Building MySQL from Source

Building MySQL from Source

Abstract

This is the Building MySQL from Source extract from the MySQL 5.0 Reference Manual.

Document generated on: 2009-06-02 (revision: 15165)

MySQL Installation Using a Source Distribution

Before you proceed with an installation from source, first check whether our binary is available for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options.

To obtain a source distribution for MySQL, How to Get MySQL. If you want to build MySQL from source on Windows, see Chapter 5, *Installing MySQL from Source on Windows*.

MySQL source distributions are provided as compressed `tar` archives and have names of the form `mysql-VERSION.tar.gz`, where `VERSION` is a number like `5.0.84`.

You need the following tools to build and install MySQL from source:

- GNU `gunzip` to uncompress the distribution.
- A reasonable `tar` to unpack the distribution. GNU `tar` is known to work. Some operating systems come with a preinstalled version of `tar` that is known to have problems. For example, the `tar` provided with early versions of Mac OS X `tar`, SunOS 4.x and Solaris 8 and earlier are known to have problems with long file names. On Mac OS X, you can use the preinstalled `gnutar` program. On other systems with a deficient `tar`, you should install GNU `tar` first.
- A working ANSI C++ compiler. `gcc` 2.95.2 or later, SGI C++, and SunPro C++ are some of the compilers that are known to work. `libg++` is not needed when using `gcc`. `gcc` 2.7.x has a bug that makes it impossible to compile some perfectly legal C++ files, such as `sql/sql_base.cc`. If you have only `gcc` 2.7.x, you must upgrade your `gcc` to be able to compile MySQL. `gcc` 2.8.1 is also known to have problems on some platforms, so it should be avoided if a newer compiler exists for the platform. `gcc` 2.95.2 or later is recommended.
- A good `make` program. GNU `make` is always recommended and is sometimes required. (BSD `make` fails, and vendor-provided `make` implementations may fail as well.) If you have problems, we recommend GNU `make` 3.75 or newer.
- `libtool` 1.5.24 or later is also recommended.

If you are using a version of `gcc` recent enough to understand the `-fno-exceptions` option, it is *very important* that you use this option. Otherwise, you may compile a binary that crashes randomly. We also recommend that you use `-felide-constructors` and `-fno-rtti` along with `-fno-exceptions`. When in doubt, do the following:

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \
       -fno-exceptions -fno-rtti" ./configure \
       --prefix=/usr/local/mysql --enable-assembler \
       --with-mysqld-ldflags=-all-static
```

On most systems, this gives you a fast and stable binary.

If you run into problems and need to file a bug report, please use the instructions in How to Report Bugs or Problems.

Chapter 1. Source Installation Overview

The basic commands that you must execute to install a MySQL source distribution are:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> gunzip < mysql-VERSION.tar.gz | tar -xvf -
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> cd /usr/local/mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> bin/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql var
shell> bin/mysqld_safe --user=mysql &
```

If you start from a source RPM, do the following:

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

This makes a binary RPM that you can install. For older versions of RPM, you may have to replace the command `rpmbuild` with `rpm` instead.

> **Note**
>
> This procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to Post-Installation Setup and Testing, for post-installation setup and testing.

A more detailed version of the preceding description for installing MySQL from a source distribution follows:

1. Add a login user and group for `mysqld` to run as:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql mysql
   ```

   These commands add the `mysql` group and the `mysql` user. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

   You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following steps.

2. Perform the following steps as the `mysql` user, except as noted.

3. Pick the directory under which you want to unpack the distribution and change location into it.

4. Obtain a distribution file using the instructions in How to Get MySQL.

5. Unpack the distribution into the current directory:

   ```
   shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named `mysql-VERSION`.

   With GNU `tar`, no separate invocation of `gunzip` is necessary. You can use the following alternative command to uncompress and extract the distribution:

   ```
   shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
   ```

6. Change location into the top-level directory of the unpacked distribution:

   ```
   shell> cd mysql-VERSION
   ```

   Note that currently you must configure and build MySQL from this top-level directory. You cannot build it in a different directory.

7. Configure the release and compile everything:

```
shell> ./configure --prefix=/usr/local/mysql
shell> make
```

When you run `configure`, you might want to specify other options. Run `./configure --help` for a list of options. Chapter 2, *Typical `configure` Options*, discusses some of the more useful options.

If `configure` fails and you are going to send mail to a MySQL mailing list to ask for assistance, please include any lines from `config.log` that you think can help solve the problem. Also include the last couple of lines of output from `configure`. To file a bug report, please use the instructions in How to Report Bugs or Problems.

If the compile fails, see Chapter 4, *Dealing with Problems Compiling MySQL*, for help.

8. Install the distribution:

```
shell> make install
```

You might need to run this command as `root`.

If you want to set up an option file, use one of those present in the `support-files` directory as a template. For example:

```
shell> cp support-files/my-medium.cnf /etc/my.cnf
```

You might need to run this command as `root`.

If you want to configure support for `InnoDB` tables, you should edit the `/etc/my.cnf` file, remove the # character before the option lines that start with `innodb_...`, and modify the option values to be what you want. See Using Option Files, and `InnoDB` Configuration.

9. Change location into the installation directory:

```
shell> cd /usr/local/mysql
```

10. If you ran the `make install` command as `root`, the installed files will be owned by `root`. Ensure that the installation is accessible to `mysql` by executing the following commands as `root` in the installation directory:

```
shell> chown -R mysql .
shell> chgrp -R mysql .
```

The first command changes the owner attribute of the files to the `mysql` user. The second changes the group attribute to the `mysql` group.

11. If you have not installed MySQL before, you must create the MySQL data directory and initialize the grant tables:

```
shell> bin/mysql_install_db --user=mysql
```

If you run the command as `root`, include the `--user` option as shown. If you run the command while logged in as `mysql`, you can omit the `--user` option.

The command should create the data directory and its contents with `mysql` as the owner.

After using `mysql_install_db` to create the grant tables for MySQL, you must restart the server manually. The `mysqld_safe` command to do this is shown in a later step.

