Qt **DOCUMENTATION**

☰

Qt Creator Manual 8.0.2

🔍 Search   Topics ❯

Qt Creator Manual ❯ Launching the Debugger

Launching the Debugger

To start an application from an open project under the control of a debugger, select the 🖿 (**Start Debugging of Startup Project**) button or press **F5**.

Qt Creator checks whether the compiled program is up-to-date, and rebuilds and deploys it if the **Build before deploying** field is set to build the whole project or the application to run and the **Always deploy before running** check box is selected in **Edit** > **Preferences** > **Build & Run** > **General**. To debug the program without deploying it, select **Debug** > **Start Debugging** > **Start Debugging Without Deployment**.

The debugger then takes over and starts the program with suitable parameters.

When using GDB or CDB as debug backend, you can specify additional commands to execute before and after the backend and debugged program are started or attached in **Edit** > **Preferences** > **Debugger** > **GDB** and **CDB**. For more information, see Specifying Debugger Settings.

To allow reading the user's default .gdbinit file on debugger startup, select the **Load .gdbinit file on startup** check box in GDB settings. For more information, see Specifying GDB Settings.

> **Note:** Starting a C++ program in the debugger can take a long time, typically in the range of several seconds to minutes if complex features are used.

Launching the Debugger in Different Modes

The debugger plugin can run the native debuggers in various operating modes depending on where and how the debugged process is started and run. Some of the modes are only available for a particular operating system or platform.

In general, **F5** and the **Start Debugging of Startup Project** button are set up in a way to start the operating mode that is commonly used in a given context. So if the current project is set up as a C++ application using the MinGW toolchain targeting desktop Windows, the GDB engine will be started in Start Internal mode. If the current project is a QML application using C++ plugins, a "mixed" QML/C++ engine will be started, with the C++ parts being handled by GDB and GDB server remote debugging.

Change the run configuration parameters (such as **Run in Terminal**) in the run settings of the project, or select options from the **Debug** > **Start Debugging** menu to select other modes of operation.

The debugger can run in the following modes:

> **Start Internal** to debug applications developed inside Qt Creator such as a Qt based GUI application.
> **Start External** to start and debug processes without a proper Qt Creator project setup, either locally or on a remote machine.
> **Attach** to debug processes already started and running outside Qt Creator, either locally or on a remote machine.
> **Core** to debug crashed processes on Unix.
> **Post-mortem** to debug crashed processes on Windows.

Launching in Start Internal Mode

Start Internal mode is the default start mode for most projects, including all projects using a desktop Qt version and plain C++ projects.

If you need a console window to operate your application, for example because it accepts console input from the user, go to **Projects** >

Qt DOCUMENTATION

by using CDB's native console. Select **Edit** > **Preferences** > **Debugger** > **CDB** > **Use CDB console** to override the console set in the Windows system environment variables. Note that the native console does not prompt on application exit.

To launch the debugger in Start Internal mode, click the **Start Debugging** button for the active project.

You can specify breakpoints before or after launching the debugger. For more information, see Setting Breakpoints.

Launching in Start External Mode

You can debug any executable already present on your local or on a remote machine without using a project. You specify a build and run kit that identifies the device to debug the application on.

While this mode does not strictly require a project to be opened in Qt Creator, opening it makes setting breakpoints and stepping through the code easier.

To start and debug an external application:

1. Select **Debug** > **Start Debugging** > **Start and Debug External Application**.
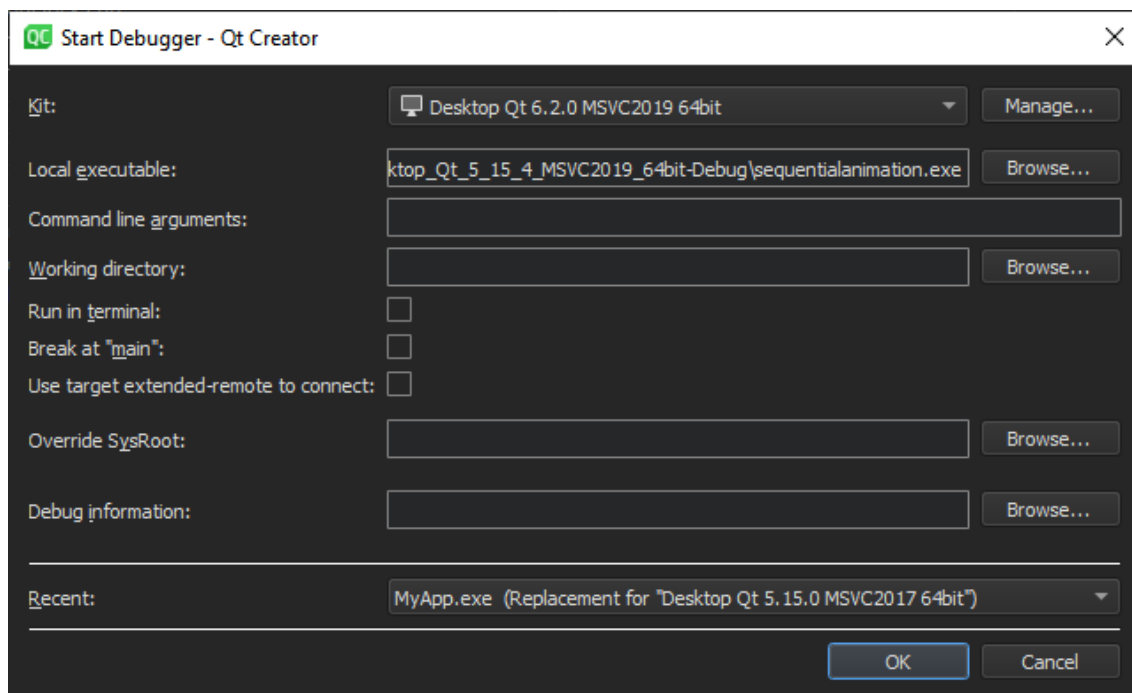

2. In the **Kit** field, select the build and run kit to use for building the project.

3. In the **Local executable** field, specify the path to the application executable on the local machine.

4. In the **Command line arguments** field, specify command line arguments to be passed to the executable.

5. In the **Working directory** field, specify the working directory. It defaults to the directory of the build result.

6. Select the **Run in terminal** check box for console applications.

7. Select the **Break at "main"** check box to stop the debugger at the main function.

8. Select the **Use target extended-remote to connect** check box to create the connection in the `target extended-remote` `mode`. In this mode, when the debugged application exits or you detach from it, the debugger remains connected to the target. You can rerun the application, attach to a running application, or use monitor commands specific to the target. For example, GDB does not exit unless it was invoked using the `--once` option, but you can make it exit by using the `monitor exit` command.

9. In the **Override SysRoot** field, specify the path to the `sysroot` to use instead of the default `sysroot`.

10. In the **Debug information** field, specify the location for storing debug information. You cannot use an empty path.

11. In the **Recent** field, you can select a recent configuration to use.

Launching in Attach Mode

You can attach the debugger to applications that are already running or instruct the debugger to attach to an application when it starts.

