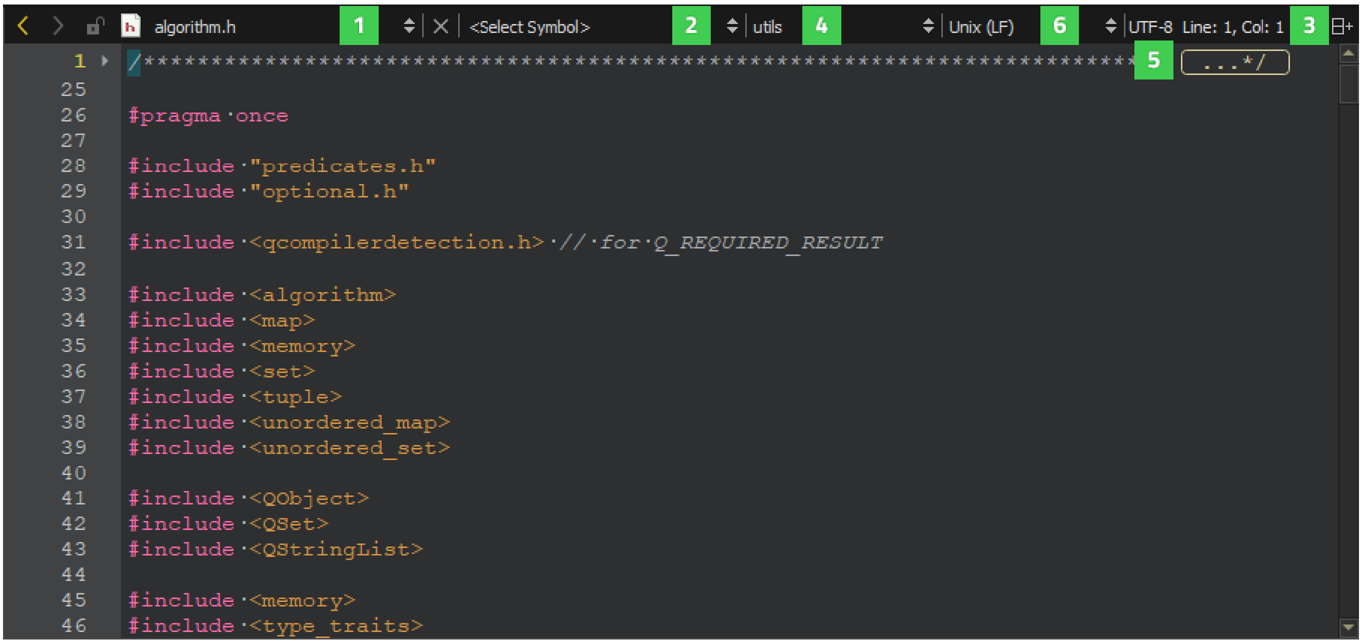
Working in Edit Mode

This section describes how to use the **Edit** mode. For more information about using the sidebar, see Browsing Project Contents.

Using the Editor Toolbar

The editor toolbar is located at the top of the editor view. The editor toolbar is context sensitive and shows items relevant to the file currently open in the editor.



```
/*************************************************************************
 
#pragma once

#include "predicates.h"
#include "optional.h"

#include <qcompilerdetection.h> // for Q_REQUIRED_RESULT

#include <algorithm>
#include <map>
#include <memory>
#include <set>
#include <tuple>
#include <unordered_map>
#include <unordered_set>

#include <QObject>
#include <QSet>
#include <QStringList>

#include <memory>
#include <type_traits>
```

Navigating Between Open Files and Symbols

Use the toolbar, **Window** menu items, or keyboard shortcuts to navigate between open files and symbols in use.

To browse backward or forward through your location history, click < (**Go Back**) and > (**Go Forward**).

To return to the last location where you made a change, select **Window** > **Go to Last Edit**.

To go to any open file, select it from the **Open files** drop-down menu (1). To open a context menu that contains commands for managing open files, right-click the file name or icon on the toolbar. In addition to the commands also available in the **File** menu, you can copy the path and name of the current file and the number of the line where the cursor is currently located to the clipboard by selecting **Copy Full Path**, **Copy File Name**, or **Copy Path and Line Number**.

