MySQL Connector/C++ 8.0 Developer Guide
Abstract

This manual describes how to install and configure MySQL Connector/C++ 8.0, which provides C++ and plain C interfaces for communicating with MySQL servers, and how to use Connector/C++ to develop database applications.

Connector/C++ 8.0 is highly recommended for use with MySQL Server 8.0 and 5.7. Please upgrade to Connector/C++ 8.0.

For notes detailing the changes in each release of Connector/C++, see MySQL Connector/C++ Release Notes.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Licensing information. This product may include third-party software, used under license. If you are using a Commercial release of MySQL Connector/C++, see the MySQL Connector/C++ Commercial Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a Community release of MySQL Connector/C++, see the MySQL Connector/C++ Community Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Document generated on: 2020-03-19 (revision: 65394)
# Table of Contents

Preface and Legal Notices .................................................................................................................. v  
1 Introduction to Connector/C++ ....................................................................................................... 1  
2 Obtaining Connector/C++ .................................................................................................................. 5  
3 Installing Connector/C++ from a Binary Distribution ................................................................. 7  
4 Installing Connector/C++ from Source .............................................................................................. 11  
   4.1 Source Installation System Prerequisites ......................................................................................... 11  
   4.2 Obtaining and Unpacking a Connector/C++ Source Distribution ..................................................... 12  
   4.3 Installing Connector/C++ from Source ............................................................................................ 13  
   4.4 Connector/C++ Source-Configuration Options ................................................................................ 16  
5 Building Connector/C++ Applications .............................................................................................. 21  
   5.1 Building Connector/C++ Applications: General Considerations .................................................... 21  
   5.2 Building Connector/C++ Applications: Platform-Specific Considerations ........................................ 27  
      5.2.1 Windows Notes .......................................................................................................................... 27  
      5.2.2 macOS Notes ............................................................................................................................ 32  
      5.2.3 Solaris Notes .............................................................................................................................. 32  
6 Connector/C++ Known Issues ........................................................................................................... 35  
7 Connector/C++ Support ...................................................................................................................... 37  
Index ...................................................................................................................................................... 39
Preface and Legal Notices

This manual describes how to install and configure MySQL Connector/C++ 8.0, and how to use it to develop database applications.

Legal Notices

Copyright © 2008, 2020, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:
You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

Documentation Accessibility

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.
Chapter 1 Introduction to Connector/C++

MySQL Connector/C++ 8.0 is a MySQL database connector for C++ applications that connect to MySQL servers. Connector/C++ can be used to access MySQL servers that implement a document store, or in a traditional way using SQL statements. The preferred development environment for Connector/C++ 8.0 is to enable development of C++ applications using X DevAPI, or plain C applications using X DevAPI for C, but Connector/C++ 8.0 also enables development of C++ applications that use the legacy JDBC-based API from Connector/C++ 1.1.

Connector/C++ applications that use X DevAPI or X DevAPI for C require a MySQL server that has X Plugin enabled. Connector/C++ applications that use the legacy JDBC-based API neither require nor support X Plugin.

For more detailed requirements about required MySQL versions for Connector/C++ applications, see Platform Support and Prerequisites.

For notes detailing the changes in each release of Connector/C++, see MySQL Connector/C++ Release Notes.

• Connector/C++ Benefits
• X DevAPI and X DevAPI for C
• Legacy JDBC API and JDBC Compatibility
• Platform Support and Prerequisites

Connector/C++ Benefits

MySQL Connector/C++ offers the following benefits for C++ users compared to the MySQL C API provided by the MySQL client library:

• Convenience of pure C++.
• Support for these application programming interfaces:
  • X DevAPI
  • X DevAPI for C
  • Legacy JDBC 4.0-based API
• Support for the object-oriented programming paradigm.
• Reduced development time.
• Licensed under the GPL with the FLOSS License Exception.
• Available under a commercial license upon request.

X DevAPI and X DevAPI for C

Connector/C++ implements X DevAPI, which enables connecting to MySQL servers that implement a document store with X Plugin. X DevAPI also enables applications to execute SQL statements.

Connector/C++ also implements a similar interface called X DevAPI for C for use by applications written in plain C.
Legacy JDBC API and JDBC Compatibility

Connector/C++ implements the JDBC 4.0 API, if built to include the legacy JDBC connector:

- Connector/C++ binary distributions include the JDBC connector.
- If you build Connector/C++ from source, the JDBC connector is not built by default, but can be included by enabling the `WITH_JDBC` CMake option. See Chapter 4, Installing Connector/C++ from Source.

The Connector/C++ JDBC API is compatible with the JDBC 4.0 API. Connector/C++ does not implement the entire JDBC 4.0 API, but does feature these classes: `Connection`, `DatabaseMetaData`, `Driver`, `PreparedStatement`, `ResultSet`, `ResultSetMetaData`, `Savepoint`, `Statement`.

The JDBC 4.0 API defines approximately 450 methods for the classes just mentioned. Connector/C++ implements approximately 80% of these.

Note

The legacy JDBC connector in Connector/C++ 8.0 is based on the connector provided by Connector/C++ 1.1. For more information about using the JDBC API in Connector/C++ 8.0, see MySQL Connector/C++ 1.1 Developer Guide.

Platform Support and Prerequisites

To see which platforms are supported, visit the Connector/C++ downloads page.

On Windows platforms, Commercial and Community Connector/C++ distributions require the Visual C++ Redistributable for Visual Studio. The Redistributable is available at the Visual Studio Download Center; install it before installing Connector/C++. The acceptable Redistributable versions depend on your Connector/C++ version:

- Connector/C++ 8.0.19 and higher: VC++ Redistributable 2017 or higher.
- Connector/C++ 8.0.14 to 8.0.18: VC++ Redistributable 2015 or higher.

The following requirements apply to building and running Connector/C++ applications, and to building Connector/C++ itself if you build it from source:

- To run Connector/C++ applications, the MySQL server requirements depend on the API the application uses:
  - Connector/C++ applications that use X DevAPI or X DevAPI for C require a server from MySQL 8.0 (8.0.11 or higher) or MySQL 5.7 (5.7.12 or higher), with X Plugin enabled. For MySQL 8.0, X Plugin is enabled by default. For MySQL 5.7, X Plugin must be enabled explicitly. (Some X Protocol features may not work with MySQL 5.7.)
  - Applications that use the JDBC API can use a server from MySQL 5.6 or higher. X Plugin is neither required nor supported.
- To build Connector/C++ applications:
  - The MySQL version does not apply.
• On Windows, Microsoft Visual Studio is required. The acceptable MSVC versions depend on your Connector/C++ version and the type of linking you use:

  • Connector/C++ 8.0.20 and higher: Same as Connector/C++ 8.0.19, with the addition that binary distributions are also compatible with MSVC 2017 using the static X DevAPI connector library. This means that binary distributions are fully compatible with MSVC 2019, and fully compatible with MSVC 2017 with the exception of the static legacy (JDBC) connector library.

