- "bldapp.bat -- Builds Windows Micro Focus Cobol applications"
- "client.cbl -- How to set and query a client (MF COBOL)"
- "updat.sqb -- How to update, delete and insert table data (MF COBOL)"
- "embprep.bat -- Prep and binds a C/C++ or Micro Focus COBOL embedded SQL program on Windows"

Batch File for Micro Focus COBOL Applications

```
@echo off
rem BATCH FILE: bldapp.bat
rem Builds Windows Micro Focus Cobol applications
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
```

Windows Micro Focus COBOL Application Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL embedded SQL and DB2 API applications on Windows with the Micro Focus COBOL compiler, as demonstrated in the `bldapp.bat` batch file.

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.

Related tasks:

- "Building Micro Focus COBOL Applications on Windows" on page 317

Related samples:

- "bldapp.bat -- Builds Windows Micro Focus Cobol applications"

Building Micro Focus COBOL Routines on Windows

DB2 provides batch files for compiling and linking DB2 API and embedded SQL programs in Micro Focus COBOL. These are located in the `sqllib\samples\cobol_mf` directory, along with sample programs that can be built with these files.

The batch file `bldrtn.bat` contains the commands to build embedded SQL routines (stored procedures). The batch file compiles the routines into a DLL on the server. The batch file takes two parameters, represented inside the batch file by the variables `%1` and `%2`.

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

Procedure:

To build the sample program `outsrv` from the source file `outsrv.sqb`, if connecting to the sample database, enter:

```
bldrtn outsrv
```

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

```
bldrtn outsrv database
```

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

Once you build the DLL, `outsrv`, you can build the client application, `outcli`, that calls the routine within the DLL (which has the same name as the DLL). You can build `outcli` using the batch file, `bldapp.bat`.

To call the `outsrv` routine, 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 for the user ID.

The client application accesses the DLL, `outsrv`, which executes the routine of the same name on the server database. The output is then returned to the client application.

Related concepts:

- "Build Files" on page 94

Related reference:

- "Windows Micro Focus COBOL Routine Compile and Link Options" on page 322
- "COBOL Samples" on page 80

Related samples:

- "bldrtn.bat -- Builds Windows Micro Focus Cobol routines (stored procedures)"
- "outcli.sqb -- Call stored procedures using the SQLDA structure (MF COBOL)"
- "outsrv.sqb -- Demonstrates stored procedures using the SQLDA structure (MF COBOL)"
- "embprep.bat -- Prep and binds a C/C++ or Micro Focus COBOL embedded SQL program on Windows"

Batch File for Micro Focus COBOL Routines

```
@echo off
rem BATCH FILE: bldrtn.bat
rem Builds Windows Micro Focus Cobol routines (stored procedures)
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
```

Windows Micro Focus COBOL Routine Compile and Link Options

The following are the compile and link options recommended by DB2 for building COBOL routines (stored procedures and user-defined functions) on Windows with the Micro Focus COBOL compiler, as demonstrated in the `bldrtn.bat` batch file.

Compile and Link Options for bldrtn	
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.	

Related tasks:

- "Building Micro Focus COBOL Routines on Windows" on page 320

Related samples:

- "bldrtn.bat -- Builds Windows Micro Focus Cobol routines (stored procedures)"

Object REXX

Building Object REXX Applications on Windows

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, 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 `sqllib\samples\rexx`.

Procedure:

To run the sample REXX program `updat`, enter:

```
rexx updat.cmd
```

Related reference:

- "REXX Samples" on page 91

Part 4. Appendixes

Appendix A. DB2 Universal Database technical information

Overview of DB2 Universal Database technical information

DB2 Universal Database technical information can be obtained in the following formats:

- Books (PDF and hard-copy formats)
- A topic tree (HTML format)
- Help for DB2 tools (HTML format)
- Sample programs (HTML format)
- Command line help
- Tutorials

This section is an overview of the technical information that is provided and how you can access it.

Categories of DB2 technical information

The DB2 technical information is categorized by the following headings:

- Core DB2 information
- Administration information
- Application development information
- Business intelligence information
- DB2 Connect information
- Getting started information
- Tutorial information
- Optional component information
- Release notes

The following tables describe, for each book in the DB2 library, the information needed to order the hard copy, print or view the PDF, or locate the HTML directory for that book. A full description of each of the books in the DB2 library is available from the IBM Publications Center at www.ibm.com/shop/publications/order

The installation directory for the HTML documentation CD differs for each category of information:

htmlcdpath/doc/htmlcd/%L/*category*

where:

- *htmlcdpath* is the directory where the HTML CD is installed.
- *%L* is the language identifier. For example, en_US.
- *category* is the category identifier. For example, core for the core DB2 information.

