How-tos

How do I:

Switch between modes

Qt Creator uses different modes for different purposes. You can quickly switch between these modes with the

- **Edit** mode **Ctrl+2**
- **Design** mode **Ctrl+3**
- **Debug** mode **Ctrl+4**
- **Projects** mode **Ctrl+5**
- **Help** mode **Ctrl+6**

For more information about Qt Creator modes, see Selecting Modes.

Move between open files

To quickly move between currently open files, press **Ctrl+Tab**.

To move forward in the location history, press **Alt+Right** (**Cmd+Opt+Right** on macOS). To move backward, press **Alt+Left** (**Cmd+Opt+Left** on macOS). For example, if you use the **Locator** to jump to a symbol in the same file, you can jump back to your original location in that file by pressing **Alt+Left**.

Switch to Edit mode

To move to the **Edit** mode and currently active file, press **Esc**.

If you already are in the **Edit** mode:

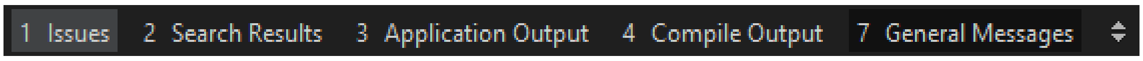
- The first press moves focus to the editor
- The second press closes secondary windows

Find a specific setting

To find specific settings in **Edit** > **Preferences**, use the filter located at the top left of the **Preferences** dialog.

View output

The taskbar provides different views to output from several sources, such as a list of errors and warnings encountered during a build, detailed output from the compiler, status of a program when it is executed, debug output, or search results.



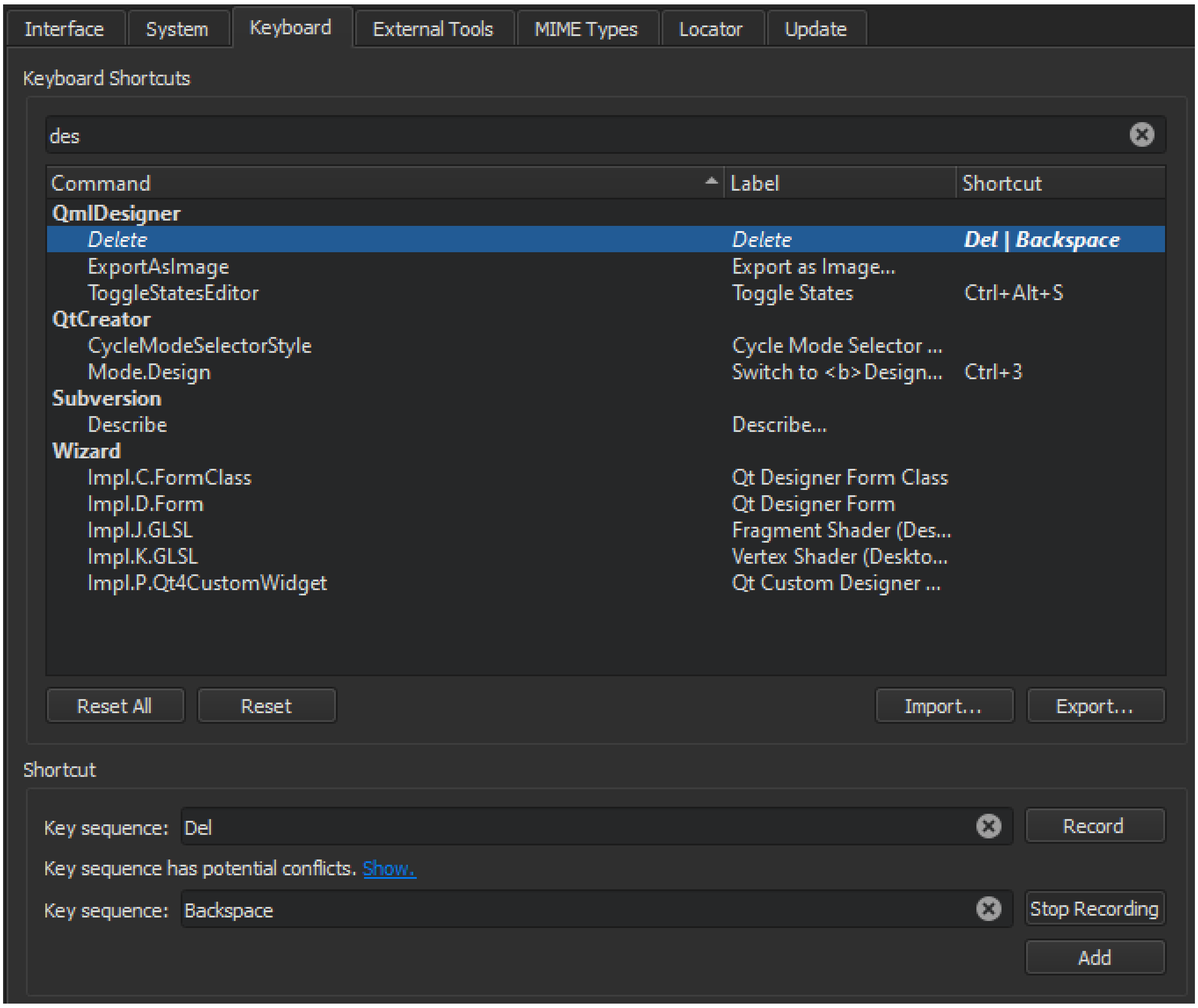
To view different types of output, use the following shortcuts:

- **Issues** - **Alt+1** (**Cmd+1** on macOS)
- **Search Results** - **Alt+2** (**Cmd+2** on macOS)
- **Application Output** - **Alt+3** (**Cmd+3** on macOS)
- **Compile Output** - **Alt+4** (**Cmd+4** on macOS)

For additional ways to view other types of output, see Viewing Output.