12. Most of the MySQL installation can be owned by `root` if you like. The exception is that the data directory must be owned by `mysql`. To accomplish this, run the following commands as `root` in the installation directory:

```
shell> chown -R root .
shell> chown -R mysql var
```

13. If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself; see also Starting and Stopping MySQL Automatically.

14. You can set up new accounts using the `bin/mysql_setpermission` script if you install the `DBI` and `DBD::mysql` Perl modules. See `mysql_setpermission`. For Perl module installation instructions, see Perl Installation Notes.

After everything has been installed, you should test your distribution. To start the MySQL server, use the following command:

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

If you run the command as `root`, you should use the `--user` option as shown. The value of the option is the name of the login account that you created in the first step to use for running the server. If you run the command while logged in as that user, you can omit the `--user` option.

If the command fails immediately and prints `mysqld ended`, you can find some information in the *`host_name`*`.err` file in the data directory.

More information about `mysqld_safe` is given in `mysqld_safe`.

> **Note**
>
> The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Post-Installation Setup and Testing.

Chapter 2. Typical `configure` Options

The `configure` script gives you a great deal of control over how you configure a MySQL source distribution. Typically you do this using options on the `configure` command line. You can also affect `configure` using certain environment variables. See Environment Variables. For a full list of options supported by `configure`, run this command:

```
shell> ./configure --help
```

A list of the available `configure` options is provided in the table below.

Table 2.1. Build (`configure`) Reference

Formats	Description	Default	Introduced	Removed
--bindir=DIR	User executables	EPREFIX/bin		
--build=BUILD	Configure for building on BUILD	guessed		
--cache-file=FILE	Cache test results in FILE	disabled		
-C	Alias for `--cache-file=config.cache'			
--config-cache				
--datadir=DIR	Read-only architecture-independent data	PREFIX/share		
--disable-FEATURE	Do not include FEATURE			
--disable-dependency-tracking	Disable dependency tracking			
--disable-grant-options	Disable GRANT options		5.0.34	
--disable-largefile	Omit support for large files			
--disable-libtool-lock	Disable libtool lock			
--disable-profiling	Build a version without query profiling code		5.0.37	5.0.45
--enable-FEATURE	Enable FEATURE			
--enable-assembler	Use assembler versions of some string functions if available			
--enable-dependency-tracking	Do not reject slow dependency extractors			
--enable-fast-install	Optimize for fast installation	yes		
--enable-local-infile	Enable LOAD DATA LOCAL INFILE	disabled		
--enable-shared	Build shared libraries	yes		
--enable-static	Build static libraries	yes		
--enable-thread-safe-client	Compile the client with threads			
--exec-prefix=EPREFIX	Install architecture-dependent files in EPREFIX			
-h	Display this help and exit			
--help				
--help=short	Display options specific to this package			
--help=recursive	Display the short help of all the included packages			
--host=HOST	Cross-compile to build programs to run on HOST			
--includedir=DIR	C header files	PREFIX/include		
--infodir=DIR	Info documentation	PREFIX/info		
--libdir=DIR	Object code libraries	EPREFIX/lib		
--libexecdir=DIR	Program executables	EPREFIX/libexec		
--localstatedir=DIR	Modifiable single-machine data	PREFIX/var		
--mandir=DIR	man documentation	PREFIX/man		

Formats	Description	Default	Introduced	Removed
-n	Do not create output files			
--no-create				
--oldincludedir=DIR	C header files for non-gcc	/usr/include		
--prefix=PREFIX	Install architecture-independent files in PREFIX			
--program-prefix=PREFIX	Prepend PREFIX to installed program names			
--program-suffix=SUFFIX	Append SUFFIX to installed program names			
- -pro- gram-transform-name=PROGRAM	run sed PROGRAM on installed program names			
-q	Do not print `checking...' messages			
--quiet				
--sbindir=DIR	System admin executables	EPREFIX/sbin		
--sharedstatedir=DIR	Modifiable architecture-independent data	PREFIX/com		
--srcdir=DIR	Find the sources in DIR	configure directory or ..		
--sysconfdir=DIR	Read-only single-machine data	PREFIX/etc		
--target=TARGET	Configure for building compilers for TARGET			
-V	Display version information and exit			
--version				
--with-PACKAGE	Use PACKAGE			
--with-archive-storage-engine	Enable the Archive Storage Engine	no		
--with-berkeley-db	Use BerkeleyDB located in DIR	no		
--with-berkeley-db-includes	Find Berkeley DB headers in DIR			
--with-berkeley-db-libs	Find Berkeley DB libraries in DIR			
--with-big-tables	Support tables with more than 4 G rows even on 32 bit platforms		5.0.4	
--with-blackhole-storage-engine	Enable the Blackhole Storage Engine	no	5.0.4	
--with-charset	Default character set			
--with-client-ldflags	Extra linking arguments for clients			
--with-collation	Default collation			
--with-comment	Comment about compilation environment			
--with-csv-storage-engine	Enable the CSV Storage Engine	yes		
--with-darwin-mwcc	Use Metrowerks CodeWarrior wrappers on OS X/Darwin		5.0.6	
--with-embedded-privilege-control	Build parts to check user's privileges (only affects embedded library)			
--with-embedded-server	Build the embedded server			
--with-example-storage-engine	Enable the Example Storage Engine	no		
--with-extra-charsets	Use charsets in addition to default			
--with-gnu-ld	Assume the C compiler uses GNU ld	no		
--with-isam	Enable the ISAM table type			
--with-lib-ccflags	Extra CC options for libraries			
--with-libwrap=DIR	Compile in libwrap (tcp_wrappers) support			
--with-low-memory	Try to use less memory to compile to avoid memory limitations			
--with-machine-type	Set the machine type, like "powerpc"		5.0.44	