In the PDF file name column in the following tables, the character in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d1e80` identifies the English version of the *Administration Guide: Planning* and the file name `db2d1g80` identifies the German 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**
Arabic	w
Brazilian Portuguese	b
Bulgarian	u
Croatian	9
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
Romanian	8
Russian	r
Simp. Chinese	c
Slovakian	7
Slovenian	l
Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

No form number indicates that the book is only available online and does not have a printed version.

Core DB2 information

The information in this category cover DB2 topics that are fundamental to all DB2 users. You will find the information in this category useful whether you are a programmer, a database administrator, or you work with DB2 Connect, DB2 Warehouse Manager, or other DB2 products.

The installation directory for this category is doc/htmlcd/%L/core.

Table 27. Core DB2 information

Name	Form Number	PDF File Name
IBM DB2 Universal Database Command Reference	SC09-4828	db2n0x80
IBM DB2 Universal Database Glossary	No form number	db2t0x80
IBM DB2 Universal Database Master Index	SC09-4839	db2w0x80
IBM DB2 Universal Database Message Reference, Volume 1	GC09-4840	db2m1x80
IBM DB2 Universal Database Message Reference, Volume 2	GC09-4841	db2m2x80
IBM DB2 Universal Database What's New	SC09-4848	db2q0x80

Administration information

The information in this category covers those topics required to effectively design, implement, and maintain DB2 databases, data warehouses, and federated systems.

The installation directory for this category is doc/htmlcd/%L/admin.

Table 28. Administration information

Name	Form number	PDF file name
IBM DB2 Universal Database Administration Guide: Planning	SC09-4822	db2d1x80
IBM DB2 Universal Database Administration Guide: Implementation	SC09-4820	db2d2x80
IBM DB2 Universal Database Administration Guide: Performance	SC09-4821	db2d3x80
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x80

Table 28. Administration information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx80
IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference	SC09-4831	db2hax80
IBM DB2 Universal Database Data Warehouse Center Administration Guide	SC27-1123	db2ddx80
IBM DB2 Universal Database Federated Systems Guide	GC27-1224	db2fpx80
IBM DB2 Universal Database Guide to GUI Tools for Administration and Development	SC09-4851	db2atx80
IBM DB2 Universal Database Replication Guide and Reference	SC27-1121	db2e0x80
IBM DB2 Installing and Administering a Satellite Environment	GC09-4823	db2dsx80
IBM DB2 Universal Database SQL Reference, Volume 1	SC09-4844	db2s1x80
IBM DB2 Universal Database SQL Reference, Volume 2	SC09-4845	db2s2x80
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x80

Application development information

The information in this category is of special interest to application developers or programmers working with DB2. You will find information about supported languages and compilers, as well as the documentation required to access DB2 using the various supported programming interfaces, such as embedded SQL, ODBC, JDBC, SQLj, and CLI. If you view this information online in HTML you can also access a set of DB2 sample programs in HTML.

The installation directory for this category is doc/htmlcd/%L/ad.

Table 29. Application development information

Name	Form number	PDF file name
IBM DB2 Universal Database Application Development Guide: Building and Running Applications	SC09-4825	db2axx80
IBM DB2 Universal Database Application Development Guide: Programming Client Applications	SC09-4826	db2a1x80
IBM DB2 Universal Database Application Development Guide: Programming Server Applications	SC09-4827	db2a2x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x80
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx80
IBM DB2 XML Extender Administration and Programming	SC27-1234	db2sxx80

Business intelligence information

The information in this category describes how to use components that enhance the data warehousing and analytical capabilities of DB2 Universal Database.

The installation directory for this category is doc/htmlcd/%L/wareh.

Table 30. Business intelligence information

Name	Form number	PDF file name
IBM DB2 Warehouse Manager Information Catalog Center Administration Guide	SC27-1125	db2dix80
IBM DB2 Warehouse Manager Installation Guide	GC27-1122	db2idx80

DB2 Connect information

The information in this category describes how to access host or iSeries data using DB2 Connect Enterprise Edition or DB2 Connect Personal Edition.

The installation directory for this category is doc/htmlcd/%L/conn.