Qt Creator provides many useful keyboard shortcuts. You can see the keyboard shortcut for a menu command in the menu or the tooltip for a button.

To customize, import, or export keyboard shortcuts, select **Edit** > **Preferences** > **Environment** > **Keyboard**.



Keyboard Shortcuts

des

Command	Label	Shortcut
QmlDesigner		
Delete	*Delete*	***Del | Backspace***
ExportAsImage	Export as Image...	
ToggleStatesEditor	Toggle States	Ctrl+Alt+S
QtCreator		
CycleModeSelectorStyle	Cycle Mode Selector ...	
Mode.Design	Switch to <b>Design...	Ctrl+3
Subversion		
Describe	Describe...	
Wizard		
Impl.C.FormClass	Qt Designer Form Class	
Impl.D.Form	Qt Designer Form	
Impl.J.GLSL	Fragment Shader (Des...	
Impl.K.GLSL	Vertex Shader (Deskto...	
Impl.P.Qt4CustomWidget	Qt Custom Designer ...	

Shortcut

Key sequence: Del

Key sequence has potential conflicts. Show.

Key sequence: Backspace

Run Qt Creator from the command line

You can launch Qt Creator from the command line using the name of an existing session or project file by entering the name as the command argument.

For example, running `qtcreator somesession`, launches Qt Creator and loads the session called *somesession*.

For more information, see Using Command Line Options.

Show and hide sidebars

You can toggle the left and right sidebar in some Qt Creator modes.

To toggle the left sidebar, click (**Hide Left Sidebar/Show Left Sidebar**) or press **Alt+0** (**Cmd+0** on macOS).

macOS).

For more information on using the sidebars, see Browsing Project Contents.

Move to symbols

To move straight to a symbol used in a project, select the symbol in the **Editor** toolbar drop-down menu. For more information on the editor toolbar, see Using the Editor Toolbar.

To jump to a symbol in the current file, press **Ctrl+K** to open the **Locator**, enter a period (.), and start typing the symbol name. Then select the symbol in the list. For more information on using the locator, see Searching with the Locator.

Press **Ctrl** (**Cmd** on macOS) and click a symbol to move directly to the definition or the declaration of the symbol. You can also move the cursor on the symbol and press **F2**. For more information, see Moving to Symbol Definition or Declaration.

Inspect signal-slot connections while debugging

If an instance of a class is derived from QObject, and you would like to find all other objects connected to one of your object's slots using Qt's signals and slots mechanism, select **Edit** > **Preferences** > **Debugger** > **Locals & Expressions** > **Use Debugging Helpers**.

In the **Locals** view, expand the object's entry and open the slot in the *slots* subitem. The objects connected to this slot are shown as children of the slot. This method works with signals too.

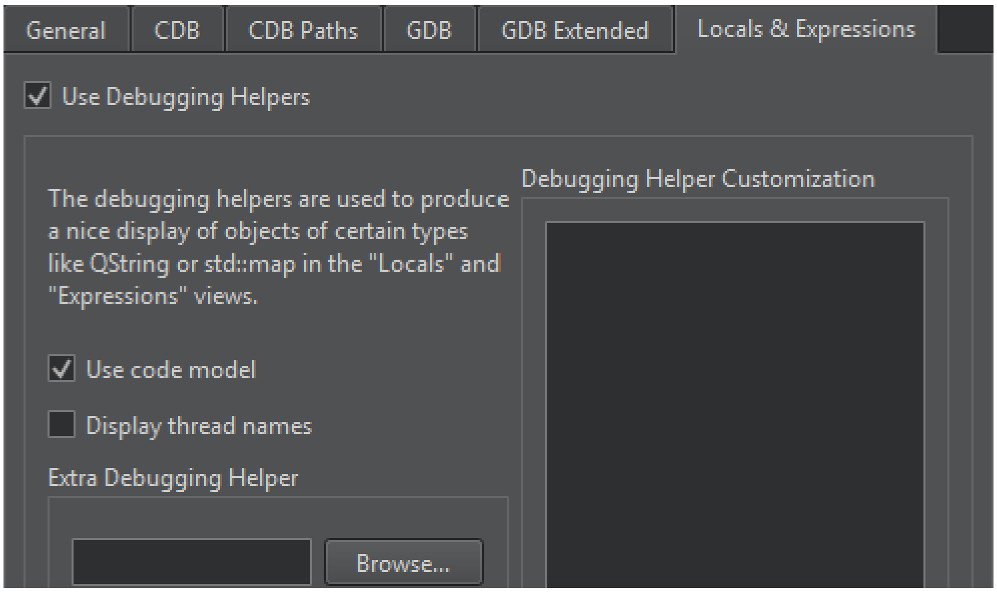
For more information about the **Locals** view, see Local Variables and Function Parameters.

Display low-level data in the debugger

If special debugging of Qt objects fails due to data corruption within the debugged objects, you can switch off the debugging helpers. When debugging helpers are switched off, low-level structures become visible.