Attaching to Running Applications

Qt DOCUMENTATION

1. Select **Debug** > **Start Debugging** > **Attach to Running Application**.
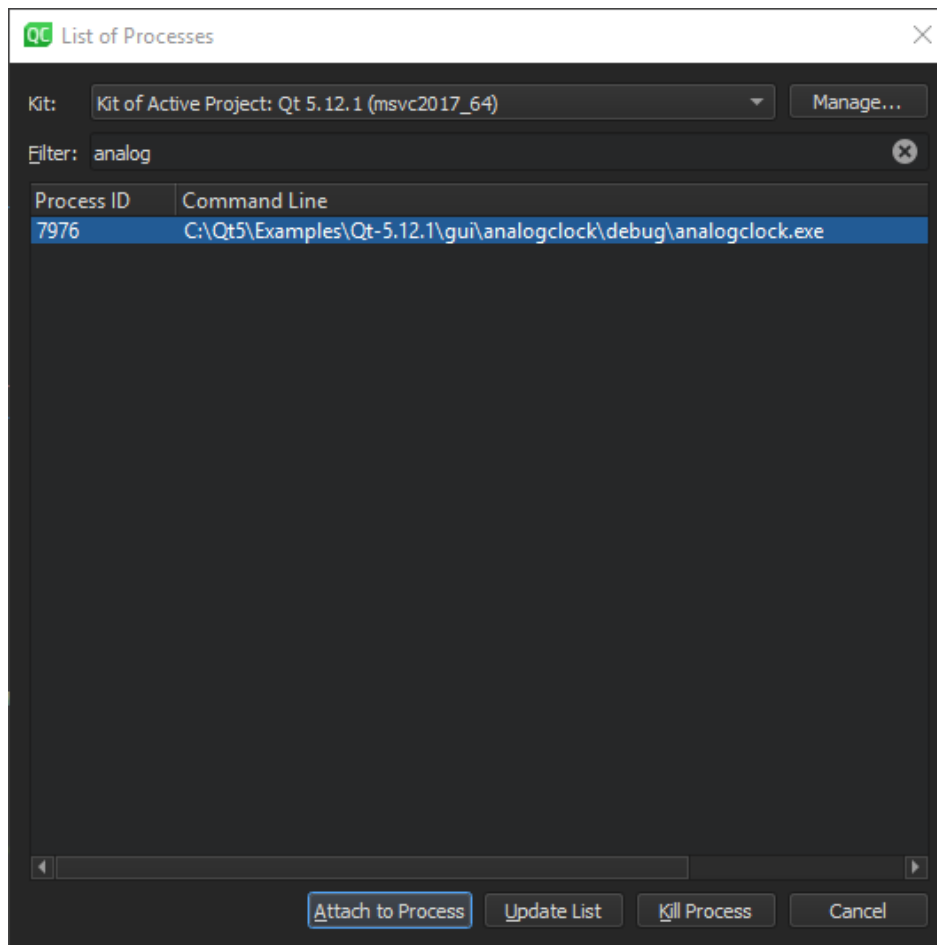
2. In the **Filter** field, enter a string to filter processes by their process ID or name.

3. Select a process in the list, and then select **Attach to Process** to start debugging.

To refresh the list of running processes, select **Update List**.

To terminate the selected process, select **Kill Process**.

While this mode does not strictly require a project to be opened in Qt Creator, opening it makes setting breakpoints and stepping through the code easier.

You can specify breakpoints before or after attaching the debugger to the application. For more information, see Setting Breakpoints.

Attaching to Processes when They Start

To instruct the debugger to watch an application process and to attach to it when it starts:

1. Select **Debug** > **Start Debugging** > **Attach to Unstarted Application**.

Qt DOCUMENTATION

≡
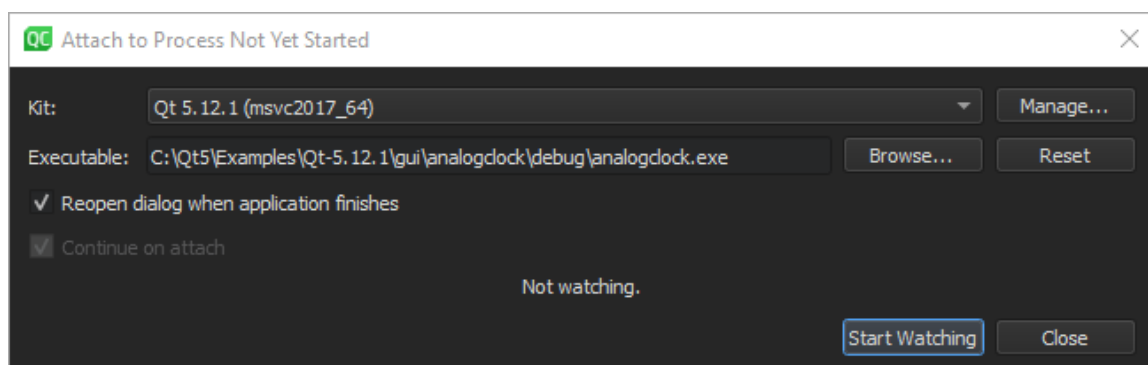
4. Select the **Reopen dialog when application finishes** check box to return to this dialog when the application is closed.

5. Select the **Continue on attach** check box to instruct the debugger to keep the application running after attaching to it.

