Qt **DOCUMENTATION**

Search

Qt Creator Manual 8.0.2

Topics 〉

Qt Creator Manual 〉 Indenting Text or Code

Indenting Text or Code

When you type text or code, it is indented automatically according to the selected text editor or code style preferences. Select a block to indent it when you press **Tab**. Press **Shift+Tab** to decrease the indentation. You can disable automatic indentation.

You can specify indentation for:
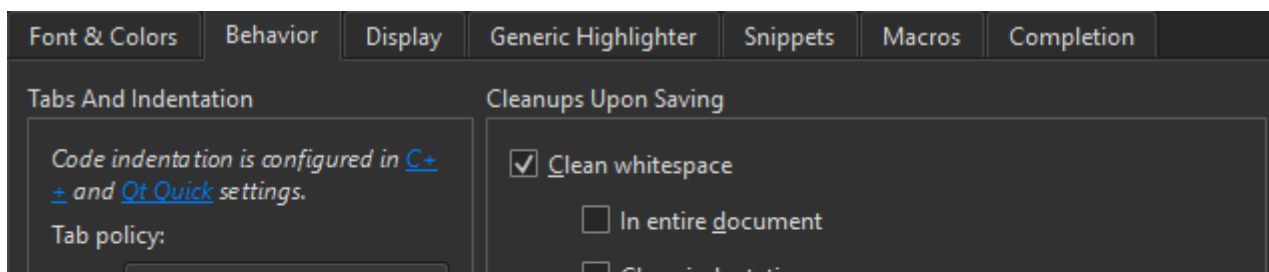
› C++ files
› QML files
› Nim files
› Other text files

To fix the indentation in the file currently open in the editor, select options in the **Edit** > **Advanced** menu or use keyboard shortcuts:

› To automatically indent the highlighted text, select **Auto-indent Selection** or press **Ctrl+I**.
› To automatically format the highlighted text, select **Auto-format Selection** or press **Ctrl+;**.
› To adjust the wrapping of the selected paragraph, select **Rewrap Paragraph** or press **Ctrl+E** followed by **R**.
› To toggle text wrapping, select **Enable Text Wrapping** or press **Ctrl+E** followed by **Ctrl+W**.
› To visualize whitespace in the editor, select **Visualize Whitespace** or press **Ctrl+E** followed by **Ctrl+V**.
› To clear all whitespace characters from the currently open file, select **Clean Whitespace**.

Specifying Indentation Settings

You can also specify indentation separately for each project. You can specify several sets of code style settings and easily switch between them. In addition, you can import and export code style settings.

To automatically fix indentation according to the indentation settings when you save the file, select **Edit** > **Preferences** > **Text Editor** > **Behavior** > **Clean whitespace** > **Clean indentation**. Select the **Skip clean whitespace for file types** check box to exclude the specified file types.

Qt DOCUMENTATION

☰

To visualize whitespace in the editor, select **Edit** > **Preferences** > **Text Editor** > **Display** > **Visualize whitespace**.