To switch off the debugging helpers:

1. Select **Edit** > **Preferences** > **Debugger** > **Locals & Expressions**.



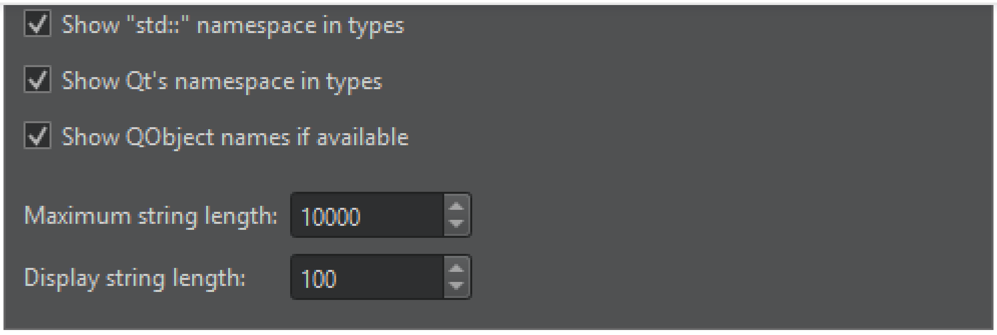
☑ Use Debugging Helpers

The debugging helpers are used to produce a nice display of objects of certain types like QString or std::map in the "Locals" and "Expressions" views.

☑ Use code model

☐ Display thread names

Extra Debugging Helper



- [✓] Show "std::" namespace in types
- [✓] Show Qt's namespace in types
- [✓] Show QObject names if available

Maximum string length: 10000

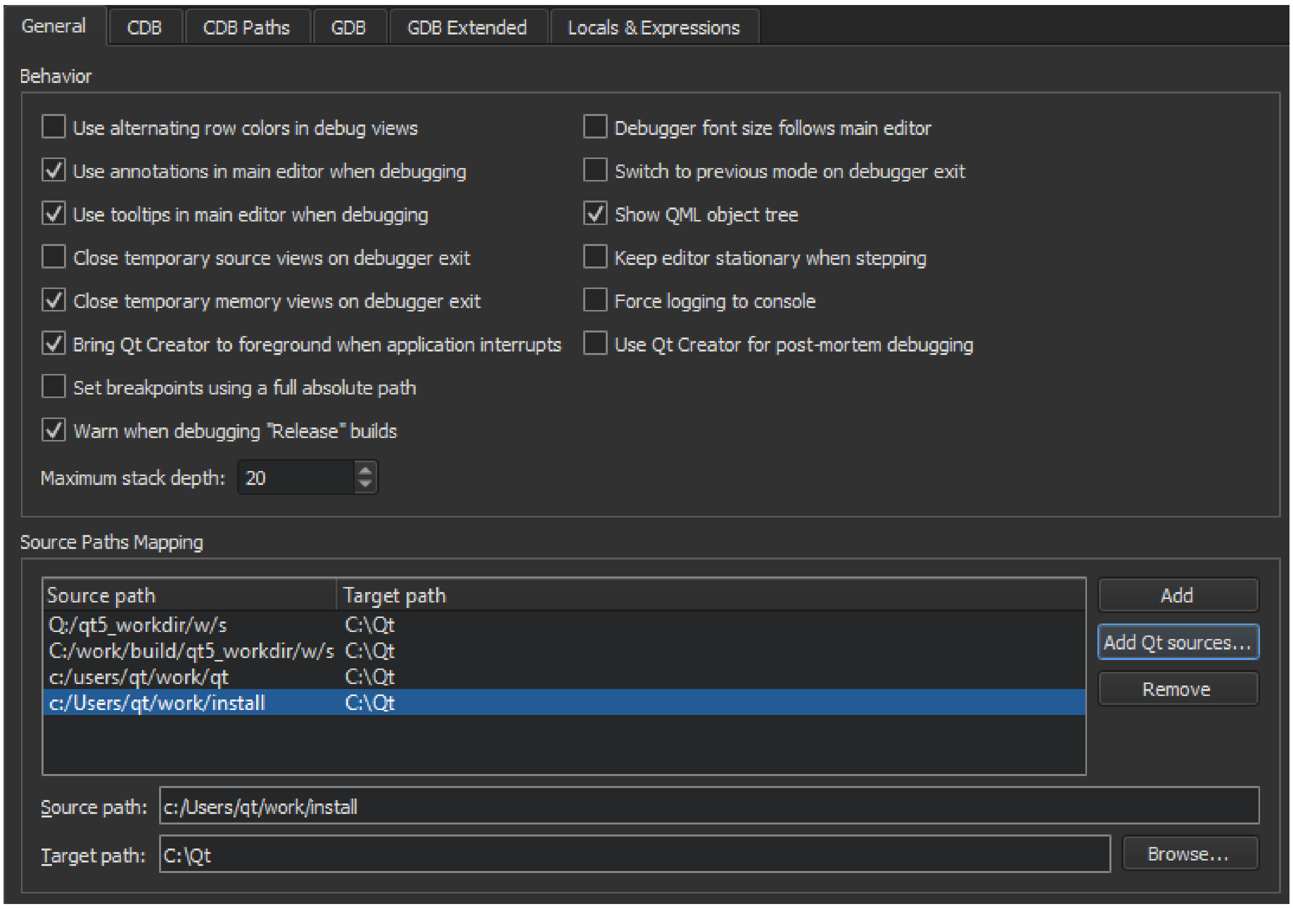
Display string length: 100

2. Deselect **Use Debugging Helpers**.

See the value of variables in tooltips while debugging

To inspect the value of variables from the editor, you can turn on tooltips. Tooltips are hidden by default for performance reasons.