6. Select **Start Watching** to wait for the application process to start.

Launching in Core Mode

The Core mode is used to inspect *core* files (crash dumps) that are generated from crashed processes on Linux and Unix systems if the system is set up to allow this.

To enable the dumping of core files on a Unix system, enter the following command in the shell from which the application will be launched:

```
ulimit -c unlimited
```

To launch the debugger in the core mode:

1. Select **Debug** > **Start Debugging** > **Load Core File**.
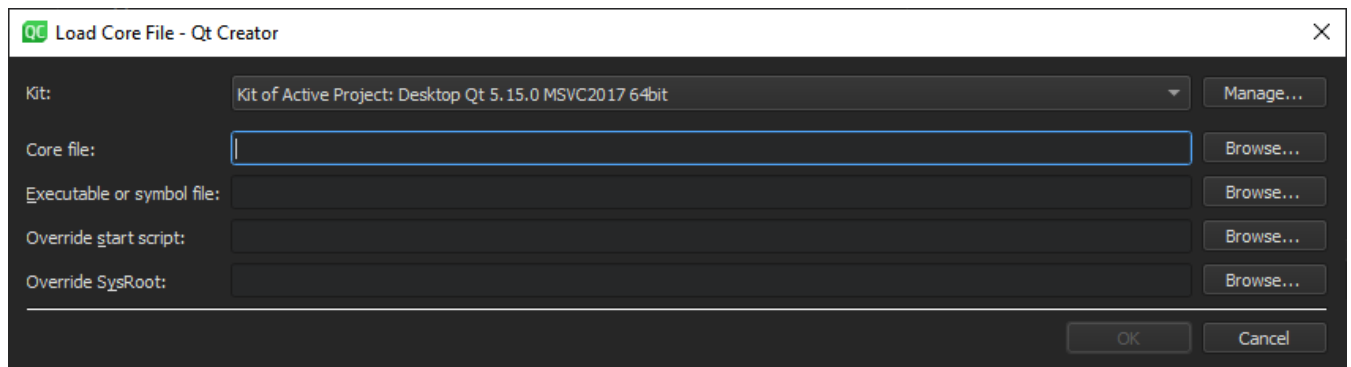
2. In the **Kit** field, select a build and run kit that was used for building the binary for which the core file was created. If the core file stems from a binary not built by Qt Creator or a process not initiated by Qt Creator, select a kit that matches the setup used as closely as possible, in respect to the specified device, tool chain, debugger, and sysroot.

3. In the **Core file** field, specify the core file to inspect.

4. In the **Executable of symbol file** field, specify a file that contains debug information corresponding to the core file. Typically, this is the executable file or a `.debug` file if the debug information is stored separately from the executable.

5. In the **Override start script** field, specify a script file to run instead of the default start script.

6. In the **Override SysRoot** field, specify the path to the `sysroot` to use instead of the default `sysroot`.

Also in this mode, using a properly configured project containing the sources of the crashed program is not strictly necessary, but helpful.

Launching in Post-Mortem Mode

The post-mortem mode is available only on Windows, if you have installed the debugging tools for Windows.

The Qt Creator installation program asks you whether you want to register Qt Creator as a post-mortem debugger. To change the setting, select **Edit** > **Preferences** > **Debugger** > **General** > **Use Qt Creator for post-mortem debugging**.

You can launch the debugger in the post-mortem mode if an application crashes on Windows. Click the **Debug in Qt Creator** button in the error message that is displayed by the Windows operating system.

Remote Debugging

Qt Creator provides very easy access to remote debugging.

Qt DOCUMENTATION

CDB.

While this setup might look daunting, it is mostly invisible to the user of Qt Creator. To start debugging on a remote target with the necessary helper processes running, select the corresponding kit in **Projects** > **Build & Run**, and then select a function to start remote debugging in the **Debug** > **Start Debugging** menu.

Special use cases, such as attaching to a running process on the target, might still require manual setup.

Using GDB

When debugging on a target supported by GDB server, a local GDB process talks to a GDB server running on the remote machine that controls the process to be debugged.