  • Connector/C++ 8.0.19: Connector/C++ binary distributions are compatible with projects built using MSVC 2019 (using either dynamic or static connector libraries) or MSVC 2017 (using dynamic connector libraries).

  • Connector/C++ 8.0.14 to 8.0.18: MSVC 2017 or 2015.

  • Connector/C++ prior to 8.0.14: MSVC 2015.

• To build Connector/C++ from source:

  • The MySQL C API client library may be required:

    • For Connector/C++ built without the JDBC connector (which is the default), the client library is not needed.

    • To build Connector/C++ with the JDBC connector, configure Connector/C++ with the `WITH_JDBC` CMake option enabled. In this case, the JDBC connector requires a client library from MySQL 8.0 (8.0.11 or higher) or MySQL 5.7 (5.7.9 or higher).

• On Windows, Microsoft Visual Studio is required. The acceptable MSVC versions depend on your Connector/C++ version:

  • Connector/C++ 8.0.19 and higher: MSVC 2019 or 2017.

  • Connector/C++ 8.0.14 to 8.0.18: MSVC 2017 or 2015.

  • Connector/C++ prior to 8.0.14: MSVC 2015.
Chapter 2 Obtaining Connector/C++

Connector/C++ binary and source distributions are available, in platform-specific packaging formats. To obtain a distribution, visit the Connector/C++ downloads page. It is also possible to clone the Connector/C++ Git source repository.

- Connector/C++ binary distributions are available for Microsoft Windows, and for Unix and Unix-like platforms. See Chapter 3, Installing Connector/C++ from a Binary Distribution.

- Connector/C++ source distributions are available as compressed tar files or Zip archives and can be used on any supported platform. See Chapter 4, Installing Connector/C++ from Source.

- The Connector/C++ source code repository uses Git and is available at GitHub. See Chapter 4, Installing Connector/C++ from Source.
Chapter 3 Installing Connector/C++ from a Binary Distribution

To obtain a Connector/C++ binary distribution, visit the Connector/C++ downloads page.

For some platforms, Connector/C++ binary distributions are available in platform-specific packaging formats. Binary distributions are also available in more generic format, in the form of compressed tar files or Zip archives.

For descriptions here that refer to documentation files, those files have names such as CONTRIBUTING.md, README.md, README.txt, LICENSE.txt, LICENSE, INFO_BIN, and INFO_SRC. (Prior to Connector/C++ 8.0.14, the information file is BUILDINFO.txt rather than INFO_BIN and INFO_SRC.)

- Installation on Windows
- Installation on Linux
- Installation on macOS
- Installation on Solaris
- Installation Using a tar or Zip Package

Installation on Windows

<table>
<thead>
<tr>
<th>Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>On Windows platforms, Commercial and Community Connector/C++ distributions require the Visual C++ Redistributable for Visual Studio. The Redistributable is available at the Visual Studio Download Center; install it before installing Connector/C++. For information about which VC++ Restributable versions are acceptable, see Platform Support and Prerequisites.</td>
</tr>
</tbody>
</table>

These methods of installing binary distributions are available on Windows:

- **MySQL Installer.** The simplest and recommended method of installing Connector/C++ on Windows platforms is to download MySQL Installer and let it install and configure all the MySQL products on your system. For details, see MySQL Installer for Windows.

- **Windows MSI installer.** As of Connector/C++ 8.0.12, an MSI Installer is available for Windows. To use the MSI Installer (.msi file), launch it and follow the prompts in the screens it presents. The MSI Installer can install components for these connectors:
  - The connector for X DevAPI (including X DevAPI for C).
  - The connector for the legacy JDBC API.

For each connector, there are two components:

- The DLL component includes the connector DLLs and libraries to satisfy runtime dependencies. The DLL component is required to run Connector/C++ application binaries that use the connector.

- The Developer component includes header files, static libraries, and import libraries for DLLs. The Developer component is required to build from source Connector/C++ applications that use the connector.
The MSI Installer requires administrative privileges. It begins by presenting a welcome screen that enables you to continue the installation or cancel it. If you continue the installation, the MSI Installer overview screen enables you to select the type of installation to perform:

- The **Complete** installation installs the DLL and Developer components for both connectors.
- The **Typical** installation installs the DLL component for both connectors.
- The **Custom** installation enables you to specify the installation location and select which components to install. The DLL and Developer components for the X DevAPI connector are preselected, but you can override the selection. The Developer component for a connector cannot be selected without also selecting the connector DLL component.

For all installation types, the MSI Installer performs these actions:

- It checks whether the required [Visual C++ Redistributable for Visual Studio](https://www.visualstudio.com/downloads/) is present. If not, the installer asks you to install it and exits with an error. For information about which VC++ Restributable versions are acceptable, see [Platform Support and Prerequisites](#).  
- It installs documentation files.

**Important**

Prior to Connector/C++ 8.0.13, because the Microsoft Visual C++ 2017 Redistributable installer deletes the Microsoft Visual C++ 2015 Redistributable registry keys that identify its installation, standalone MySQL MSIs may fail to detect the Microsoft Visual C++ 2015 Redistributable if both it and the Microsoft Visual C++ 2017 Redistributable are installed. The solution is to repair the Microsoft Visual C++ 2017 Redistributable via the Windows Control Panel to recreate the registry keys needed for the runtime detection. Unlike the standalone MSIs, MySQL Installer for Windows contains a workaround for the detection problem.

This workaround is unnecessary as of Connector/C++ 8.0.13.

- **Zip archive package without installer.** To install from a Zip archive package (.zip file), see [Installation Using a tar or Zip Package](#).

## Installation on Linux

These methods of installing binary distributions are available on Linux:

- **RPM package.** RPM packages are available for Linux (as of Connector/C++ 8.0.12). The packages are distinguished by their base names (the full names include the Connector/C++ version and suffixes):
  
  - **mysql-connector-c++**: This package provides the shared connector library implementing X DevAPI and X DevAPI for C.
  - **mysql-connector-c++-jdbc**: This package provides the shared legacy connector library implementing the JDBC API.
  - **mysql-connector-c++-devel**: This package installs development files required for building applications that use Connector/C++ libraries provided by the other packages, and static connector libraries. This package depends on the shared libraries provided by the other packages. It cannot be installed by itself without the other two packages.
Installation on macOS

- **Debian package.** Debian packages are available for Linux (as of Connector/C++ 8.0.14). The packages are distinguished by their base names (the full names include the Connector/C++ version and suffixes):
  
  - `libmysqlcppconn8-1`: This package provides the shared connector library implementing X DevAPI and X DevAPI for C.
  
  - `libmysqlcppconn7`: This package provides the shared legacy connector library implementing the JDBC API.
  
  - `libmysqlcppconn-dev`: This package installs development files required for building applications that use Connector/C++ libraries provided by the other packages, and static connector libraries. This package depends on the shared libraries provided by the other packages. It cannot be installed by itself without the other two packages.

- **Compressed tar file.** To install from a compressed tar file (.tar.gz file), see Installation Using a tar or Zip Package.