Formats	Description	Default	Introduced	Removed
--with-max-indexes=N	Sets the maximum number of indexes per table	64		
--with-mit-threads	Always use included thread lib			
--with-mysqld-ldflags	Extra linking arguments for mysqld			
--with-mysqld-libs	Extra libraries to link with for mysqld		5.0.44	
--with-mysqld-user	What user the mysqld daemon shall be run as			
--with-mysqlfs	Include the corba-based MySQL file system			
--with-mysqlmanager	Build the mysqlmanager binary	Build if server is built		
--with-named-curses-libs	Use specified curses libraries			
--with-named-thread-libs	Use specified thread libraries			
--with-ndb-ccflags	Extra CC options for ndb compile		5.0.3	
--with-ndb-docs	Include the NDB Cluster ndbapi and mgmapi documentation			
--with-ndb-port	Port for NDB Cluster management server			
--with-ndb-port-base	Port for NDB Cluster management server		5.0.3	
--with-ndb-sci=DIR	Provide MySQL with a custom location of sci library			
--with-ndb-shm	Include the NDB Cluster shared memory transporter			
--with-ndb-test	Include the NDB Cluster ndbapi test programs			
--with-ndbcluster	Include the NDB Cluster table handler	no		
--with-openssl=DIR	Include the OpenSSL support			
--with-openssl-includes	Find OpenSSL headers in DIR			
--with-openssl-libs	Find OpenSSL libraries in DIR			
--with-other-libc=DIR	Link against libc and other standard libraries installed in the specified non-standard location			
--with-pic	Try to use only PIC/non-PIC objects	Use both		
--with-pstack	Use the pstack backtrace library			
--with-pthread	Force use of pthread library			
--with-raid	Enable RAID Support			
--with-server-suffix	Append value to the version string			
--with-system-type	Set the system type, like "sun-solaris10"		5.0.44	
--with-tags	Include additional configurations	automatic		
--with-tcp-port	Which port to use for MySQL services	3306		
--with-unix-socket-path	Where to put the unix-domain socket			
--with-vio	Include the Virtual IO support			
--with-yassl	Include the yaSSL support		5.0.6	
--with-zlib-dir=no|bundled|DIR	Provide MySQL with a custom location of compression library			
--without-PACKAGE	Do not use PACKAGE			
--without-bench	Skip building of the benchmark suite			
--without-debug	Build a production version without debugging code			
--without-docs	Skip building of the documentation			

Formats	Description	Default	Introduced	Removed
--without-extra-tools	Skip building utilities in the tools directory			
--without-geometry	Do not build geometry-related parts			
--without-innodb	Do not include the InnoDB table handler			
--without-libedit	Use system libedit instead of bundled copy			
--without-man	Skip building of the man pages			
--without-ndb-debug	Disable special ndb debug features		5.0.3	
--without-query-cache	Do not build query cache			
--without-readline	Use system readline instead of bundled copy			
--without-server	Only build the client			
--without-uca	Skip building of the national Unicode collations		5.0.3	

Some of the configure options available are described here. For options that may be of use if you have difficulties building MySQL, see Chapter 4, *Dealing with Problems Compiling MySQL*.

- To compile just the MySQL client libraries and client programs and not the server, use the --without-server option:

```
shell> ./configure --without-server
```

  If you have no C++ compiler, some client programs such as mysql cannot be compiled because they require C++.. In this case, you can remove the code in configure that tests for the C++ compiler and then run ./configure with the --without-server option. The compile step should still try to build all clients, but you can ignore any warnings about files such as mysql.cc. (If make stops, try make -k to tell it to continue with the rest of the build even if errors occur.)

- If you want to build the embedded MySQL library (libmysqld.a), use the --with-embedded-server option.

- If you don't want your log files and database directories located under /usr/local/var, use a configure command something like one of these:

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
          --localstatedir=/usr/local/mysql/data
```

  The first command changes the installation prefix so that everything is installed under /usr/local/mysql rather than the default of /usr/local. The second command preserves the default installation prefix, but overrides the default location for database directories (normally /usr/local/var) and changes it to /usr/local/mysql/data.

  You can also specify the installation directory and data directory locations at server startup time by using the --basedir and --datadir options. These can be given on the command line or in an MySQL option file, although it is more common to use an option file. See Using Option Files.

- If you are using Unix and you want the MySQL socket file location to be somewhere other than the default location (normally in the directory /tmp or /var/run), use a configure command like this:

```
shell> ./configure \
          --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

  The socket file name must be an absolute path name. You can also change the location of mysql.sock at server startup by using a MySQL option file. See How to Protect or Change the MySQL Unix Socket File.

- If you want to compile statically linked programs (for example, to make a binary distribution, to get better performance, or to work around problems with some Red Hat Linux distributions), run configure like this:

```
shell> ./configure --with-client-ldflags=-all-static \
          --with-mysqld-ldflags=-all-static
```

- If you are using gcc and don't have libg++ or libstdc++ installed, you can tell configure to use gcc as your C++ compiler:

```
shell> CC=gcc CXX=gcc ./configure
```

When you use gcc as your C++ compiler, it does not attempt to link in libg++ or libstdc++. This may be a good thing to do even if you have those libraries installed. Some versions of them have caused strange problems for MySQL users in the past.

The following list indicates some compilers and environment variable settings that are commonly used with each one.