The GDB server process is started on the remote machines by passing a port number and the executable:
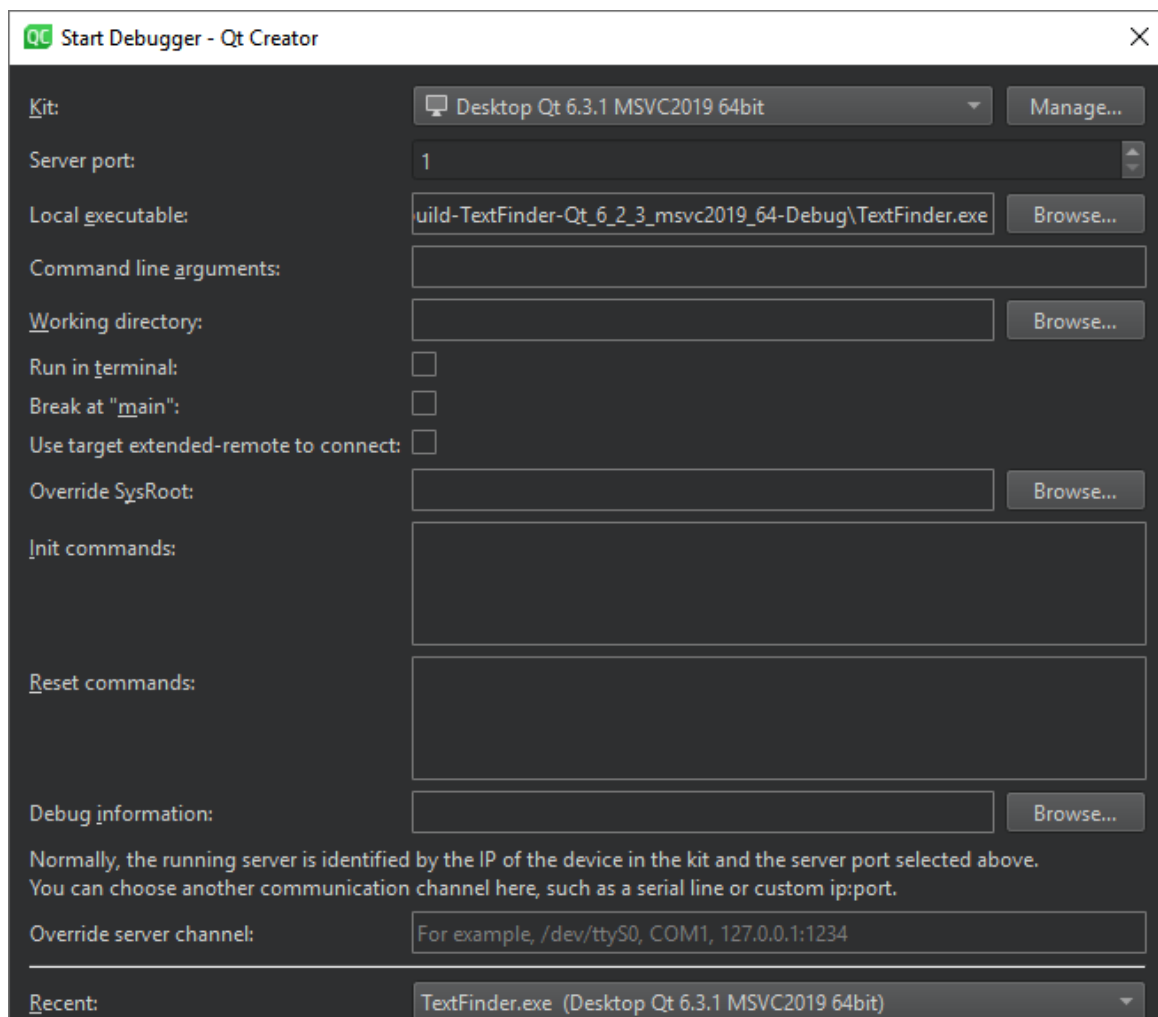
```
gdbserver :1234 <executable>
```

It then typically responds:

```
Process bin/qtcreator created; pid = 5159
Listening on port 1234
```

On the local machine that runs Qt Creator:

1. Select **Debug** > **Start Debugging** > **Attach to Running Debug Server**.

Qt DOCUMENTATION

2. In the **Kit** field, select the build and run kit to use for building the project.

3. In the **Server port** field, enter the name of the remote machine and the port number to use.

4. In the **Local executable** field, specify the path to the application executable on the local machine.

5. In the **Command line arguments** field, specify command line arguments to be passed to the executable.

6. In the **Working directory** field, specify the working directory. It defaults to the directory of the build result.

7. Select the **Run in terminal** check box for console applications.

8. Select the **Break at "main"** check box to stop the debugger at the main function.

9. Select the **Use target extended-remote to connect** check box to create the connection in the `target extended-remote` mode. In this mode, when the debugged application exits or you detach from it, the debugger remains connected to the target. You can rerun the application, attach to a running application, or use monitor commands specific to the target. For example, GDB does not exit unless it was invoked using the `--once` option, but you can make it exit by using the `monitor exit` command.

10. In the **Override SysRoot** field, specify the path to the `sysroot` to use instead of the default `sysroot`.

11. In the **Init commands** field, enter the commands to execute immediately after the connection to a target has been established.

12. In the **Reset commands** field, enter the commands to execute when resetting the connection to a target.

13. In the **Debug information** field, specify the location for storing debug information. You cannot use an empty path.

14. In the **Override server channel** field, specify a communication channel to use, such as a serial line or custom port.

15. In the **Recent** field, you can select a recent configuration to use.

16. Select **OK** to start debugging.

By default, a non-responsive GDB process is terminated after 20 seconds. To increase the timeout in the **GDB timeout** field, select **Edit** > **Preferences** > **Debugger** > **GDB**. For more information about settings that you can specify to manage the GDB process, see Specifying GDB Settings and Specifying Extended GDB Settings.

For more informaton about connecting with `target extended-remote` mode in GDB, see Debugging with GDB: Connecting to a Remote Target.

Using CDB

In remote mode, the local CDB process talks to a CDB process that runs on the remote machine. The process is started with special command line options that switch it into server mode. The remote CDB process must load the Qt Creator CDB extension library that is shipped with Qt Creator:

1. Install the *Debugging Tools for Windows* on the remote machine. The installation folder contains the CDB command line executable (`cdb.exe`).

2. Copy the Qt Creator CDB extension library and the dependencies from the Qt installation directory to a new folder on the remote machine (32 or 64 bit version depending on the version of the Debugging Tools for Windows used):

   > `\lib\qtcreatorcdbext32` (32 bit)

   > `\lib\qtcreatorcdbext64` (64 bit)

3. Set the _NT_DEBUGGER_EXTENSION_PATH environment variable to point to that folder.

4. To use TCP/IP as communication protocol, launch remote CDB as follows:

   ```
   cdb.exe -server tcp:port=1234 <executable>
   ```

5. On the local machine running Qt Creator, select **Debug** > **Start Debugging** > **Attach to Remote CDB Session**.

6. In the **Connection** field enter the connection parameters. For example, for TCP/IP:

   ```
   Server:Port
   ```

If you chose some other protocol, specify one of the alternative formats:

Qt DOCUMENTATION

```
tcp:clicon=Server,port=Port[,password=Password][,ipversion=6]
npipe:server=Server,pipe=PipeName[,password=Password]
com:port=COMPort,baud=BaudRate,channel=COMChannel[,password=Password]
spipe:proto=Protocol,{certuser=Cert|machuser=Cert},server=Server,pipe=PipeName[,password=Password]
ssl:proto=Protocol,{certuser=Cert|machuser=Cert},server=Server,port=Socket[,password=Password]
ssl:proto=Protocol,{certuser=Cert|machuser=Cert},clicon=Server,port=Socket[,password=Password]
```

7. Click **OK** to start debugging.

To specify settings for managing the CDB process, select **Edit** > **Preferences** > **Debugger** > **CDB**. For more information, see Specifying CDB Settings.

< Setting Up Debugger   Interacting with the Debugger >

Qt The Qt Company

Contact Us

Company

About Us
Investors
Newsroom
Careers
Office Locations

Licensing

Terms & Conditions
Open Source
FAQ

Support

Support Services
Professional Services
Partners
Training

For Customers

Support Center
Downloads
Qt Login
Contact Us
Customer Success

Community

Contribute to Qt
Forum
Wiki
Downloads

Qt DOCUMENTATION

Feedback   Sign In

Feedback   Sign In