**Installation on macOS**

These methods of installing binary distributions are available on macOS:

- **DMG package.** DMG (disk image) packages for macOS are available as of Connector/C++ 8.0.12. A DMG package provides shared and static connector libraries implementing X DevAPI and X DevAPI for C, and the legacy connector library implementing the JDBC API. The package also includes OpenSSL libraries, public header files, and documentation files.

- **Compressed tar file.** To install from a compressed tar file (.tar.gz file), see Installation Using a tar or Zip Package.

**Installation on Solaris**

**Important**

The installation packages have a dependency on the Oracle Developer Studio 12.6 Runtime Libraries, which must be installed before you run the MySQL installation package. See the download options for Oracle Developer Studio here. The installation package enables you to install the runtime libraries only instead of the full Oracle Developer Studio; see instructions in Installing Only the Runtime Libraries on Oracle Solaris 11.

These methods of installing binary distributions are available on Solaris:

- **Compressed tar file.** To install from a compressed tar file (.tar.gz file), see Installation Using a tar or Zip Package.

**Installation Using a tar or Zip Package**

Connector/C++ binary distributions are available for several platforms, packaged in the form of compressed tar files or Zip archives, denoted here as `PACKAGE.tar.gz` or `PACKAGE.zip`.

To unpack a compressed tar file, use this command in the intended installation directory:

```
tar xzvf PACKAGE.tar.gz
```

To install from a Zip archive package (.zip file), use WinZip or another tool that can read .zip files to unpack the file into the location of your choosing.
Chapter 4 Installing Connector/C++ from Source

Table of Contents

4.1 Source Installation System Prerequisites .............................................................. 11
4.2 Obtaining and Unpacking a Connector/C++ Source Distribution ................................ 12
4.3 Installing Connector/C++ from Source ................................................................... 13
4.4 Connector/C++ Source-Configuration Options ....................................................... 16

This chapter describes how to install Connector/C++ using a source distribution or a copy of the Git source repository.

4.1 Source Installation System Prerequisites

To install Connector/C++ from source, the following system requirements must be satisfied:

- **Build Tools**
- **MySQL Client Library**
- **Boost C++ Libraries**
- **SSL Support**

**Build Tools**

You must have the cross-platform build tool **CMake** (3.0 or higher).

You must have a C++ compiler that supports C++11.

**MySQL Client Library**

To build Connector/C++ from source, the MySQL C API client library may be required:

- Building the JDBC connector requires a client library from MySQL 8.0 (8.0.11 or higher) or MySQL 5.7 (5.7.9 or higher). This occurs when Connector/C++ is configured with the `WITH_JDBC CMake` option enabled to include the JDBC connector.
- For Connector/C++ built without the JDBC connector, the client library is not needed.

Typically, the MySQL client library is installed when MySQL is installed. However, check your operating system documentation for other installation options.

To specify where to find the client library, set the `MYSQL_DIR CMake` option appropriately at configuration time as necessary (see Section 4.4, “Connector/C++ Source-Configuration Options”).

**Boost C++ Libraries**

To compile Connector/C++ the Boost C++ libraries are needed only if you build the legacy JDBC API or if the version of the C++ standard library on your system does not implement the UTF8 converter (`codecvt_utf8`).

If the Boost C++ libraries are needed, Boost 1.59.0 or newer must be installed. To obtain Boost and its installation instructions, visit the official Boost site.
SSL Support

After Boost is installed, use the `WITH_BOOST` CMake option to indicate where the Boost files are located (see Section 4.4, “Connector/C++ Source-Configuration Options”):

```
cmake [other_options] -DWITH_BOOST=/usr/local/boost_1_59_0
```

Adjust the path as necessary to match your installation.

SSL Support

Use the `WITH_SSL` CMake option to specify which SSL library to use when compiling Connector/C++. OpenSSL 1.0.x or higher is required. As of Connector/C++ 8.0.18, it is possible to compile against OpenSSL 1.1.

For more information about `WITH_SSL` and SSL libraries, see Section 4.4, “Connector/C++ Source-Configuration Options”.

4.2 Obtaining and Unpacking a Connector/C++ Source Distribution

To obtain a Connector/C++ source distribution, visit the Connector/C++ downloads page. Alternatively, clone the Connector/C++ Git source repository.

A Connector/C++ source distribution is packaged as a compressed tar file or Zip archive, denoted here as `PACKAGE.tar.gz` or `PACKAGE.zip`. A source distribution in tar file or Zip archive format can be used on any supported platform.

The distribution when unpacked includes an `INFO_SRC` file that provides information about the product version and the source repository from which the distribution was produced. The distribution also includes other documentation files such as those listed in Chapter 3, Installing Connector/C++ from a Binary Distribution.

To unpack a compressed tar file, use this command in the intended installation directory:

```
tar zxfv PACKAGE.tar.gz
```

After unpacking the distribution, build it using the appropriate instructions for your platform later in this chapter.

To install from a Zip archive package (.zip file), use WinZip or another tool that can read .zip files to unpack the file into the location of your choosing. After unpacking the distribution, build it using the appropriate instructions for your platform later in this chapter.

To clone the Connector/C++ code from the source code repository located on GitHub at https://github.com/mysql/mysql-connector-cpp, use this command:

```
git clone https://github.com/mysql/mysql-connector-cpp.git
```

That command should create a `mysql-connector-cpp` directory containing a copy of the entire Connector/C++ source tree.

The `git clone` command sets the sources to the `master` branch, which is the branch that contains the latest sources. Released code is in the `8.0` branch (the `8.0` branch contains the same sources as the `master` branch). If necessary, use `git checkout` in the source directory to select the desired branch. For example, to build Connector/C++ 8.0:

```
    cd mysql-connector-cpp
    git checkout 8.0
```

After cloning the repository, build it using the appropriate instructions for your platform later in this chapter.
Installing Connector/C++ from Source

After the initial checkout operation to get the source tree, run `git pull` periodically to update your source to the latest version.

### 4.3 Installing Connector/C++ from Source

To install Connector/C++ from source, verify that your system satisfies the requirements outlined in Section 4.1, “Source Installation System Prerequisites”.

- Configuring Connector/C++
- Building Connector/C++
- Installing Connector/C++
- Verifying Connector/C++ Functionality

#### Configuring Connector/C++

Use `CMake` to configure and build Connector/C++. Only out-of-source-builds are supported, so create a directory to use for the build and change location into it. Then configure the build using this command, where `concpp_source` is the directory containing the Connector/C++ source code:

```bash
cmake concpp_source
```

It may be necessary to specify other options on the configuration command. Some examples:

- By default, these installation locations are used:
  - `/usr/local/mysql/connector-c++-8.0` (Unix and Unix-like systems)
  - `User_home/MySQL/"MySQL Connector C++ 8.0"` (Windows)

  To specify the installation location explicitly, use the `CMAKE_INSTALL_PREFIX` option:

  ```bash
  -DCMAKE_INSTALL_PREFIX=path_name
  ```

- On Windows, you can use the `-G` and `-A` options to select a particular generator:

  ```bash
  -G "Visual Studio 16" -A x64 (64-bit builds)
  -G "Visual Studio 16" -A Win32 (32-bit builds)
  ```

  Consult the `CMake` manual or check `cmake --help` to find out which generators are supported by your `CMake` version. (However, it may be that your version of `CMake` supports more generators than can actually be used to build Connector/C++.)