Qt DOCUMENTATION
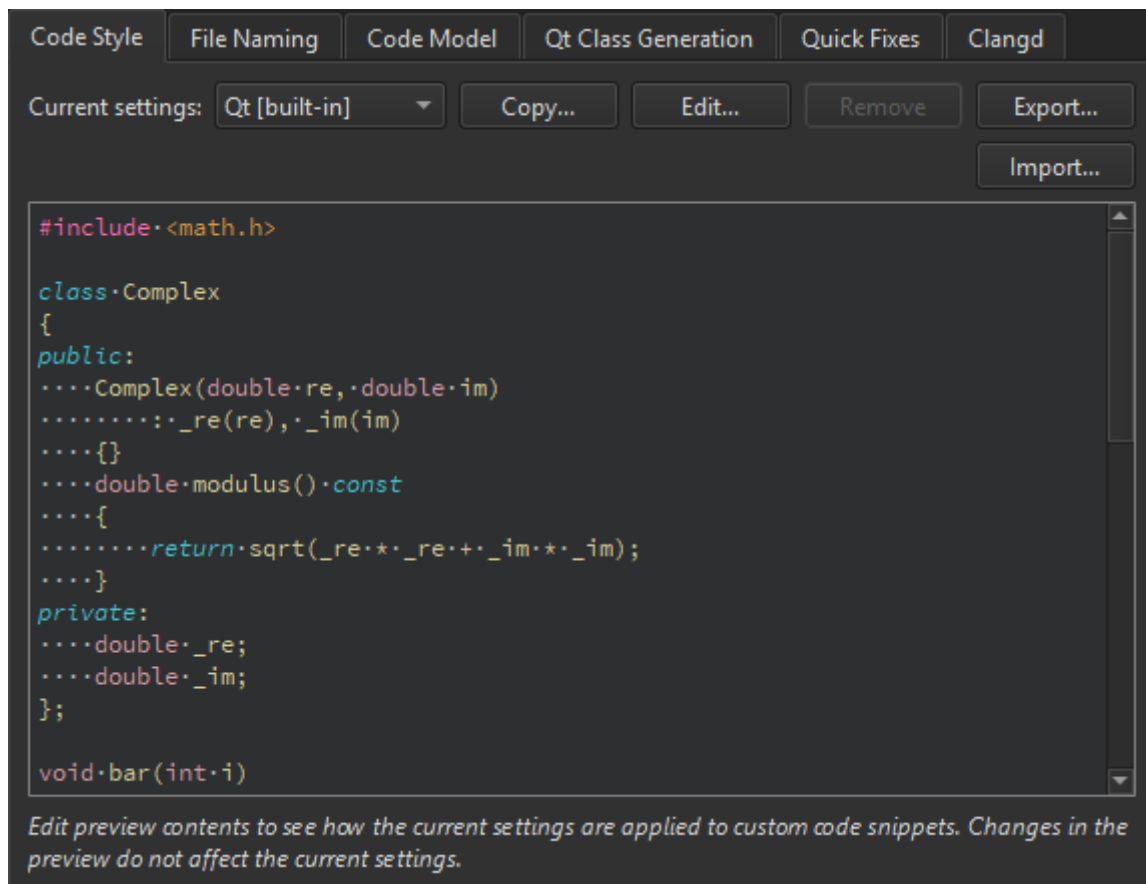
To help you keep line length at a particular number of characters, set the number of characters in the **Display right margin at column** field. To use a context-specific margin when available, select the **Use context-specific margin** check box. For example, the margin could be set by the `ColumnLimit` option of the Clang Format plugin.
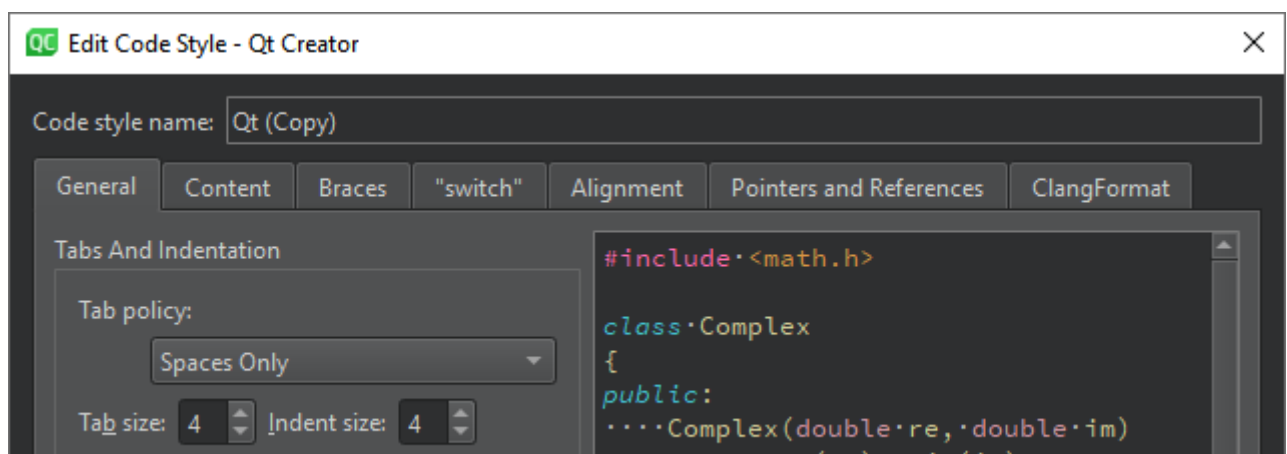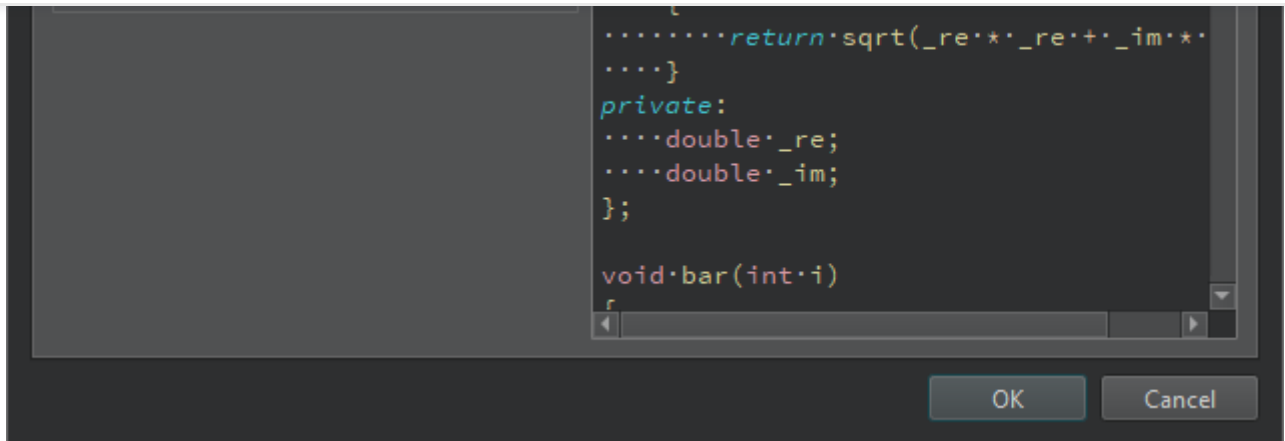
Indenting C++ Files

To specify indentation settings for the C++ editor:

1. Select **Edit** > **Preferences** > **C++**.

2. In the **Current settings** field, select the settings to modify and click **Copy**.

```
Code Style   File Naming   Code Model   Qt Class Generation   Quick Fixes   Clangd

Current settings:  Qt [built-in]    ▼     Copy...      Edit...     Remove       Export...

                                                                              Import...

#include·<math.h>

class·Complex
{
public:
····Complex(double·re,·double·im)
········:·_re(re),·_im(im)
····{}
····double·modulus()·const
····{
········return·sqrt(_re·*·_re·+·_im·*·_im);
····}
private:
····double·_re;
····double·_im;
};

void·bar(int·i)
```

Edit preview contents to see how the current settings are applied to custom code snippets. Changes in the preview do not affect the current settings.

