

User Interface Text Guidelines

Follow the guidelines in this section to make sure that your extensions are consistent with the Qt Creator UI and that they can be easily localized into different languages.

When you write UI text, make sure that it is:

- › Consistent with existing Qt Creator UI terms
- › Short and concise
- › Neutral, descriptive, and factually correct
- › Unambiguous
- › Translatable into different languages

Grammar and Style

All UI text must be grammatically correct English and use the standard form of written language. Do not use dialect or slang words. Use idiomatic language, that is, expressions that are commonly used in English. If possible, ask a native English speaker for a review.

UI text should be concise and economically formulated. Avoid unnecessary content words and phrases. However, it is more important that the text is useful and easy to understand.

Avoid addressing the user in the second person. Use a neutral tone or passive voice but use a formal address when necessary. Avoid using the word *Please* when addressing the user. Avoid using some copyright text and short imperative sentences that might otherwise sound abrupt. For example, *Please wait*.

Avoid abbreviations in the menu names and items. If there is no room for the full spelling or hyphenation of a word, abbreviate the text according to the English abbreviation rules.

Avoid contractions. For example, write *cannot* instead of *can't*.

Punctuation

Avoid using punctuation marks or special characters in menu names and items.

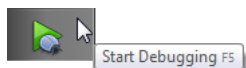
Use punctuation marks as follows:

- › Use full stops in messages.
- › Never use full stops (.) at the end of menu item names.
- › Place three full stops (...) at the end of menu item names that open a dialog requiring user action.
- › Use exclamation marks (!) only in text that demands extra attention from the user or carries special weight.
- › Use quotation marks (") around variable values. For example, **Close Project "qtcreator"**. For consistency, use double quotes to emphasize or set apart file names, directory names, and user visible strings.
- › Do not use leading, trailing, or multiple spaces to align text in messages, as translation tools might not handle them correctly.

Writing Tooltips

Tooltips contain useful information about icons, menu items, or other UI elements. They appear when users place the mouse pointer over an UI element. You can also add descriptive text that is always visible.

For an icon, you can use the command name as a tool tip. In that case, use book style capitalization and do not add a period after the tool tip.



Tooltips can also contain full sentences. Try to make them as short and concise as possible, while still making them grammatically correct. Use sentence style capitalization and punctuation for any sentence.



Writing Tooltips in Design Mode

In Qt Designer, use plain text for tooltips. For extra formatting, write short, canonical HTML in the source tab of the rich text editor: `<html><head/><body>Note: text</body></html>`. Qt Designer has a feature that simplifies the rich text (on by default), but still, you should verify by looking at the **Source** tab.

Writing Messages

Check that messages are concise and economically formulated. However, it is more important that the messages are useful and easy to understand.

- › Cannot send log as selected message type. text is too long.
- › Cannot receive image.
- › Cannot insert picture. Maximum text length is 120 characters.
- › Image name already in use.
- › Folder name already in use.

UI Text Capitalization

Two styles are used, book title and sentence style:

- › Example of Book Title Capitalization
- › Example of sentence style capitalization

Using Book Style Capitalization

When using book style capitalization, capitalize all words, except prepositions that are shorter than five letters (for example, 'with' but 'Without'), conjunctions (for example, and, or, the). However, always capitalize the first and last word.

Use book style capitalization for:

- › Titles (window, dialog, group box, tab, list view columns, and so on)
- › Functions (menu items, buttons)
- › Selectable items (combobox items, listbox items, tree list items, and so on)

Checking Book Style Capitalization

You can use the `to-title-case.js` script in the `\doc\titlecase` folder to check book style capitalization of UI text or headings in documentation:

1. Open `to-title-case.html` in a browser.
2. Enter the UI text in the field.
3. Click **Convert**.

The UI text with suggested book style capitalization is displayed in the field to the right.

Note: The script is based on word lists; it does not perform grammatical analysis. Therefore, it might get the capitalization wrong if you use a rare meaning of a word. For example feathers and not direction when you write *down*. However, you should be able to trust it in most cases in the context of writing UI text and technical documentation.

Using Sentence Style Capitalization

When using sentence style capitalization, capitalize only the first letter, except proper names.

Use sentence style capitalization for:

- › Labels
- › Tool tips
- › Descriptive text
- › Other non-heading or title text

Preparing for Localization

Qt Creator is localized into several languages. Consistency and conciseness make UI text easier to translate.

Marking UI Text for Translation

Make sure the text strings presented to the user are easy to translate. The user interface text strings are enclosed in `tr()` calls and extracted from the source code during the translation. Therefore, the translator might not know the source code context of the messages.