- If the Boost C++ libraries are needed, use the `WITH_BOOST` option to specify their location:

  ```bash
  -DWITH_BOOST=path_name
  ```

- By default, the build creates dynamic (shared) libraries. To build static libraries, enable the `BUILD_STATIC` option:

  ```bash
  -DBUILD_STATIC=ON
  ```

- By default, the legacy JDBC connector is not built. If you plan to build this connector, an additional `git` command is needed to perform submodule initialization (do this in the top-level source directory):

  ```bash
  git submodule update --init
  ```
To include the JDBC connector in the build, enable the `WITH_JDBC` option:

```
-DWITH_JDBC=ON
```

**Note**

If you configure and build the test programs later, use the same *CMake* options to configure them as the ones you use to configure Connector/C++ (`-G WITH_BOOST, BUILD_STATIC`, and so forth). Exceptions: Path name arguments will differ, and you need not specify `CMAKE_INSTALL_PREFIX`.

For information about *CMake* configuration options, see Section 4.4, "Connector/C++ Source-Configuration Options".

**Building Connector/C++**

After configuring the Connector/C++ distribution, build it using this command:

```
cmake --build . --config build_type
```

The `--config` option is optional. It specifies the build configuration to use, such as `Release` or `Debug`. If you omit `--config`, the default is `Debug`.

**Important**

If you specify the `--config` option on the preceding command, specify the same `--config` option for later steps, such as the steps that install Connector/C++ or that build test programs.

If the build is successful, it creates the connector libraries in the build directory. (For Windows, look for the libraries in a subdirectory with the same name as the `build_type` value specified for the `--config` option.)

- If you build dynamic libraries, they have these names:
  - `libmysqlcppconn8.so.1` (Unix)
  - `libmysqlcppconn8.1.dylib` (macOS)
  - `mysqlcppconn8-1-vs14.dll` (Windows)
- If you build static libraries, they have these names:
  - `libmysqlcppconn8-static.a` (Unix, macOS)
  - `mysqlcppconn8-static.lib` (Windows)

If you enabled the `WITH_JDBC` option to include the legacy JDBC connector in the build, the following additional library files are created.

- If you build legacy dynamic libraries, they have these names:
  - `libmysqlcppconn.so.7` (Unix)
  - `libmysqlcppconn.7.dylib` (macOS)
  - `mysqlcppconn-7-vs14.dll` (Windows)
Installing Connector/C++

To install Connector/C++, use this command:

```bash
cmake --build . --target install --config build_type
```

Verifying Connector/C++ Functionality

To verify connector functionality, build and run one or more of the test programs included in the `testapp` directory of the source distribution. Create a directory to use and change location into it. Then issue the following commands:

```bash
cmake [other_options] -DWITH_CONCPP=concpp_install concpp_source/testapp
cmake --build . --config=build_type
```

`WITH_CONCPP` is an option used only to configure the test application. `other_options` consists of the options that you used to configure Connector/C++ itself (`-G, WITH_BOOST, BUILD_STATIC, and so forth`). `concpp_source` is the directory containing the Connector/C++ source code, and `concpp_install` is the directory where Connector/C++ is installed.

The preceding commands should create the `devapi_test` and `xapi_test` programs in the `run` directory of the build location. If you enable `WITH_JDBC` when configuring the test programs, the build also creates the `jdbc_test` program.

Before running test programs, ensure that a MySQL server instance is running with X Plugin enabled. The easiest way to arrange this is to use the `mysql-test-run.pl` script from the MySQL distribution. For MySQL 8.0, X Plugin is enabled by default, so invoke this command in the `mysql-test` directory of that distribution:

```bash
perl mysql-test-run.pl --start-and-exit
```

For MySQL 5.7, X Plugin must be enabled explicitly, so add an option to do that:

```bash
perl mysql-test-run.pl --start-and-exit --mysqld=--plugin-load=mysqlx
```

The command should start a test server instance with X Plugin enabled and listening on port 13009 instead of its standard port (33060).

Now you can run one of the test programs. They accept a connection-string argument, so if the server was started as just described, you can run them like this:

```bash
run/devapi_test mysqlx://root@127.0.0.1:13009
run/xapi_test mysqlx://root@127.0.0.1:13009
```

The connection string assumes availability of a `root` user account without any password and the programs assume that there is a `test` schema available (assumptions that hold for a server started using `mysql-test-run.pl`).

To test `jdbc_test`, you need a MySQL server, but X Plugin is not required. Also, the connection options must be in the form specified by the JDBC API. Pass the user name as the second argument. For example:
Connector/C++ Source-Configuration Options

4.4 Connector/C++ Source-Configuration Options

Connector/C++ recognizes the `CMake` options described in this section.

Table 4.1 Connector/C++ Source-Configuration Option Reference

<table>
<thead>
<tr>
<th>Formats</th>
<th>Description</th>
<th>Default</th>
<th>Introduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD_STATIC</td>
<td>Whether to build a static library</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>BUNDLE_DEPENDENCIES</td>
<td>Whether to bundle external dependency libraries with the connector</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>CMAKE_BUILD_TYPE</td>
<td>Type of build to produce</td>
<td>Debug</td>
<td></td>
</tr>
<tr>
<td>CMAKE_INSTALL_DOCDIR</td>
<td>Documentation installation directory</td>
<td></td>
<td>8.0.14</td>
</tr>
<tr>
<td>CMAKE_INSTALL_INCLUDEDIR</td>
<td>Header file installation directory</td>
<td></td>
<td>8.0.14</td>
</tr>
<tr>
<td>CMAKE_INSTALL_LIBDIR</td>
<td>Library installation directory</td>
<td></td>
<td>8.0.14</td>
</tr>
<tr>
<td>CMAKE_INSTALL_PREFIX</td>
<td>Installation base directory</td>
<td>/usr/local</td>
<td></td>
</tr>
<tr>
<td>MAINTAINER_MODE</td>
<td>For internal use only</td>
<td>OFF</td>
<td>8.0.12</td>
</tr>
<tr>
<td>MYSQLCLIENT_STATIC_BINDING</td>
<td>Whether to link to the shared MySQL client library</td>
<td>ON</td>
<td>8.0.16</td>
</tr>
<tr>
<td>MYSQLCLIENT_STATIC_LINKING</td>
<td>Whether to statically link to the MySQL client library</td>
<td>ON</td>
<td>8.0.16</td>
</tr>
<tr>
<td>MYSQL_CONFIG_EXECUTABLE</td>
<td>Path to the mysql_config program</td>
<td>$(MYSQL_DIR)/bin/mysql_config</td>
<td></td>
</tr>
<tr>
<td>MYSQL_DIR</td>
<td>MySQL Server installation directory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATIC_MSVCR</td>
<td>Use the static runtime library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WITH_BOOST</td>
<td>The Boost source directory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WITH_DOC</td>
<td>Whether to generate Doxygen documentation</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>WITH_JDBC</td>
<td>Whether to build legacy JDBC library</td>
<td>OFF</td>
<td>8.0.7</td>
</tr>
<tr>
<td>WITH_SSL</td>
<td>Type of SSL support</td>
<td>system</td>
<td>8.0.7</td>
</tr>
</tbody>
</table>

- `-DBUILD_STATIC=bool`

By default, dynamic (shared) libraries are built. If this option is enabled, static libraries are built instead.

