Applying Refactoring Actions

Qt Creator allows you to quickly and conveniently apply actions (quick fixes) to refactor your code by selecting them in a context menu. The actions available depend on the position of the cursor in the code editor.

To apply refactoring actions to C++ code, right-click an operand, conditional statement, string, or name to open a context menu. To apply refactoring actions to QML code, right-click an item ID or name.

In the context menu, select **Refactoring** and then select a refactoring action.

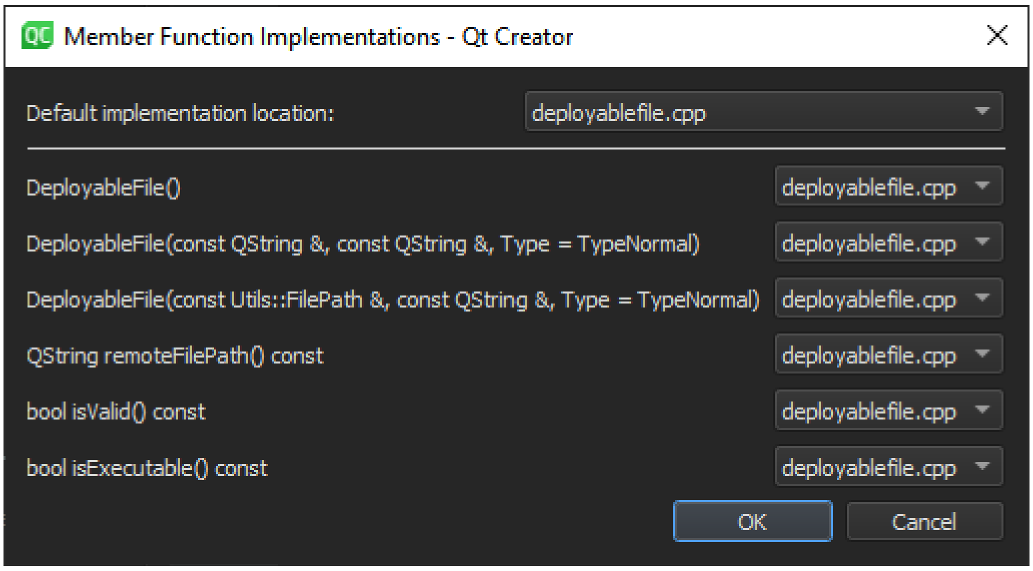
You can also press **Alt+Enter** to open a context menu that contains refactoring actions available in the current cursor position.

Creating Functions

You can apply refactoring actions to implement member functions, insert virtual functions of base classes, create getter and setter functions, and generate constructors. You can specify settings for generating the functions either globally for all projects or separately for each project in the build and run settings of the project.

Implementing Member Functions

You can apply the **Create Implementations for Member Functions** refactoring action to create 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.

