5.1 Building Connector/C++ Applications: General Considerations

This section discusses general considerations to keep in mind when building Connector/C++ applications. For information that applies to particular platforms, see the section that applies to your platform in Section 5.2, “Building Connector/C++ Applications: Platform-Specific Considerations”.

Commands shown here are as given from the command line (for example, as invoked from a `Makefile`). The commands apply to any platform that supports **make** and command-line build tools such as **g++**, **cc**, or **clang**, but may need adjustment for your build environment.

- Build Tools and Configuration Settings
- C++11 Support
- Connector/C++ Header Files
- Connector/C++ Version Macros
- Boost Header Files
- Link Libraries
- Runtime Libraries
- Using the Connector/C++ Dynamic Library
- Using the Connector/C++ Static Library

Build Tools and Configuration Settings

It is important that the tools you use to build your Connector/C++ applications are compatible with the tools used to build Connector/C++ itself. Ideally, build your applications with the same tools that were used to build the Connector/C++ binaries.

To avoid issues, ensure that these factors are the same for your applications and Connector/C++ itself:

- Compiler version.
- Runtime library.
- Runtime linker configuration settings.

To avoid potential crashes, the build configuration of Connector/C++ should match the build configuration of the application using it. For example, do not use a release build of Connector/C++ with a debug build of the client application.

To use a different compiler version, release configuration, or runtime library, first build Connector/C++ from source using your desired settings (see Chapter 4, *Installing Connector/C++ from Source*), then build your applications using those same settings.

Connector/C++ binary distributions include an `INFO_BIN` file that describes the environment and configuration options used to build the distribution. If you installed Connector/C++ from a binary distribution and experience build-related issues on a platform, it may help to check the settings that were used to build the distribution on that platform. Binary distributions also include an `INFO_SRC` file that provides information about the product version and the source repository from which the distribution was produced. (Prior to Connector/C++ 8.0.14, look for `BUILDINFO.txt` rather than `INFO_BIN` and `INFO_SRC`.)

C++11 Support

X DevAPI uses C++11 language features. To compile Connector/C++ applications that use X DevAPI, enable C++11 support in the compiler using the `-std=c++11` option. This option is not needed for applications that use X DevAPI for C (which is a plain C API) or the legacy JDBC API (which is based on plain C++), unless the application code uses C++11.

Connector/C++ Header Files

The API an application uses determines which Connector/C++ header files it should include. The following include directives work under the assumption that the include path contains `$MYSQL_CPPCONN_DIR/include`, where `$MYSQL_CPPCONN_DIR` is the Connector/C++ installation location. Pass an `-I $MYSQL_CPPCONN_DIR/include` option on the compiler invocation command to ensure this.

- For applications that use X DevAPI:

```
#include <mysqlx/xdevapi.h>
```

- For applications that use X DevAPI for C:

```
#include <mysqlx/xapi.h>
```

- For applications that use the legacy JDBC API, the header files are version dependent:

- As of Connector/C++ 8.0.16, a single `#include` directive suffices:

```
#include <mysql/jdbc.h>
```

- Prior to Connector/C++ 8.0.16, use this set of `#include` directives:

```
#include <jdbc/mysql_driver.h>
#include <jdbc/mysql_connection.h>
#include <jdbc/cppconn/*.h>
```

  The notation `<jdbc/cppconn/*.h>` means that you should include all header files from the `jdbc/cppconn` directory that are needed by your application. The particular files needed depend on the application.

- Legacy code that uses Connector/C++ 1.1 has `#include` directives of this form:

```
#include <mysql_driver.h>
#include <mysql_connection.h>
#include <cppconn/*.h>
```

  To build such code with Connector/C++ 8.0 without modifying it, add `$MYSQL_CPPCONN_DIR/include/jdbc` to the include path.

To compile code that you intend to link statically against Connector/C++, define a macro that adjusts API declarations in the header files for usage with the static library. For details, see Using the Connector/C++ Static Library.

Connector/C++ Version Macros

Starting with Connector/C++ 8.0.30, version-related macros are defined in public header files. The intent of the macros is to provide a way to systematically and predictably maintain version numbering of the Connector/C++ product. The following table describes the version-related macros.

