

应用重构操作

Qt Creator 允许您通过在上下文菜单中选择操作（快速修复）来快速方便地应用操作（快速修复）来重构代码。可用的操作取决于光标在代码编辑器中的位置。

若要将重构操作应用于 C++ 代码，请右键单击操作数、条件语句、字符串或名称以打开上下文菜单。要将重构操作应用于 QML 代码，请右键单击项目 ID 或名称。

在上下文菜单中，选择“**重构**”，然后选择重构操作。

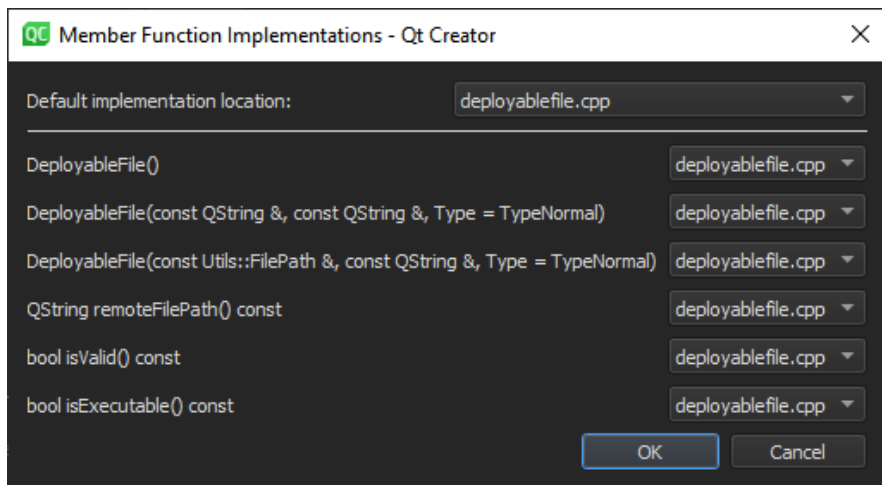
还可以按 **Alt+Enter** 打开上下文菜单，其中包含当前光标位置中可用的重构操作。

创建函数

您可以应用重构操作来实现成员函数、插入基类的虚函数、创建 getter 和 setter 函数以及生成构造函数。您可以指定用于为所有项目全局生成函数的设置，也可以为项目的 **生成和运行** 设置中的每个项目单独生成函数。

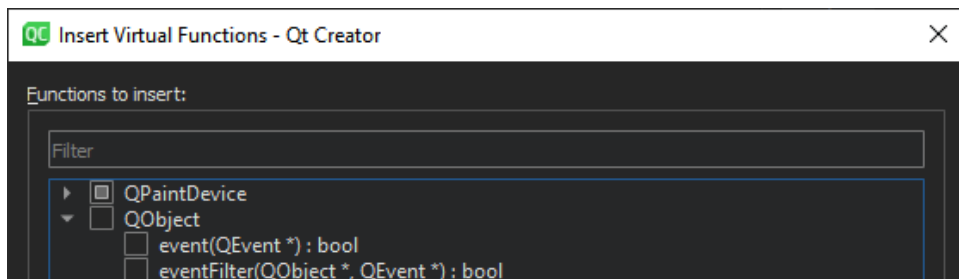
实现成员函数

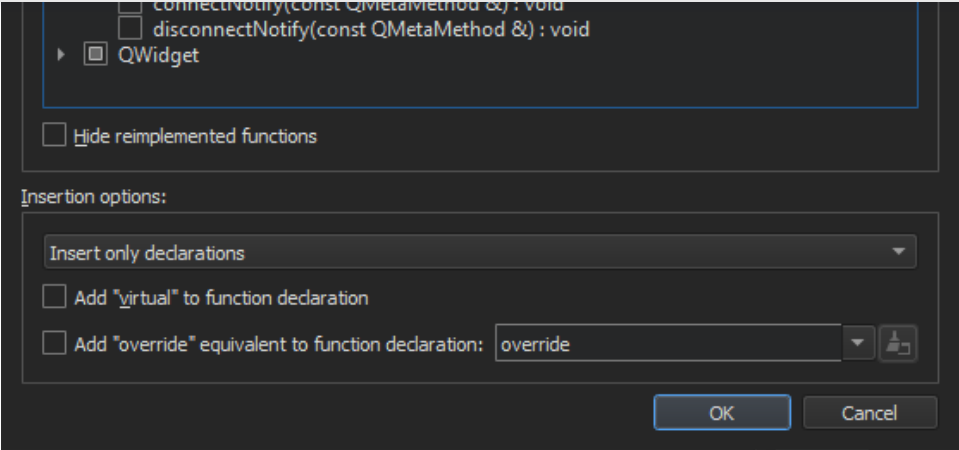
您可以应用“**创建成员函数的实现**”重构操作，一次性为所有成员函数创建实现。在“**成员函数实现**”对话框中，可以指定成员函数是在类内部生成还是在类外部生成。