1. Select **Edit** > **Preferences** > **Debugger** > **General**.



Behavior

- [ ] Use alternating row colors in debug views
- [✓] Use annotations in main editor when debugging
- [✓] Use tooltips in main editor when debugging
- [ ] Close temporary source views on debugger exit
- [✓] Close temporary memory views on debugger exit
- [✓] Bring Qt Creator to foreground when application interrupts
- [ ] Set breakpoints using a full absolute path
- [✓] Warn when debugging "Release" builds
- [ ] Debugger font size follows main editor
- [ ] Switch to previous mode on debugger exit
- [✓] Show QML object tree
- [ ] Keep editor stationary when stepping
- [ ] Force logging to console
- [ ] Use Qt Creator for post-mortem debugging

Maximum stack depth: 20

Source Paths Mapping

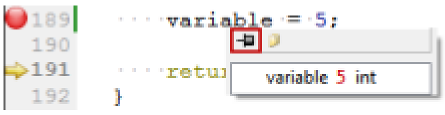
Source path	Target path
Q:/qt5_workdir/w/s	C:\Qt
C:/work/build/qt5_workdir/w/s	C:\Qt
c:/users/qt/work/qt	C:\Qt
c:/Users/qt/work/install	C:\Qt

Source path: c:/Users/qt/work/install

Target path: C:\Qt

2. Select **Use tooltips in main editor when debugging**.

When you hover over a variable in the code editor in **Debug** mode, a tooltip is displayed. To keep the tooltip visible, click the pin button. You can expand pinned tooltips to view their full content.