- `-DBUNDLE_DEPENDENCIES=bool`

This is an internal option used for creating Connector/C++ distribution packages.

- `-DCMAKE_BUILD_TYPE=type`

The type of build to produce:

- **Debug**: Disable optimizations and generate debugging information. This is the default.
• **Release**: Enable optimizations.

• **RelWithDebInfo**: Enable optimizations and generate debugging information.

• **-DCMAKE_INSTALL_DOCDIR=dir_name**

  The documentation installation directory, relative to `CMAKE_INSTALL_PREFIX`. If not specified, the default is to install in `CMAKE_INSTALL_PREFIX`.

  This option requires that **WITH_DOC** be enabled.

  This option was added in Connector/C++ 8.0.14.

• **-DCMAKE_INSTALL_INCLUDEDIR=dir_name**

  The header file installation directory, relative to `CMAKE_INSTALL_PREFIX`. If not specified, the default is `include`.

  This option was added in Connector/C++ 8.0.14.

• **-DCMAKE_INSTALL_LIBDIR=dir_name**

  The library installation directory, relative to `CMAKE_INSTALL_PREFIX`. If not specified, the default is `lib64` or `lib`.

  This option was added in Connector/C++ 8.0.14.

• **-DCMAKE_INSTALL_PREFIX=dir_name**

  The installation base directory (where to install Connector/C++).

• **-DMAINTAINER_MODE=bool**

  This is an internal option used for creating Connector/C++ distribution packages. It was added in Connector/C++ 8.0.12.

• **-DMYSQLCLIENT_STATIC_BINDING=bool**

  Whether to link to the shared MySQL client library. This option is used only if `MYSQLCLIENT_STATIC_LINKING` is disabled to enable dynamic linking of the MySQL client library. In that case, if `MYSQLCLIENT_STATIC_BINDING` is enabled (the default), Connector/C++ is linked to the shared MySQL client library. Otherwise, the shared MySQL client library is loaded and mapped at runtime.

  This option applies only if you are building the legacy JDBC connector (that is, only if `WITH_JDBC` is enabled). It was added in Connector/C++ 8.0.16.

• **-DMYSQLCLIENT_STATIC_LINKING=bool**

  Whether to link statically to the MySQL client library. The default is ON (use static linking to the client library). Disabling this option enables dynamic linking to the client library.

  This option applies only if you are building the legacy JDBC connector (that is, only if `WITH_JDBC` is enabled). It was added in Connector/C++ 8.0.16.

• **-DMYSQL_CONFIG_EXECUTABLE=file_name**

  The path to the `mysql_config` program.
On non-Windows systems, CMake checks to see whether `MYSQL_CONFIG_EXECUTABLE` is set. If not, CMake tries to locate `mysql_config` in the default locations.

This option applies only if you are building the legacy JDBC connector (that is, only if `WITH_JDBC` is enabled).

- **-DMYSQL_DIR=dir_name**
  The directory where MySQL is installed.
  This option applies only if you are building the legacy JDBC connector (that is, only if `WITH_JDBC` is enabled).

- **-DSTATIC_MSVCRT=bool**
  (Windows only) Use the static runtime library (the `/MT*` compiler option). This option might be necessary if code that uses Connector/C++ also uses the static runtime library.

- **-DWITH_BOOST=dir_name**
  The directory where the Boost sources are installed.

- **-DWITH_DOC=bool**
  Whether to enable generating the Doxygen documentation. As of Connector/C++ 8.0.16, enabling this option also causes the Doxygen documentation to be built by the all target.

- **-DWITH_JDBC=bool**
  Whether to build the legacy JDBC connector. This option is disabled by default. If it is enabled, Connector/C++ 8.0 applications can use the legacy JDBC API, just like Connector/C++ 1.1 applications.

- **-DWITH_SSL=ssl_type|path_name**
  This option specifies which SSL library to use when compiling Connector/C++. The option value indicates the type of SSL support to include or the path name to the SSL installation to use:

  - **ssl_type** can be one of the following values:
    - **system**: Use the system OpenSSL library.

    When running an application that is linked to the connector dynamic library, the OpenSSL libraries on which the connector depends should be correctly found if they are placed in the file system next to the connector library. The application should also work when the OpenSSL libraries are installed at the standard system-wide locations. This assumes that the version of OpenSSL is as expected by Connector/C++.

    Compressed tar files or Zip archive distributions for for Windows, Linux, and macOS should contain the required OpenSSL libraries in the same location as the connector library.

    Except for Windows, it should be possible to run an application linked to the connector dynamic library when the connector library and the OpenSSL libraries are placed in a nonstandard location, provided that these locations were stored as runtime paths when building the application (gcc -rpath option).

    For Windows, an application that is linked to the connector shared library can be run only if the connector library and the OpenSSL libraries are stored either:
• In the Windows system folder

• In the same folder as the application

• In a folder listed in the PATH environment variable

If the application is linked to the connector static library, it remains true that the required OpenSSL libraries must be found in one of the preceding locations.

• path_name is the path name to the SSL installation to use. It should be the path to the installed OpenSSL library, and must point to a directory containing a lib subdirectory with OpenSSL libraries that are already built.

Specifying a path name for the OpenSSL installation can be preferable to using the ssl_type value of system because it can prevent CMake from detecting and using an older or incorrect OpenSSL version installed on the system.
Chapter 5 Building Connector/C++ Applications

Table of Contents

5.1 Building Connector/C++ Applications: General Considerations ........................................... 21
5.2 Building Connector/C++ Applications: Platform-Specific Considerations .......................... 27
  5.2.1 Windows Notes .............................................................................................................. 27
  5.2.2 macOS Notes ............................................................................................................. 32
  5.2.3 Solaris Notes .............................................................................................................. 32

This chapter provides guidance on building Connector/C++ applications:

• General considerations for building Connector/C++ applications successfully. See Section 5.1, “Building Connector/C++ Applications: General Considerations”.

• Information about building Connector/C++ applications that applies to specific platforms such as Windows, macOS, and Solaris. See Section 5.2, “Building Connector/C++ Applications: Platform-Specific Considerations”.

For discussion of the programming interfaces available to Connector/C++ applications, see Chapter 1, Introduction to Connector/C++.

