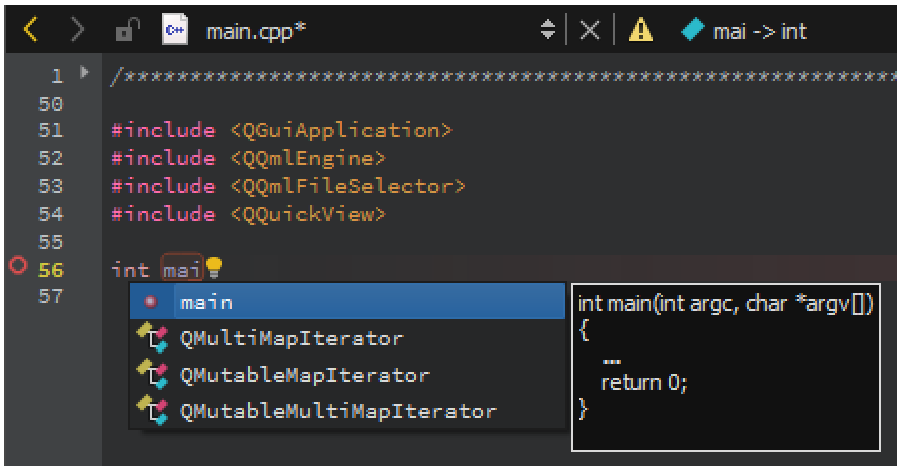
完成代码

在您编写代码时，Qt Creator 会建议属性、ID 和代码片段来完成代码。它提供了对当前游标下的语句的建议列表。按 **Tab 或 Enter** 接受所选建议并完成代码。

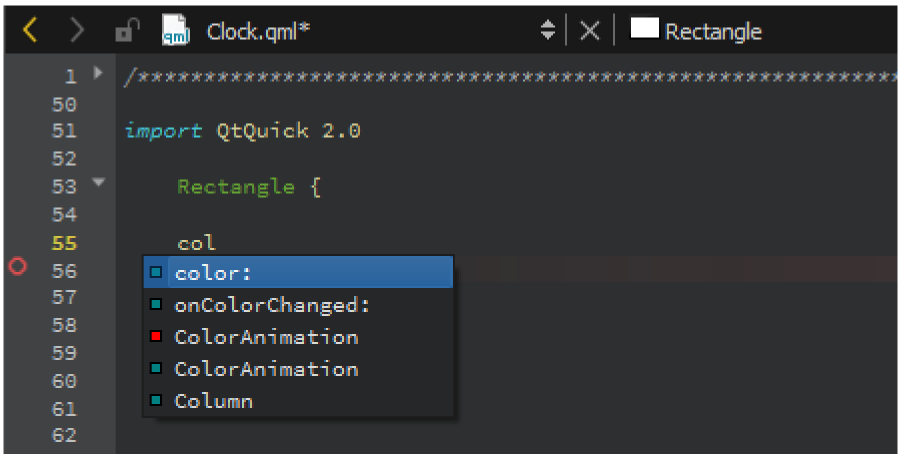
下图显示了完成C++代码的建议：



```
/*******************************************************************

#include <QGuiApplication>
#include <QQmlEngine>
#include <QQmlFileSelector>
#include <QQuickView>

int mai
    main
    QMultiMapIterator
    QMutableMapIterator
    QMutableMultiMapIterator
```

```
int main(int argc, char *argv[])
{
  ...
  return 0;
}
```