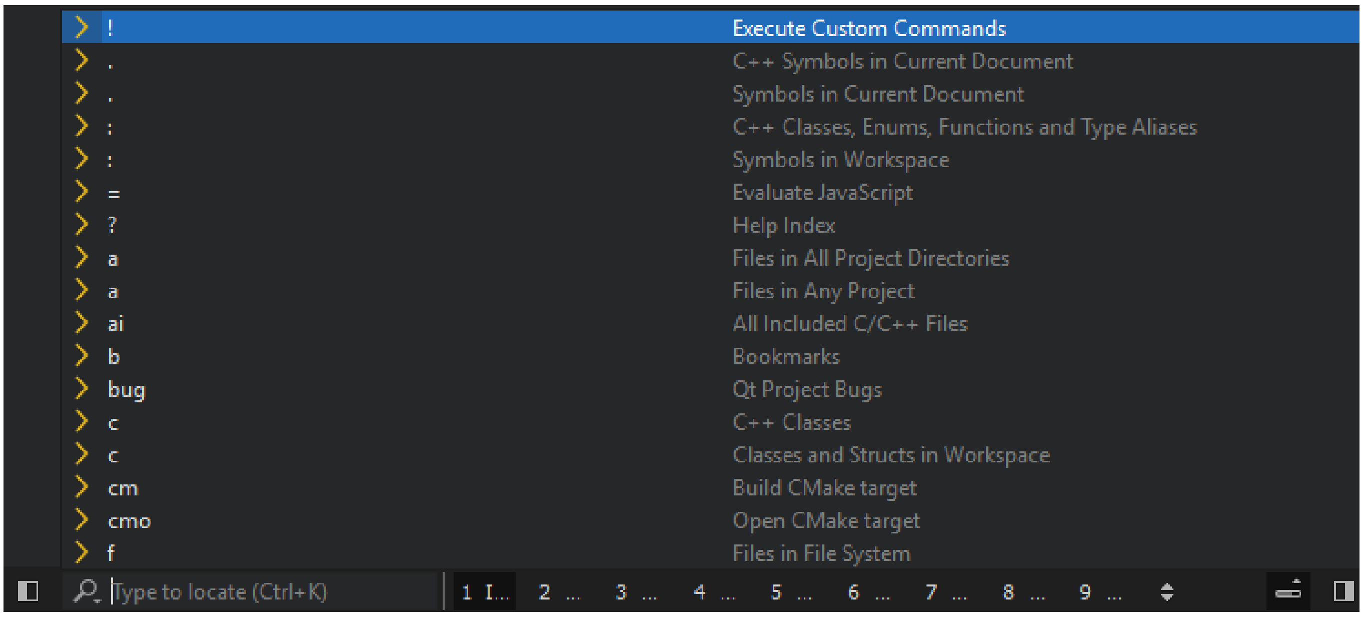
```
    variable = 5;

    retu
}
```

variable 5 int

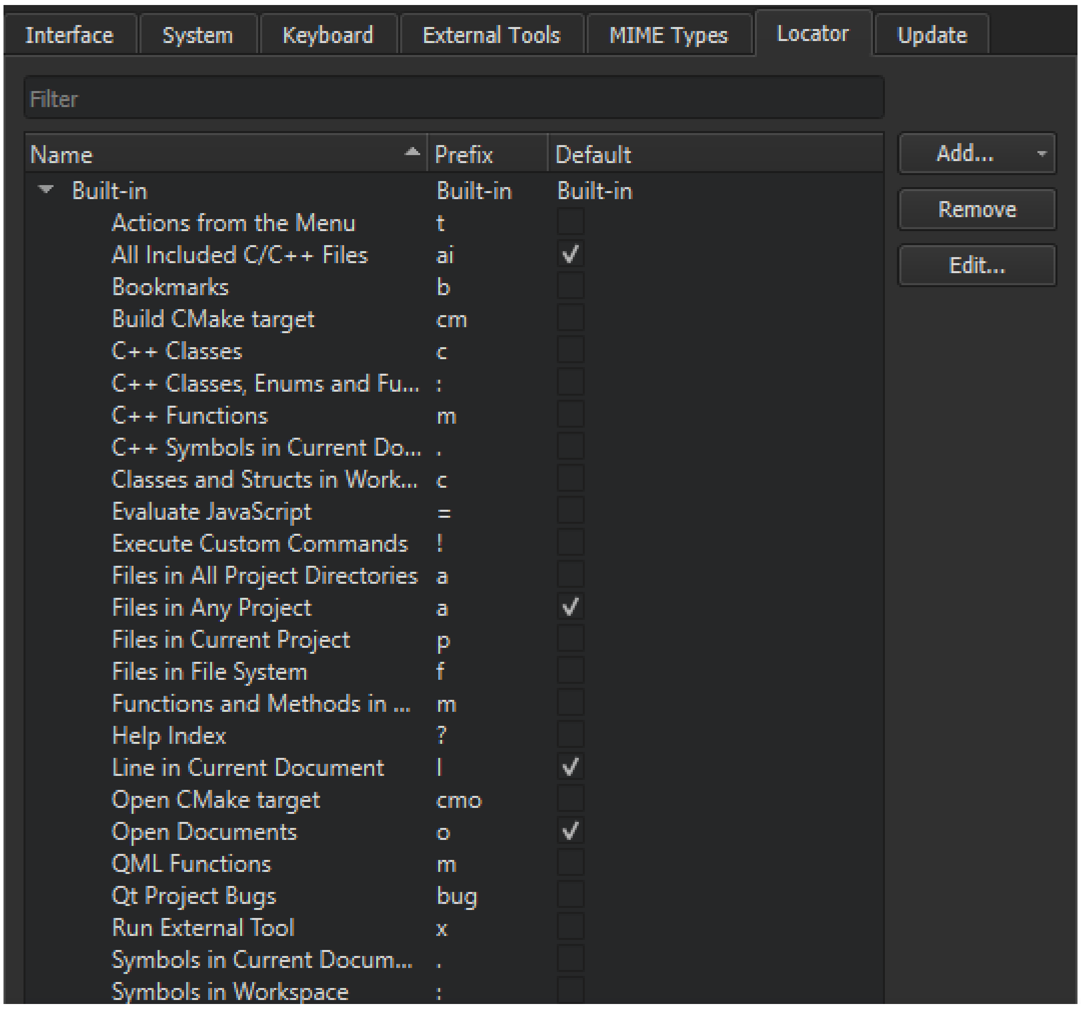
Quickly locate files using the keyboard

The **Locator** provides one of the easiest ways in Qt Creator to browse through projects, files, classes, functions, documentation, and file systems. To quickly access files not directly mentioned in your project, you can create your own locator filters. That way you can locate files in a directory structure you have defined.

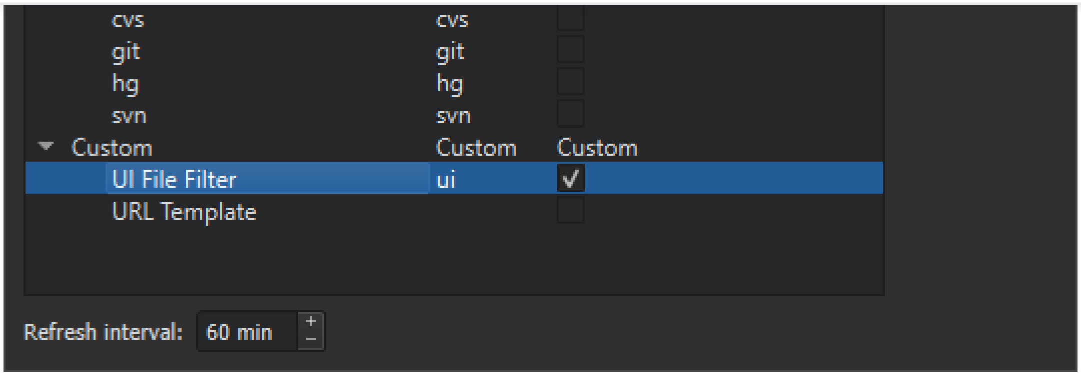


!	Execute Custom Commands
.	C++ Symbols in Current Document
.	Symbols in Current Document
:	C++ Classes, Enums, Functions and Type Aliases
:	Symbols in Workspace
=	Evaluate JavaScript
?	Help Index
a	Files in All Project Directories
a	Files in Any Project
ai	All Included C/C++ Files
b	Bookmarks
bug	Qt Project Bugs
c	C++ Classes
c	Classes and Structs in Workspace
cm	Build CMake target
cmo	Open CMake target
f	Files in File System

To create locator filters, select **Edit** > **Preferences** > **Environment** > **Locator** > **Add**.



Name	Prefix	Default
Built-in	Built-in	Built-in
Actions from the Menu	t	
All Included C/C++ Files	ai	✓
Bookmarks	b	
Build CMake target	cm	
C++ Classes	c	
C++ Classes, Enums and Fu...	:	
C++ Functions	m	
C++ Symbols in Current Do...	.	
Classes and Structs in Work...	c	
Evaluate JavaScript	=	
Execute Custom Commands	!	
Files in All Project Directories	a	
Files in Any Project	a	✓
Files in Current Project	p	
Files in File System	f	
Functions and Methods in ...	m	
Help Index	?	
Line in Current Document	l	✓
Open CMake target	cmo	
Open Documents	o	✓
QML Functions	m	
Qt Project Bugs	bug	
Run External Tool	x	
Symbols in Current Docum...	.	
Symbols in Workspace	:	



cvs	cvs	
git	git	
hg	hg	
svn	svn	
Custom	Custom	Custom
UI File Filter	ui	✓
URL Template		

Refresh interval: 60 min

For more information, see Creating Locator Filters.