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

• 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:
C++11 Support

- 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 <mysql/xdevapi.h>
  ```

- For applications that use X DevAPI for C:
  ```
  #include <mysql/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 <jdbc/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.

Boost Header Files

The Boost header files are needed under these circumstances:

• To compile Connector/C++ applications that use the legacy JDBC API.

• 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).

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”.

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 \( \text{\texttt{A}} \) in the library name represents the ABI version:

- \( \text{libmysqlcppconn8.so.A} \) (Unix)
- \( \text{libmysqlcppconn8.A.dylib} \) (macOS)
- \( \text{mysqlcppconn8-A-vsNN.dll, with import library vsNN/mysqlcppconn8.lib} \) (Windows)

For the legacy JDBC API, the dynamic libraries are named as follows, where \( \text{\texttt{B}} \) in the library name represents the ABI version:

- \( \text{libmysqlcppconn.so.B} \) (Unix)
- \( \text{libmysqlcppconn.B.dylib} \) (macOS)
- \( \text{mysqlcppconn-B-vsNN.dll, with import library vsNN/mysqlcppconn-static.lib} \) (Windows)

On Windows, the \( \text{vsNN} \) value in library names depends on the MSVC toolchain version used to build the libraries. (Connector/C++ libraries provided by Oracle use \( \text{vs14} \), 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 \( \text{-lmysqlcppconn8} \) to the linker options. To build code that uses the legacy JDBC API, add \( \text{-lmysqlcppconn} \).

You must also indicate whether to use the 64-bit or 32-bit libraries by specifying the appropriate library directory. Use an \( \text{-L} \) linker option to specify \( \text{SMYSQL_CONCPP_DIR/lib64} \) (64-bit libraries) or \( \text{SMYSQL_CONCPP_DIR/lib} \) (32-bit libraries), where \( \text{SMYSQL_CONCPP_DIR} \) 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 \text{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 -I $(MYSQL_CONCPP_DIR)/lib64
LDDLIBS = -lmysqlcppconn8
CXXFLAGS = -std=c++11
app : app.cc
```

With that Makefile, the command \texttt{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 \text{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 -I $(MYSQL_CONCPP_DIR)/lib64
LDDLIBS = -lmysqlcppconn8
app : app.c
```

With that Makefile, the command \texttt{make app} generates the following compiler invocation:
Using the Connector/C++ Static Library

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_CONNCPP_DIR = Connector/C++ installation location
CPPFLAGS = -I $(MYSQL_CONNCPP_DIR)/include -L $(MYSQL_CONNCPP_DIR)/lib64
LDDLBS = -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)
- vsNN/mysqlcppconn8-static.lib (Windows)

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

- libmysqlcppconn-static.a (Unix, macOS)
- vsNN/mysqlcppconn-static.lib (Windows)

On Windows, the vsNN 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.)
Using the Connector/C++ Static Library

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 \texttt{-D} option on the compiler invocation command:

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

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

\begin{verbatim}
MYSQL_CONCPP_DIR = Connector/C++ installation location
CPPFLAGS = \texttt{-DSTATIC_CONCPP} \texttt{-I $(MYSQL_CONCPP_DIR)/include}
LDLIBS = \texttt{$(MYSQL_CONCPP_DIR)/lib64/libmysqlcppconn6-static.a -lssl -lcrypto -lpthread}
CXXFLAGS = \texttt{-std=c++11}
app : app.cc
\end{verbatim}

With that \texttt{Makefile}, the command \texttt{make app} generates the following compiler invocation:

\begin{verbatim}
g++ \texttt{-std=c++11 -DSTATIC_CONCPP -I .../include app.cc}
.../lib64/libmysqlcppconn6-static.a -lssl -lcrypto -lpthread -o app
\end{verbatim}

\textbf{Note}

To avoid having the linker report unresolved symbols, the compile line must include the OpenSSL libraries and the \texttt{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 \texttt{socket}, \texttt{rt}, and \texttt{nsl} libraries might be needed.

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

\begin{verbatim}
app : app.c
\end{verbatim}

With that \texttt{Makefile}, the command \texttt{make app} generates the following compiler invocation:

\begin{verbatim}
cc \texttt{-DSTATIC_CONCPP -I .../include app.c}
.../lib64/libmysqlcppconn6-static.a -lssl -lcrypto -lpthread -o app
\end{verbatim}

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

\begin{verbatim}
app : app.c
\end{verbatim}
Building Connector/C++ Applications: Platform-Specific Considerations

CPPFLAGS = -DCPPCONN_PUBLIC_FUNC= -I $(MYSQL_CONCPP_DIR)/include
LDLIBS = $(MYSQL_CONCPP_DIR)/lib64/libmysqlcppconn-static.a -lssl -lcrypto -lpthread

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 -lpthread
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 -lpthread -lstdc++
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.

5.2 Building Connector/C++ Applications: Platform-Specific Considerations

This section discusses platform-specific considerations to keep in mind when building Connector/C++ applications. For general considerations that apply on a platform-independent basis, see Section 5.1, “Building Connector/C++ Applications: General Considerations”.

5.2.1 Windows Notes
This section describes Windows-specific aspects of building Connector/C++ applications. For general application-building information, see Section 5.1, “Building Connector/C++ Applications: General Considerations”.

Developers using Microsoft Windows must satisfy these conditions to build Connector/C++ applications:

- An acceptable version of Microsoft Visual Studio is required.
- Your applications should use the same linker configuration as Connector/C++. For example, use one of /MD, /MDd, /MT, or /MTd.
- Target hosts running client applications must have an acceptable version of the Visual C++ Redistributable for Visual Studio installed.

For information about which Visual Studio and VC++ Restributable versions are acceptable, see Platform Support and Prerequisites.

On Windows, applications can be built in different modes (also called build configurations), which determine the type of the runtime library that is used by the final executable:

- An application can be built in 32-bit or 64-bit mode.
- An application can be built in debug or release mode.
- You can choose between the static runtime (/MT) or dynamic runtime (/MD). Different versions of the MSVC compiler also use different versions of the runtime.

Binary distributions of Connector/C++ 8.0 are available as 64-bit and 32-bit packages, which store libraries in directories named `lib64` and `lib`, respectively. Package names and certain library file and directory names also include `vsNN`. The `vsNN` value in these names depends on the MSVC toolchain version used to build the libraries. This convention enables using libraries built with different versions of MSVC on the same system.

**Note**

The `vsNN` value represents the major version of the MSVC toolchain used to build the libraries. Currently it is `vs14`, which is the toolchain used by MSVC 2015 through 2019.

It is important to ensure that the compiler version and the build mode of an application match the corresponding parameters used to build the connector library, to ensure that the connector and the application use the same runtime library.

Binary distributions of Connector/C++ 8.0 ship libraries built in release mode using the dynamic runtime (/MD). The libraries are compatible with MSVC 2019 and 2017, and code that uses these libraries can be built with either MSVC 2019 or 2017 in /MD mode. To build code in a different mode, first build Connector/C++ from source in that mode (see Section 4.3, “Installing Connector/C++ from Source”), then build your applications using the same mode.