下图显示了完成 QML 代码的建议：

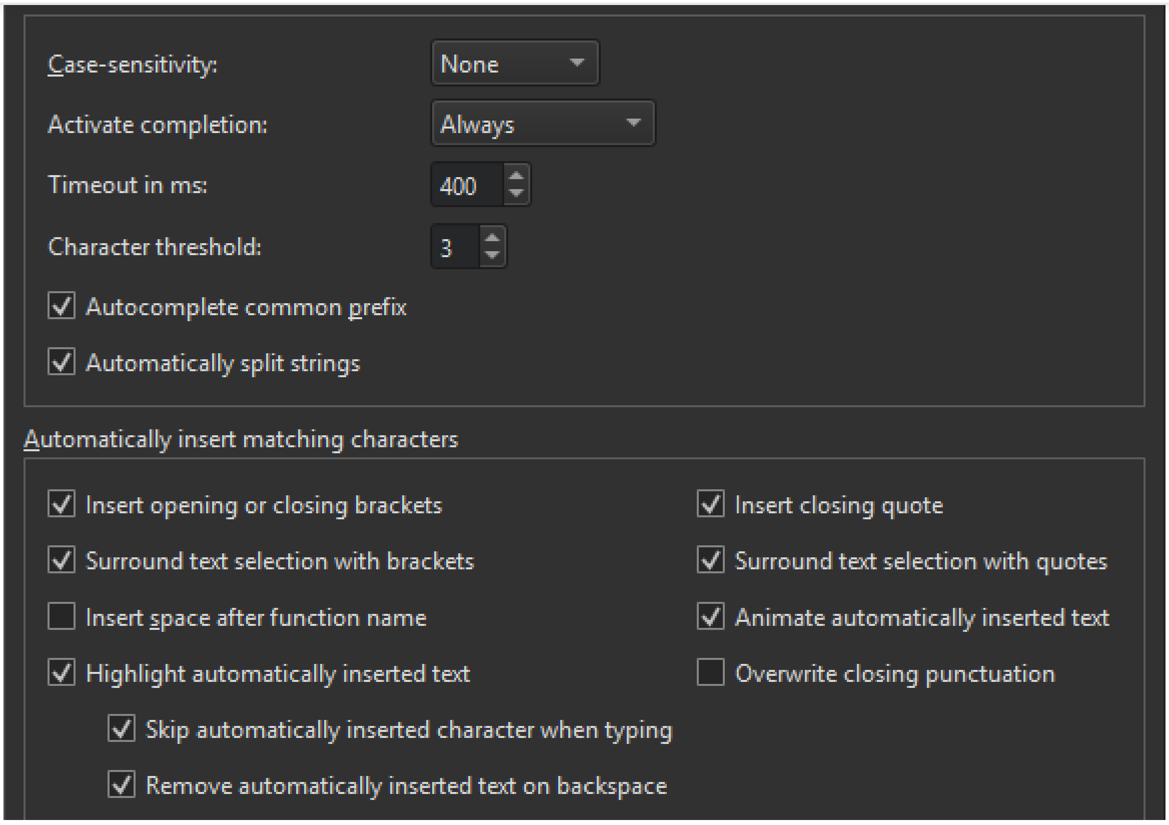


```
/*******************************************************************

import QtQuick 2.0

    Rectangle {

    col
     color:
     onColorChanged:
     ColorAnimation
     ColorAnimation
     Column
```

若要随时打开建议列表，请按 **Ctrl+空格键**。如果只有一个选项可用，Qt创建者会自动插入它。

指定完成设置

若要指定代码完成的设置，请选择"**编辑**>**首选项**">"**文本编辑器**">**完成**"。



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
- [✓] Surround text selection with brackets
- [✓] Surround text selection with quotes
- [ ] Insert space after function name
- [✓] Animate automatically inserted text
- [✓] Highlight automatically inserted text
- [ ] Overwrite closing punctuation
  - [✓] Skip automatically inserted character when typing
  - [✓] Remove automatically inserted text on backspace

默认情况下，代码完成不考虑大小写。若要应用完整字母或首字母区分大小写，请在"区分**大小写**"字段中选择"**完整**"或"**首字母**"。

默认情况下，始终调用代码完成，但您可以在"**激活完成**"字段中更改此行为，以**手动**或**触发时**调用它。

可以在"超时（毫秒）"字段中设置代码完成的超时（以毫秒**为单位**）。

在"**字符阈值**"字段中，指定在触发代码完成之前需要输入的字符数。

手动调用完成时，Qt 创建者将完成建议列表的通用前缀。这对于具有多个名称相似的成员的类特别有用。要禁用此功能，请取消选中"**自动完成通用前缀**"复选框。

选中"**自动拆分字符串**"复选框，通过在光标位置添加结束引号（按 **Enter** 键时）将字符串拆分为两行，并在下一行的开头（在字符串的其余部分之前）添加一个开始引号。此外，按 **Shift+Enter 会在**光标位置插入一个转义字符，并将字符串的其余部分移动到下一行。

Summary of Available Types

The following table lists available types for code completion and icon used for each.

