启动调试器

若要在调试器的控制下从打开的项目启动应用程序，请选择“（**启动启动项目的调试**）”按钮或按 F5。

Qt Creator 检查编译的程序是否是最新的，如果将“部署**前生成**”字段设置为生成整个项目或要运行的应用程序，并且在“编辑>首选项”>“**生成**>**常规**”中选中“运行**前始终部署**”复选框，则重新**生成并部署**该**程序**。若要在不部署程序的情况下调试程序，请选择“**调试>启动调试**>”**启动调试而不进行部署**“。

然后，调试器接管并使用合适的参数启动程序。

使用 GDB 或 CDB 作为调试后端时，可以在 **GDB** 和 **CDB 的“编辑>首选项”>调试器**“中指定要在后端和调试程序启动或附加之前和之后 >执行的其他命令。有关更多信息，请参见指定调试器设置。

若要允许在调试器启动时读取用户的默认 .gdbinit 文件，请在 GDB 设置中选中“**启动时加载 .gdbinit 文件**”复选框。有关更多信息，请参阅指定 GDB 设置。

> **注意：** 在调试器中启动C++程序可能需要很长时间，如果使用复杂的功能，通常需要几秒钟到几分钟的时间。

以不同模式启动调试器

调试器插件可以在各种操作模式下运行本机调试器，具体取决于调试进程的启动和运行位置和方式。某些模式仅适用于特定的操作系统或平台。

通常，F5 和**“启动项目的调试”**按钮的设置方式是启动给定上下文中常用的操作模式。因此，如果使用面向桌面 Windows 的 MinGW 工具链将当前项目设置为C++应用程序，则 GDB 引擎将在“启动内部”模式下启动。如果当前项目是使用C++插件的QML应用程序，则将启动“混合“QML/C++引擎，C++部分由GDB和GDB服务器远程调试处理。

在项目的运行设置中更改运行配置参数（如**“在终端**中运行”），或从**“调试”**>**“启动调试”**菜单中选择选项以选择其他操作模式。

调试器可以在以下模式下运行：

- **启动内部**以调试在 Qt 创建器内部开发的应用程序，例如基于 Qt 的 GUI 应用程序。
- **启动外部**以在本地或远程计算机上启动和调试进程，而无需正确的Qt Creator项目设置。
- **附加到**已在 Qt Creator 外部启动和运行的调试进程，无论是在本地还是在远程计算机上。
- **用于**调试 Unix 上崩溃的进程的核心。
- **事后分析**以在窗口上调试崩溃的进程。

在启动内部模式下启动

启动内部模式是大多数项目的默认启动模式，包括使用桌面 Qt 版本的所有项目和普通C++项目。

如果需要控制台窗口来操作应用程序（例如，因为它接受来自用户的控制台输入），请转到**“项目”**>**“运行设置”**，然后选中“**在终端中运行**”复选框。

如果控制台应用程序未在配置的控制台中正确启动，并且后续连接失败，则可以使用 CDB 的本机控制台诊断问题。选择**“编辑>首选项”**>**调试器**> CDB >**使用 CDB 控制台**覆盖 Windows 系统环境变量中设置的控制台。请注意，本机控制台不会在应用程序退出时提示。