Perform calculations

Open the **Locator** with **Ctrl+K** and type =, followed by a space. You can now do basic calculations, with options to copy the results to the clipboard by navigating through the entries and pressing **Enter**.

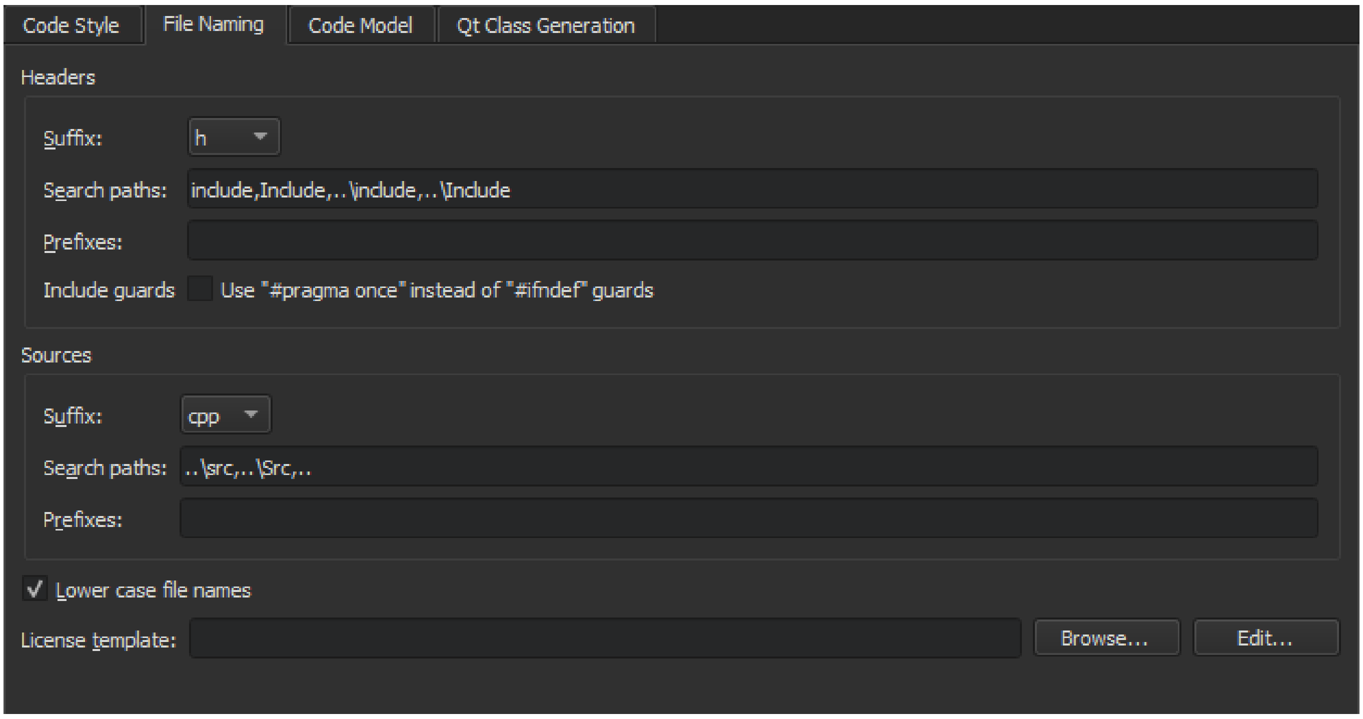
For more information, see Executing JavaScript.

Jump to a function in QML code

Open the **Locator** with **Ctrl+K** and type m, followed by a space. You can now go directly to any QML method in the file.

Add a license header template for C++ code

Specify a file containing a license header for C++ in **Edit** > **Preferences** > **C++** > **File Naming** > **License template**.



Headers

Suffix: h

Search paths: include,Include,..\include,..\Include

Prefixes:

Include guards ☐ Use "#pragma once" instead of "#ifndef" guards

Sources

Suffix: cpp

Search paths: ..\src,..\Src,..

Prefixes:

✓ Lower case file names

License template:

The license file may contain special placeholders enclosed in %% that are replaced when generating a new file:

3. %DAY%: Day of the month
4. %DATE%: Date
5. %USER%: Username
6. %FILENAME%: File name
7. %CLASS%: Class name (if applicable)
8. %$VARIABLE%: Contents of environment variable VARIABLE.

Paste text from my clipboard history

Qt Creator stores copied text in clipboard history. To retrieve clips from the history, press **Ctrl+Shift+V** until the clip appears. The number of clips in the history is fixed to 10.