You can add comments that are visible in Qt Linguist (`//:`) to clarify the context. For example:

```
//: Contact book "Add person" button label
return tr("Add");
```

If the class is not `QObject`, use `CoreApplication::translate("class context", "message")` or consider using `Q_DECLARE_TR_FUNCTIONS`. Do not use `QObject` confusing because the messages appear grouped by class context in Qt Linguist and messages tied to `QObject` do not have a class context.

Use `QDir::toNativeSeparators()` for file and directory names that you pass to `tr().arg()`.

Do not use markup that spans the whole string, because that can be confusing for translators. For example, instead of:

```
tr("<html><head></head><body><span>UI Text</span></body></html>")
```

```
QLatin1String("<html><head/><body><span>") + tr("UI Text") + QLatin1String("/span></body></html>")
```

Features of Languages or Writing Systems

To allow for localization of your extensions, consider the impact that languages and writing systems have on the implementation.

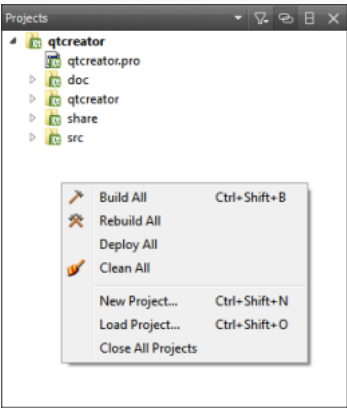
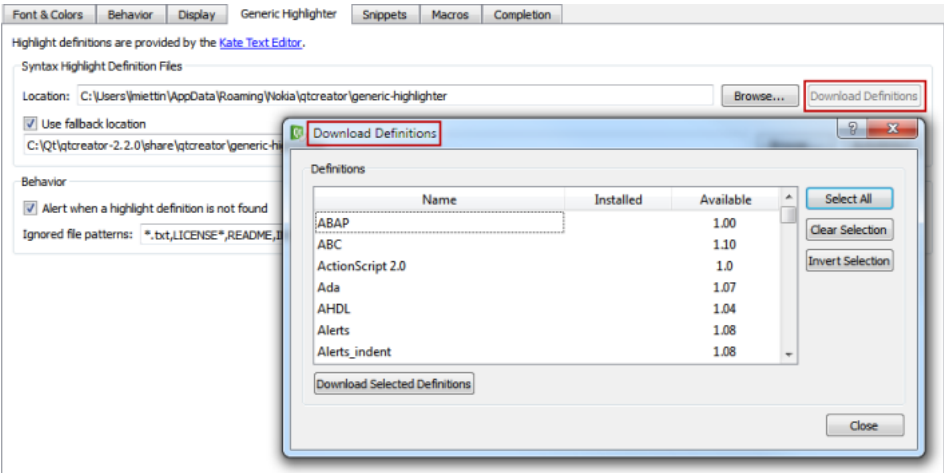

Features of Languages or Writing Systems	Impact on Implementation
Word order	Different languages have different word order rules. Do not use run-time concatenation. Use complete phrases and "%1" formatting instead. For example, use: tr("Foo failed: %1").arg(message) instead of tr("Foo failed: ") + message
Singular vs. plural vs. dual forms	Some languages do not have plural form (for example, Chinese and Japanese), whereas some have a different form for dual. Allow room for text expansion in the layout design. Some languages need more space to indicate plurality or duality to convey the need. For example, use tr("%n files found", 0, number) instead of tr("%1 files found").arg(number)
Gender	Some languages have gender (feminine, masculine, neutral), whereas some do not (for example, Finnish) or do not use it extensively (if). Do not reuse text strings. The same term may not work in another context due to the gender of the base word. Articles have a grammatical gender in some languages and sentences cannot be as easily constructed as in English. Avoid following by tr("%1 failed").arg(someCondition ? "the operation" : "opening a file")