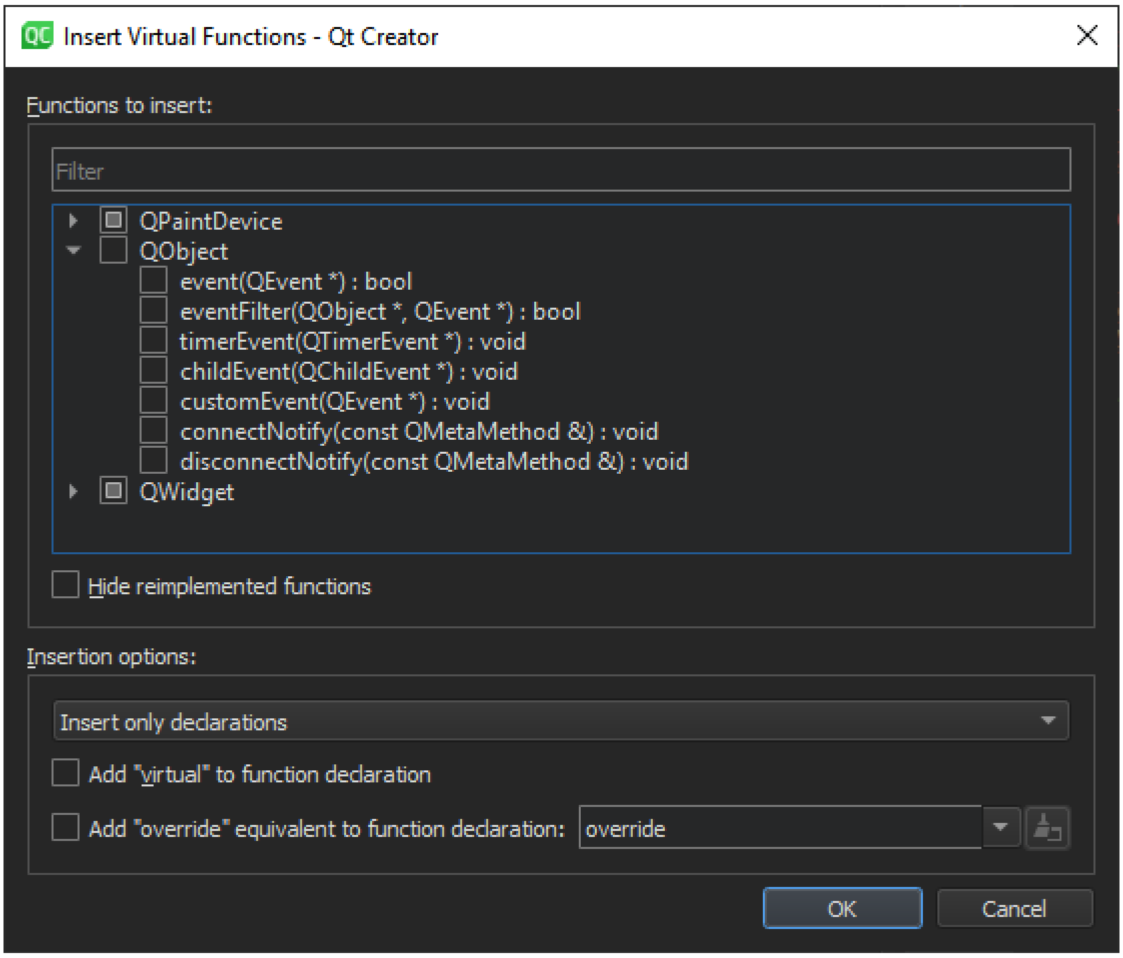

Member Function Implementations - Qt Creator

Default implementation location:	deployablefile.cpp
DeployableFile()	deployablefile.cpp
DeployableFile(const QString &, const QString &, Type = TypeNormal)	deployablefile.cpp
DeployableFile(const Utils::FilePath &, const QString &, Type = TypeNormal)	deployablefile.cpp
QString remoteFilePath() const	deployablefile.cpp
bool isValid() const	deployablefile.cpp
bool isExecutable() const	deployablefile.cpp

Inserting Virtual Functions

Qt DOCUMENTATION

corresponding definitions inside or outside the class or in an implementation file (if it exists).


Insert Virtual Functions - Qt Creator

Functions to insert:

Filter

- QPaintDevice
- QObject
  - event(QEvent *) : bool
  - eventFilter(QObject *, QEvent *) : bool
  - timerEvent(QTimerEvent *) : void
  - childEvent(QChildEvent *) : void
  - customEvent(QEvent *) : void
  - connectNotify(const QMetaMethod &) : void
  - disconnectNotify(const QMetaMethod &) : void
- QWidget

Hide reimplemented functions

Insertion options:

Insert only declarations

Add "virtual" to function declaration

Add "override" equivalent to function declaration: override

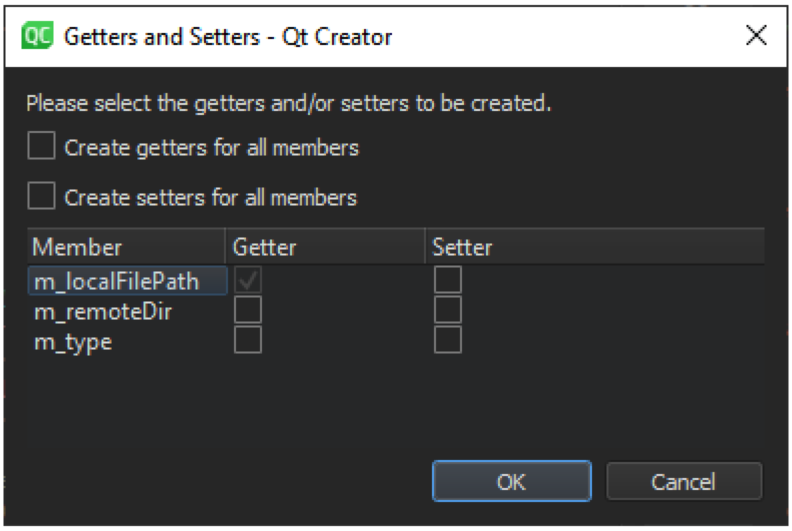
OK Cancel

Select the functions to insert in the list of available functions. You can filter the list and hide reimplemented functions from it.

You can add *virtual* or the *override* equivalent to the function declaration.

Creating Getters and Setters

You can apply the **Create Getter and Setter Member Functions** refactoring action to create either both getter and setter member functions for member variables or only a getter or setter.


Getters and Setters - Qt Creator

Please select the getters and/or setters to be created.