3. Give a name to the settings and click **OK**.

4. Click **Edit** to specify code style settings for the project.

```
QC Edit Code Style - Qt Creator                                               ✕

Code style name:  Qt (Copy)

  General    Content   Braces   "switch"   Alignment   Pointers and References   ClangFormat

Tabs And Indentation                            #include·<math.h>

  Tab policy:                                   class·Complex
                                                {
      Spaces Only                    ▼          public:
                                                ····Complex(double·re,·double·im)
  Tab size:  4  ▲▼   Indent size:  4  ▲▼        ·········:·_re(re)·:·_im(im)
```

You can specify how to:

> Interpret the **Tab** and **Backspace** key presses.

> Indent the contents of classes, functions, blocks, and namespaces.

> Indent braces in classes, namespaces, enums, functions, and blocks.

> Control switch statements and their contents.

> Align continuation lines.

> Bind pointers (*) and references (&) in types and declarations to identifiers, type names, or left or right `const` or `volatile` keywords.

> Name getter functions.

You can use the live preview to see how the preferences change the indentation.

To specify different settings for a particular project, select **Projects** > **Code Style**.

Automatic Formatting and Indentation

The Clang Format plugin uses the LibFormat library for automatic formatting and indentation.

To enable the plugin, select **Help** > **About Plugins** > **C++** > **ClangFormat**. Then select **Restart Now** to restart Qt Creator and load the plugin.

> **Note:** If you enable Clang Format, do not use the Beautifier because combining them can provide unexpected results.

You can use Clang Format to enforce a coding style for a project or the whole organization. Create a `.clang-format` file that contains the Clang Format Style Options to use and save it in the root folder of the project or one of its parent folders. The plugin searches for the Clang format file recursively from the directory that contains the source file up to the file system root.

To override the `.clang-format` file globally for all projects, select **Edit** > **Preferences** > **C++** > **Copy** > **Edit** > **ClangFormat** > **Override Clang Format configuration file**.

Qt DOCUMENTATION
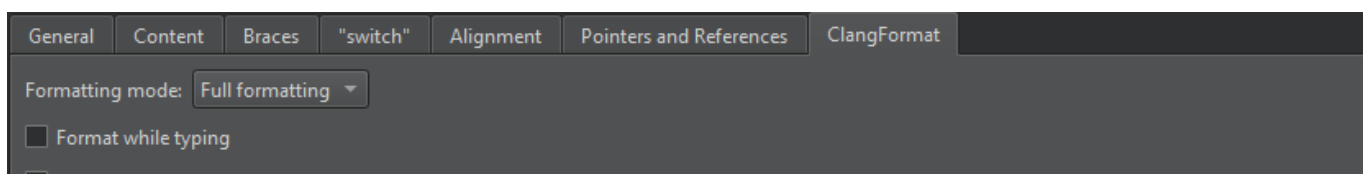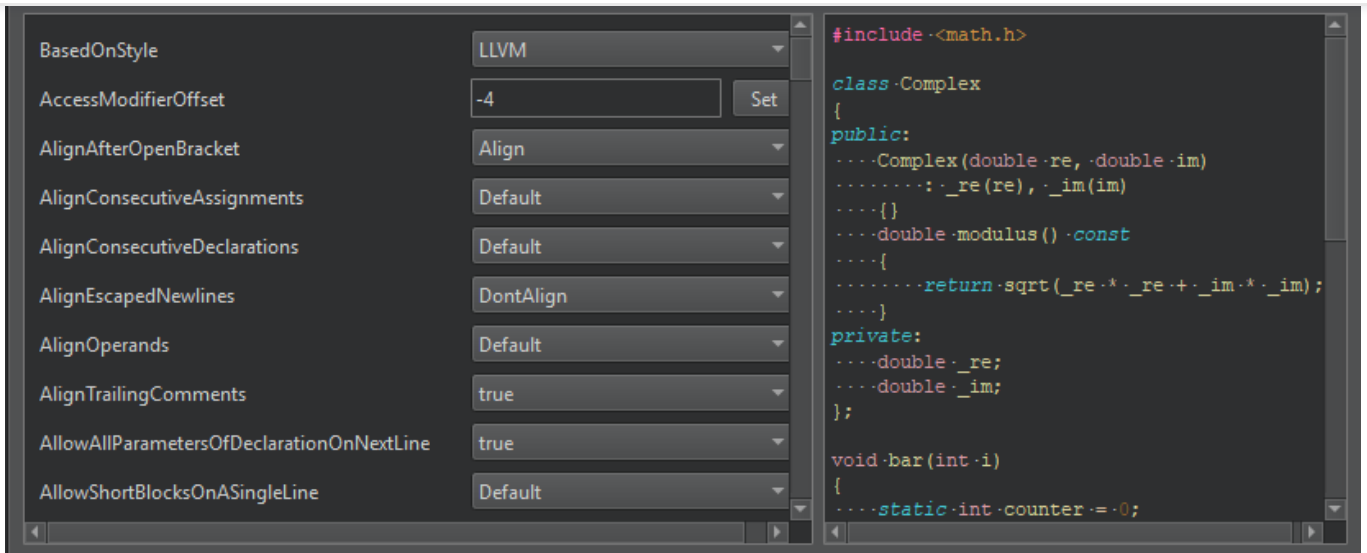
In **Formatting mode**, select **Indenting Only** to only indent code. Select **Full Formatting** to use the **Ctrl+I** keyboard shortcut to format code instead of indenting. To apply the formatting while you type, select **Format while typing**. To apply the formatting to the edited code when you save the file, select **Format edited code on file save**.