Table 31. DB2 Connect information

Name	Form number	PDF file name
APPC, CPI-C, and SNA Sense Codes	No form number	db2apx80
IBM Connectivity Supplement	No form number	db2h1x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Enterprise Edition	GC09-4833	db2c6x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition	GC09-4834	db2c1x80
IBM DB2 Connect User's Guide	SC09-4835	db2c0x80

Getting started information

The information in this category is useful when you are installing and configuring servers, clients, and other DB2 products.

The installation directory for this category is doc/htmlcd/%L/start.

Table 32. Getting started information

Name	Form number	PDF file name
IBM DB2 Universal Database Quick Beginnings for DB2 Clients	GC09-4832	db2itx80
IBM DB2 Universal Database Quick Beginnings for DB2 Servers	GC09-4836	db2isx80
IBM DB2 Universal Database Quick Beginnings for DB2 Personal Edition	GC09-4838	db2i1x80
IBM DB2 Universal Database Installation and Configuration Supplement	GC09-4837	db2iyx80
IBM DB2 Universal Database Quick Beginnings for DB2 Data Links Manager	GC09-4829	db2z6x80

Tutorial information

Tutorial information introduces DB2 features and teaches how to perform various tasks.

The installation directory for this category is doc/htmlcd/%L/tutr.

Table 33. Tutorial information

Name	Form number	PDF file name
Business Intelligence Tutorial: Introduction to the Data Warehouse	No form number	db2tux80
Business Intelligence Tutorial: Extended Lessons in Data Warehousing	No form number	db2tax80
Development Center Tutorial for Video Online using Microsoft Visual Basic	No form number	db2tdx80
Information Catalog Center Tutorial	No form number	db2aix80
Video Central for e-business Tutorial	No form number	db2twx80
Visual Explain Tutorial	No form number	db2tvx80

Optional component information

The information in this category describes how to work with optional DB2 components.

The installation directory for this category is doc/htmlcd/%L/opt.

Table 34. Optional component information

Name	Form number	PDF file name
IBM DB2 Life Sciences Data Connect Planning, Installation, and Configuration Guide	GC27-1235	db2lsx80
IBM DB2 Spatial Extender User's Guide and Reference	SC27-1226	db2sbx80
IBM DB2 Universal Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x80

Table 34. Optional component information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Net Search Extender Administration and Programming Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6740	N/A

Release notes

The release notes provide additional information specific to your product's release and FixPak level. They also provides summaries of the documentation updates incorporated in each release and FixPak.

Table 35. Release notes

Name	Form number	PDF file name	HTML directory
DB2 Release Notes	See note.	See note.	doc/prodcd/%L/db2ir where %L is the language identifier.
DB2 Connect Release Notes	See note.	See note.	doc/prodcd/%L/db2cr where %L is the language identifier.
DB2 Installation Notes	Available on product CD-ROM only.	Available on product CD-ROM only.	

Note: The HTML version of the release notes 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/opt/db2_08_01 on AIX
  - /opt/IBM/db2/V8.1 on all other UNIX operating systems
- On other platforms, see the RELEASE.TXT file. This file is located in the directory where the product is installed.

Related tasks:

- "Printing DB2 books from PDF files" on page 335

- "Ordering printed DB2 books" on page 336
- "Accessing online help" on page 336
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 340
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 341

Printing DB2 books from PDF files

You can print DB2 books from the PDF files on the *DB2 PDF Documentation* CD. Using Adobe Acrobat Reader, you can print either the entire book or a specific range of pages.

Prerequisites:

Ensure that you have Adobe Acrobat Reader. It is available from the Adobe Web site at www.adobe.com

Procedure:

To print a DB2 book from a PDF file:

1. Insert the *DB2 PDF Documentation* CD. On UNIX operating systems, mount the DB2 PDF Documentation CD. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start Adobe Acrobat Reader.
3. Open the PDF file from one of the following locations:
   - On Windows operating systems:

     x:\doc*language* directory, where *x* represents the CD-ROM drive letter and *language* represents the two-character territory code that represents your language (for example, EN for English).
   - On UNIX operating systems:

     /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.