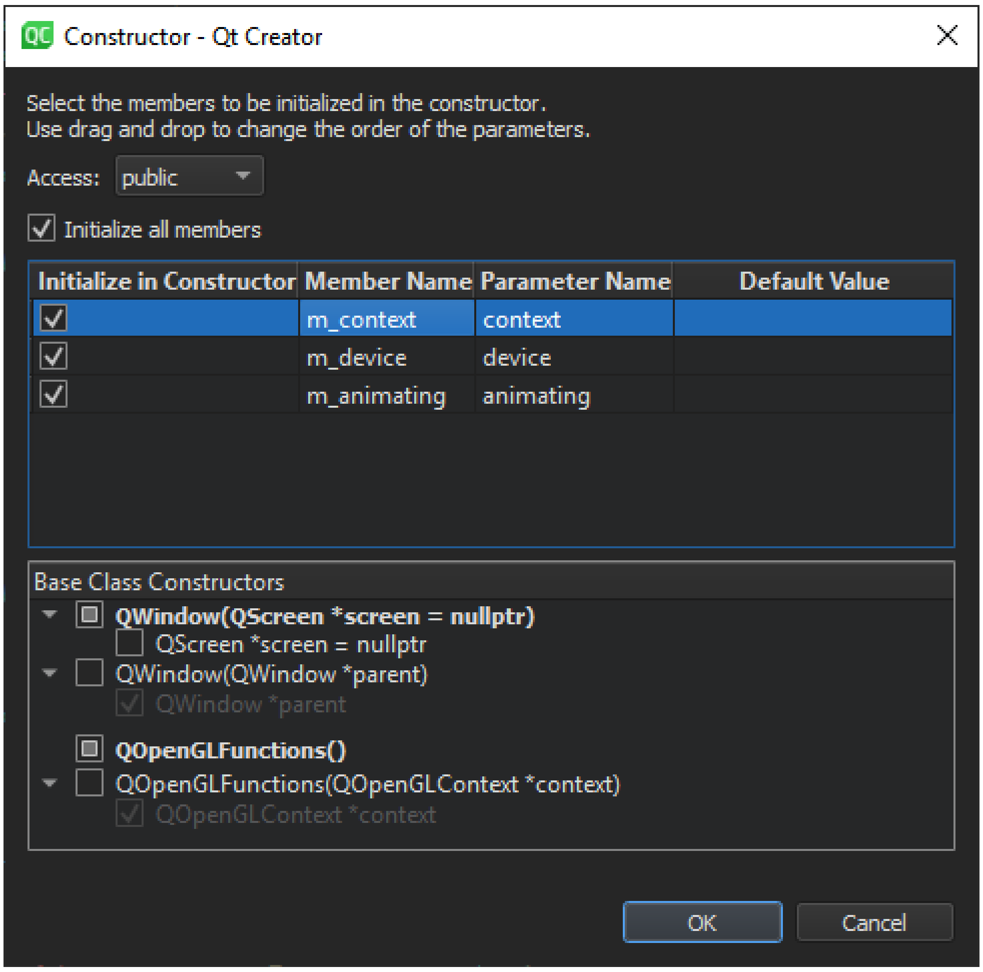
Create getters for all members

Create setters for all members

Member	Getter	Setter
m_localFilePath	✓	
m_remoteDir		
m_type		

OK Cancel

class. Select the class members to initialize in the constructor. Drag and drop the parameters to specify their order in the constructor.



Select the members to be initialized in the constructor.
Use drag and drop to change the order of the parameters.

Access: public

✓ Initialize all members

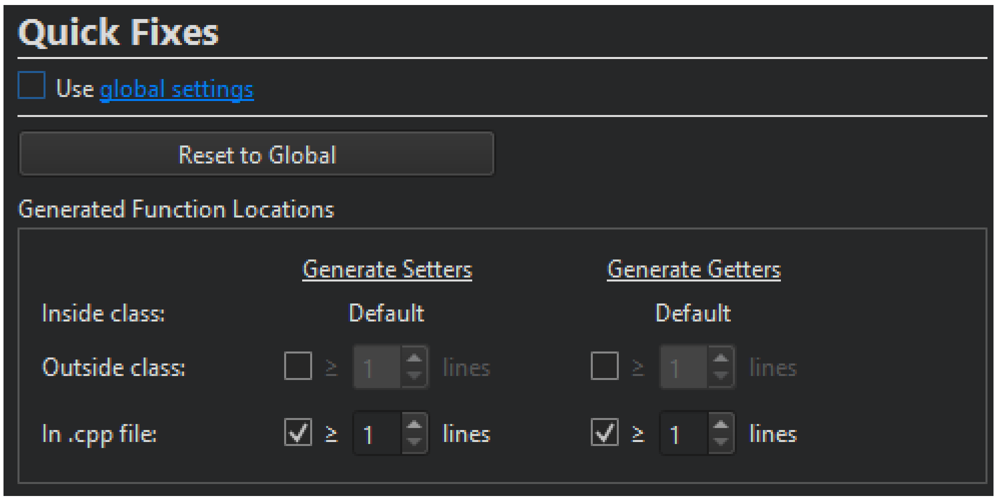
Initialize in Constructor	Member Name	Parameter Name	Default Value
✓	m_context	context	
✓	m_device	device	
✓	m_animating	animating	

Base Class Constructors
- QWindow(QScreen *screen = nullptr)
  - QScreen *screen = nullptr
- QWindow(QWindow *parent)
  - ✓ QWindow *parent
- QOpenGLFunctions()
- QOpenGLFunctions(QOpenGLContext *context)
  - ✓ QOpenGLContext *context

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**.



Use global settings

Generated Function Locations

	Generate Setters	Generate Getters
Inside class:	Default	Default
Outside class:	☐ ≥ 1 lines	☐ ≥ 1 lines
In .cpp file:	✓ ≥ 1 lines	✓ ≥ 1 lines