**Note**

When linking dynamically, it is possible to build your code in debug mode even if the connector libraries are built in release mode. However, in that case, it is not possible to step inside connector code during a debug session. To be able to do that, or to build in debug mode while linking statically to the connector, you must first build Connector/C++ in debug mode.
• Linking Connector/C++ to Applications

• Building Connector/C++ Applications with Microsoft Visual Studio

**Linking Connector/C++ to Applications**

Connector/C++ is available as a dynamic or static library to use with your application.

A dynamic connector library name has a .dll extension and is used with an import library that has a .lib extension in the vsNN subdirectory. Thus, a connector dynamic library named mysqlcppconn8-2-vs14.dll is used with an import library named vs14/mysqlcppconn8.lib. The 2 in the dynamic library name is the major ABI version number. (This helps when using compatibility libraries with an old ABI together with new libraries having a different ABI.) The libraries installed on your system may have a different ABI version in their file names. The corresponding static library is named vs14/mysqlcppconn8-static.lib.

A legacy JDBC connector dynamic library named mysqlcppconn-7-vs14.dll is used with an import library named vs14/mysqlcppconn.lib. The corresponding static library is named vs14/mysqlcppconn-static.lib.

The following tables indicate which dynamic and import library files to use for dynamic linking, and which static library files to use for static linking. LIB denotes the Connector/C++ installation library path name. The name of the last path component is lib64 (for 64-bit packages) or lib (for 32-bit packages).

**Table 5.1 Connector/C++ Dynamic and Import Libraries**

<table>
<thead>
<tr>
<th>Connector Type</th>
<th>Dynamic Library File Name</th>
<th>Import Library File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>X DevAPI, X DevAPI for C</td>
<td>LIB/mysqlcppconn8-2-vs14.dll</td>
<td>LIB/vs14/mysqlcppconn8.lib</td>
</tr>
<tr>
<td>JDBC</td>
<td>LIB/mysqlcppconn-7-vs14.dll</td>
<td>LIB/vs14/mysqlcppconn.lib</td>
</tr>
</tbody>
</table>

**Table 5.2 Connector/C++ Static Libraries**

<table>
<thead>
<tr>
<th>Connector Type</th>
<th>Static Library File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>X DevAPI, X DevAPI for C</td>
<td>LIB/vs14/mysqlcppconn8-static.lib</td>
</tr>
<tr>
<td>JDBC</td>
<td>LIB/vs14/mysqlcppconn-static.lib</td>
</tr>
</tbody>
</table>

When building code that uses Connector/C++ libraries, use these guidelines for setting build options in the project configuration:

• As an additional include directory, specify $MYSQL_CPPCONN_DIR/include.

• As an additional library directory, specify $MYSQL_CONCPP_DIR/lib64 (for 64-bit libraries) or $MYSQL_CONCPP_DIR/lib (for 32-bit libraries).

• To use a dynamic library file (.dll extension), link your application with a .lib import library: add vs14/mysqlcppconn8.lib to the linker options, or vs14/mysqlcppconn.lib for legacy code. At runtime, the application must have access to the .dll library.

• To use a static library file (.lib extension), link your application with the library: add vs14/mysqlcppconn8-static.lib, or vs14/mysqlcppconn-static.lib for legacy code.

If linking statically, the linker must find the link libraries (with .lib extension) for the required OpenSSL libraries. If the connector was installed from a binary package provided by Oracle, they are present in the vs14 subdirectory under the main library directory ($MYSQL_CONCPP_DIR/lib64 or $MYSQL_CONCPP_DIR/lib for 32-bit libraries).
Windows Notes

$MYSQL_CONCPP_DIR/lib), and the corresponding OpenSSL .dll libraries are present in the main library directory, next to the connector .dll libraries.

Note
A Windows application that uses the connector dynamic library must be able to locate it at runtime, as well as its dependencies such as OpenSSL. The common way of arranging this is to put the required DLLs in the same location as the executable.

Building Connector/C++ Applications with Microsoft Visual Studio

The initial steps for building an application are the same whether you use the dynamic or static library. Some additional steps vary, depending on whether you use the dynamic or static library.

• Initial Application-Building Steps
• Building with the Dynamic Library
• Building with the Static Library

Initial Application-Building Steps

These steps are the same whether you use the dynamic or static library:


2. In the drop-down list for build configuration on the toolbar, change the configuration from the default option of Debug to Release.

Connector/C++ and Application Build Configuration Must Match

Because the application build configuration must match that of the Connector/C++ it uses, Release is required when using an Oracle-built Connector/C++, which is built in the release configuration. When linking dynamically, it is possible to build your code in debug mode even if the connector libraries are built in release mode. However, in that case, it is not possible to step inside connector code during a debug session. To be able to do that, or to build in debug mode while linking statically to the connector, you must build Connector/C++ from source yourself using the Debug configuration.

3. From the main menu select Project, Properties. This can also be accessed using the hot key ALT + F7.

4. Under Configuration Properties, open the tree view.

5. Select C/C++, General in the tree view.

6. In the Additional Include Directories text field:
   • Add the include/ directory of Connector/C++. This directory should be located within the Connector/C++ installation directory.
   • If Boost is required to build the application, also add the Boost library root directory. See Section 5.1, “Building Connector/C++ Applications: General Considerations”.

7. In the tree view, open Linker, General, Additional Library Directories.
8. In the **Additional Library Directories** text field, add the Connector/C++ library directory. This directory should be located within the Connector/C++ installation directory. The directory name ends with `lib64` (for 64-bit builds) or `lib` (for 32-bit builds).

The remaining steps depend on whether you are building an application to use the Connector/C++ dynamic or static library.

**Building with the Dynamic Library**

To build an application to use the Connector/C++ dynamic library, follow these steps:

1. Open **Linker, Input** in the **Property Pages** dialog.

2. Add the appropriate import library name into the **Additional Dependencies** text field. For example, use `vs14/mysqlcppconn8.lib`, or `vs14/mysqlcppconn.lib` for legacy applications; see Linking Connector/C++ to Applications.

3. Choose the C++ Runtime Library to link to. In the **Property Pages** dialog, open **C++, Code Generation** in the tree view, and then select the appropriate option for **Runtime Library**.

   Link to the dynamic version of the C++ Runtime Library by selecting the `/MD` compiler option. Also, target hosts running the client application must have the **Visual C++ Redistributable for Visual Studio** installed. For information about which VC++ Restributable versions are acceptable, see Platform Support and Prerequisites.

   Do not use the `/MTd` or `/MDd` option if you are using an Oracle-built Connector/C++. For an explanation, see this discussion: Connector/C++ and Application Build Configuration Must Match.

4. Copy the appropriate dynamic library to the same directory as the application executable (see Linking Connector/C++ to Applications). Alternatively, extend the `PATH` environment variable using `SET PATH=%PATH%;C:\path\to\cpp`, or copy the dynamic library to the Windows installation directory, typically `C:\windows`.