若要在“启动内部”模式下启动调试器，请单击活动项目的“**启动调试**”按钮

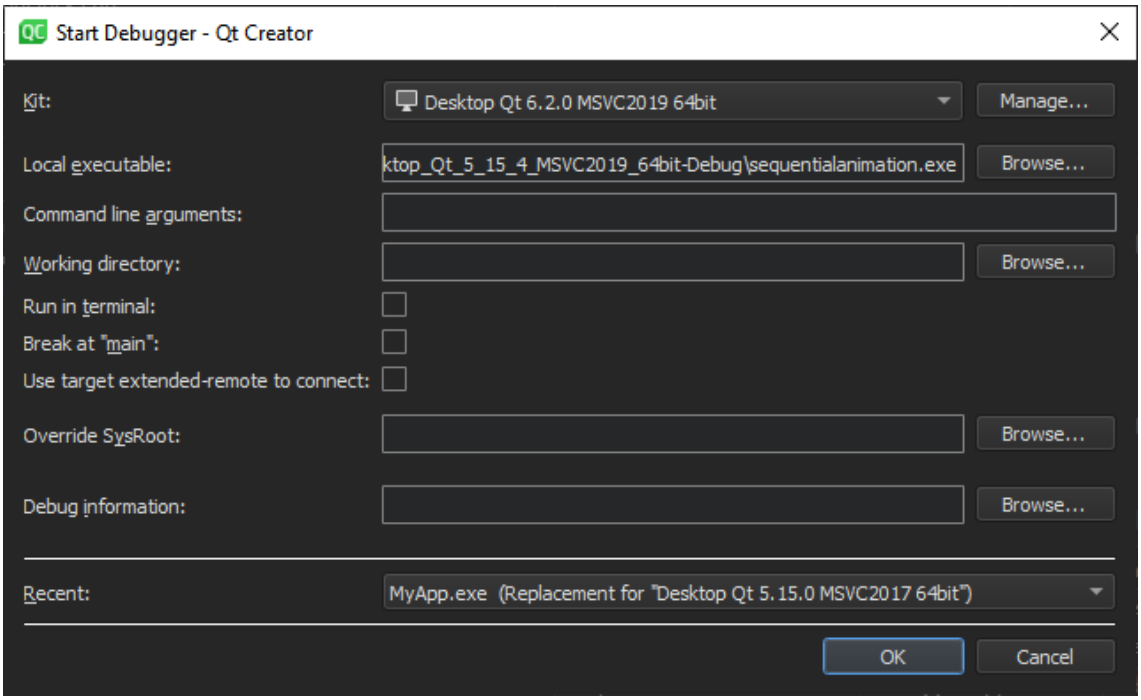
在启动外部模式下启动

您可以调试本地或远程计算机上已存在的任何可执行文件，而无需使用项目。指定一个生成和运行工具包，用于标识要在其上调试应用程序的设备。

虽然此模式并不严格要求在Qt Creator中打开项目，但打开它使设置断点和单步执行代码变得更加容易。

启动和调试外部应用程序：

1. 选择**"调试>启动调试"**>**"启动和调试外部应用程序"**。



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 . 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 option, but you can make it exit by using the command.`target extended-remote mode--oncemonitor exit`
9. In the **Override SysRoot** field, specify the path to the to use instead of the default .`sysrootsysroot`
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.

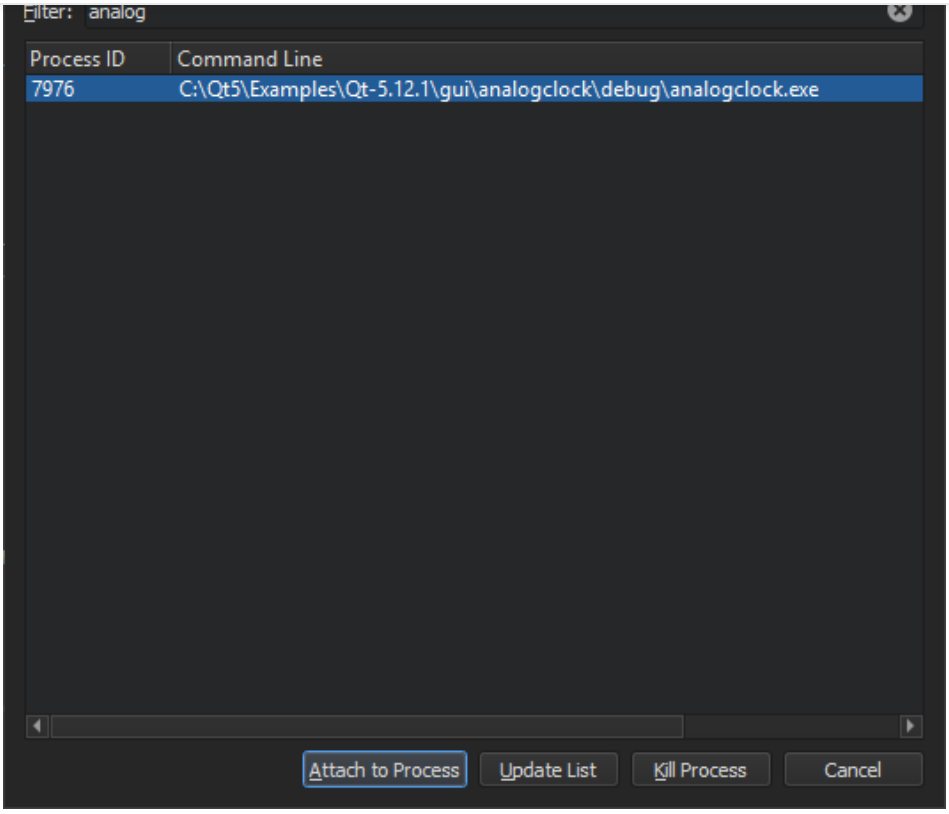
Launching in Attach Mode

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

To attach the debugger to an application already running on your local or on a remote machine:

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**.