Icon	Description
◆	A class
▤	An enum
▤	An enumerator (value of an enum)
◆	A function
◆	A private function
◆	A protected function
◆	A variable

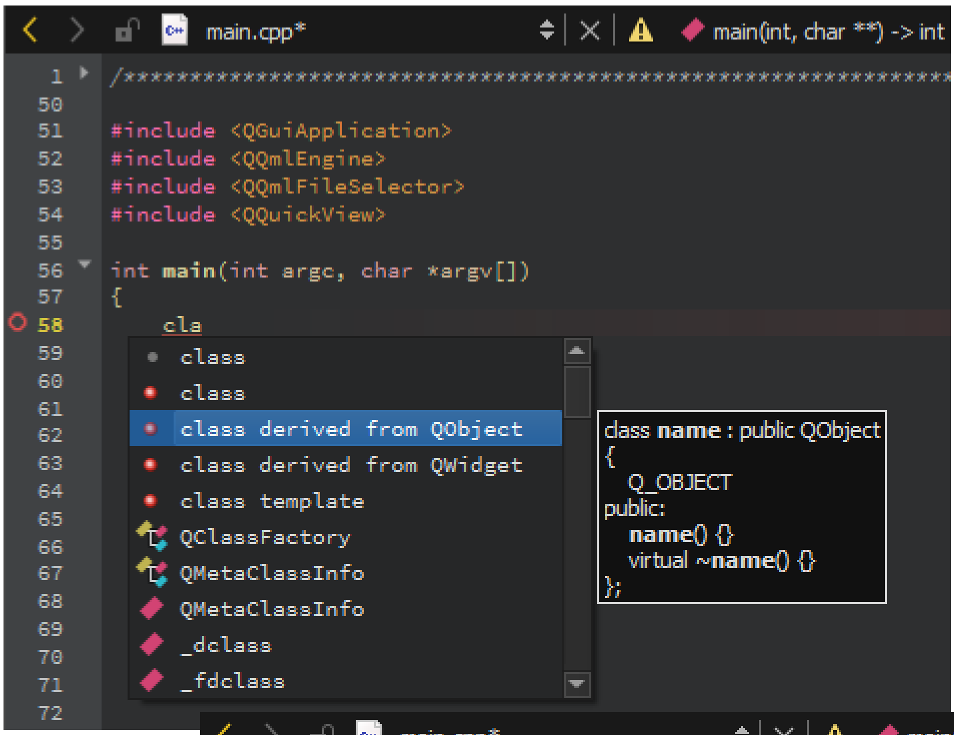
Icon Description

🔑	A protected variable
◆	A signal
◆	A slot
🔒	A private slot
🔑	A protected slot
●	A C++ keyword
●	A C++ code snippet
■	A QML type
■	A QML code snippet
▭	A macro
⠿	A namespace

Completing Code Snippets

Code snippets can consist of multiple variables that you specify values for. Select an item in the list and press **Tab** or **Enter** to complete the code. Press **Tab** to move between the variables and specify values for them. When you specify a value for a variable, all instances of the variable within the snippet are renamed.

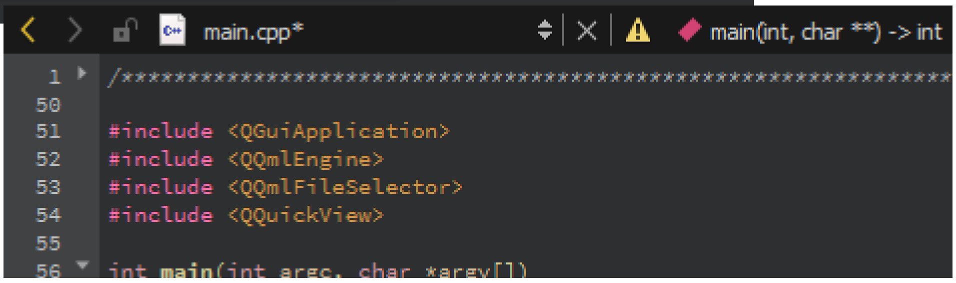
The following image shows a C++ code snippet:



main.cpp* main(int, char **) -> int

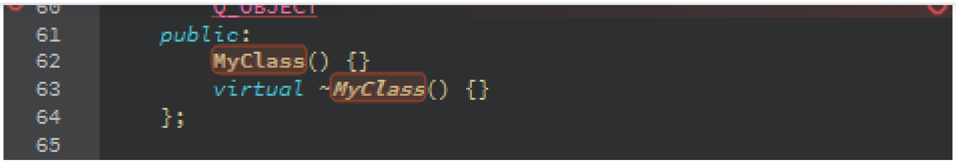
```
/***********************************************************************

#include <QGuiApplication>
#include <QQmlEngine>
#include <QQmlFileSelector>
#include <QQuickView>

int main(int argc, char *argv[])
{
    cla
```

```
class name : public QObject
{
    Q_OBJECT
public:
    name() {}
    virtual ~name() {}
};
```