This creates a local configuration file that overrides the one stored in the file system.

To override the `.clang-format` file for a particular project, create a copy of the built-in style and edit its settings by selecting **Projects** > **Project Settings** > **Code Style** > **Copy** > **Edit** > **ClangFormat** > **Override Clang Format configuration file**.
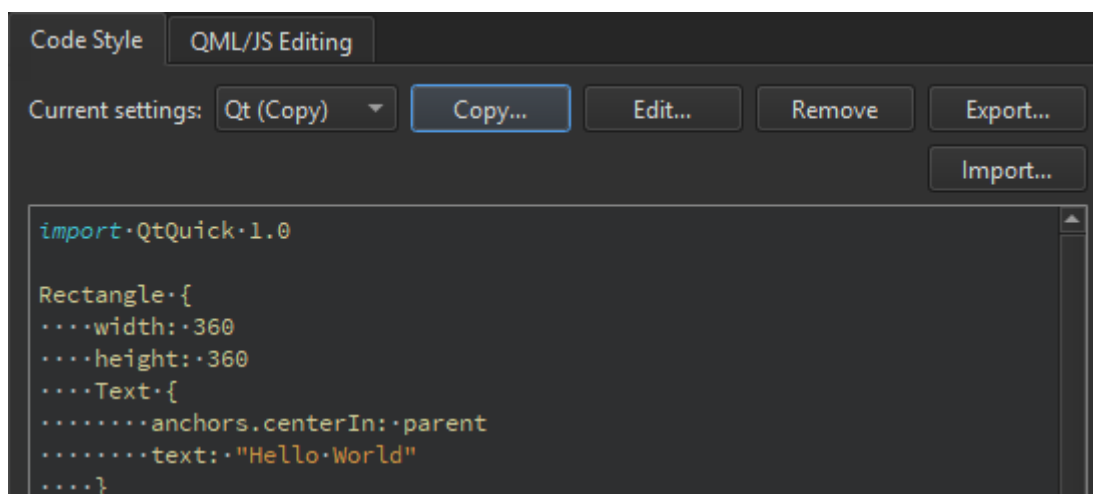
You can create `.clang-format` files that contain the configuration options of a certain predefined style from the command line. For example, to create a format file for the LLVM style, enter the following command:
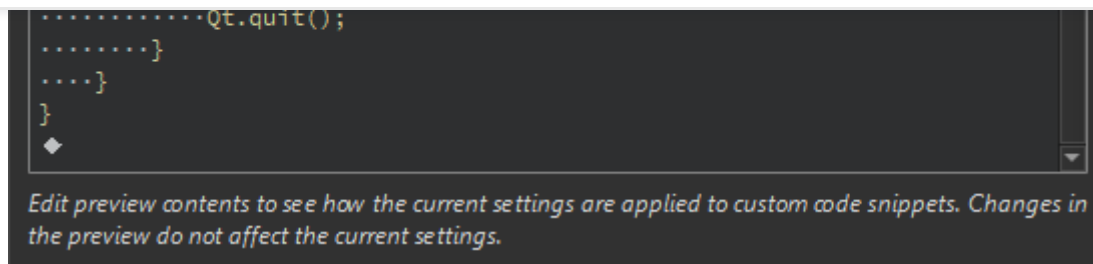
```
clang-format -style=llvm -dump-config > .clang-format
```
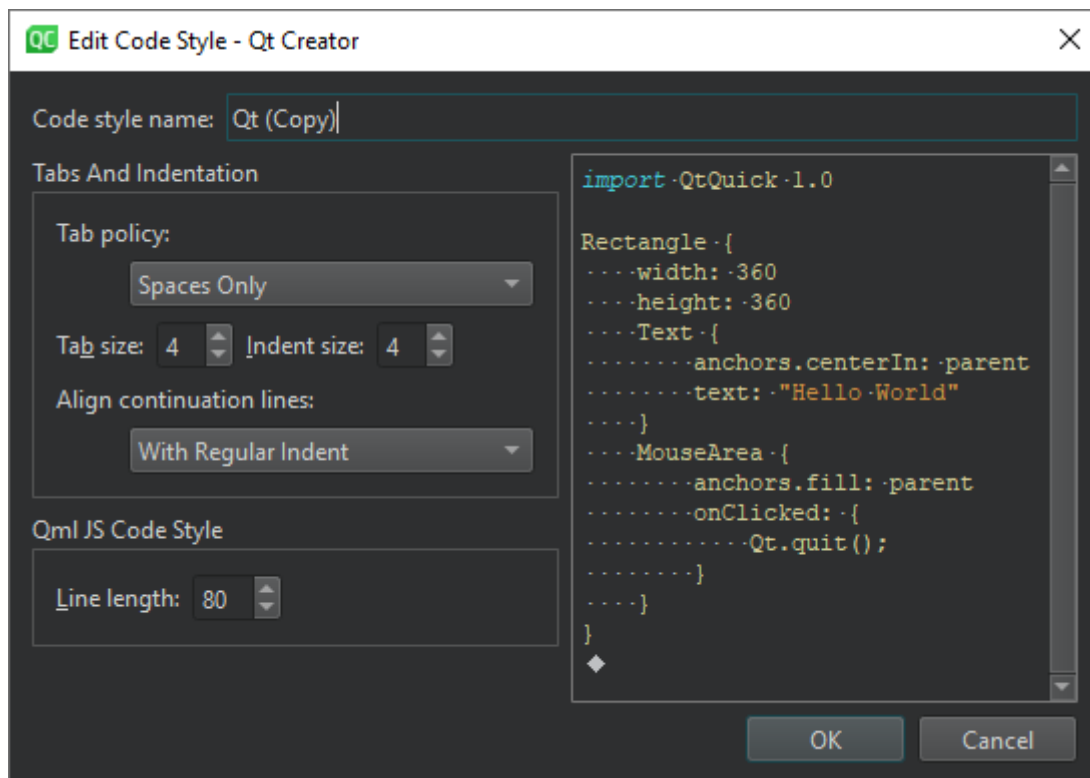
Indenting QML Files