插入虚函数

可以应用“**插入基类的虚函数**”重构操作，在类内部或外部或实现文件（如果存在）中插入声明和相应的定义。



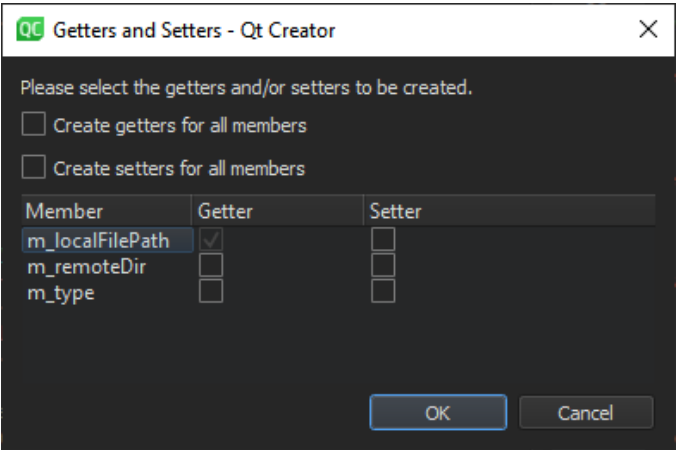


选择要插入到可用函数列表中的函数。您可以过滤列表并从中隐藏重新实现的函数。

您可以添加*虚拟*或与函数声明等效的*重写*。

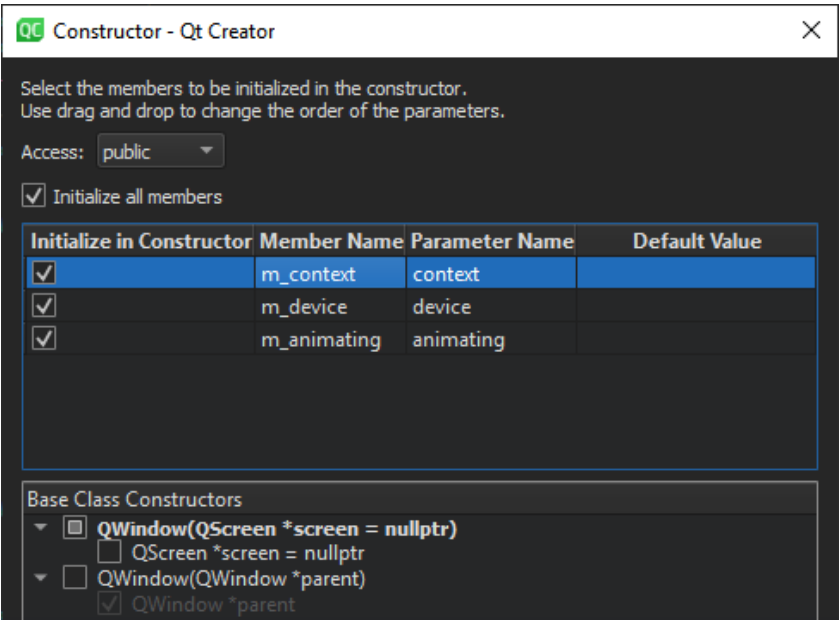
创建获取者和设置者

您可以应用“**创建获取器**”和“**设置器成员函数**”重构操作，为成员变量创建 getter 和 setter 成员函数，或者仅创建 getter 或 setter。



Generating Constructors

You can apply the **Generate Constructor** refactoring action to create a public, protected, or private constructor for a class. Select the class members to initialize in the constructor. Drag and drop the parameters to specify their order in the constructor.