Macro Name	Description
`MYSQL_CONCPP_VERSION_MAJOR`	Major number of the product version; currently **8**.
`MYSQL_CONCPP_VERSION_MINOR`	Minor number of the product version; currently **00**.
`MYSQL_CONCPP_VERSION_MICRO`	Micro number of the product version; initially **30**.
`MYSQL_CONCPP_VERSION_NUMBER`	Full Connector/C++ version number, which combines the major, minor, and micro numbers. For example, the combined version number

Macro Name	Description
	8000030 represents Connector/C++ 8.0.30.

> **Note**
>
> The version numbers maintained by these macros apply to the Connector/C++ product only and are unrelated to API or ABI versions, which are handled separately.

Connector/C++ applications that use X DevAPI, X DevAPI for C, or the legacy JDBC API can specify the `MYSQL_CONCPP_VERSION_NUMBER` macro to add conditional tests that determine the inclusion or exclusion of feature dependencies, based on which Connector/C++ version introduced the dependency. For example, it is possible to use the `MYSQL_CONCPP_VERSION_NUMBER` macro in the following cases:

- When a Connector/C++ application needs a guard that checks for features introduced after the specified version. The following example specifies version 8.0.32, which has the macro defined in public header files. The same conditional-compilation directive also works when the macro is not defined (with pre-8.0.30 header files), because the value is treated as 0.

```
#if MYSQL_CONCPP_VERSION_NUMBER > 8000032
  // use some 8.0.32+ feature
#endif
```

- When a Connector/C++ application requires all features introduced before the specified version.

```
#if MYSQL_CONCPP_VERSION_NUMBER < 8000032
  // this usage is OK; it compiles with 8.0.31 and all previous versions
#endif
```

- When a Connector/C++ application that uses X DevAPI also uses the `CharacterSet::utf8mb3` enumeration constant or any of the new `utf8mb4` collation members. If the application compiles with the pre-8.0.30 connector, then it is possible to guard the use of these new API elements.

```
#if MYSQL_CONCPP_VERSION_NUMBER >= 8000030
  if (CharacterSet::utf8mb3 == cs)
#else
  if (CharacterSet::utf8 == cs)
#endif
  {
```

```
    // cs is the id of the utf8 character set
  }
```

- When a Connector/C++ application that uses X DevAPI needs to check the name of the `utf8mb3` character set or any of its collations, and it must also be compiled with the pre-8.0.30 connector.

```
#if MYSQL_CONCPP_VERSION_NUMBER >= 8000030
  if ("utf8mb3" == characterSetName(cs))
#else
  if ("utf8" == characterSetName(cs))
#endif
  {
    // cs is the id of the utf8 character set
  }
```

> **Note**
>
> Alternatively, you can compare against numeric enumeration constant value, which should work regardless of the connector version.

- When a Connector/C++ application that uses the legacy JDBC API needs to check the name of the `utf8mb3` character set or any of its collations, and it must also be compiled with the pre-8.0.30 connector.

```
#if MYSQL_CONCPP_VERSION_NUMBER >= 8000030
  if ("utf8mb3" == metadata->getColumnCharset(column))
#else
  if ("utf8" == metadata->getColumnCharset(column))
#endif
  {
    // column is the column index using the utf8 character set
  }
```

Do not use the `MYSQL_CONCPP_VERSION_NUMBER` macro to check against versions earlier than Connector/C++ 8.0.30, which can produce unreliable results. For example:

```
#if MYSQL_CONCPP_VERSION_NUMBER > 8000028
  // this does not compile the with 8.0.29 connector!
#endif

#if MYSQL_CONCPP_VERSION_NUMBER < 8000028
  // this compiles with the 8.0.29 connector!
```

```
#endif
```

Boost Header Files

The Boost header files are needed under these circumstances:

- Prior to Connector/C++ 8.0.16, on Unix and Unix-like platforms for applications that use X DevAPI or X DevAPI for C, if you build using **gcc** and the version of the C++ standard library on your system does not implement the UTF8 converter (`codecvt_utf8`).
- Prior to Connector/C++ 8.0.23, to compile Connector/C++ applications that use the legacy JDBC API.

If the Boost header files are needed, Boost 1.59.0 or newer must be installed, and the location of the headers must be added to the include path. To obtain Boost and its installation instructions, visit the official Boost site.