Related tasks:

- "Ordering printed DB2 books" on page 336
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 340
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 341

Related reference:

- "Overview of DB2 Universal Database technical information" on page 327

Ordering printed DB2 books

Procedure:

To order printed books:

- Contact your IBM authorized dealer or marketing representative. To find a local IBM representative, check the IBM Worldwide Directory of Contacts at www.ibm.com/shop/planetwide
- Phone `1-800-879-2755` in the United States or `1-800-IBM-4YOU` in Canada.
- Visit the IBM Publications Center at www.ibm.com/shop/publications/order

Related tasks:

- "Printing DB2 books from PDF files" on page 335
- "Finding topics by accessing the DB2 Information Center from a browser" on page 338
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 341

Related reference:

- "Overview of DB2 Universal Database technical information" on page 327

Accessing online help

The online help that comes with all DB2 components is available in three types:

- Window and notebook help
- Command line help
- SQL statement help

Window and notebook help explain the tasks that you can perform in a window or notebook and describe the controls. This help has two types:

- Help accessible from the **Help** button
- Infopops

The **Help** button gives you access to overview and prerequisite information. The infopops describe the controls in the window or notebook. Window and notebook help are available from DB2 centers and components that have user interfaces.

Command line help includes Command help and Message help. Command help explains the syntax of commands in the command line processor. Message help describes the cause of an error message and describes any action you should take in response to the error.

SQL statement help includes SQL help and SQLSTATE help. DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the syntax of SQL statements (SQL states and class codes).

Note: SQL help is not available for UNIX operating systems.

Procedure:

To access online help:

- For window and notebook help, click **Help** or click that control, then click **F1**. If the **Automatically display infopops** check box on the **General** page of the **Tool Settings** notebook is selected, you can also see the infopop for a particular control by holding the mouse cursor over the control.
- For command line help, open the command line processor and enter:
  - For Command help:

    ```
    ? 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.
- For Message help:

  ```
  ? XXXnnnnn
  ```

  where *XXXnnnnn* represents a valid message identifier.

  For example, `? SQL30081` displays help about the SQL30081 message.
- For SQL statement help, open the command line processor and enter:
  - For SQL help:

    ```
    ? 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.
  - For SQLSTATE help:

```
help statement
```

where *statement* represents an SQL statement.

For example, help SELECT displays help about the SELECT statement.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 338
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 341

Finding topics by accessing the DB2 Information Center from a browser

The DB2 Information Center accessed from a browser enables you to access the information you need to take full advantage of DB2 Universal Database and DB2 Connect. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, metadata, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser is composed of the following major elements:

Navigation tree
The navigation tree is located in the left frame of the browser window. The tree expands and collapses to show and hide topics, the glossary, and the master index in the DB2 Information Center.

Navigation toolbar
The navigation toolbar is located in the top right frame of the browser window. The navigation toolbar contains buttons that enable you to search the DB2 Information Center, hide the navigation tree, and find the currently displayed topic in the navigation tree.

Content frame
The content frame is located in the bottom right frame of the browser window. The content frame displays topics from the DB2 Information Center when you click on a link in the navigation tree, click on a search result, or follow a link from another topic or from the master index.

Prerequisites:

To access the DB2 Information Center from a browser, you must use one of the following browsers:

- Microsoft Explorer, version 5 or later
- Netscape Navigator, version 6.1 or later

Restrictions:

The DB2 Information Center contains only those sets of topics that you chose to install from the *DB2 HTML Documentation CD*. If your Web browser returns a `File not found` error when you try to follow a link to a topic, you must install one or more additional sets of topics *DB2 HTML Documentation CD*.

Procedure:

To find a topic by searching with keywords:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

To find a topic in the navigation tree:

1. In the navigation tree, click the book icon of the category of topics related to your area of interest. A list of subcategories displays underneath the icon.
2. Continue to click the book icons until you find the category containing the topics in which you are interested. Categories that link to topics display the category title as an underscored link when you move the cursor over the category title. The navigation tree identifies topics with a page icon.
3. Click the topic link. The topic displays in the content frame.

To find a topic or term in the master index:

1. In the navigation tree, click the “Index” category. The category expands to display a list of links arranged in alphabetical order in the navigation tree.
2. In the navigation tree, click the link corresponding to the first character of the term relating to the topic in which you are interested. A list of terms with that initial character displays in the content frame. Terms that have multiple index entries are identified by a book icon.
3. Click the book icon corresponding to the term in which you are interested. A list of subterms and topics displays below the term you clicked. Topics are identified by page icons with an underscored title.
4. Click on the title of the topic that meets your needs. The topic displays in the content frame.