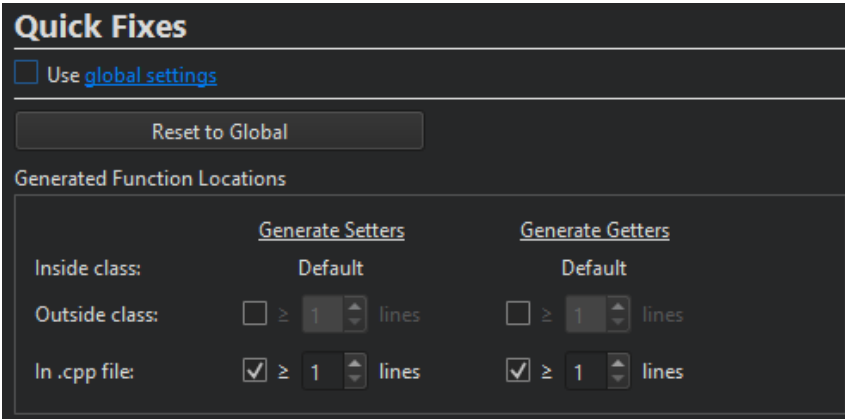




Specifying Settings for Refactoring Actions

You can specify settings for the refactoring actions either globally for all projects or separately for each project. To specify global options, select **Edit > Preferences > C++ > Quick Fixes**.

To specify custom settings for a particular project, select **Projects > Project Settings > Quick Fixes**, and then deselect **Use global settings**.



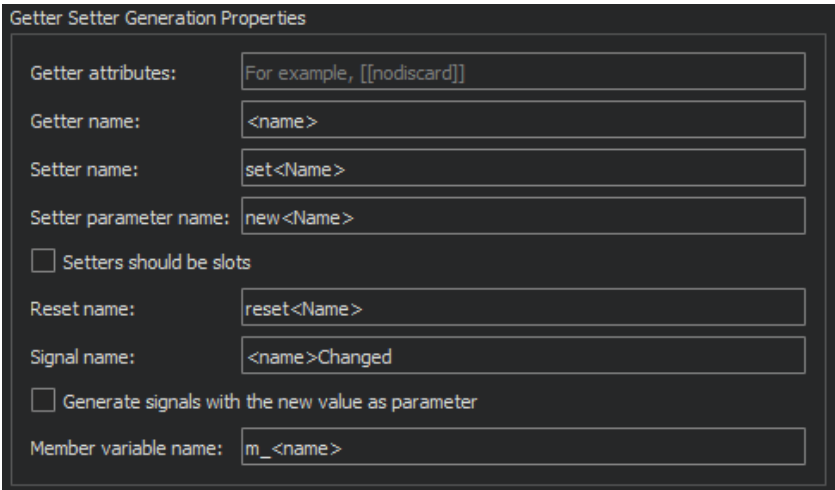
To revert to global settings, select **Reset to Global**. To delete the custom settings, select **Use global settings**, and then select **Delete Custom Settings File**.

Function Locations

In the **Generated Function Locations** group, you can determine whether refactoring actions should generate getter and setter functions in the header file (inside or outside the class) or in the implementation file.

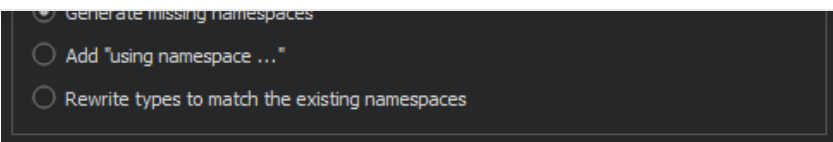
Function Names and Attributes

In the **Getter Setter Generation Properties** group, you can specify additional settings for getter and setter names, attributes, and parameters. You can specify that setter functions should be created as *slots* and that signals should be generated with the new value as a parameter.