Link Libraries

Building Connector/C++ using OpenSSL makes the connector library dependent on OpenSSL dynamic libraries. In that case:

- When linking an application to Connector/C++ dynamically, this dependency is relevant only at runtime.
- When linking an application to Connector/C++ statically, link to the OpenSSL libraries as well. On Linux, this means adding `-lssl -lcrypto` explicitly to the compile/link command. On Windows, this is handled automatically.

On Windows, link to the dynamic version of the C++ Runtime Library.

Runtime Libraries

X DevAPI for C applications need `libstdc++` at runtime. Depending on your platform or build tools, a different library may apply. For example, the library is `libc++` on macOS; see Section 5.2.2, “macOS Notes”.

If an application is built using dynamic link libraries, those libraries must be present not just on the build host, but on target hosts where the application runs. The dynamic linker must be properly configured to find those libraries and their runtime dependencies, as well as to find Connector/C++ libraries and their runtime dependencies.

Connector/C++ libraries built by Oracle depend on the OpenSSL libraries. The latter must be installed on the system in order to run code that links against Connector/C++ libraries. Another option is to put

the OpenSSL libraries in the same location as Connector/C++, in which case, the dynamic linker should find them next to the connector library. See also Section 5.2.1, “Windows Notes”, and Section 5.2.2, “macOS Notes”.

> **Note**
>
> The TLSv1 and TLSv1.1 connection protocols are no longer supported as of Connector/C++ 8.0.28, making TLSv1.2 the earliest supported connection protocol.

Using the Connector/C++ Dynamic Library

The Connector/C++ dynamic library name depends on the platform. These libraries implement X DevAPI and X DevAPI for C, where ***A*** in the library name represents the ABI version:

- `libmysqlcppconn8.so.`***A*** (Unix)
- `libmysqlcppconn8.`***A***`.dylib` (macOS)
- `mysqlcppconn8-`***A***`-vs`***NN***`.dll`, with import library `vs`***NN***`/mysqlcppconn8.lib` (Windows)

For the legacy JDBC API, the dynamic libraries are named as follows, where ***B*** in the library name represents the ABI version:

- `libmysqlcppconn.so.`***B*** (Unix)
- `libmysqlcppconn.`***B***`.dylib` (macOS)
- `mysqlcppconn-`***B***`-vs`***NN***`.dll`, with import library `vs`***NN***`/mysqlcppconn-static.lib` (Windows)

On Windows, the `vs`***NN*** value in library names depends on the MSVC toolchain version used to build the libraries. (Connector/C++ libraries provided by Oracle use `vs14`, and they are compatible with MSVC 2019 and 2017.) This convention enables using libraries built with different versions of MSVC on the same system. See also Section 5.2.1, “Windows Notes”.

To build code that uses X DevAPI or X DevAPI for C, add `-lmysqlcppconn8` to the linker options. To build code that uses the legacy JDBC API, add `-lmysqlcppconn`.

You must also indicate whether to use the 64-bit or 32-bit libraries by specifying the appropriate library directory. Use an `-L` linker option to specify `$MYSQL_CONCPP_DIR/lib64` (64-bit libraries) or `$MYSQL_CONCPP_DIR/lib` (32-bit libraries), where `$MYSQL_CPPCONN_DIR` is the Connector/C++ installation location. On FreeBSD, `/lib64` is not used. The library name always ends with `/lib`.

To build a Connector/C++ application that uses X DevAPI, has sources in `app.cc`, and links dynamically to the connector library, the `Makefile` might look like this:

```
MYSQL_CONCPP_DIR = Connector/C++ installation location
CPPFLAGS = -I $(MYSQL_CONCPP_DIR)/include -L $(MYSQL_CONCPP_DIR)/lib64
LDLIBS = -lmysqlcppconn8
CXXFLAGS = -std=c++11
app : app.cc
```

With that `Makefile`, the command **make app** generates the following compiler invocation:

```
g++ -std=c++11 -I .../include -L .../lib64 app.cc -lmysqlcppconn8 -o app
```

To build a plain C application that uses X DevAPI for C, has sources in `app.c`, and links dynamically to the connector library, the `Makefile` might look like this:

```
MYSQL_CONCPP_DIR = Connector/C++ installation location
CPPFLAGS = -I $(MYSQL_CONCPP_DIR)/include -L $(MYSQL_CONCPP_DIR)/lib64
LDLIBS = -lmysqlcppconn8
app : app.c
```

With that `Makefile`, the command **make app** generates the following compiler invocation:

```
cc -I .../include -L .../lib64 app.c -lmysqlcppconn8 -o app
```

> **Note**
>
> The resulting code, even though it is compiled as plain C, depends on the C++ runtime (typically `libstdc++`, though this may differ depending on platform or build tools; see Runtime Libraries).