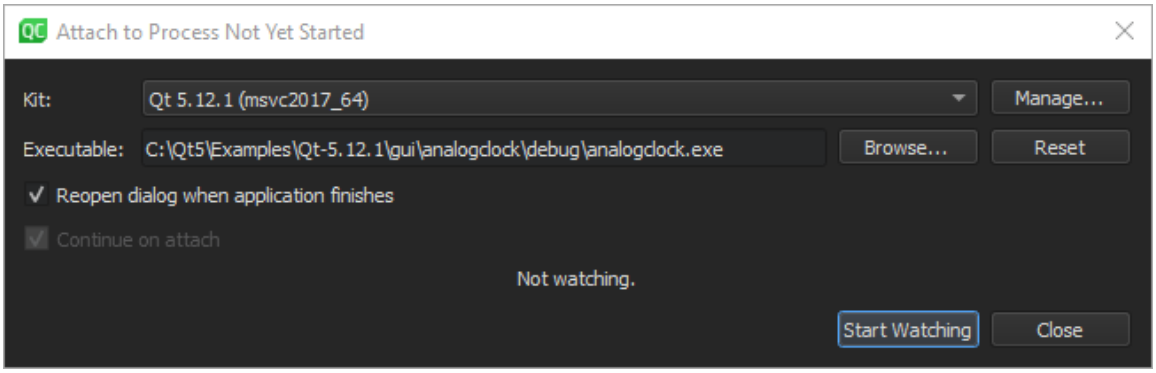
While this mode does not strictly require a project to be opened in Qt Creator, opening it makes setting breakpoints and stepping through the code easier.

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**.



2. In the **Kit** field, select the build and run kit to use for building the project.
3. In the **Executable** field, specify the path to the application executable.
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.

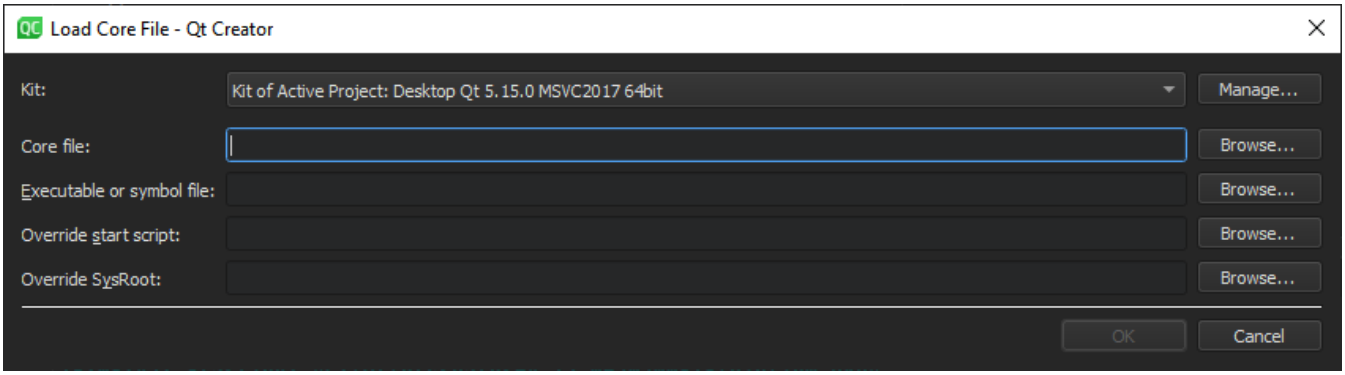
The Core mode is used to inspect *core* files (crash dumps) that are generated from crashed processes on Linux and Unix systems if the system is set up to allow this.

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 file if the debug information is stored separately from the executable. `.debug`
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 to use instead of the default `.sysrootsysroot`

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.

In general, the remote debugging setup consist of a probe running on the remote machine and a counterpart running on the host side. The probe is either integrated into the running process (e.g. for QML debugging) or runs a separate process (e.g. when using GDB server on embedded Linux). The host side typically consists of Qt Creator itself, often with the help of an external process, such as GDB or 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.