Alphabetically to arrange the symbols in alphabetic order.

To jump to a line and column in the current file, select the line and column indicator (3) or press **Ctrl+K** (or **Cmd+K** on macOS) to open the locator. Enter the line number and column number in the locator, separated by a colon (:).

> **Note:** Other convenient ways of navigating in Qt Creator are provided by the sidebars.

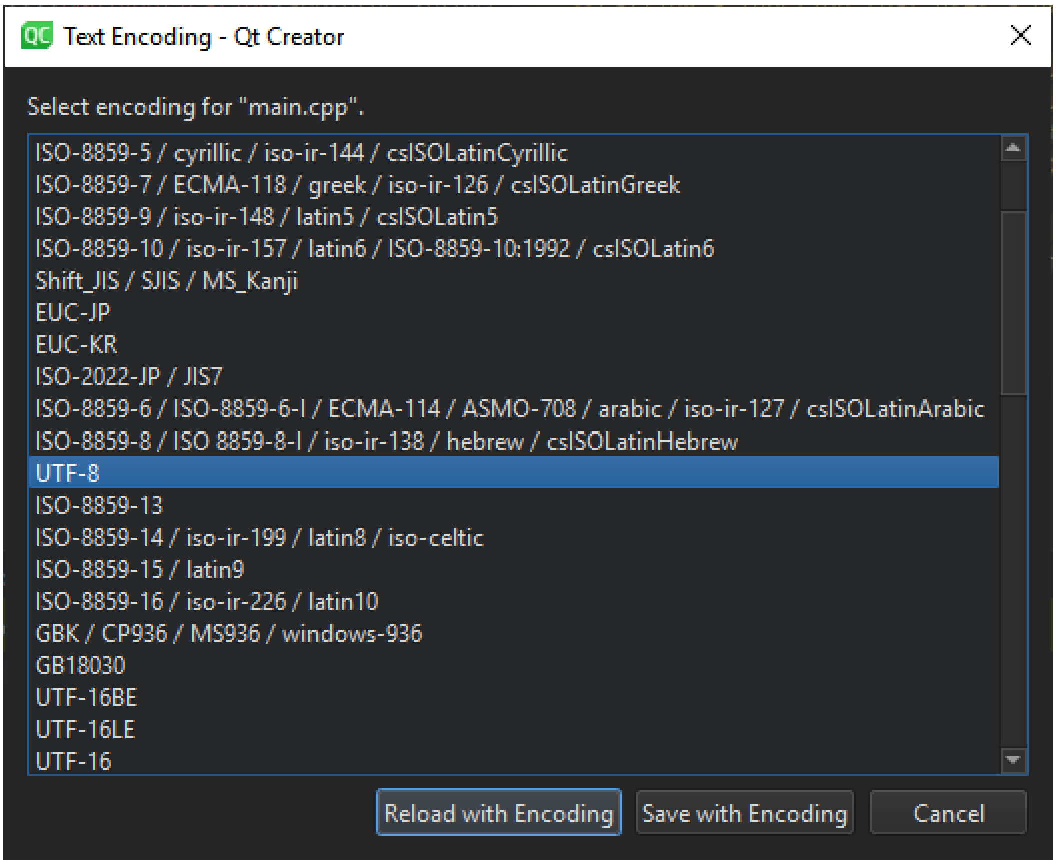
Selecting Parse Context

Code might be interpreted differently in different contexts. A file can be used by different projects or subprojects with different defines, or it can be included in the context of C, C++, Objective-C, or Objective-C++. To change the active parse context, select an available parse context in the **Active Parse Context** menu (4). The menu is visible only when several parse contexts are available. To reset the parse context, right-click on the menu to open a context menu, and then select **Clear Preferred Parse Context**. If the information needed for parsing the project is still incomplete or incorrect, select **Additional Preprocessor Directives** to add preprocessor directives.