Namespace Handling

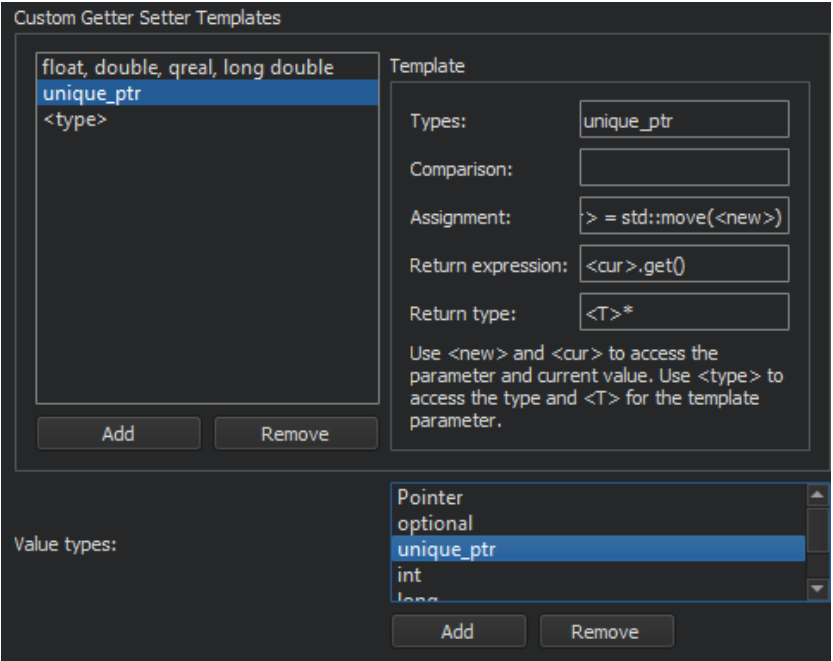
In the **Missing Namespace Handling** group, select whether to generate missing namespaces, add where necessary, or rewrite types to match the existing namespaces.using namespace



Custom Parameter Types

In the **Custom Getter Setter Templates** group, specify how the code of a getter or setter function for a certain data type should look like. This is necessary for types where assignment cannot use `,` as in the pre-defined settings for `unique_ptr` or where `==` is not suitable for comparison, as in the pre-defined settings for floating-point types. For example, if you have a special type `MyClass`, you can specify that a function `compare` should be used for comparison rather than the default of `operator==`.

To specify special handling for a custom parameter type, select **Add** and set the parameter type, comparison, return expression, and return type. In the **Return type** field, you can use `<cur>` to access the parameter and current value. Use `<type>` to access the type and `<T>` for the template parameter.



Usually, arguments are passed by using a reference. To pass arguments of a particular type as values, list them in the **Value types** field. Namespaces and template arguments are removed. The real Type must contain the given Type. For example, `std::optional<vector<int>>` matches but not `std::optional<vector<int>>`.

Summary of Refactoring Actions

If you use the **Clang code model** to parse the C++ files, the **Clang fix-it hints** that have been integrated into Qt Creator are also available to you. In addition to the standard ways of activating refactoring actions, you can select the actions that are applicable on a line in the context menu in the left margin of the code editor.

Refactoring C++ Code

You can apply the following types of refactoring actions to C++ code:

- Change binary operands
- Simplify if and while conditions (for example, move declarations out of if conditions)
- Modify strings (for example, set the encoding for a string to Latin-1, mark strings translatable, and convert symbol names to camel case)
- Create variable declarations
- Create function declarations and definitions

The following table summarizes the refactoring actions for C++ code. The action is available when the cursor is in the position described in the Activation column.

Add Curly Braces	<div>Adds curly braces to an if statement that does not contain a compound statement. For example, rewrites</div> <div><pre>if (a) b;</pre></div> <div>as</div> <div><pre>if (a) { b; }</pre></div>	if
Move Declaration out of Condition	<div>Moves a declaration out of an if or while condition to simplify the condition. For example, rewrites</div> <div><pre>if (Type name = foo()) {}</pre></div> <div>as</div> <div><pre>Type name = foo; if (name) {}</pre></div>	Name of the introduced variable
Rewrite Condition Using	<div>Rewrites the expression according to De Morgan's laws. For example, rewrites:</div> <div><pre>!a && !b</pre></div> <div>as</div> <div><pre>!(a b)</pre></div>	&&
Rewrite Using operator	<div>Rewrites an expression negating it and using the inverse operator. For example, rewrites:</div> <div><pre>> a op b</pre></div> <div>as</div> <div><pre>!(a invop b)</pre></div> <div><pre>> (a op b)</pre></div>	<=, ..., or <>=== !=
Refactoring Action		Activation

	<div>!(a invop b)</div> <div>> !(a op b)</div> <div>as</div> <div>(a invob b)</div>	
Split Declaration	<div>Splits a simple declaration into several declarations. For example, rewrites:</div> <div>int *a, b;</div> <div>as</div> <div>int *a; int b;</div>	Type name or variable name
Split if Statement	<div>Splits an if statement into several statements. For example, rewrites:</div> <div>if (something && something_else) { }</div> <div>as</div> <div>if (something) { if (something_else) { } }</div> <div>and</div> <div>if (something something_else) x;</div> <div>with</div> <div>if (something) x; else if (something_else) x;</div>	&& or
Refactoring Action	Description	Activation