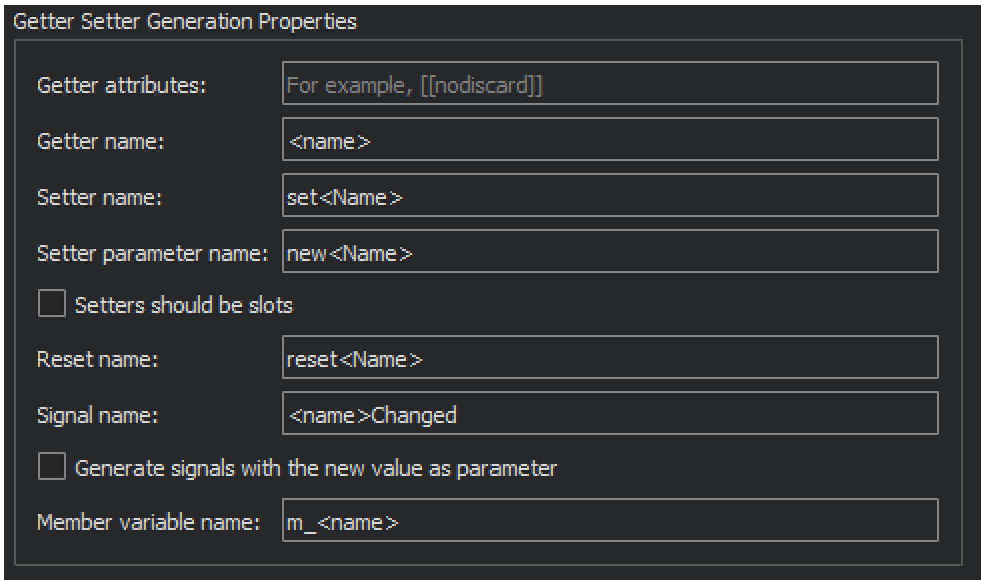
To revert to global settings, select **Reset to Global**. To delete the custom settings, select **Use global settings**, and

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

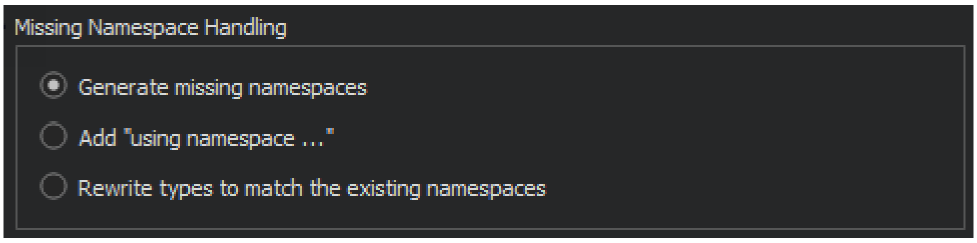
In the **Getter Setter Generation Properties** group, you can specify additional settings for getter and setter names, attributes, and parameters. You can specify that setter functions should be created as *slots* and that signals should be generated with the new value as a parameter.


Getter Setter Generation Properties

Getter attributes:	For example, [[nodiscard]]
Getter name:	<name>
Setter name:	set<Name>
Setter parameter name:	new<Name>
☐ Setters should be slots	
Reset name:	reset<Name>
Signal name:	<name>Changed
☐ Generate signals with the new value as parameter	
Member variable name:	m_<name>