Sort lines alphabetically

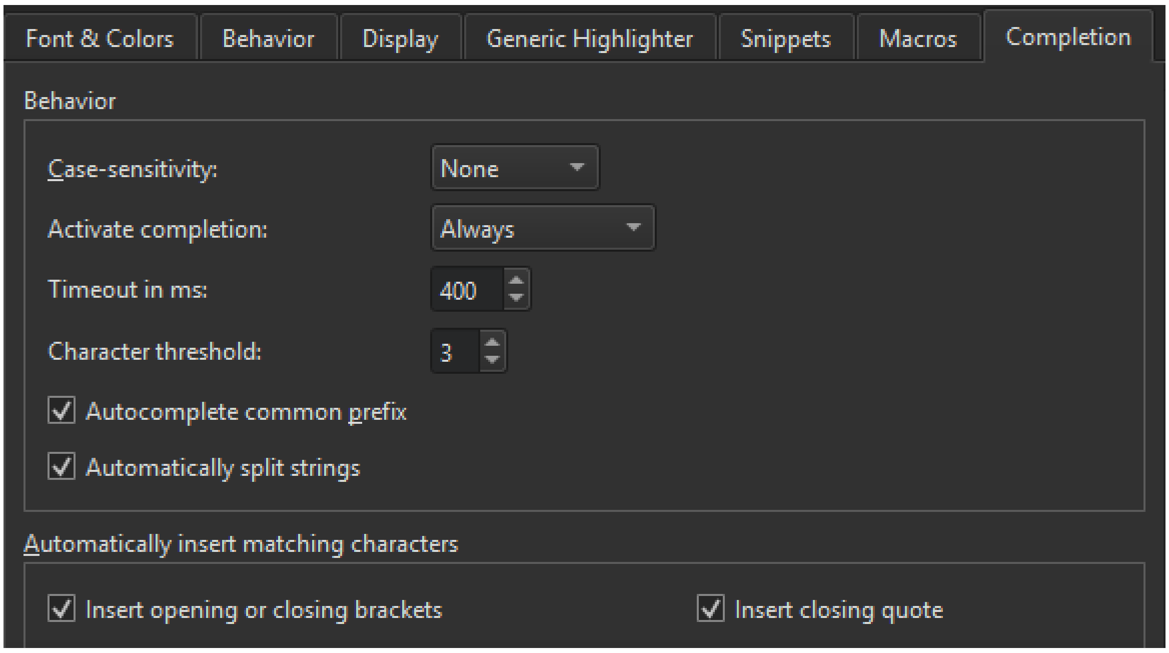
To sort selected lines alphabetically, select **Edit** > **Advanced** > **Sort Selected Lines** or press **Alt+Shift+S** (or **Ctrl+Shift+S** on macOS).

Enclose selected code in curly braces, parentheses, or double quotes

When you have selected code and enter one of the following opening characters, the matching closing character is added automatically at the end of the selection:

- {
- (
- "

To specify whether to automatically insert matching characters, select **Edit** > **Preferences** > **Text Editor** > **Completion**.



Behavior

Case-sensitivity: None

Activate completion: Always

Timeout in ms: 400

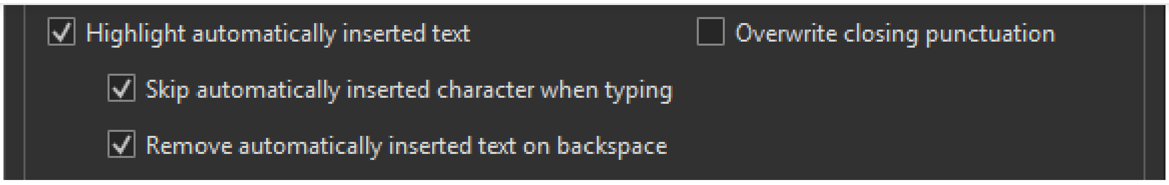
Character threshold: 3

☑ Autocomplete common prefix

☑ Automatically split strings

Automatically insert matching characters

☑ Insert opening or closing brackets ☑ Insert closing quote

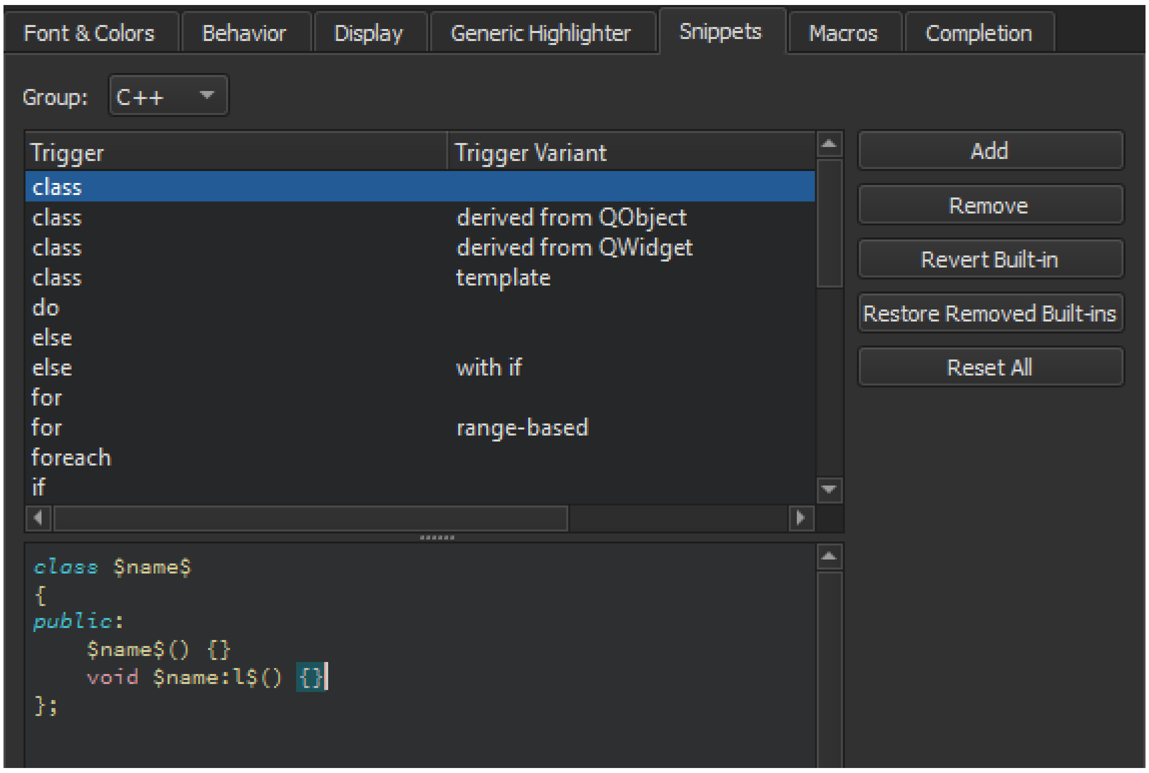

☑ Highlight automatically inserted text ☐ Overwrite closing punctuation
☑ Skip automatically inserted character when typing
☑ Remove automatically inserted text on backspace

Select the enclosing block in C++

Press **Ctrl+U**.