To specify settings for the Qt Quick editor:

1. Select **Edit** > **Preferences** > **Qt Quick**.
2. In the **Current settings** field, select the settings to modify and click **Copy**.

Edit preview contents to see how the current settings are applied to custom code snippets. Changes in the preview do not affect the current settings.

3. Give a name to the settings and click **OK**.

4. Click **Edit** to specify code style settings for the project.

You can specify how to interpret the **Tab** key presses and how to align continuation lines.
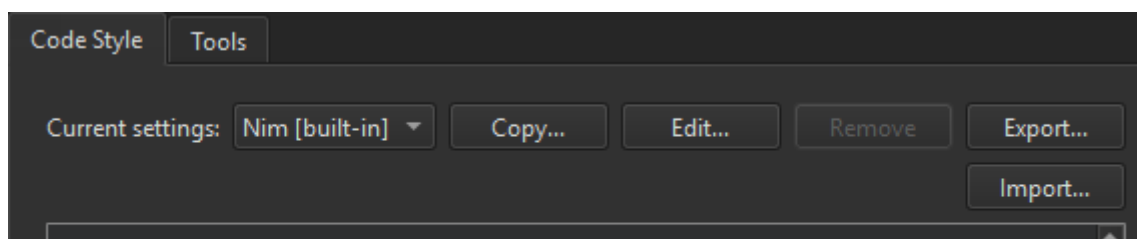
In **Line length**, you can adjust the maximum line length for code lines.

To specify different settings for a particular project, select **Projects** > **Code Style**.

Indenting Nim Files

To specify settings for the Nim editor (experimental):

1. Select **Edit** > **Preferences** > **Nim**.

2. In the **Current settings** field, select the settings to modify and click **Copy**.
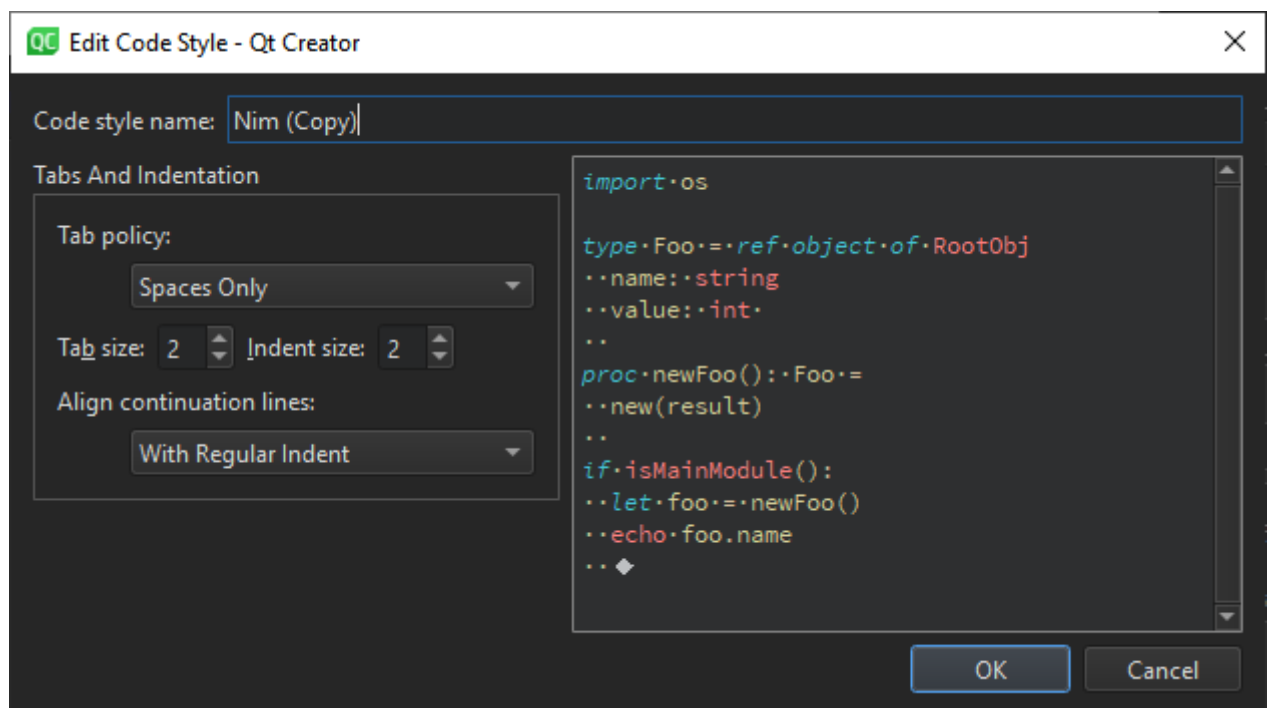
3. Give a name to the settings and click **OK**.