Swap Operands	<p>rewrites an expression in the inverse order using the inverse operator. For example, rewrites:</p> <pre>a op b</pre> <p>as</p> <pre>b flipop a</pre>	<pre><>===!=&& </pre>
Convert to Decimal	Converts an integer literal to decimal representation	Numeric literal
Convert to Hexadecimal	Converts an integer literal to hexadecimal representation	Numeric literal
Convert to Octal	Converts an integer literal to octal representation	Numeric literal
Convert to Objective-C String Literal	<p>Converts a string literal to an Objective-C string literal if the file type is Objective-C(++). For example, rewrites the following strings</p> <pre>"abcd" QLatin1String("abcd") QLatin1Literal("abcd")</pre> <p>as</p> <pre>@"abcd"</pre>	String literal
Enclose in QLatin1Char()	<p>Sets the encoding for a character to Latin-1, unless the character is already enclosed in <code>QLatin1Char</code>, <code>QT_TRANSLATE_NOOP</code>, <code>tr</code>, <code>trUtf8</code>, <code>QLatin1Literal</code>, or <code>QLatin1String</code>. For example, rewrites</p> <pre>'a'</pre> <p>as</p> <pre>QLatin1Char('a')</pre>	String literal
Enclose in QLatin1String()	<p>Sets the encoding for a string to Latin-1, unless the string is already enclosed in <code>QLatin1Char</code>, <code>QT_TRANSLATE_NOOP</code>, <code>tr</code>, <code>trUtf8</code>, <code>QLatin1Literal</code>, or <code>QLatin1String</code>. For example, rewrites</p> <pre>"abcd"</pre> <p>as</p> <pre>QLatin1String("abcd")</pre>	String literal
Refactoring Action	Description	Activation

Mark as Translatable	<p>Marks a string translatable. For example, rewrites with one of the following options, depending on which of them is available:"abcd"</p> <pre>tr("abcd") CoreApplication::translate("CONTEXT", "abcd") QT_TRANSLATE_NOOP("GLOBAL", "abcd")</pre>	String literal
Add Definition in ...	<p>Inserts a definition stub for a function declaration either in the header file (inside or outside the class) or in the implementation file. For free functions, inserts the definition after the declaration of the function or in the implementation file. Qualified names are minimized when possible, instead of always being fully expanded. For example, rewrites</p> <pre>Class Foo { void bar(); };</pre> <p>as (inside class)</p> <pre>Class Foo { void bar() { } };</pre> <p>as (outside class)</p> <pre>Class Foo { void bar(); }; void Foo::bar() { }</pre> <p>as (in implementation file)</p> <pre>// Header file Class Foo { void bar(); }; // Implementation file void Foo::bar() { }</pre>	Function name
Refactoring Action	Description	Activation
Add	Inserts the member function declaration that matches the member function definition into	Function name

Add Class Member	Adds a member declaration for the class member being initialized if it is not yet declared. You must enter the data type of the member.	Identifier
Create Implementations for Member Functions	Creates implementations for all member functions in one go. In the Member Function Implementations dialog, you can specify whether the member functions are generated inline or outside the class.	Function name
Switch with Next/Previous Parameter	Moves a parameter down or up one position in a parameter list.	Parameter in the declaration or definition of a function
Extract Function	Moves the selected code to a new function and replaces the block of code with a call to the new function. Enter a name for the function in the Extract Function Refactoring dialog.	Block of code selected
Extract Constant as Function Parameter	Replaces the selected literal and all its occurrences with the function parameter . The parameter will have the original literal as the default value. newParameternewParameter	Block of code selected
Add Local Declaration	Adds the type of an assignee, if the type of the right-hand side of the assignment is known. For example, rewrites <div>a = foo();</div> as <div>Type a = foo();</div> where Type is the return type of foo()	Assignee
Convert to Camel Case	Converts a symbol name to camel case, where elements of the name are joined without delimiter characters and the initial character of each element is capitalized. For example, rewrites as and as an_example_symbolanExampleSymbolAN_EXAMPLE_SYMBOLAnExampleSymbol	Identifier
Complete Switch Statement	Adds all possible cases to a switch statement of the type enum	switch
Generate Missing Q_PROPERTY Members	Adds missing members to a :Q_PROPERTY <div>> read function</div> <div>> wr i te function, if there is a WRITE</div> <div>> onChanged signal, if there is a NOTIFY</div> <div>> data member with the name m_<propertyName></div>	Q_PROPERTY
Generate Q_PROPERTY and Missing Members	Generates a Q_PROPERTY and adds missing members to it, as described above.	Class member
Generate Constant Q_PROPERTY and Missing Members	Generates a constant Q_PROPERTY and adds missing members to it, as described above.	Class member
Generate Q_PROPERTY and Missing Members with Reset Function	Generates a Q_PROPERTY and adds missing members to it, as described above, but with an additional function.reset	Class member
Apply Changes	Keeps function declarations and definitions synchronized by checking for the matching declaration or definition when you edit a function signature and by applying the changes to the matching code.	Function signature. When this action is available, a light
Refactoring Action	Description	Activation