Changing Text Encoding

To show the file encoding of the current file on the editor toolbar (5), select **Edit** > **Preferences** > **Text Editor** > **Display** > **Display file encoding**.

To change the text encoding, click it on the toolbar and select new encoding in the **Text Encoding** dialog:



Text Encoding - Qt Creator

Select encoding for "main.cpp".

ISO-8859-5 / cyrillic / iso-ir-144 / csISOLatinCyrillic
ISO-8859-7 / ECMA-118 / greek / iso-ir-126 / csISOLatinGreek
ISO-8859-9 / iso-ir-148 / latin5 / csISOLatin5
ISO-8859-10 / iso-ir-157 / latin6 / ISO-8859-10:1992 / csISOLatin6
Shift_JIS / SJIS / MS_Kanji
EUC-JP
EUC-KR
ISO-2022-JP / JIS7
ISO-8859-6 / ISO-8859-6-I / ECMA-114 / ASMO-708 / arabic / iso-ir-127 / csISOLatinArabic
ISO-8859-8 / ISO 8859-8-I / iso-ir-138 / hebrew / csISOLatinHebrew
UTF-8
ISO-8859-13
ISO-8859-14 / iso-ir-199 / latin8 / iso-celtic
ISO-8859-15 / latin9
ISO-8859-16 / iso-ir-226 / latin10
GBK / CP936 / MS936 / windows-936
GB18030
UTF-16BE
UTF-16LE
UTF-16

To reload the file with the selected encoding, select **Reload with Encoding**. To save the file with the new encoding, select **Save with Encoding**.

Selecting Line Ending Style

To switch between Windows line endings (CRLF) and Unix line endings (LF), select the ending style on the editor select the ending style in the **Default line endings** field.

To set the line endings to use for a project, select **Projects** > **Project Settings** > **Editor**. For more information, see Specifying Editor Settings.

Editing Selected Lines

The **Edit** > **Advanced** menu contains options for editing selected lines of text.

To duplicate the selected lines, select **Duplicate Selection**. To format the duplicated lines as a comment, select **Duplicate Selection and Comment**.

To turn selected text into lowercase, select **Lowercase Selection** or press **Alt+U**. To turn it into uppercase, select **Uppercase Selection** or press **Alt+Shift+U**.

To sort selected lines alphabetically, select **Sort Selected Lines** or press **Alt+Shift+S**.

Select **Add Next Occurrence to Selection** or press **Ctrl+D** to add a cursor at the next occurrence of selected text for multi-cursor editing.

Multi-Cursor Editing

To apply a change to several places simultaneously, press and hold **Alt**, and click to place cursors in several places. Any changes you make are applied simultaneously at all the cursor positions.

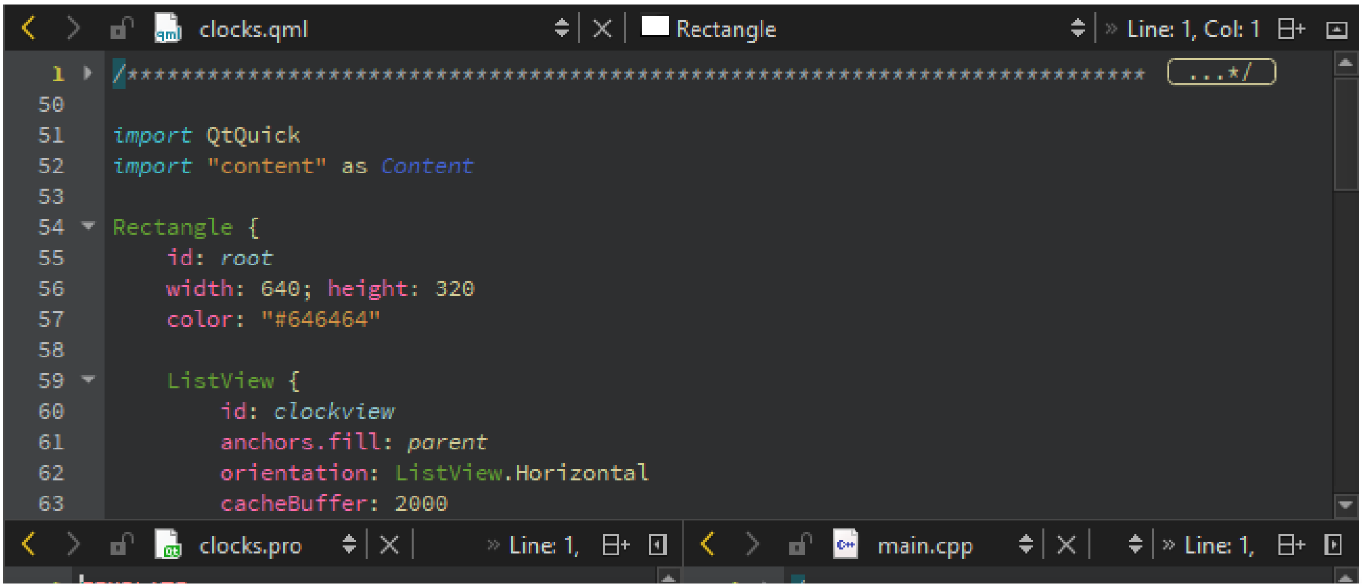
Use the arrow keys to move all the cursors up and down. The **Home** and **End** key move all the cursors to the beginning or to the end of the line.