To build a plain C++ application that uses the legacy JDBC API, has sources in `app.c`, and links dynamically to the connector library, the `Makefile` might look like this:

```
MYSQL_CONCPP_DIR = Connector/C++ installation location
CPPFLAGS = -I $(MYSQL_CONCPP_DIR)/include -L $(MYSQL_CONCPP_DIR)/lib64
LDLIBS = -lmysqlcppconn
app : app.c
```

The library option in this case is `-lmysqlcppcon`, rather than `-lmysqlcppcon8` as for an X DevAPI or X DevAPI for C application.

With that `Makefile`, the command **make app** generates the following compiler invocation:

```
cc -I .../include -L .../lib64 app.c -lmysqlcppconn -o app
```

> **Note**
>
> When running an application that uses the Connector/C++ dynamic library, the library and its runtime dependencies must be found by the dynamic linker. See Runtime Libraries.

Using the Connector/C++ Static Library

It is possible to link your application with the Connector/C++ static library. This way there is no runtime dependency on the connector, and the resulting binary can run on systems where Connector/C++ is not installed.

> **Note**
>
> Even when linking statically, the resulting code still depends on all runtime dependencies of the Connector/C++ library. For example, if Connector/C++ is built using OpenSSL, the code has a runtime dependency on the OpenSSL libraries. See Runtime Libraries.

The Connector/C++ static library name depends on the platform. These libraries implement X DevAPI and X DevAPI for C:

- `libmysqlcppconn8-static.a` (Unix, macOS)
- `vs`***`NN`***`/mysqlcppconn8-static.lib` (Windows)

For the legacy JDBC API, the static libraries are named as follows:

- `libmysqlcppconn-static.a` (Unix, macOS)
- `vs`***`NN`***`/mysqlcppconn-static.lib` (Windows)

On Windows, the `vs`***`NN`*** value in library names depends on the MSVC toolchain version used to build the libraries. (Connector/C++ libraries provided by Oracle use `vs14`, and they are compatible with

MSVC 2019 and 2017.) This convention enables using libraries built with different versions of MSVC on the same system. See also Section 5.2.1, “Windows Notes”.

To compile code that you intend to link statically against Connector/C++, define a macro that adjusts API declarations in the header files for usage with the static library. One way to define the macro is by passing a `-D` option on the compiler invocation command:

- For applications that use X DevAPI, X DevAPI for C, or (as of Connector/C++ 8.0.16) the legacy JDBC API, define the `STATIC_CONCPP` macro. All that matters is that you define it; the value does not matter. For example: `-DSTATIC_CONCPP`
- Prior to Connector/C++ 8.0.16, for applications that use the legacy JDBC API, define the `CPPCONN_PUBLIC_FUNC` macro as an empty string. To ensure this, define the macro as `CPPCONN_PUBLIC_FUNC=`, not as `CPPCONN_PUBLIC_FUNC`. For example: `-DCPPCONN_PUBLIC_FUNC=`

To build a Connector/C++ application that uses X DevAPI, has sources in `app.cc`, and links statically to the connector library, the `Makefile` might look like this:

```
MYSQL_CONCPP_DIR = Connector/C++ installation location
CPPFLAGS = -DSTATIC_CONCPP -I $(MYSQL_CONCPP_DIR)/include
LDLIBS = $(MYSQL_CONCPP_DIR)/lib64/libmysqlcppconn8-static.a -lssl -lcrypto -lpthr
CXXFLAGS = -std=c++11
app : app.cc
```

With that `Makefile`, the command **make app** generates the following compiler invocation:

```
g++ -std=c++11 -DSTATIC_CONCPP -I .../include app.cc
  .../lib64/libmysqlcppconn8-static.a -lssl -lcrypto -lpthread -o app
```

> **Note**
>
> To avoid having the linker report unresolved symbols, the compile line must include the OpenSSL libraries and the `pthread` library on which Connector/C++ code depends.
>
> OpenSSL libraries are not needed if Connector/C++ is built without them, but Connector/C++ distributions built by Oracle do depend on OpenSSL.
>
> The exact list of libraries required by Connector/C++ library depends on the platform. For example, on Solaris, the `socket`, `rt`, and `nsl` libraries might be needed.