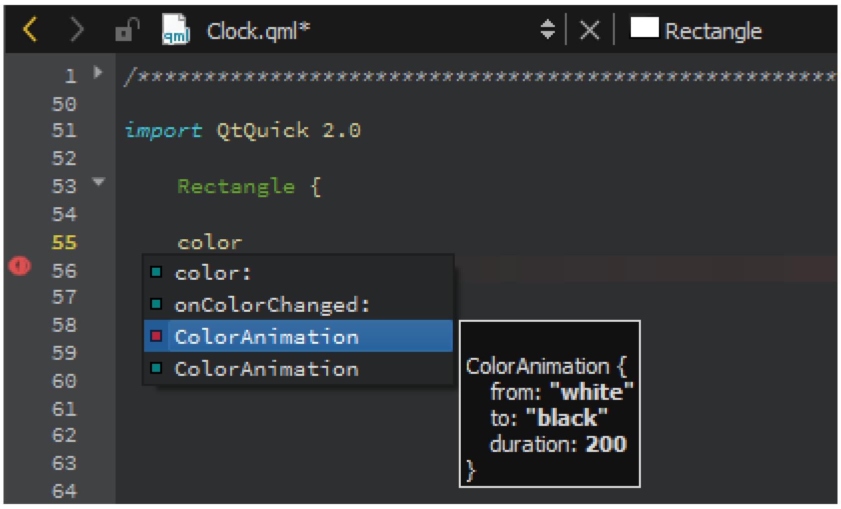


main.cpp* main(int, char **) -> int

```
/***********************************************************************

#include <QGuiApplication>
#include <QQmlEngine>
#include <QQmlFileSelector>
#include <QQuickView>

int main(int argc, char *argv[])
```


```
    Q_OBJECT
public:
    MyClass() {}
    virtual ~MyClass() {}
};
```

The following image shows a QML code snippet:

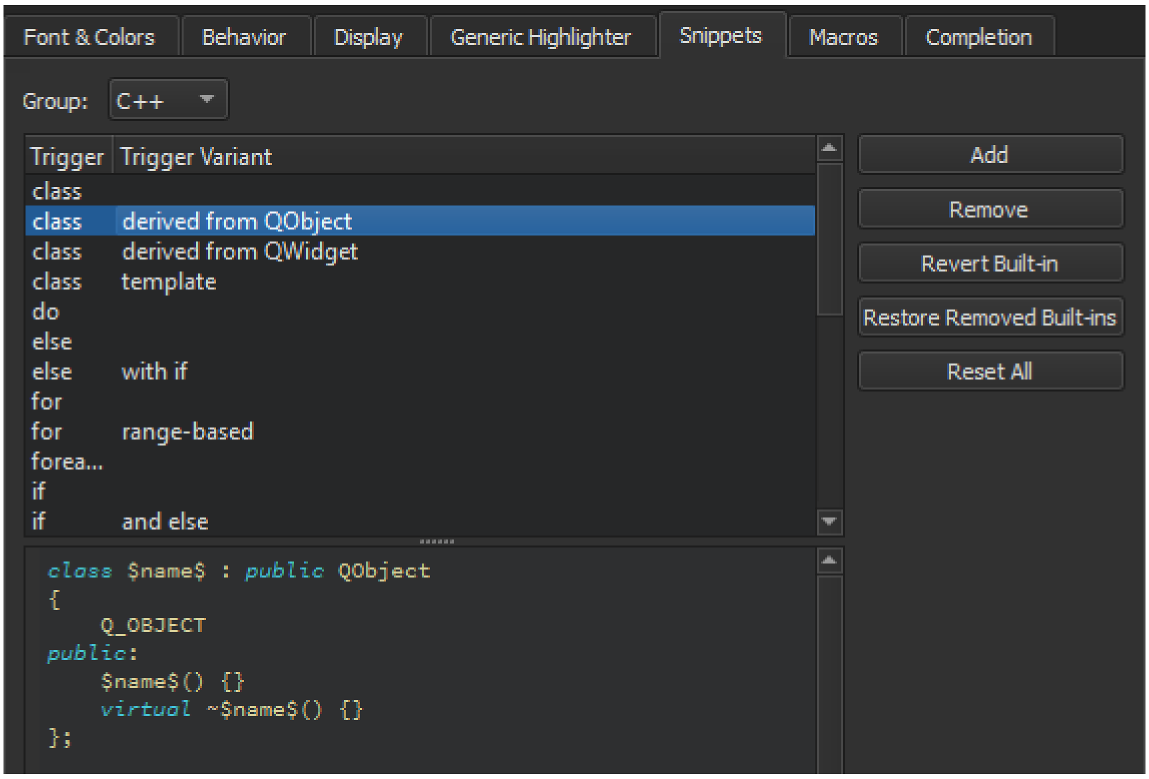


```
/*********************************************************
import QtQuick 2.0

    Rectangle {

    color
    color:
    onColorChanged:
    ColorAnimation
    ColorAnimation
```
```
ColorAnimation {
    from: "white"
    to: "black"
    duration: 200
}
```