Add my own code snippets to the auto-complete menu

You can add, modify, and remove snippets in the snippet editor. To open the editor, select **Edit** > **Preferences** > **Text Editor** > **Snippets**.



Group: C++

Trigger	Trigger Variant
class	
class	derived from QObject
class	derived from QWidget
class	template
do	
else	
else	with if
for	
for	range-based
foreach	
if	

```
class $name$
{
public:
    $name$() {}
    void $name:l$() {}
};
```

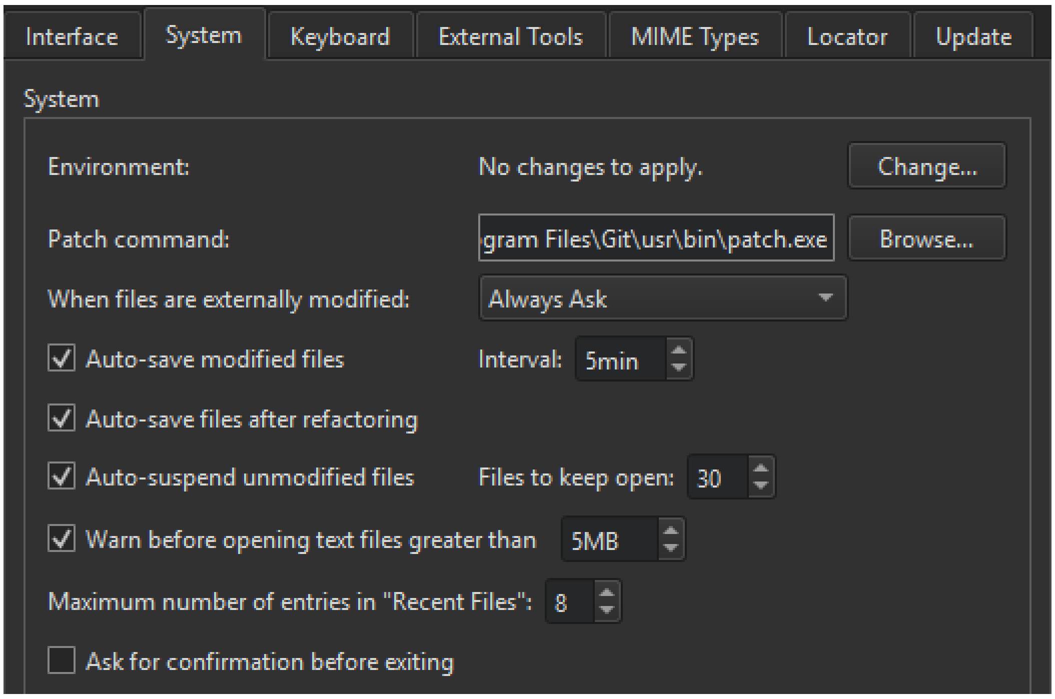
For more information, see Adding and Editing Snippets.

Quickly write down notes somewhere

Select **File** > **New File** > **General** > **Scratch Buffer**. Alternatively, **Ctrl+N** can be used to open this dialog, which is fully navigable via keyboard by using the up and down arrow keys and the tab key.

This creates a new empty text file and saves it to the temporary directory on your machine. You can use it to write down notes without having to worry about deleting the file afterwards. The operating system will eventually remove the file automatically. If you want to keep the file, you can easily save it as a new file somewhere else. If you accidentally close the file, you can find it in the **File** > **Recent Files** menu.

Configure the amount of recent files shown



System

Environment: No changes to apply. Change...

Patch command: gram Files\Git\usr\bin\patch.exe Browse...

When files are externally modified: Always Ask

[✓] Auto-save modified files Interval: 5min

[✓] Auto-save files after refactoring

[✓] Auto-suspend unmodified files Files to keep open: 30

[✓] Warn before opening text files greater than 5MB

Maximum number of entries in "Recent Files": 8

[ ] Ask for confirmation before exiting

Search and replace across files using a regular expression

As an example, say you want to replace equality checks (`foo == bar`) with a function (`foo.equals(bar)`):

1. Ensure that any work you have done is committed to version control, as the changes cannot be undone.
2. Press **Ctrl+Shift+F** to bring up the **Advanced Find** form.
3. Change the scope to whatever is appropriate for your search.
4. Under the **Search for** text field, select the **Use regular expressions** check box.
5. Enter the following text in the **Search for** text field:

```
if \((.*) == (.*)\)
```

6. Press **Search & Replace** to see a list of search results.
7. In the **Replace with** text field, enter the following text:

```
if (\1.strictlyEquals(\2))
```

8. Press **Replace** to replace all instances of the text.

For more information, see Advanced Search.

< FAQ Known Issues >

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
Marketplace

Feedback Sign In