Namespace Handling

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


Missing Namespace Handling

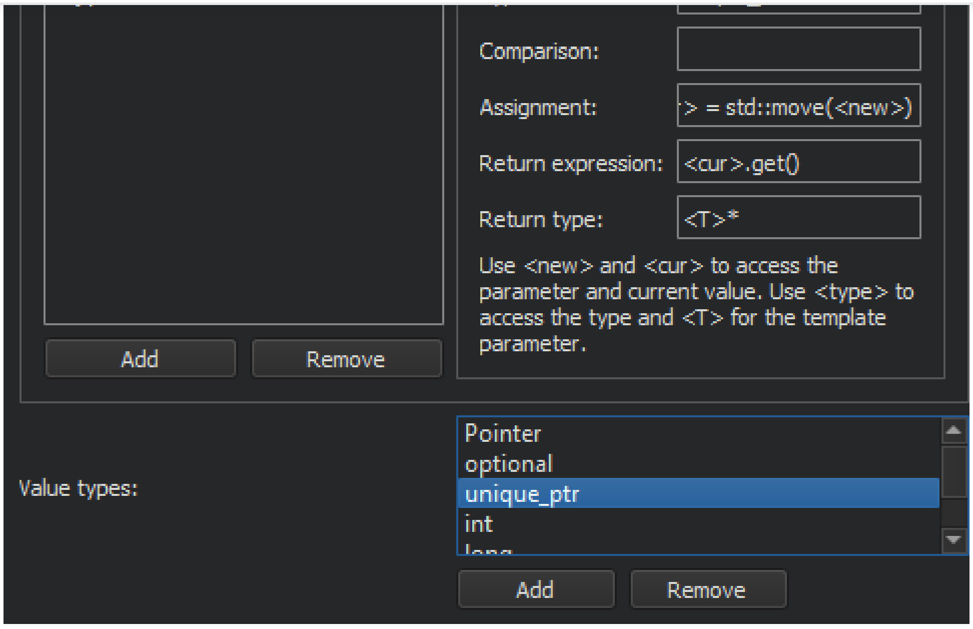
● Generate missing namespaces
○ Add "using namespace ..."
○ Rewrite types to match the existing namespaces

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 `operator=`, as in the pre-defined settings for `unique_ptr` or where `operator==` 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, `myCompare`, should be used for comparison rather than the default of `==`.

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 `<new>` and `<cur>` to access the parameter and current value. Use `<type>` to access the type and `<T>` for the template parameter.

Custom Getter Setter Templates


Comparison:
Assignment: > = std::move(<new>)
Return expression: <cur>.get()
Return type: <T>*
Use <new> and <cur> to access the parameter and current value. Use <type> to access the type and <T> for the template parameter.
Value types: Pointer, optional, unique_ptr, int, long

Usually, arguments are passed by using a `const` 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, `int` matches `int32_t` but not `vector<int>`, and `vector` matches `std::pmr::vector<int>` 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.

Refactoring Action	Description	Activation

Add Curly Braces	Adds curly braces to an if statement that does not contain a compound statement. For example, rewrites
	`if (a)`
	`    b;`
		`if`

Refactoring Action	Description	Activation

	`if (a) {`
	`    b;`
	`}`	

Move Declaration out of Condition	Moves a declaration out of an if or while condition to simplify the condition. For example, rewrites
	`if (Type name = foo()) {}`
	as
	`Type name = foo;`
	`if (name) {}`
		Name of the introduced variable

Rewrite Condition Using ||	Rewrites the expression according to De Morgan's laws. For example, rewrites:
	`!a && !b`
	as
	`!(a || b)`
		&&

Rewrite Using *operator*	Rewrites an expression negating it and using the inverse operator. For example, rewrites: › `a op b` as `!(a invop b)` › `(a op b)`	<=, <, >, >=, == or !=

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

Qt DOCUMENTATION

Refactoring Action	Description	Activation

	with
	`if (something)`
	`    x;`
	`else if (something_else)`
	`    x;`	

Swap Operands	Rewrites an expression in the inverse order using the inverse operator. For example, rewrites:
	`a op b`
	as
	`b flipop a`
		<=, <, >, >=, ==, !=, && or ||

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	Converts a string literal to an Objective-C string literal if the file type is Objective-C(++). For example, rewrites the following strings
	`"abcd"`
	`QLatin1String("abcd")`
	`QLatin1Literal("abcd")`
	as
	`@"abcd"`
		String literal

Enclose in QLatin1Char()	Sets the encoding for a character to Latin-1, unless the character is already enclosed in QLatin1Char, QT_TRANSLATE_NOOP, tr, trUtf8, QLatin1Literal, or QLatin1String. For example, rewrites `'a'`	String literal