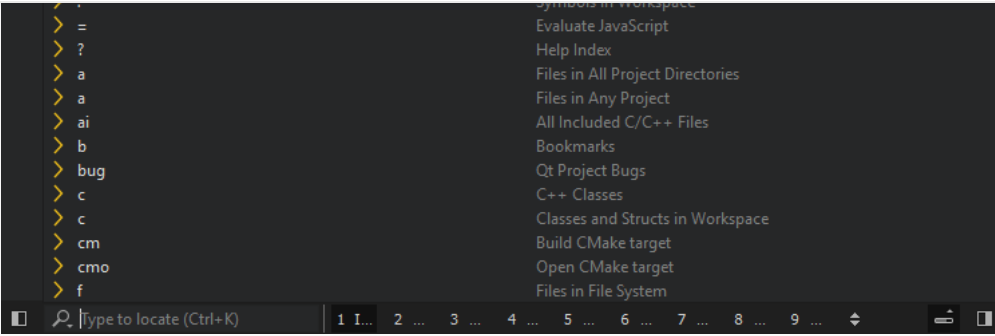
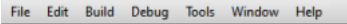
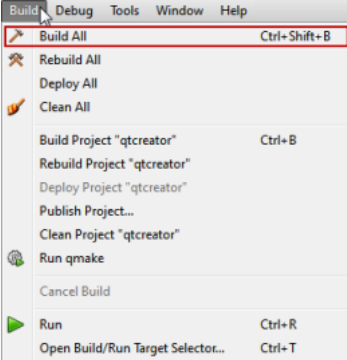
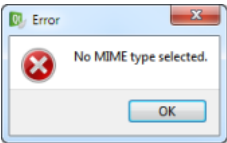

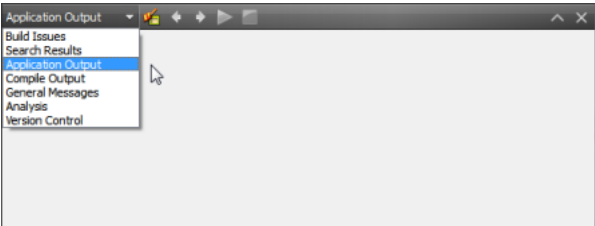
Common Qt Creator Terms

This section summarizes the terminology used for common Qt Creator UI components. It also describes the conventions for naming different types of UI components.

Always check that the term you plan to use is not used to mean something else in the UI. If a suitable term already exists, use it. For example, use *Find* for searching and *New* for wizard objects.

For more information on how to add UI components, see [Common Extension Tasks](#).

UI Text	Usage
Context menu	Opens when users right-click on the screen. Contents depend on the context. 
Dialog	User interface element that usually contains a number of choices or allows the user to give input to the application. Opens when users select a menu item or button. 
Locator	Allows you to browse not only files, but any items defined by locator filters.
UI Text	

	
Menu	<p>Contains menu items that represent commands or options and that are logically grouped and displayed. A menu can also contain submenus.</p> 
Menu item	<p>Represents a command or an option for users to choose.</p> 
Message box	<p>Dialog that provides feedback to users, in the form of status information, a warning, or an error message.</p>  <p>Output from Qt Creator should be displayed in output panes, instead.</p>
Mode	<p>Modes correspond to complete screens of controls, specialized for a task.</p> 
Output pane	<p>A pane displayed in the task pane that displays output from Qt Creator.</p> 
Side pane	<p>A view available in the Edit and Debug modes that you can use to browse projects, files, and bookmarks, and to view the class hierarchy.</p>

doc

qtcreator

share

File System

Projects

Open Documents

Bookmarks

File System

Class View

Outline

Type Hierarchy

dist

doc

lib

scripts

View

An area of the screen that displays information for users and provides them with functions for managing the information. Available in **Debug** mode, for interaction with t program that is running under the control of the debugger.

Debugger CDB for "analogdock" Threads: #2 Stopped.

Level	Function	File	Line	Number	Function
1	NtUserMsgWaitForMultipleObjectsEx	win32u			
2	MsgWaitForMultipleObjectsEx	USER32			
3	QByteArray::operator!=	Qt5Cored			
4	qt_plugin_query_metadata	qwindowsd			
5	QByteArray::operator!=	Qt5Cored			
6	QByteArray::operator!=	Qt5Cored			
7	QByteArray::operator!=	Qt5Cored			
8	QOpenGLFunctions_3_3_Compatibility::glMultiTexCoord1dv	Qt5Guid			
9	main	main.cpp	169		
10	invoke_main	ese-common.inl	79		
11	__scrt_common_main_seh				
12	__scrt_common_main				
13	mainCRTStartup				
14	BaseThreadInitThunk				
15	RtlUserThreadStart				

Address: 0x7ff7:447b:91cc
Function: main
File: C:
\\Qt5\\Examples\\Qt-5.12.1\\gui\\analogclock\\main.cpp
Line: 169
Module: analogclock

/Note: Sources for this frame are available.
Double-click on the file name to open an editor.

< External Tool Specification Files

Qt

© 2021 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 Docum 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



Contact Us

Company

- About Us
- Investors
- Newsroom
- Careers
- Office Locations

Support

- Support Services
- Professional Services
- Partners
- Training

Community

- Contribute to Qt
- Forum
- Wiki
- Downloads

Licensing

- Terms & Conditions
- Open Source
- FAQ

For Customers

- Support Center
- Downloads
- Qt Login
- Contact Us
- Customer Success