Related concepts:

- "Accessibility" on page 347
- "DB2 Information Center for topics" on page 349

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 340
- "Updating the HTML documentation installed on your machine" on page 342
- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 344
- "Searching the DB2 documentation" on page 345

Related reference:

- "Overview of DB2 Universal Database technical information" on page 327

Finding product information by accessing the DB2 Information Center from the administration tools

The DB2 Information Center provides quick access to DB2 product information and is available on all operating systems for which the DB2 administration tools are available.

The DB2 Information Center accessed from the tools provides six types of information.

Tasks Key tasks you can perform using DB2.

Concepts
Key concepts for DB2.

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

Troubleshooting
Error messages and information to help you with common DB2 problems.

Samples
Links to HTML listings of the sample programs provided with DB2.

Tutorials
Instructional aid designed to help you learn a DB2 feature.

Prerequisites:

Some links in the DB2 Information Center point to Web sites on the Internet. To display the content for these links, you will first have to connect to the Internet.

Procedure:

To find product information by accessing the DB2 Information Center from the tools:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Click the tab of the information type related to the information you are attempting to find.
3. Navigate through the tree and click on the topic in which you are interested. The Information Center will then launch a Web browser to display the information.
4. To find information without browsing the lists, click the **Search** icon to the right of the list.

   Once the Information Center has launched a browser to display the information, you can perform a full-text search by clicking the **Search** icon in the navigation toolbar.

Related concepts:

- "Accessibility" on page 347
- "DB2 Information Center for topics" on page 349

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 338
- "Searching the DB2 documentation" on page 345

Viewing technical documentation online directly from the DB2 HTML Documentation CD

All of the HTML topics that you can install from the *DB2 HTML Documentation CD* can also be read directly from the CD. Therefore, you can view the documentation without having to install it.

Restrictions:

Because the following items are installed from the DB2 product CD and not the *DB2 HTML Documentation CD*, you must install the DB2 product to view these items:

- Tools help
- DB2 Quick Tour
- Release notes

Procedure:

1. Insert the *DB2 HTML Documentation* CD. On UNIX operating systems, mount the *DB2 HTML Documentation CD*. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start your HTML browser and open the appropriate file:
   - For Windows operating systems:

     ```
     e:\Program Files\sqllib\doc\htmlcd\%L\index.htm
     ```

     where *e* represents the CD-ROM drive, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.
   - For UNIX operating systems:

     ```
     /cdrom/Program Files/sqllib/doc/htmlcd/%L/index.htm
     ```

     where */cdrom/* represents where the CD is mounted, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 338
- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 344

Related reference:

- "Overview of DB2 Universal Database technical information" on page 327

Updating the HTML documentation installed on your machine

It is now possible to update the HTML installed from the *DB2 HTML Documentation CD* when updates are made available from IBM. This can be done in one of two ways:

- Using the Information Center (if you have the DB2 administration GUI tools installed).
- By downloading and applying a DB2 HTML documentation FixPak .

Note: This will NOT update the DB2 code; it will only update the HTML documentation installed from the *DB2 HTML Documentation CD*.

Procedure:

To use the Information Center to update your local documentation:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Ensure your machine has access to the external Internet; the updater will download the latest documentation FixPak from the IBM server if required.
3. Select **Information Center —> Update Local Documentation** from the menu to start the update.
4. Supply your proxy information (if required) to connect to the external Internet.

The Information Center will download and apply the latest documentation FixPak, if one is available.

To manually download and apply the documentation FixPak :

1. Ensure your machine is connected to the Internet.
2. Open the DB2 support page in your Web browser at: www.ibm.com/software/data/db2/udb/winos2unix/support.
3. Follow the link for version 8 and look for the ″Documentation FixPaks″ link.
4. Determine if the version of your local documentation is out of date by comparing the documentation FixPak level to the documentation level you have installed. This current documentation on your machine is at the following level: **DB2 v8.1 GA**.
5. If there is a more recent version of the documentation available then download the FixPak applicable to your operating system. There is one FixPak for all Windows platforms, and one FixPak for all UNIX platforms.
6. Apply the FixPak:
   - For Windows operating systems: The documentation FixPak is a self extracting zip file. Place the downloaded documentation FixPak in an empty directory, and run it. It will create a **setup** command which you can run to install the documentation FixPak.
   - For UNIX operating systems: The documentation FixPak is a compressed tar.Z file. Uncompress and untar the file. It will create a directory named `delta_install` with a script called **installdocfix**. Run this script to install the documentation FixPak.