   The dynamic library must be in the same directory as the application executable, or somewhere on the system's path, so that the application can access the Connector/C++ dynamic library at runtime.

**Building with the Static Library**

To build an application to use the Connector/C++ static library, follow these steps:

1. Open **Linker, Input** in the **Property Pages** dialog.

2. Add the appropriate static library name into the **Additional Dependencies** text field. For example, use `vs14/mysqlcppconn8-static.lib`, or `vs14/mysqlcppconn-static.lib` for legacy applications; see Linking Connector/C++ to Applications.

3. To compile code that is linked statically with the connector library, define a macro that adjusts API declarations in the header files for usage with the static library. By default, the macro is defined to declare functions to be compatible with an application that calls a DLL.

   In the **Project, Properties** tree view, under **C++, Preprocessor**, enter the appropriate macro into the **Preprocessor Definitions** text field:

   - 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`
4. Choose the C++ Runtime Library to link to. In the Property Pages dialog, open C++, Code Generation in the tree view, and then select the appropriate option for Runtime Library.

Link to the dynamic version of the C++ Runtime Library by selecting the /MD compiler option. Also, target hosts running the client application must have the Visual C++ Redistributable for Visual Studio installed. For information about which VC++ Restributable versions are acceptable, see Platform Support and Prerequisites.

Do not use the /MTd or /MDd option if you are using an Oracle-built Connector/C++. For an explanation, see this discussion: Connector/C++ and Application Build Configuration Must Match.

5.2.2 macOS Notes

This section describes macOS-specific aspects of building Connector/C++ applications. For general application-building information, see Section 5.1, “Building Connector/C++ Applications: General Considerations”.

The binary distribution of Connector/C++ for macOS is compiled using the macOS native clang compiler. For that reason, an application that uses Connector/C++ should be built with the same clang compiler.

The clang compiler can use two different implementations of the C++ runtime library: either the native libc++ or the GNU libstdc++ library. It is important that an application uses the same runtime implementation as Connector/C++ that is, the native libc++. To ensure that, the -stdlib=libc++ option should be passed to the compiler and the linker invocations.

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

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

Binary packages for macOS include OpenSSL libraries that are required by code linked with the connector. These libraries are installed in the same location as the connector libraries and should be found there by the dynamic linker.

5.2.3 Solaris Notes

This section describes Solaris-specific aspects of building Connector/C++ applications. For general application-building information, see Section 5.1, “Building Connector/C++ Applications: General Considerations”.

As of Connector/C++ 8.0.13, it is possible to build Connector/C++ applications on Solaris. This requires the SunPro 5.15 or higher compiler (from Developer Studio 12.6). Earlier versions and building with GCC are not supported.

To use a Connector/C++ package provided by Oracle, application code must be built with SunPro 5.15 or higher under the following options: -m64 -std=c++11. The C++ runtime libraries and atomics library used should be the defaults (-library=stdcpp, -xatomics=studio).
Important

The connector library and any code that uses it depends on the GCC runtime libraries shipped with Oracle Developer Studio 12.6, which must be installed before you run the application. See the download options for Oracle Developer Studio. The installation package enables you to install the runtime libraries only instead of the full Oracle Developer Studio; see instructions in Installing Only the Runtime Libraries on Oracle Solaris 11.

Target hosts running client applications must have the runtime libraries from Developer Studio 12.6 installed.
Chapter 6 Connector/C++ Known Issues

To report bugs, use the MySQL Bug System. See How to Report Bugs or Problems.

For notes detailing the changes in each release of Connector/C++, see MySQL Connector/C++ Release Notes.

• Generally speaking, C++ library binaries are less portable than C library binaries. Issues can be caused by name mangling, different Standard Template Library (STL) versions, and using different compilers and linkers for linking against the libraries than were used for building the library itself.

Even a small change in the compiler version can cause problems. If you obtain error messages that you suspect are related to binary incompatibilities, build Connector/C++ from source, using the same compiler and linker that you use to build and link your application.

Due to variations between Linux distributions, compiler versions, linker versions, and STL versions, it is not possible to provide binaries for every possible configuration. However, Connector/C++ binary distributions include an INFO_BIN file that describes the environment and configuration options used to build the binary versions of the connector libraries. 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.)

• 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.
Chapter 7 Connector/C++ Support

For general discussion of Connector/C++, please use the C/C++ community forum.

To report bugs, use the MySQL Bug System. See How to Report Bugs or Problems.

For notes detailing the changes in each release of Connector/C++, see MySQL Connector/C++ Release Notes.

For Licensing questions, and to purchase MySQL Products and Services, please see http://www.mysql.com/buy-mysql/.
Index

B
BUILD_STATIC option
CMake, 16
BUNDLE_DEPENDENCIES option
CMake, 16

C
CMake
BUILD_STATIC option, 16
BUNDLE_DEPENDENCIES option, 16
CMAKE_BUILD_TYPE option, 16
CMAKE_INSTALL_DOCDIR option, 17
CMAKE_INSTALL_INCLUDEDIR option, 17
CMAKE_INSTALL_LIBDIR option, 17
CMAKE_INSTALL_PREFIX option, 17
MAINTAINER_MODE option, 17
MYSQLCLIENT_STATIC_BINDING option, 17
MYSQLCLIENT_STATIC_LINKING option, 17
MYSQL_CONFIG_EXECUTABLE option, 17
MYSQL_DIR option, 18
STATIC_MSVCRT option, 18
WITH_BOOST option, 18
WITH_DOC option, 18
WITH_JDBC option, 18
WITH_SSL option, 18
CMAKE_BUILD_TYPE option
CMake, 16
CMAKE_INSTALL_DOCDIR option
CMake, 17
CMAKE_INSTALL_INCLUDEDIR option
CMake, 17
CMAKE_INSTALL_LIBDIR option
CMake, 17
CMAKE_INSTALL_PREFIX option
CMake, 17
MAINTAINER_MODE option
CMake, 17
MYSQLCLIENT_STATIC_BINDING option
CMake, 17
MYSQLCLIENT_STATIC_LINKING option
CMake, 17
mysqlcppconn-static.lib, 27
mysqlcppconn.dll, 28
MYSQL_CONFIG_EXECUTABLE option
CMake, 17
MYSQL_DIR option
CMake, 18

S
STATIC_MSVCRT option
CMake, 18

W
WITH_BOOST option
CMake, 18
WITH_DOC option
CMake, 18
WITH_JDBC option
CMake, 18
WITH_SSL option
CMake, 18
Connector/C++, 1

M
MAINTAINER_MODE option
CMake, 17
MYSQLCLIENT_STATIC_BINDING option
CMake, 17
MYSQLCLIENT_STATIC_LINKING option
CMake, 17
mysqlcppconn-static.lib, 27
mysqlcppconn.dll, 28
MYSQL_CONFIG_EXECUTABLE option
CMake, 17
MYSQL_DIR option
CMake, 18