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:

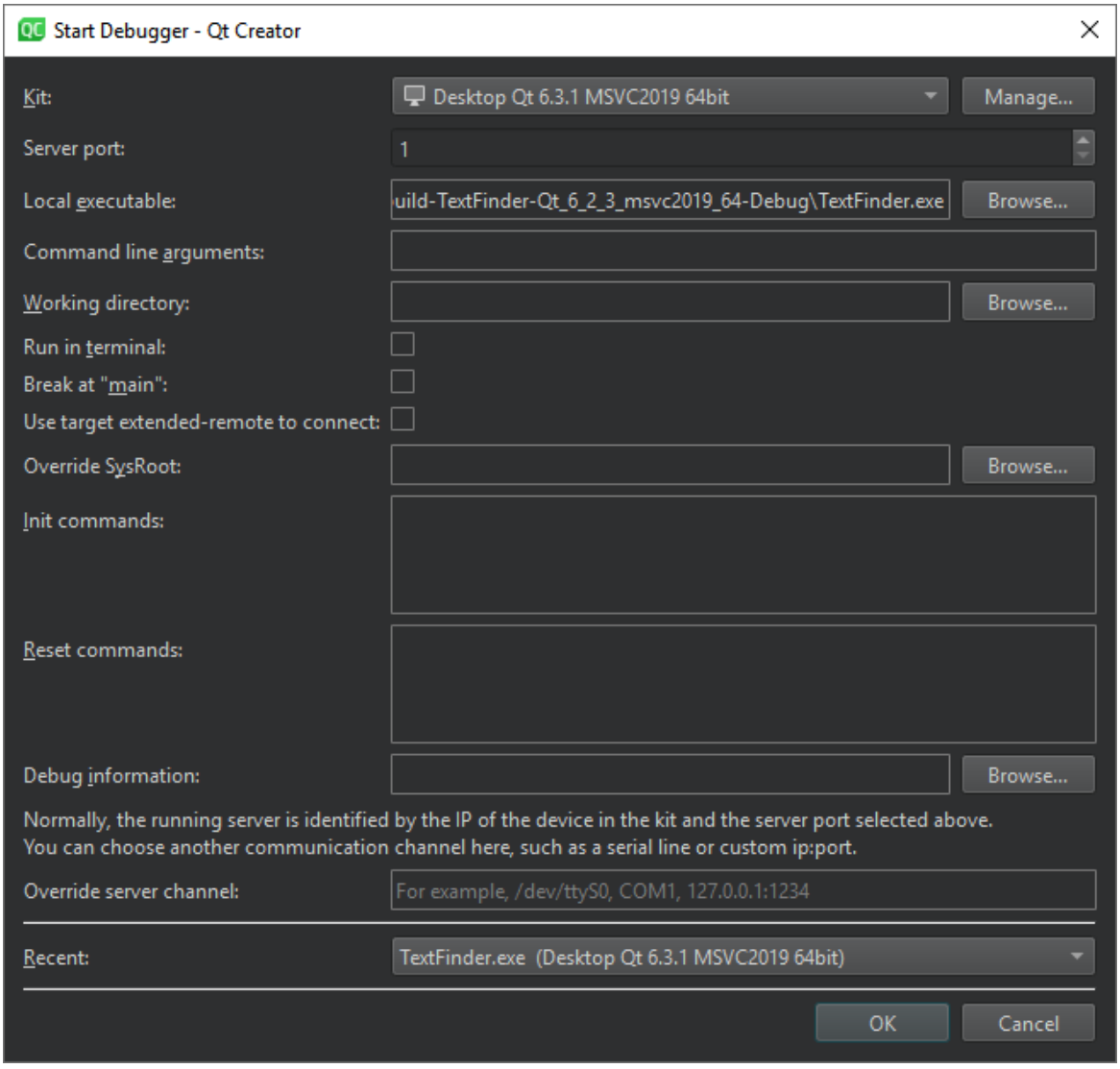
```
gdbserver :1234 <executable>
```

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**.



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.

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 . 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 option, but you can make it exit by using the command.`target extended-remote mode--oncemonitor exit`
10. In the **Override SysRoot** field, specify the path to the to use instead of the default .`sysrootsysroot`
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 mode in GDB, see Debugging with GDB: Connecting to a Remote Target.`target extended-remote`

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:

```
tcp:server=Server,port=Port[,password=Password][,ipversion=6]
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]
```

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 ›