Editing Code Snippets

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

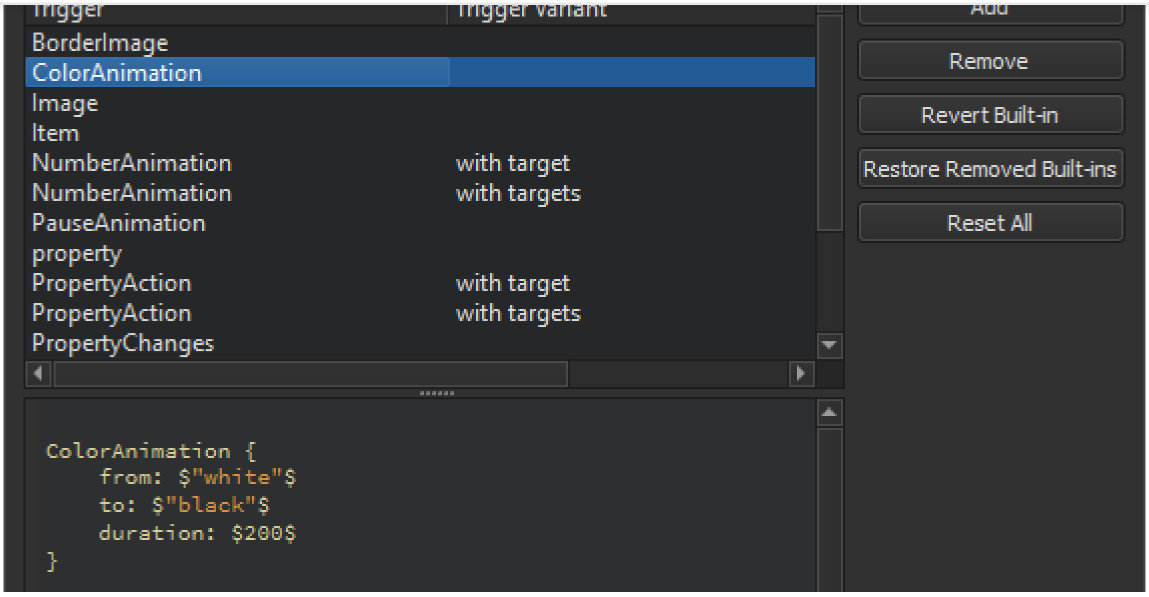
The following image shows built-in C++ code snippets:



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

```
class $name$ : public QObject
{
    Q_OBJECT
public:
    $name$() {}
    virtual ~$name$() {}
};
```

The following image shows built-in QML code snippets:



Trigger	Trigger Variant
BorderImage	
ColorAnimation	
Image	
Item	
NumberAnimation	with target
NumberAnimation	with targets
PauseAnimation	
property	
PropertyAction	with target
PropertyAction	with targets
PropertyChanges	

```
ColorAnimation {
    from: $"white"$
    to: $"black"$
    duration: $200$
}
```

Qt Creator provides you with built-in snippets in the following categories:

- Text snippets, which can contain any text string. For example, code comments
- C++ code snippets, which specify C++ code constructs
- CMake code snippets that you can use when editing files in the CMake editor`CMakeLists.txt`
- QML code snippets, which specify QML code constructs
- Nim code snippets, which specify Nim code constructs

Adding and Editing Snippets

Select a snippet in the list to edit it in the snippet editor. To add a new snippet, select **Add**. Specify a trigger and, if the trigger is already in use, an optional variant, which appear in the list of suggestions when you write code. Also specify a text string or C++ or QML code construct in the snippet editor, depending on the snippet category. You can use predefined variables in snippets.