Add #include for undeclared or forward declared identifier	Adds an directive to the current file to make the definition of a symbol available. <code>#include</code>	Undeclared identifier
Add Forward Declaration	Adds a forward declaration for an undeclared identifier operation.	Undeclared identifier
Reformat Pointers or References	<p>Reformats declarations with pointers or references according to the code style settings for the current project. In case no project is open, the current global code style settings are used.</p> <p>For example, rewrites:</p> <pre>char*s;</pre> <p>as</p> <pre>char *s;</pre> <p>When applied to selections, all suitable declarations in the selection are rewritten.</p>	Declarations with pointers or references and selections containing such declarations
Create Getter and Setter Member Functions	Creates either both getter and setter member functions for member variables or only a getter or setter.	Member variable in class definition
Generate Getter and Setter	Creates getter and setter member functions for a member variable.	Member variable in class definition
Generate Getter	Creates a getter member function for a member variable.	Member variable in class definition
Generate Setter	Creates a setter member function for a member variable.	Member variable in class definition
Generate Constructor	Creates a constructor for a class.	Class definition
Move Function Definition	<p>Moves a function definition to the implementation file, outside the class or back to its declaration. For example, rewrites:</p> <pre>class Foo { void bar() { // do stuff here } };</pre> <p>as</p> <pre>class Foo { void bar(); }; void Foo::bar() { // do stuff here }</pre>	Function signature
Refactoring Action	Description	Activation

	<pre>class Foo { void bar() { // do stuff here } void baz() { // do stuff here } };</pre> <p>as</p> <pre>class Foo { void bar(); void baz(); }; void Foo::bar() { // do stuff here } void Foo::baz() { // do stuff here }</pre>	
Assign to Local Variable	<p>Adds a local variable which stores the return value of a function call or a new expression. For example, rewrites:</p> <pre>QString s; s.toLatin1();</pre> <p>as</p> <pre>QString s; QByteArray latin1 = s.toLatin1();</pre> <p>and</p> <pre>new Foo;</pre> <p>as</p> <pre>Foo * localFoo = new Foo;</pre>	Function call or class name
Insert Virtual	Inserts declarations and the corresponding definitions inside or outside the class or in an	Class or base
Refactoring Action	Description	Activation

Optimize for-Loop	<p>Rewrites post increment operators as pre increment operators and post decrement operators as pre decrement operators. It also moves other than string or numeric literals and id expressions from the condition of a for loop to its initializer. For example, rewrites:</p> <pre>for (int i = 0; i < 3 * 2; i++)</pre> <p>as</p> <pre>for (int i = 0, total = 3 * 2; i < total; ++i)</pre>	for
Escape String Literal as UTF-8	<p>Escapes non-ASCII characters in a string literal to hexadecimal escape sequences. String Literals are handled as UTF-8.</p>	String literal
Unescape String Literal as UTF-8	<p>Unescapes octal or hexadecimal escape sequences in a string literal. String Literals are handled as UTF-8.</p>	String literal
Convert to Stack Variable	<p>Converts the selected pointer to a stack variable. For example, rewrites:</p> <pre>QByteArray *foo = new QByteArray("foo"); foo->append("bar");</pre> <p>as</p> <pre>QByteArray foo("foo"); foo.append("bar");</pre> <p>This operation is limited to work only within function scope. Also, the coding style for pointers and references is not respected yet.</p>	Pointer Variable
Convert to Pointer	<p>Converts the selected stack variable to a pointer. For example, rewrites:</p> <pre>QByteArray foo = "foo"; foo.append("bar");</pre> <p>as</p> <pre>QByteArray *foo = new QByteArray("foo"); foo->append("bar");</pre> <p>This operation is limited to work only within function scope. Also, the coding style for pointers and references is not respected yet.</p>	Stack Variable
Remove and Adjust Type Names Accordinglyusing namespace	<p>Remove occurrences of in the local scope and adjust type names accordingly.using namespace</p>	using directive
Remove All Occurrences of in Global Scope and Adjust Type Names	<p>Remove all occurrences of in the global scope and adjust type names accordingly.using namespace</p>	using directive
Refactoring Action	Description	Activation