Refactoring Action	Description	Activation
	as `QLatin1Char('a')`	

Enclose in QLatin1String()	Sets the encoding for a string to Latin-1, unless the string is already enclosed in QLatin1Char, QT_TRANSLATE_NOOP, tr, trUtf8, QLatin1Literal, or QLatin1String. For example, rewrites
	`"abcd"`
	as
	`QLatin1String("abcd")`
		String literal

Mark as Translatable	Marks a string translatable. For example, rewrites `"abcd"` with one of the following options, depending on which of them is available:
	`tr("abcd")`
	`QCoreApplication::translate("CONTEXT", "abcd")`
	`QT_TRANSLATE_NOOP("GLOBAL", "abcd")`
		String literal

Add Definition in ...	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
	`Class Foo {`
	`    void bar();`
	`};`
	as (inside class)
	`Class Foo {`
	`    void bar() {`
	
	`    }`
	`};`
		Function name

```
Class Foo {
    void bar();
};

void Foo::bar()
{

}
```

as (in implementation file)

```
// Header file
Class Foo {
    void bar();
};

// Implementation file
void Foo::bar()
{

}
```

Refactoring Action	Description	Activation
Add `Function` Declaration	Inserts the member function declaration that matches the member function definition into the class declaration. The function can be `public`, `protected`, `private`, `public slot`, `protected slot`, or `private slot`.	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 `newParameter`. The parameter `newParameter` will have the original literal as the default value.	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	Assignee

Refactoring Action	Description	Activation
	as `Type a = foo();` where Type is the return type of `foo()`	
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 `an_example_symbol` as `anExampleSymbol` and `AN_EXAMPLE_SYMBOL` as `AnExampleSymbol`	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`: › `read` function › `write` function, if there is a WRITE › `onChanged` signal, if there is a NOTIFY › data member with the name `m_<propertyName>`	`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 `reset` function.	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 bulb icon appears: 💡
Add #include for undeclared or forward declared identifier	Adds an `#include` directive to the current file to make the definition of a symbol available.	Undeclared identifier
Add Forward Declaration	Adds a forward declaration for an undeclared identifier operation.	Undeclared identifier
Reformat Pointers or References	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.	Declarations with pointers or references and

Refactoring Action	Description	Activation

	`char*s;`
	
	as
	
	`char *s;`
	
	When applied to selections, all suitable declarations in the selection are rewritten.	

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	Moves a function definition to the implementation file, outside the class or back to its declaration. For example, rewrites:
	
	`class Foo`
	`{`
	`  void bar()`
	`  {`
	`      // do stuff here`
	`  }`
	`};`
	
	as
	
	`class Foo`
	`{`
	`  void bar();`
	`};`
	
	`void Foo::bar() {`
	`    // do stuff here`
	`}`
		Function signature

Move All Function	Moves all function definitions to the implementation file or outside the	Class name

Refactoring Action	Description	Activation

	`class Foo`
	`{`
	`  void bar()`
	`  {`
	`      // do stuff here`
	`  }`
	`  void baz()`
	`  {`
	`      // do stuff here`
	`  }`
	`};`
	
	as
	
	`class Foo`
	`{`
	`  void bar();`
	`  void baz();`
	`};`
	
	`void Foo::bar() {`
	`    // do stuff here`
	`}`
	
	`void Foo::baz() {`
	`    // do stuff here`
	`}`	

Assign to Local Variable	Adds a local variable which stores the return value of a function call or a new expression. For example, rewrites:
	
	`QString s;`
	`s.toLatin1();`
	
	as
	
	`QString s;`
	`QByteArray latin1 = s.toLatin1();`
	
	and
	
	`new Foo;`
		Function call or class name

Refactoring Action	Description	Activation
	`Foo * localFoo = new Foo;`	
Insert Virtual Functions of Base Classes	Inserts declarations and the corresponding definitions inside or outside the class or in an implementation file (if it exists). For more information, see Inserting Virtual Functions.	Class or base class name

Optimize for-Loop	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:
	`for (int i = 0; i < 3 * 2; i++)`
	as
	`for (int i = 0, total = 3 * 2; i < total; ++i)`
		`for`