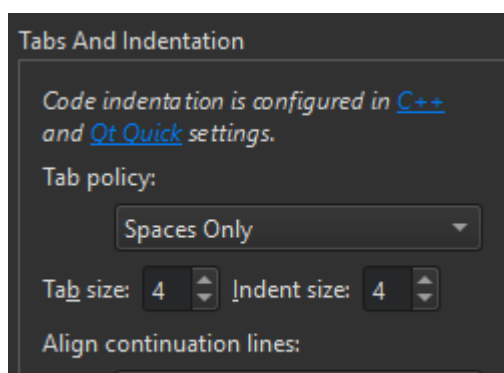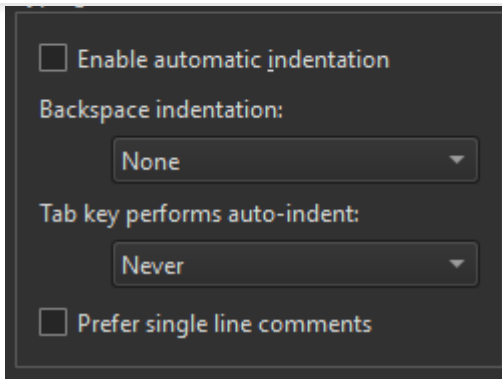4. Click **Edit** to specify code style settings for the project.

You can specify how to interpret the **Tab** key presses and how to align continuation lines.

To specify different settings for a particular project, select **Projects** > **Code Style**.

Indenting Other Text Files

To specify indentation settings for text files that do not contain C++ or QML code (such as Python code files), select **Edit** > **Preferences** > **Text Editor** > **Behavior**.

Qt DOCUMENTATION

☐ Enable automatic indentation

Backspace indentation:

None ▾

Tab key performs auto-indent:

Never ▾

☐ Prefer single line comments

To specify different settings for a particular project, select **Projects** > **Editor**.

You can specify how to interpret the **Tab** and **Backspace** key presses and how to align continuation lines.

Specifying Tab Settings

You can specify tab settings at the following levels:

> For all C++ files

> For all QML files

> For all other text files

> For C++ files in a project

> For QML files in a project

> For other text files in a project

Specifying Tabs and Indentation

You can specify tab policy and tab size in the **Tabs and Indentation** group. In the **Tab policy** field, select whether to use only spaces or only tabs for indentation, or to use a mixture of them.

By default, the tab length in code editor is 8 spaces and the indent size is 4 spaces. You can specify the tab length and indent size separately for each project and for different types of files.

You can have continuation lines aligned with the previous line. In the **Align continuation lines** field, select **Not at all** to disable automatic alignment and indent continuation lines to the logical depth. To always use spaces for alignment, select **With Spaces**. To follow the **Tab policy**, select **With Regular Indent**.

Setting Typing Preferences

When you type text or code, it is indented automatically according to the selected text editor or code style preferences. To set typing preferences, select **Edit** > **Preferences** > **Text Editor** > **Behavior** > **Typing**.

To disable automatic indentation, deselect the **Enable automatic indentation** check box.

You can specify how the indentation is decreased when you press **Backspace** in the **Backspace indentation** field. To go back one space at a time, select **None**. To decrease indentation in leading white space by one level, select **Follows Previous Indents**. To move back one tab length if the character to the left of the cursor is a space, select **Unindents**.

You can specify whether the **Tab** key automatically indents text when you press it. To automatically indent text, select **Always** in the **Tab key performs auto-indent** field. To only indent text when the cursor is located within

Qt DOCUMENTATION

comment definition when commenting out a selection, select **Prefer single line comments**.
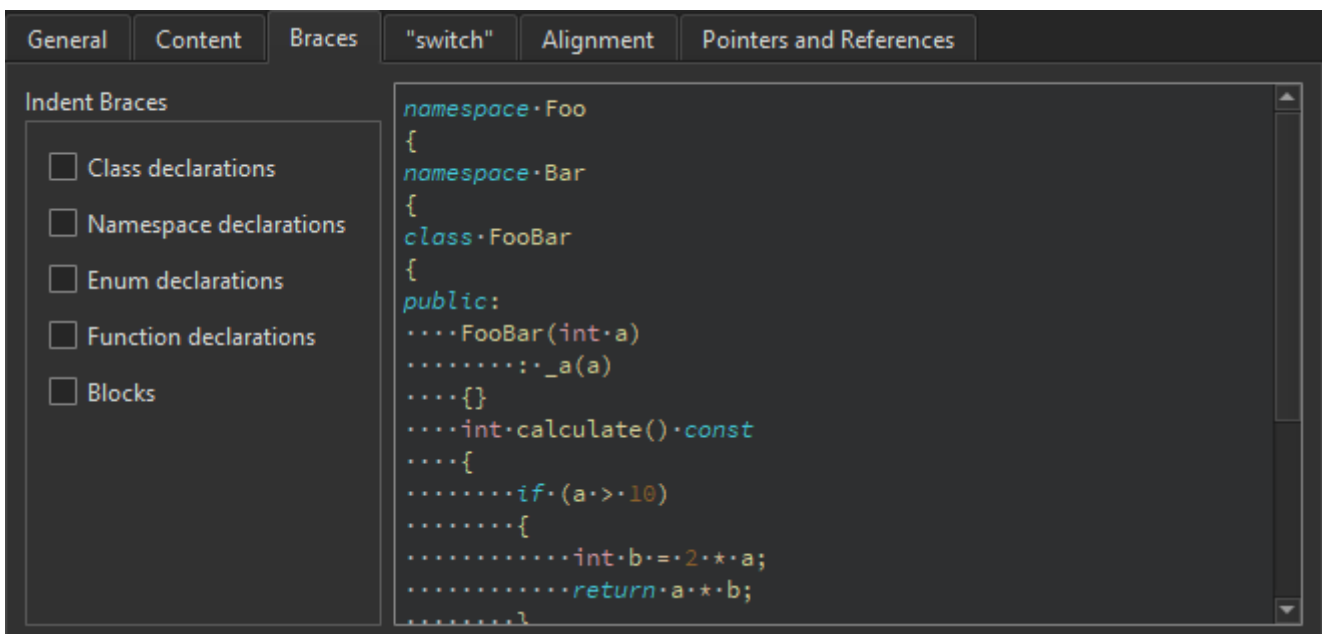
Specifying Settings for Content