To build a plain C application that uses X DevAPI for C, has sources in `app.c`, and links statically to the connector library, the `Makefile` might look like this:

```
MYSQL_CONCPP_DIR = Connector/C++ installation location
CPPFLAGS = -DSTATIC_CONCPP -I $(MYSQL_CONCPP_DIR)/include
LDLIBS = $(MYSQL_CONCPP_DIR)/lib64/libmysqlcppconn8-static.a -lssl -lcrypto -lpthr
app : app.c
```

With that `Makefile`, the command **make app** generates the following compiler invocation:

```
cc -DSTATIC_CONCPP -I .../include app.c
  .../lib64/libmysqlcppconn8-static.a -lssl -lcrypto -lpthread -o app
```

To build a plain C application that uses the legacy JDBC API, has sources in `app.c`, and links statically to the connector library, the `Makefile` might look like this:

```
MYSQL_CONCPP_DIR = Connector/C++ installation location
CPPFLAGS = -DCPPCONN_PUBLIC_FUNC= -I $(MYSQL_CONCPP_DIR)/include
LDLIBS = $(MYSQL_CONCPP_DIR)/lib64/libmysqlcppconn-static.a -lssl -lcrypto -lpthre
app : app.c
```

The library option in this case names `libmysqlcppcon-static.a`, rather than `libmysqlcppcon8-static.a` as for an X DevAPI or X DevAPI for C application.

With that `Makefile`, the command **make app** generates the following compiler invocation:

```
cc -std=c++11 --DCPPCONN_PUBLIC_FUNC= -I .../include app.c
  .../lib64/libmysqlcppconn-static.a -lssl -lcrypto -lpthread -o app
```

When building plain C code, it is important to take care of connector's dependency on the C++ runtime, which is introduced by the connector library even though the code that uses it is plain C:

- One approach is to ensure that a C++ linker is used to build the final code. This approach is taken by the `Makefile` shown here:

  ```
  MYSQL_CONCPP_DIR = Connector/C++ installation location
  CPPFLAGS = -DSTATIC_CONCPP -I $(MYSQL_CONCPP_DIR)/include
  LDLIBS = $(MYSQL_CONCPP_DIR)/lib64/libmysqlcppconn8-static.a -lssl -lcrypto -l
  LINK.o = $(LINK.cc) # use C++ linker
  app : app.o
  ```

With that `Makefile`, the build process has two steps: first compile the application source in `app.c` using a plain C compiler to produce `app.o`, then link the final executable (`app`) using the C++ linker, which takes care of the dependency on the C++ runtime. The commands look something like this:

```
cc -DSTATIC_CONCPP -I .../include -c -o app.o app.c
g++ -DSTATIC_CONCPP -I .../include app.o
  .../libmysqlcppconn8-static.a -lssl -lcrypto -lpthread -o app
```

- Another approach is to use a plain C compiler and linker, but add the `libstdc++` C++ runtime library as an explicit option to the linker. This approach is taken by the `Makefile` shown here:

```
MYSQL_CONCPP_DIR = Connector/C++ installation location
CPPFLAGS = -DSTATIC_CONCPP -I $(MYSQL_CONCPP_DIR)/include
LDLIBS = $(MYSQL_CONCPP_DIR)/lib64/libmysqlcppconn8-static.a -lssl -lcrypto -l
app : app.c
```

With that `Makefile`, the compiler is invoked as follows:

```
cc -DSTATIC_CONCPP -I .../include app.c
  .../libmysqlcppconn8-static.a -lssl -lcrypto -lpthread -lstdc++ -o app
```

> **Note**
>
> Even if the application that uses Connector/C++ is written in plain C, the final executable depends on the C++ runtime which must be installed on the target computer on which the application is to run.