Escape String Literal as UTF-8	Escapes non-ASCII characters in a string literal to hexadecimal escape sequences. String Literals are handled as UTF-8.	String literal
Unescape String Literal as UTF-8	Unescapes octal or hexadecimal escape sequences in a string literal. String Literals are handled as UTF-8.	String literal

Convert to Stack Variable	Converts the selected pointer to a stack variable. For example, rewrites:
	`QByteArray *foo = new QByteArray("foo");`
	`foo->append("bar");`
	as
	`QByteArray foo("foo");`
	`foo.append("bar");`
	This operation is limited to work only within function scope. Also, the coding style for pointers and references is not respected yet.
		Pointer Variable

Convert to Pointer	Converts the selected stack variable to a pointer. For example, rewrites:
	`QByteArray foo = "foo";`
	`foo.append("bar");`
		Stack Variable

Qt DOCUMENTATION

	as
	`QByteArray *foo = new QByteArray("foo");`
	`foo->append("bar");`
	This operation is limited to work only within function scope. Also, the coding style for pointers and references is not respected yet.	

Remove `using namespace` and Adjust Type Names Accordingly	Remove occurrences of `using namespace` in the local scope and adjust type names accordingly.	`using` directive
Remove All Occurrences of `using namespace` in Global Scope and Adjust Type Names Accordingly	Remove all occurrences of `using namespace` in the global scope and adjust type names accordingly.	`using` directive
Convert connect() to Qt 5 Style	Converts a Qt 4 QObject::connect() to Qt 5 style.	QObject::connect() (Qt 4 style)

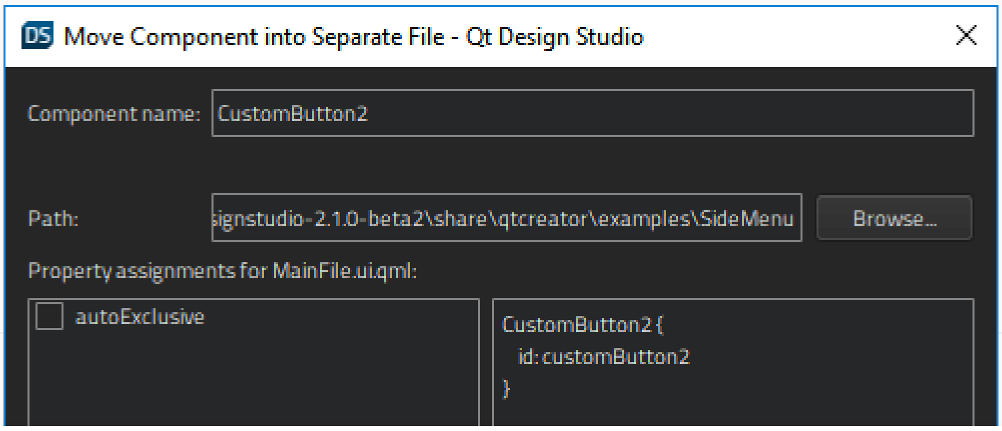
Refactoring QML Code

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	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.
	
	Move Component into Separate File - Qt Design Studio
	Component name: CustomButton2
	Path: signstudio-2.1.0-beta2\share\qtcreator\examples\SideMenu
	Property assignments for MainFile.ui.qml:
	☐ autoExclusive
	`CustomButton2 {`
	`    id: customButton2`
	`}`
		QML type name.

Refactoring Action Activation

Split Initializer	Reformats a one-line type into a multi-line type. For example, rewrites
	`Item { x: 10; y: 20; width: 10 }`
	as
	`Item {`
	`    x: 10;`
	`    y: 20;`
	`    width: 10`
	`}`
		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

< Refactoring Beautifying Source Code >

The Qt Company

Contact Us

Qt DOCUMENTATION

Investors
Newsroom
Careers
Office Locations

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