Related tasks:

- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 344

Related reference:

- "Overview of DB2 Universal Database technical information" on page 327

Copying files from the DB2 HTML Documentation CD to a Web Server

The entire DB2 information library is delivered to you on the *DB2 HTML Documentation CD*, so you can install the library on a Web server for easier access. Simply copy to your Web server the documentation for the languages that you want.

Procedure:

To copy files from the *DB2 HTML Documentation CD* to a Web server, use the appropriate path:

- For Windows operating systems:

  ```
  E:\Program Files\sqllib\doc\htmlcd\%L\*.*
  ```

  where *E* represents the CD-ROM drive and *%L* represents the language identifier.
- For UNIX operating systems:

  ```
  /cdrom:Program Files/sqllib/doc/htmlcd/%L/*.*
  ```

  where *cdrom* represents the CD-ROM drive and *%L* represents the language identifier.

Related tasks:

- "Searching the DB2 documentation" on page 345

Related reference:

- "Supported DB2 interface languages, locales, and code pages" in the *Quick Beginnings for DB2 Servers*
- "Overview of DB2 Universal Database technical information" on page 327

Troubleshooting DB2 documentation search with Netscape 4.x

Most search problems are related to the Java support provided by web browsers. This task describes possible workarounds.

Procedure:

A common problem with Netscape 4.x involves a missing or misplaced security class. Try the following workaround, especially if you see the following line in the browser Java console:

```
Cannot find class  java/security/InvalidParameterException
```

- On Windows operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied *x*:Program Files\sqllib\doc\htmlcd*locale*\InvalidParameterException.class file to the java\classes\java\security\ directory relative to your Netscape browser installation, where *x* represents the CD-ROM drive letter and *locale* represents the name of the desired locale.

  Note: You may have to create the java\security\ subdirectory structure.

- On UNIX operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied */cdrom*/Program Files/sqllib/doc/htmlcd/*locale*/InvalidParameterException.class file to the java/classes/java/security/ directory relative to your Netscape browser installation, where *cdrom* represents the mount point of the CD-ROM and *locale* represents the name of the desired locale.

  Note: You may have to create the java/security/ subdirectory structure.

If your Netscape browser still fails to display the search input window, try the following:

- Stop all instances of Netscape browsers to ensure that there is no Netscape code running on the machine. Then open a new instance of the Netscape browser and try to start the search again.
- Purge the browser's cache.
- Try a different version of Netscape, or a different browser.

Related tasks:

- "Searching the DB2 documentation" on page 345

Searching the DB2 documentation

To search DB2's documentation, you need Netscape 6.1 or higher, or Microsoft's Internet Explorer 5 or higher. Ensure that your browser's Java support is enabled.

A pop-up search window opens when you click the search icon in the navigation toolbar of the Information Center accessed from a browser. If you are using the search for the first time it may take a minute or so to load into the search window.

Restrictions:

The following restrictions apply when you use the documentation search:

- Boolean searches are not supported. The boolean search qualifiers *and* and *or* will be ignored in a search. For example, the following searches would produce the same results:
  - servlets *and* beans
  - servlets *or* beans
- Wildcard searches are not supported. A search on *java** will only look for the literal string *java** and would not, for example, find *javadoc*.

In general, you will get better search results if you search for phrases instead of single words.

Procedure:

To search the DB2 documentation:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

Note: When you perform a search, the first result is automatically loaded into your browser frame. To view the contents of other search results, click on the result in results lists.

Related tasks:

- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 344

Online DB2 troubleshooting information

With the release of DB2® UDB Version 8, there will no longer be a *Troubleshooting Guide*. The troubleshooting information once contained in this guide has been integrated into the DB2 publications. By doing this, we are able to deliver the most up-to-date information possible. To find information on the troubleshooting utilities and functions of DB2, access the DB2 Information Center from any of the tools.

Refer to the DB2 Online Support site if you are experiencing problems and want help finding possible causes and solutions. The support site contains a

large, constantly updated database of DB2 publications, TechNotes, APAR (product problem) records, FixPaks, and other resources. You can use the support site to search through this knowledge base and find possible solutions to your problems.

Access the Online Support site at www.ibm.com/software/data/db2/udb/winos2unix/support, or by clicking the **Online Support** button in the DB2 Information Center. Frequently changing information, such as the listing of internal DB2 error codes, is now also available from this site.