The snippet editor provides you with:

- Highlighting
- Indentation
- Parentheses matching
- Basic code completion

Specify the variables for the snippets in the following format:

```
$variable$
```

Specify Qt Creator variables in the following format:

```
%{variable}
```

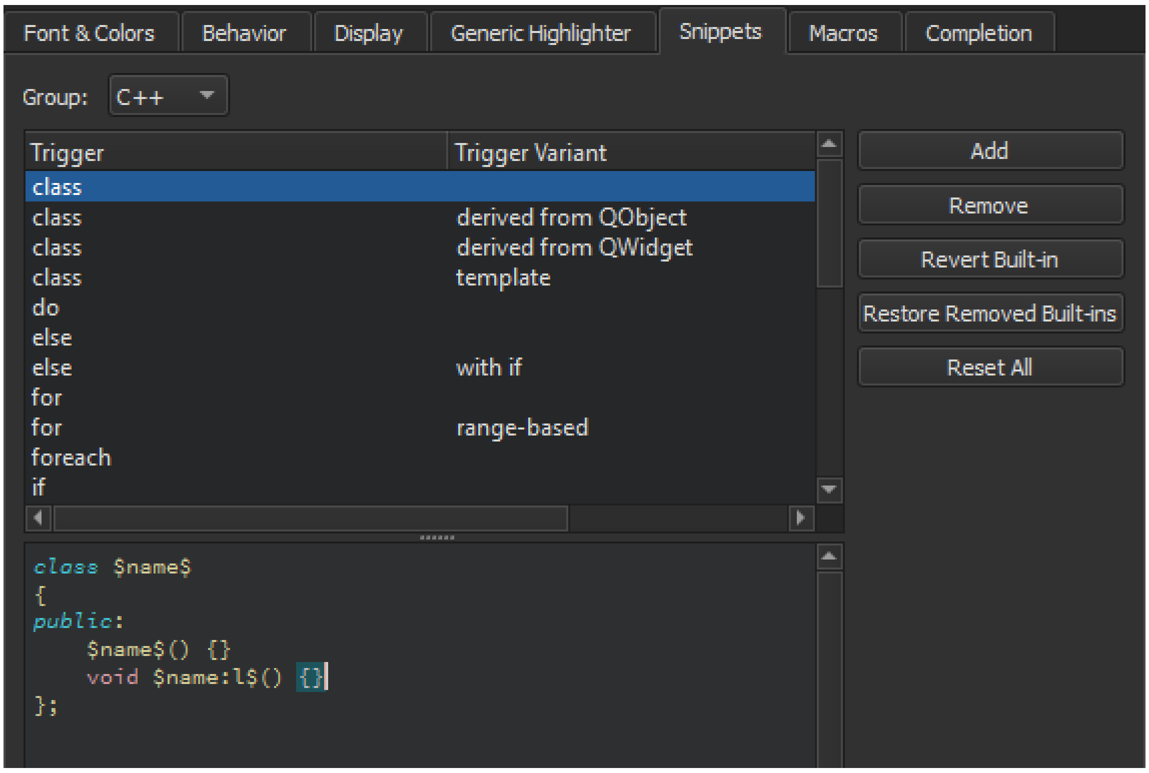
Use unique variable names within a snippet because all instances of a variable are renamed when you specify a value for it.

To determine the case of values you enter in snippets, use the following modifiers:

- `:c` converts the initial letter of the string to upper case
- `:l` converts the string to lower case
- `:u` converts the string to upper case

For example, add the following line to the snippet to specify that the function name is converted to all lower case characters regardless of how you specify the value of the variable:`class$name$`

```
void $name:l$() {}
```



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 $name:l$() {}
};
```

The snippet editor does not check the syntax of the snippets that you edit or add. However, when you use the snippets, the code editor marks any errors by underlining them in red.

To discard the changes you made to a built-in snippet, select **Revert Built-in**.

Removing Snippets

Several similar built-in snippets might be provided for different use cases. To make the list of suggestions shorter when you write code, remove the built-in snippets that you do not need. If you need them later, you can restore them.

To remove snippets, select a snippet in the list, and then select **Remove**. To restore the removed snippets, select **Restore Removed Built-ins**.

Note: If you now select **OK** or **Apply**, you permanently lose all your own snippets.

Completing Nim Code

You can use the Nimsuggest tool to query source files and obtain suggestions for code completion.`.nim`

To use Nimsuggest, you must install it on the development PC. Then select **Edit** > **Preferences** > **Nim** > **Tools**, and enter the path to the tool executable in the **Path** field.

‹ Checking Code Syntax Indenting Text or Code ›

Qt DOCUMENTATION

Wiki

Downloads

Marketplace

Feedback Sign In