You can indent public, protected, and private statements and declarations related to them within classes.

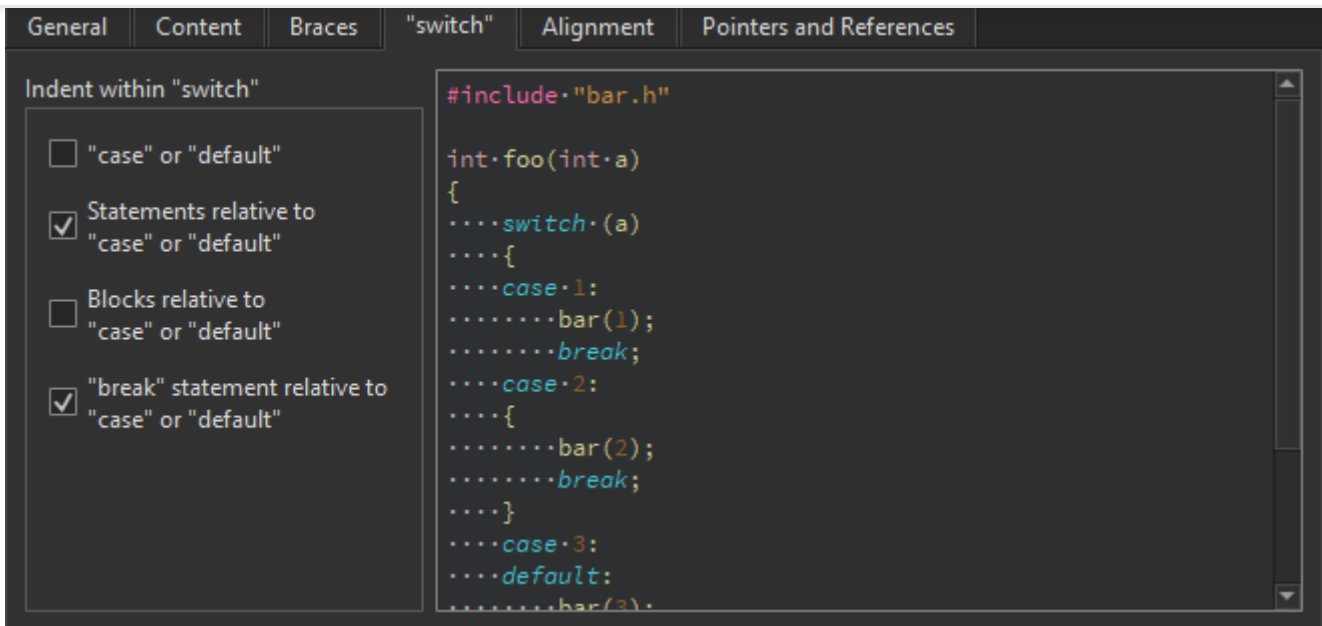You can also indent statements within functions and blocks and declarations within namespaces.
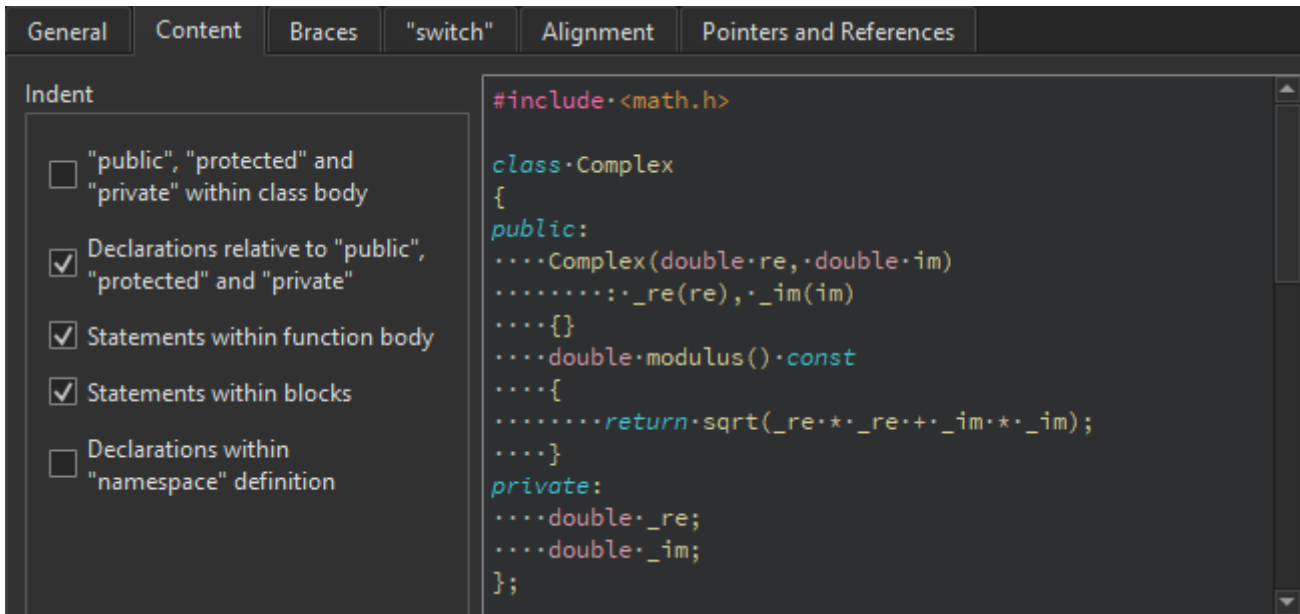
Specifying Settings for Braces