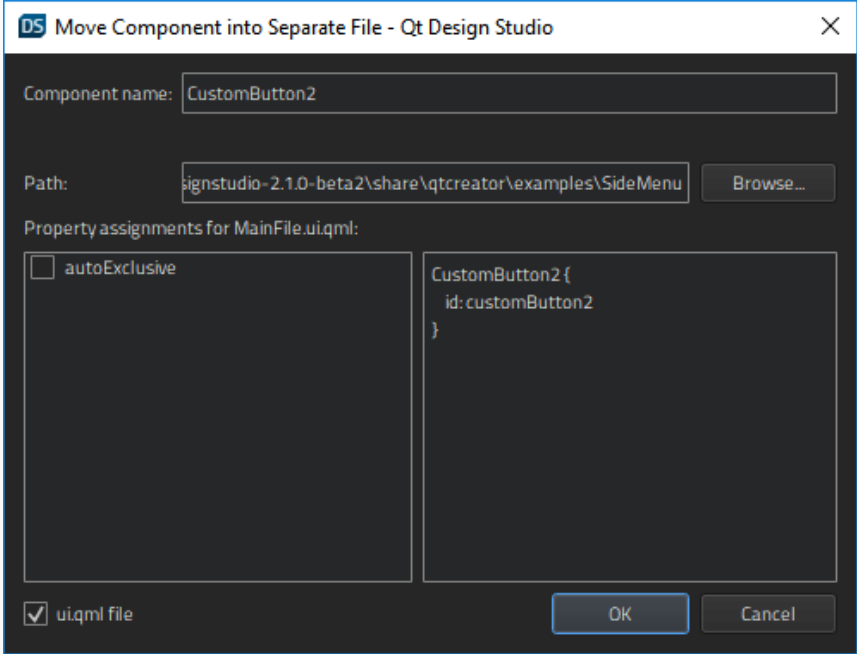
Convert connect() to Qt 5 Style	Converts a Qt 4 <code>QObject::connect()</code> to Qt 5 style.	<code>QObject::connect()</code> (Qt 4 style)
---------------------------------	--	--

Refactoring QML Code

You can apply the following types of refactoring actions to QML code:

- › Rename IDs
- › Split initializers
- › Move a QML type into a separate file to reuse it in other .qml files

The following table summarizes the refactoring actions for QML code. The action is available when the cursor is in the position described in the Activation column.

Refactoring Action	Description	Activation
Move Component into Separate File	<p>Moves a QML type into a separate file. Give the new component a name and select whether properties are set for the new component or for the original one.</p> 	QML type name.
Split Initializer	<p>Reformats a one-line type into a multi-line type. For example, rewrites</p> <pre>Item { x: 10; y: 20; width: 10 }</pre> <p>as</p> <pre>Item { x: 10; y: 20; width: 10 }</pre>	QML type property
Wrap Component in Loader	Wraps the type in a Component type and loads it dynamically in a Loader type. This is usually done to improve startup time.	QML type name
Add a message suppression comment	Prepends the line with an annotation comment that stops the message from being generated.	Error, warning or hint from static analysis

© 2022 The Qt Company Ltd. Documentation contributions included herein are the copyrights of their respective owners. The documentation provided herein is licensed under the terms of the [GNU Free Documentation License version 1.3](#) as published by the Free Software Foundation. Qt and respective logos are trademarks of The Qt Company Ltd in Finland and/or other countries worldwide. All other trademarks are property of their respective owners.



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

© 2022 The Qt Company

[Feedback](#) [Sign In](#)