- gcc 2.7.2:

  ```
  CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
  ```

- gcc 2.95.2:

  ```
  CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
  -felide-constructors -fno-exceptions -fno-rtti"
  ```

- pgcc 2.90.29 or newer:

  ```
  CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc \
  CXXFLAGS="-O3 -mpentiumpro -mstack-align-double \
  -felide-constructors -fno-exceptions -fno-rtti"
  ```

In most cases, you can get a reasonably optimized MySQL binary by using the options from the preceding list and adding the following options to the configure line:

```
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The full configure line would, in other words, be something like the following for all recent gcc versions:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti" ./configure \
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The binaries we provide on the MySQL Web site at http://dev.mysql.com/downloads/ are all compiled with full optimization and should be perfect for most users. See MySQL Binaries Compiled by MySQL AB. There are some configuration settings you can tweak to build an even faster binary, but these are only for advanced users. See How Compiling and Linking Affects the Speed of MySQL.

If the build fails and produces errors about your compiler or linker not being able to create the shared library libmysqlclient.so.*N* (where *N* is a version number), you can work around this problem by giving the --disable-shared option to configure. In this case, configure does not build a shared libmysqlclient.so.*N* library.

- By default, MySQL uses the latin1 (cp1252 West European) character set. To change the default set, use the --with-charset option:

  ```
  shell> ./configure --with-charset=CHARSET
  ```

  CHARSET may be one of binary, armscii8, ascii, big5, cp1250, cp1251, cp1256, cp1257, cp850, cp852, cp866, cp932, dec8, eucjpms, euckr, gb2312, gbk, geostd8, greek, hebrew, hp8, keybcs2, koi8r, koi8u, latin1, latin2, latin5, latin7, macce, macroman, sjis, swe7, tis620, ucs2, ujis, utf8. See The Character Set Used for Data and Sorting. (Additional character sets might be available. Check the output from ./configure --help for the current list.)

  The default collation may also be specified. MySQL uses the latin1_swedish_ci collation by default. To change this, use the --with-collation option:

  ```
  shell> ./configure --with-collation=COLLATION
  ```

  To change both the character set and the collation, use both the --with-charset and --with-collation options. The collation must be a legal collation for the character set. (Use the SHOW COLLATION statement to determine which collations are available for each character set.)

  > **Warning**
  >
  > If you change character sets after having created any tables, you must run myisamchk -r -q --set-collation=*collation_name* *on every MyISAM table*. Your indexes may be sorted incorrectly other-

wise. This can happen if you install MySQL, create some tables, and then reconfigure MySQL to use a different character set and reinstall it.

With the configure option --with-extra-charsets=*LIST*, you can define which additional character sets should be compiled into the server. *LIST* is one of the following:

- A list of character set names separated by spaces
- complex to include all character sets that can't be dynamically loaded
- all to include all character sets into the binaries

Clients that want to convert characters between the server and the client should use the SET NAMES statement. See Session System Variables, and Connection Character Sets and Collations.

- To configure MySQL with debugging code, use the --with-debug option:

```
shell> ./configure --with-debug
```

This causes a safe memory allocator to be included that can find some errors and that provides output about what is happening. See MySQL Internals: Porting.

As of MySQL 5.0.25, using --with-debug to configure MySQL with debugging support enables you to use the --debug="d,parser_debug" option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

- If your client programs are using threads, you must compile a thread-safe version of the MySQL client library with the --enable-thread-safe-client configure option. This creates a libmysqlclient_r library with which you should link your threaded applications. See How to Make a Threaded Client.
- Some features require that the server be built with compression library support, such as the COMPRESS() and UNCOMPRESS() functions, and compression of the client/server protocol. The --with-zlib-dir=no|bundled|*DIR* option provides control for compression library support. The value no explicitly disables compression support. bundled causes the zlib library bundled in the MySQL sources to be used. A *DIR* path name specifies where to find the compression library sources.
- It is possible to build MySQL 5.0 with large table support using the --with-big-tables option, beginning with MySQL 5.0.4.

  This option causes the variables that store table row counts to be declared as unsigned long long rather than unsigned long. This enables tables to hold up to approximately 1.844E+19 ($(2^{32})^2$) rows rather than 2^{32} (~4.295E+09) rows. Previously it was necessary to pass -DBIG_TABLES to the compiler manually in order to enable this feature.
- Run configure with the --disable-grant-options option to cause the --bootstrap, --skip-grant-tables, and --init-file options for mysqld to be disabled. For Windows, the configure.js script recognizes the DISABLE_GRANT_OPTIONS flag, which has the same effect. The capability is available as of MySQL 5.0.34.
- In MySQL Community Server, this option enables the statement profiling capability exposed by the SHOW PROFILE and SHOW PROFILES statements. (See SHOW PROFILES Syntax.) The option was added in MySQL 5.0.37.
- See Operating System-Specific Notes, for options that pertain to particular operating systems.
- See Using SSL Connections, for options that pertain to configuring MySQL to support secure (encrypted) connections.

Chapter 3. Installing from the Development Source Tree

> **Caution**
>
> You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL up and running on your system, you should use a standard release distribution (either a binary or source distribution).

To obtain the most recent development source tree, you first need to download and install Bazaar. You can obtain Bazaar from the Bazaar VCS Website. Bazaar is supported by any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows or Mac OS X host. Instructions for downloading and installing Bazaar on the different platforms are available on the Bazaar website.

All MySQL projects are hosted on Launchpad. MySQL projects, including MySQL server, MySQL Workbench and others are available from the Sun/MySQL Engineering page. For the repositories related only to MySQL server, see the MySQL Server page.

To build under Unix/Linux, you must have the following tools installed:

- GNU `make`, available from http://www.gnu.org/software/make/. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make`. It may already be available on your system as `gmake`.
- `autoconf` 2.58 (or newer), available from http://www.gnu.org/software/autoconf/.
- `automake` 1.8.1, available from http://www.gnu.org/software/automake/.
- `libtool` 1.5, available from http://www.gnu.org/software/libtool/.
- `m4`, available from http://www.gnu.org/software/m4/.
- `bison`, available from http://www.gnu.org/software/bison/. You should use the latest version of bison where possible. Version 1.75 and version 2.1 are known to work. There have been reported problems with `bison` 1.875. If you experience problems, upgrade to a later, rather than earlier, version. Versions of `bison` older than 1.75 may report this error:

  ```
  sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
  ```

  The maximum table size is not actually exceeded; the error is caused by bugs in older versions of `bison`.

To build under Windows you will need a copy of Microsoft Visual C++ 2005 Express Edition, Visual Studio .Net 2003 (7.1), or Visual Studio 2005 (8.0) compiler system.

Once you have the necessary tools installed, you first need to create a local branch of the MySQL source code on your machine:

1. To obtain a copy of the MySQL source code, you must create a new Bazaar branch. If you do not already have a Bazaar repository directory set up, you need to initialize a new directory:

   ```
   shell> mkdir mysql-server
   shell> bzr init-repo --trees mysql-server
   ```

   Once you have an initialized directory, you can `branch` from the public MySQL server repositories. To create a branch of a specific version:

   ```
   shell> cd mysql-server
   shell> bzr branch lp:mysql-server/5.0 mysql-5.0
   ```

   The initial download will take some time to complete, depending on the speed of your connection. Please be patient. Once you have downloaded the first tree, additional trees should take significantly less time to download.

   When building from the Bazaar branch, you may want to create a copy of your active branch so that you can make configuration and other changes without affecting the original branch contents. You can achieve this by branching from the original branch:

   ```
   shell> bzr branch mysql-5.0 mysql-5.0-build
   ```

Once you have the local branch, you can start to build MySQL server from the source code. On Windows, the build process is different from Unix/Linux. To continue building MySQL on Windows, see Chapter 5, *Installing MySQL from Source on Windows*.

On Unix/Linux you need to use the `autoconf` system to create the `configure` script so that you can configure the build environment before building.

1. The following example shows the typical commands required to configure a source tree. The first `cd` command changes location into the top-level directory of the tree; replace `mysql-5.0` with the appropriate directory name.

   > **Note**
   >
   > For MySQL 5.1.12 and earlier, you must separately configure the `INNODB` storage engine. You can do this by running the following command from the main source directory:

   ```
   shell> cd mysql-5.0
   shell> (cd bdb/deist; sh s_all)
   shell> (cd innobase; autoreconf --force --install)
   shell> autoreconf --force --install
   shell> ./configure  # Add your favorite options here
   shell> make
   ```

   Or you can use `BUILD/autorun.sh` as a shortcut for the following sequence of commands:

   ```
   shell> aclocal; autoheader
   shell> libtoolize --automake --force
   shell> automake --force --add-missing; autoconf
   shell> (cd bdb/deist; sh s_all)
   shell> (cd innobase; aclocal; autoheader; autoconf; automake)
   ```

   The command line that changes directory into the `storage/innobase` directory is used to configure the `InnoDB` storage engine. You can omit this lines if you do not require `InnoDB` support.

   If you get some strange errors during this stage, verify that you have the correct version of the `libtool` installed.

   A collection of our standard configuration scripts is located in the `BUILD/` subdirectory. For example, you may find it more convenient to use the `BUILD/compile-pentium-debug` script than the preceding set of shell commands. To compile on a different architecture, modify the script by removing flags that are Pentium-specific, or use another script that may be more appropriate. These scripts are provided on an “as-is” basis. They are not officially maintained and their contents may change from release to release.

2. When the build is done, run `make install`. Be careful with this on a production machine; the command may overwrite your live release installation. If you have another installation of MySQL, we recommend that you run `./configure` with different values for the `--prefix`, `--with-tcp-port`, and `--with-unix-socket-path` options than those used for your production server.

3. Play hard with your new installation and try to make the new features crash. Start by running `make test`. See MySQL Test Suite.

4. If you have gotten to the `make` stage, but the distribution does not compile, please enter the problem into our bugs database using the instructions given in How to Report Bugs or Problems. If you have installed the latest versions of the required GNU tools, and they crash trying to process our configuration files, please report that also. However, if you execute `aclocal` and get a `command not found` error or a similar problem, do not report it. Instead, make sure that all the necessary tools are installed and that your `PATH` variable is set correctly so that your shell can find them.

5. After initially copying the repository with `bzr` to obtain the source tree, you should use `pull` option to periodically update your local copy. To do this any time after you have set up the repository, use this command:

   ```
   shell> bzr pull
   ```

6. You can examine the changeset comments for the tree by using the `log` option to `bzr`:

   ```
   shell> bzr log
   ```

   You can also browse changesets, comments, and source code online. To browse this information for MySQL 5.0, go to http://launchpad.net/mysql-server/.

   If you see diffs or code that you have a question about, do not hesitate to send email to the MySQL `internals` mailing list. See MySQL Mailing Lists. Also, if you think you have a better idea on how to do something, send an email message to the list with a patch.

Chapter 4. Dealing with Problems Compiling MySQL

All MySQL programs compile cleanly for us with no warnings on Solaris or Linux using `gcc`. On other systems, warnings may occur due to differences in system include files. See MIT-pthreads Notes, for warnings that may occur when using MIT-pthreads. For other problems, check the following list.

The solution to many problems involves reconfiguring. If you do need to reconfigure, take note of the following:

- If `configure` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `config.cache`. When `configure` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `configure`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old configuration information or object files from being used, run these commands before re-running `configure`:

```
shell> rm config.cache
shell> make clean
```

Alternatively, you can run `make distclean`.

The following list describes some of the problems when compiling MySQL that have been found to occur most often:

- If you get errors such as the ones shown here when compiling `sql_yacc.cc`, you probably have run out of memory or swap space:

  ```
  Internal compiler error: program cc1plus got fatal signal 11
  Out of virtual memory
  Virtual memory exhausted
  ```

  The problem is that `gcc` requires a huge amount of memory to compile `sql_yacc.cc` with inline functions. Try running `configure` with the `--with-low-memory` option:

  ```
  shell> ./configure --with-low-memory
  ```

  This option causes `-fno-inline` to be added to the compile line if you are using `gcc` and `-O0` if you are using something else. You should try the `--with-low-memory` option even if you have so much memory and swap space that you think you can't possibly have run out. This problem has been observed to occur even on systems with generous hardware configurations, and the `--with-low-memory` option usually fixes it.

- By default, `configure` picks `c++` as the compiler name and GNU `c++` links with `-lg++`. If you are using `gcc`, that behavior can cause problems during configuration such as this:

  ```
  configure: error: installation or configuration problem:
  C++ compiler cannot create executables.
  ```

  You might also observe problems during compilation related to `g++`, `libg++`, or `libstdc++`.

  One cause of these problems is that you may not have `g++`, or you may have `g++` but not `libg++`, or `libstdc++`. Take a look at the `config.log` file. It should contain the exact reason why your C++ compiler didn't work. To work around these problems, you can use `gcc` as your C++ compiler. Try setting the environment variable `CXX` to `"gcc -O3"`. For example:

  ```
  shell> CXX="gcc -O3" ./configure
  ```

  This works because `gcc` compiles C++ source files as well as `g++` does, but does not link in `libg++` or `libstdc++` by default.

  Another way to fix these problems is to install `g++`, `libg++`, and `libstdc++`. However, we recommend that you not use `libg++` or `libstdc++` with MySQL because this only increases the binary size of `mysqld` without providing any benefits. Some versions of these libraries have also caused strange problems for MySQL users in the past.

- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

  ```
  making all in mit-pthreads
  make: Fatal error in reader: Makefile, line 18:
  Badly formed macro assignment
  ```

Or:

```
make: file `Makefile' line 18: Must be a separator (:
```

Or:

```
pthread.h: No such file or directory
```

Solaris and FreeBSD are known to have troublesome `make` programs.

GNU `make` 3.75 is known to work.

- If you want to define flags to be used by your C or C++ compilers, do so by adding the flags to the `CFLAGS` and `CXXFLAGS` environment variables. You can also specify the compiler names this way using `CC` and `CXX`. For example:

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

See MySQL Binaries Compiled by MySQL AB, for a list of flag definitions that have been found to be useful on various systems.

- If you get errors such as those shown here when compiling `mysqld`, `configure` did not correctly detect the type of the last argument to `accept()`, `getsockname()`, or `getpeername()`:

```
cxx: Error: mysqld.cc, line 645: In this statement, the referenced
     type of the pointer value ''length'' is ''unsigned long'',
     which is not compatible with ''int''.
new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);
```

To fix this, edit the `config.h` file (which is generated by `configure`). Look for these lines:

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

Change `XXX` to `size_t` or `int`, depending on your operating system. (You must do this each time you run `configure` because `configure` regenerates `config.h`.)

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally, the build process does not need to create `sql_yacc.cc` because MySQL comes with a pre-generated copy. However, if you do need to re-create it, you might encounter this error:

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

This is a sign that your version of `yacc` is deficient. You probably need to install `bison` (the GNU version of `yacc`) and use that instead.

- On Debian Linux 3.0, you need to install `gawk` instead of the default `mawk` if you want to compile MySQL with Berkeley DB support.

- If you need to debug `mysqld` or a MySQL client, run `configure` with the `--with-debug` option, and then recompile and link your clients with the new client library. See MySQL Internals: Porting.

- If you get a compilation error on Linux (for example, SuSE Linux 8.1 or Red Hat Linux 7.3) similar to the following one, you probably do not have `g++` installed:

```
libmysql.c:1329: warning: passing arg 5 of `gethostbyname_r' from
incompatible pointer type
libmysql.c:1329: too few arguments to function `gethostbyname_r'
libmysql.c:1329: warning: assignment makes pointer from integer
without a cast
make[2]: *** [libmysql.lo] Error 1
```

By default, the `configure` script attempts to determine the correct number of arguments by using `g++` (the GNU C++ compiler). This test yields incorrect results if `g++` is not installed. There are two ways to work around this problem:

  - Make sure that the GNU C++ `g++` is installed. On some Linux distributions, the required package is called `gpp`; on others,

it is named `gcc-c++`.

- Use `gcc` as your C++ compiler by setting the `CXX` environment variable to `gcc`:

```
export CXX="gcc"
```

You must run `configure` again after making either of those changes.

Chapter 5. Installing MySQL from Source on Windows

These instructions describe how to build binaries from source for MySQL 5.0 on Windows. Instructions are provided for building binaries from a standard source distribution or from the Bazaar tree that contains the latest development source.

> **Note**
>
> The instructions here are strictly for users who want to test MySQL on Microsoft Windows from the latest source distribution or from the Bazaar tree. For production use, we do not advise using a MySQL server built by yourself from source. Normally, it is best to use precompiled binary distributions of MySQL that are built specifically for optimal performance on Windows by Sun Microsystems, Inc. Instructions for installing binary distributions are available in Installing MySQL on Windows.

To build MySQL on Windows from source, you must satisfy the following system, compiler, and resource requirements:

- Windows 2000, Windows XP, or newer version.

  Windows Vista is supported when using Visual Studio 2005 provided you have installed the following updates:

  - Microsoft Visual Studio 2005 Professional Edition - ENU Service Pack 1 (KB926601)
  - Security Update for Microsoft Visual Studio 2005 Professional Edition - ENU (KB937061)
  - Update for Microsoft Visual Studio 2005 Professional Edition - ENU (KB932232)

- To build from the standard source distribution, you will need CMake, which can be downloaded from http://www.cmake.org. After installing, modify your path to include the `cmake` binary.
- Microsoft Visual C++ 2005 Express Edition, Visual Studio .Net 2003 (7.1), or Visual Studio 2005 (8.0) compiler system.
- If you are using Visual C++ 2005 Express Edition, you must also install an appropriate Platform SDK. More information and links to downloads for various Windows platforms is available from http://www.microsoft.com/downloads/details.aspx?familyid=0baf2b35-c656-4969-ace8-e4c0c0716adb.
- If you are compiling from a Bazaar tree or making changes to the parser, you need `bison` for Windows, which can be downloaded from http://gnuwin32.sourceforge.net/packages/bison.htm. Download the package labeled “Complete package, excluding sources”. After installing the package, modify your path to include the `bison` binary and ensure that this binary is accessible from Visual Studio.
- Cygwin might be necessary if you want to run the test script or package the compiled binaries and support files into a Zip archive. (Cygwin is needed only to test or package the distribution, not to build it.) Cygwin is available from http://cygwin.com.
- 3GB to 5GB of disk space.

The exact system requirements can be found here: http://msdn.microsoft.com/vstudio/Previous/2003/sysreqs/default.aspx and http://msdn.microsoft.com/vstudio/products/sysreqs/default.aspx

There are three solutions available for building from the source code on Windows:

- Build from the standard MySQL source distribution. For this you will need CMake and Visual C++ Express Edition or Visual Studio. Using this method you can select the storage engines that are included in your build. To use this method, see Section 5.1, “Building MySQL from the Standard Source Distribution”.
- Build from the MySQL Windows source distribution. The Windows source distribution includes ready-made Visual Studio solution files that enable support for all storage engines (except `NDB`). To build using using method you only need Visual C++ Express Edition or Visual Studio. To use this method, see Section 5.2, “Building MySQL from a Windows Source Distribution”.
- Build directly from the Bazaar source repository. For this you will need CMake, Visual C++ Express Edition or Visual Studio, and `bison`. For this method you need to create the distribution on a Unix system and then copy the generated files to your Windows build environment. To use this method, see Section 5.5, “Creating a Windows Source Package from the Bazaar Repository”.

If you find something not working as expected, or you have suggestions about ways to improve the current build process on Windows, please send a message to the `win32` mailing list. See MySQL Mailing Lists.

5.1. Building MySQL from the Standard Source Distribution

You can build MySQL on Windows by using a combination of `cmake` and Microsoft Visual Studio .NET 2003 (7.1), Microsoft Visual Studio 2005 (8.0) or Microsoft Visual C++ 2005 Express Edition. You must have the appropriate Microsoft Platform SDK installed.

> **Note**
>
> To compile from the source code using CMake you must use the standard source distribution (for example, `mysql-5.0.45.tar.gz`). You build from the same distribution as used to build MySQL on Unix, Linux and other platforms. Do *not* use the Windows Source distributions as they do not contain the necessary configuration script and other files.

Follow this procedure to build MySQL:

1. If you are installing from a packaged source distribution, create a work directory (for example, `C:\workdir`), and unpack the source distribution there using `WinZip` or another Windows tool that can read `.zip` files. This directory is the work directory in the following instructions.
2. If you are installing from a Bazaar tree, the root directory of that tree is the work directory in the following instructions.
3. Using a command shell, navigate to the work directory and run the following command:

   ```
   C:\workdir>win\configure.js options
   ```

   If you have associated the `.js` file extension with an application such as a text editor, then you may need to use the following command to force `configure.js` to be executed as a script:

   ```
   C:\workdir>cscript win\configure.js options
   ```

   These options are available for `configure.js`:

   - `WITH_INNOBASE_STORAGE_ENGINE`: Enable the `InnoDB` storage engine.
   - `WITH_PARTITION_STORAGE_ENGINE`: Enable user-defined partitioning.
   - `WITH_ARCHIVE_STORAGE_ENGINE`: Enable the `ARCHIVE` storage engine.
   - `WITH_BLACKHOLE_STORAGE_ENGINE`: Enable the `BLACKHOLE` storage engine.
   - `WITH_EXAMPLE_STORAGE_ENGINE`: Enable the `EXAMPLE` storage engine.
   - `WITH_FEDERATED_STORAGE_ENGINE`: Enable the `FEDERATED` storage engine.
   - `MYSQL_SERVER_SUFFIX=suffix`: Server suffix, default none.
   - `COMPILATION_COMMENT=comment`: Server comment, default "Source distribution".
   - `MYSQL_TCP_PORT=port`: Server port, default 3306.
   - `DISABLE_GRANT_OPTIONS`: Disables the the `--bootstrap`, `--skip-grant-tables`, and `--init-file` options for `mysqld`. This option is available as of MySQL 5.0.36.

   For example (type the command on one line):

   ```
   C:\workdir>win\configure.js WITH_INNOBASE_STORAGE_ENGINE »
              WITH_PARTITION_STORAGE_ENGINE MYSQL_SERVER_SUFFIX=-pro
   ```

4. From the work directory, execute the `win\build-vs8.bat` or `win\build-vs71.bat` file, depending on the version of Visual Studio you have installed. The script invokes CMake, which generates the `mysql.sln` solution file you will need to build MySQL using Visual Studio..

   You can also use `win\build-vs8_x64.bat` to build the 64-bit version of MySQL. However, you cannot build the 64-bit version with Visual Studio Express Edition. You must use Visual Studio 2005 (8.0) or higher.

5. From the work directory, open the generated `mysql.sln` file with Visual Studio and select the proper configuration using the CONFIGURATION menu. The menu provides Debug, Release, RelwithDebInfo, MinRelInfo options. Then select SOLUTION > Build to build the solution.

The build process will take some time. Please be patient.

Remember the configuration that you use in this step. It is important later when you run the test script because that script needs to know which configuration you used.

6. You should test you build before installation. See Section 5.4, “Testing a Windows Source Build”.

7. To install, use the instructions in Section 5.3, “Installing MySQL from a Source Build on Windows”.

5.2. Building MySQL from a Windows Source Distribution

The Windows source distribution includes the necessary solution file and the `vcproj` files required to build each component. Using this method you are not able to select the storage engines that are included in your build.

> **Note**
>
> VC++ workspace files for MySQL 4.1 and above are compatible with Microsoft Visual Studio 7.1 and tested by MySQL AB staff before each release.

Follow this procedure to build MySQL:

1. Create a work directory (for example, `C:\workdir`).

2. Unpack the source distribution in the aforementioned directory using `WinZip` or another Windows tool that can read `.zip` files.

3. Start Visual Studio .Net 2003 (7.1).

4. From the FILE menu, select Open Solution....

5. Open the `mysql.sln` solution you find in the work directory.

6. From the BUILD menu, select Configuration Manager....

7. In the ACTIVE SOLUTION CONFIGURATION pop-up menu, select the configuration to use. You likely want to use one of nt (normal server), Max nt (more engines and features), or Debug configuration.

8. From the BUILD menu, select Build Solution.

9. Debug versions of the programs and libraries are placed in the `client_debug` and `lib_debug` directories. Release versions of the programs and libraries are placed in the `client_release` and `lib_release` directories.

10. You should test you build before installation. See Section 5.4, “Testing a Windows Source Build”.

11. To install, use the instructions in Section 5.3, “Installing MySQL from a Source Build on Windows”.

5.3. Installing MySQL from a Source Build on Windows

When you are satisfied that the program you have built is working correctly, stop the server. Now you can install the distribution. There are two ways to do this, either by using the supplied installation script or by copying the files individually by hand.

To use the script method you must have Cygwin installed as the script is a Shell script. To execute the installation process, run the `make_win_bin_dist` script in the `scripts` directory of the MySQL source distribution (see `make_win_bin_dist`). This is a shell script, so you must have Cygwin installed if you want to use it. It creates a Zip archive of the built executables and support files that you can unpack to your desired installation location.

It is also possible to install MySQL by copying directories and files manually:

1. Create the directories where you want to install MySQL. For example, to install into `C:\mysql`, use these commands:

```
shell> mkdir C:\mysql
shell> mkdir C:\mysql\bin
shell> mkdir C:\mysql\data
shell> mkdir C:\mysql\share
shell> mkdir C:\mysql\scripts
```

If you want to compile other clients and link them to MySQL, you should also create several additional directories:

```
shell> mkdir C:\mysql\include
shell> mkdir C:\mysql\lib
shell> mkdir C:\mysql\lib\debug
shell> mkdir C:\mysql\lib\opt
```

If you want to benchmark MySQL, create this directory:

```
shell> mkdir C:\mysql\sql-bench
```

Benchmarking requires Perl support. See Perl Installation Notes.

2. From the work directory, copy into the `C:\mysql` directory the following directories:

```
shell> cd \workdir
C:\workdir> copy client_release\*.exe C:\mysql\bin
C:\workdir> copy client_debug\mysqld.exe C:\mysql\bin\mysqld-debug.exe
C:\workdir> xcopy scripts\*.* C:\mysql\scripts /E
C:\workdir> xcopy share\*.* C:\mysql\share /E
```

If you want to compile other clients and link them to MySQL, you should also copy several libraries and header files:

```
C:\workdir> copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
C:\workdir> copy lib_debug\libmysql.* C:\mysql\lib\debug
C:\workdir> copy lib_debug\zlib.* C:\mysql\lib\debug
C:\workdir> copy lib_release\mysqlclient.lib C:\mysql\lib\opt
C:\workdir> copy lib_release\libmysql.* C:\mysql\lib\opt
C:\workdir> copy lib_release\zlib.* C:\mysql\lib\opt
C:\workdir> copy include\*.h C:\mysql\include
C:\workdir> copy libmysql\libmysql.def C:\mysql\include
```

If you want to benchmark MySQL, you should also do this:

```
C:\workdir> xcopy sql-bench\*.* C:\mysql\bench /E
```

After installation, set up and start the server in the same way as for binary Windows distributions. See Installing MySQL on Windows.

5.4. Testing a Windows Source Build

You should test the server that you have built from source before using the distribution.

To test the server you need to run the built `mysqld`. By default, using the source build examples, the MySQL base directory and data directory are `C:\mysql` and `C:\mysql\data`. If you want to test your server using the source tree root directory and its data directory as the base directory and data directory, you need to tell the server their path names. You can either do this on the command line with the `--basedir` and `--datadir` options, or by placing appropriate options in an option file. (See Using Option Files.) If you have an existing data directory elsewhere that you want to use, you can specify its path name instead.

When the server is running in standalone fashion or as a service based on your configuration, try to connect to it from the `mysql` interactive command-line utility.

You can also run the standard test script, `mysql-test-run.pl`. This script is written in Perl, so you'll need either Cygwin or ActiveState Perl to run it. You may also need to install the modules required by the script. To run the test script, change location into the `mysql-test` directory under the work directory, set the `MTR_VS_CONFIG` environment variable to the configuration you selected earlier (or use the `--vs-config` option), and invoke `mysql-test-run.pl`. For example (using Cygwin and the `bash` shell):

```
shell> cd mysql-test
shell> export MTS_VS_CONFIG=debug
shell> ./mysql-test-run.pl --force --timer
shell> ./mysql-test-run.pl --force --timer --ps-protocol
```

5.5. Creating a Windows Source Package from the Bazaar Repository

To create a Windows source package from the current Bazaar source tree, use the instructions here. This procedure must be performed on a system running a Unix or Unix-like operating system because some of the configuration and build steps require tools

that work only on Unix. For example, the following procedure is known to work well on Linux.

1. Copy the Bazaar source tree for MySQL 5.0. For instructions on how to do this, see Chapter 3, *Installing from the Development Source Tree*.

2. Configure and build the distribution so that you have a server binary to work with. One way to do this is to run the following command in the top-level directory of your source tree:

   ```
   shell> ./BUILD/compile-pentium-max
   ```

3. After making sure that the build process completed successfully, run the following utility script from top-level directory of your source tree:

   ```
   shell> ./scripts/make_win_src_distribution
   ```

   This script creates a Windows source package to be used on your Windows system. You can supply different options to the script based on your needs. See `make_win_src_distribution`, for a list of allowable options.

   By default, `make_win_src_distribution` creates a Zip-format archive with the name `mysql-`*`VERSION`*`-win-src.zip`, where *`VERSION`* represents the version of your MySQL source tree.

4. Copy or upload the Windows source package that you have just created to your Windows machine. To compile it, use the instructions in Section 5.2, “Building MySQL from a Windows Source Distribution”.

Chapter 6. Compiling MySQL Clients on Windows

In your source files, you should include `my_global.h` before `mysql.h`:

```
#include <my_global.h>
#include <mysql.h>
```

`my_global.h` includes any other files needed for Windows compatibility (such as `windows.h`) if you compile your program on Windows.

You can either link your code with the dynamic `libmysql.lib` library, which is just a wrapper to load in `libmysql.dll` on demand, or link with the static `mysqlclient.lib` library.

The MySQL client libraries are compiled as threaded libraries, so you should also compile your code to be multi-threaded.