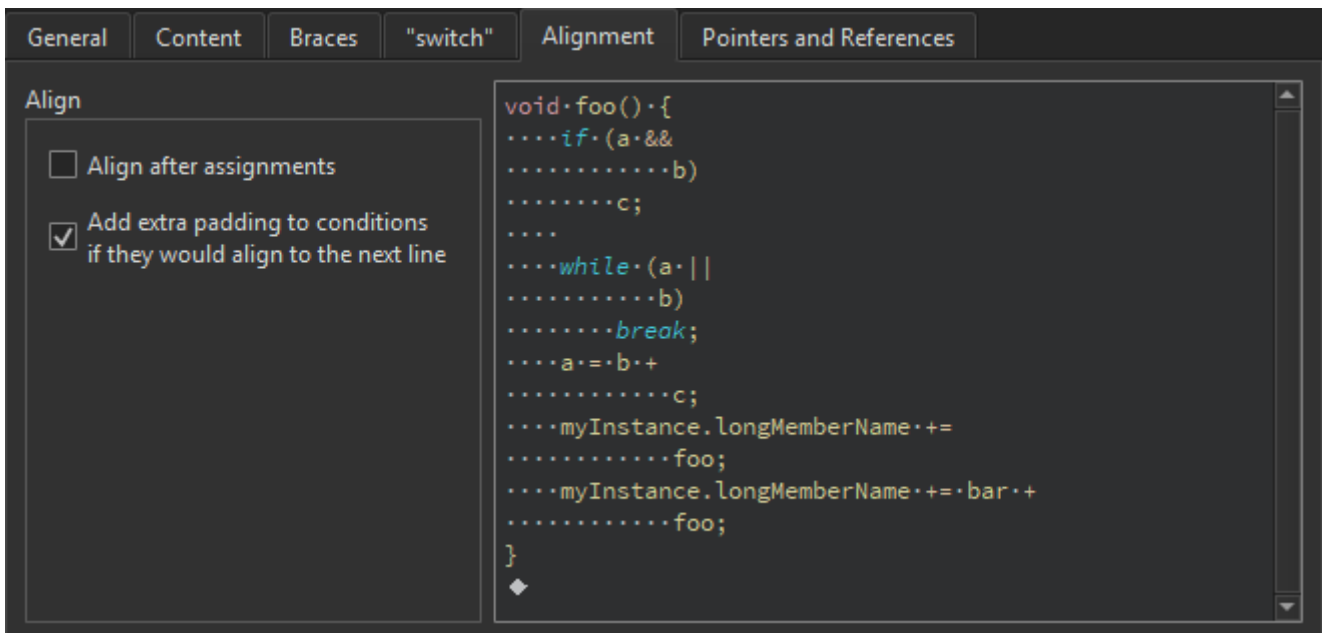You can indent class, namespace, enum and function declarations and code blocks.

Specifying Settings for Switch Statements

Qt DOCUMENTATION

| General | Content | Braces | "switch" | Alignment | Pointers and References |

Indent within "switch"

☐ "case" or "default"

☑ Statements relative to
   "case" or "default"

☐ Blocks relative to
   "case" or "default"

☑ "break" statement relative to
   "case" or "default"

```
#include "bar.h"

int foo(int a)
{
    switch (a)
    {
    case 1:
        bar(1);
        break;
    case 2:
    {
        bar(2);
        break;
    }
    case 3:
    default:
        bar(3):
```

Specifying Alignment

To align continuation lines to tokens after assignments, such as = or +=, select the **Align after assignments** check box. You can specify additional settings for aligning continuation lines in the **General** tab.

You can also add spaces to conditional statements, so that they are not aligned with the following line. Usually, this only affects `if` statements.

| General | Content | Braces | "switch" | Alignment | Pointers and References |

Align

☐ Align after assignments

☑ Add extra padding to conditions
   if they would align to the next line

```
void foo() {
    if (a &&
            b)
        c;

    while (a ||
            b)
        break;
    a = b +
            c;
    myInstance.longMemberName +=
            foo;
    myInstance.longMemberName += bar +
            foo;
}
◆
```

Binding Pointers and References

To bind pointers (*) and references (&) in types and declarations to identifiers, type names, or left or right `const` or `volatile` keywords, select the check boxes in the **Pointers and References** tab.

The * and & characters are automatically bound to identifiers of pointers to functions and pointers to arrays.

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

Qt DOCUMENTATION

Contribute to Qt

Forum

Wiki

Downloads

Marketplace

Feedback   Sign In