Related concepts:

- "DB2 Information Center for topics" on page 349

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 340

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features in DB2® Universal Database Version 8:

- DB2 allows you to operate all features using the keyboard instead of the mouse. See "Keyboard Input and Navigation".
- DB2 enables you customize the size and color of your fonts. See "Accessible Display" on page 348.
- DB2 allows you to receive either visual or audio alert cues. See "Alternative Alert Cues" on page 348.
- DB2 supports accessibility applications that use the Java™ Accessibility API. See "Compatibility with Assistive Technologies" on page 348.
- DB2 comes with documentation that is provided in an accessible format. See "Accessible Documentation" on page 348.

Keyboard Input and Navigation

Keyboard Input

You can operate the DB2 Tools using only the keyboard. You can use keys or key combinations to perform most operations that can also be done using a mouse.

Keyboard Focus

In UNIX-based systems, the position of the keyboard focus is highlighted, indicating which area of the window is active and where your keystrokes will have an effect.

Accessible Display

The DB2 Tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font Settings

The DB2 Tools allow you to select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

Non-dependence on Color

You do not need to distinguish between colors in order to use any of the functions in this product.

Alternative Alert Cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with Assistive Technologies

The DB2 Tools interface supports the Java Accessibility API enabling use by screen readers and other assistive technologies used by people with disabilities.

Accessible Documentation

Documentation for the DB2 family of products is available in HTML format. This allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

DB2 tutorials

The DB2® tutorials help you learn about various aspects of DB2 Universal Database. The tutorials provide lessons with step-by-step instructions in the areas of developing applications, tuning SQL query performance, working with data warehouses, managing metadata, and developing Web services using DB2.

Before you begin:

Before you can access these tutorials using the links below, you must install the tutorials from the *DB2 HTML Documentation* CD-ROM.

If you do not want to install the tutorials, you can view the HTML versions of the tutorials directly from the *DB2 HTML Documentation CD*. PDF versions of these tutorials are also available on the *DB2 PDF Documentation CD*.

Some tutorial lessons use sample data or code. See each individual tutorial for a description of any prerequisites for its specific tasks.

DB2 Universal Database tutorials:

If you installed the tutorials from the *DB2 HTML Documentation* CD-ROM, you can click on a tutorial title in the following list to view that tutorial.

Business Intelligence Tutorial: Introduction to the Data Warehouse Center
Perform introductory data warehousing tasks using the Data Warehouse Center.

Business Intelligence Tutorial: Extended Lessons in Data Warehousing
Perform advanced data warehousing tasks using the Data Warehouse Center.

Development Center Tutorial for Video Online using Microsoft® Visual Basic
Build various components of an application using the Development Center Add-in for Microsoft Visual Basic.

Information Catalog Center Tutorial
Create and manage an information catalog to locate and use metadata using the Information Catalog Center.

Video Central for e-business Tutorial
Develop and deploy an advanced DB2 Web Services application using WebSphere® products.

Visual Explain Tutorial
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 Information Center for topics

The DB2® Information Center gives you access to all of the information you need to take full advantage of DB2 Universal Database™ and DB2 Connect™ in your business. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, the Information Catalog Center, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser has the following features:

Regularly updated documentation
Keep your topics up-to-date by downloading updated HTML.

Search

Search all of the topics installed on your workstation by clicking **Search** in the navigation toolbar.

Integrated navigation tree

Locate any topic in the DB2 library from a single navigation tree. The navigation tree is organized by information type as follows:

- Tasks provide step-by-step instructions on how to complete a goal.
- Concepts provide an overview of a subject.
- Reference topics provide detailed information about a subject, including statement and command syntax, message help, requirements.

Master index

Access the information in topics and tools help from one master index. The index is organized in alphabetical order by index term.

Master glossary

The master glossary defines terms used in the DB2 Information Center. The glossary is organized in alphabetical order by glossary term.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 338
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 340
- "Updating the HTML documentation installed on your machine" on page 342

Appendix B. 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/region 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/region 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 that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
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 document 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 illustrate 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 are trademarks of International Business Machines Corporation in the United States, other countries, or both, and have been used in at least one of the documents in the DB2 UDB documentation library.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 UDB documentation library:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-237-5511 for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, FixPaks, news, and links to web resources.

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) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

Part Number: CT17WNA

Printed in U.S.A.

(1P) P/N: CT17WNA

SC09-4825-00

Spine information:

IBM® DB2 Universal Database™ Building and Running Applications Version 8