Press and hold **Alt** and double-click strings to select several strings simultaneously.

Press **Esc** to remove all the cursors and selections.

Splitting the Editor View

Split the editor view or open the editor in a new window when you want to work on and view multiple files on the same screen or on multiple screens.

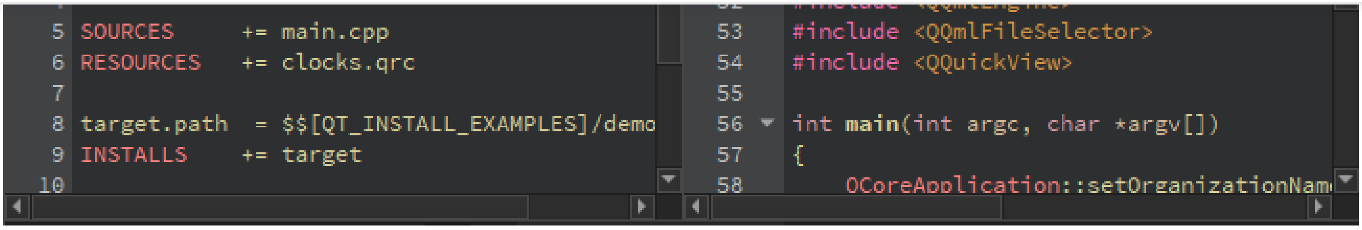


```
/*********************************************************************** ...*/

import QtQuick
import "content" as Content

Rectangle {
    id: root
    width: 640; height: 320
    color: "#646464"

    ListView {
        id: clockview
        anchors.fill: parent
        orientation: ListView.Horizontal
        cacheBuffer: 2000
```



```
SOURCES      += main.cpp
RESOURCES    += clocks.qrc

target.path  = $$[QT_INSTALL_EXAMPLES]/demo
INSTALLS     += target
```

```
#include <QQmlFileSelector>
#include <QQuickView>

int main(int argc, char *argv[])
{
    QCoreApplication::setOrganizationNam
```

You can view multiple files simultaneously in the following ways:

- To split the editor view into a top and bottom view, select **Window** > **Split**, press **Ctrl+E, 2**, or select the (**Split**) button and then select **Split**.

  Split command creates views below the currently active editor view.

- To split the editor view into adjacent views, select **Window** > **Split Side by Side**, press **Ctrl+E, 3**, or select **Split** > **Split Side by Side**.

  Side by side split command creates views to the right of the currently active editor view.

- To open the editor in a detached window, press **Ctrl+E, 4**, or select **Window** > **Open in New Window**.

  The new window behaves basically in the same way as the editor area in the main window. For example, you can split this window, as well. Documents are opened in the currently active editor window.

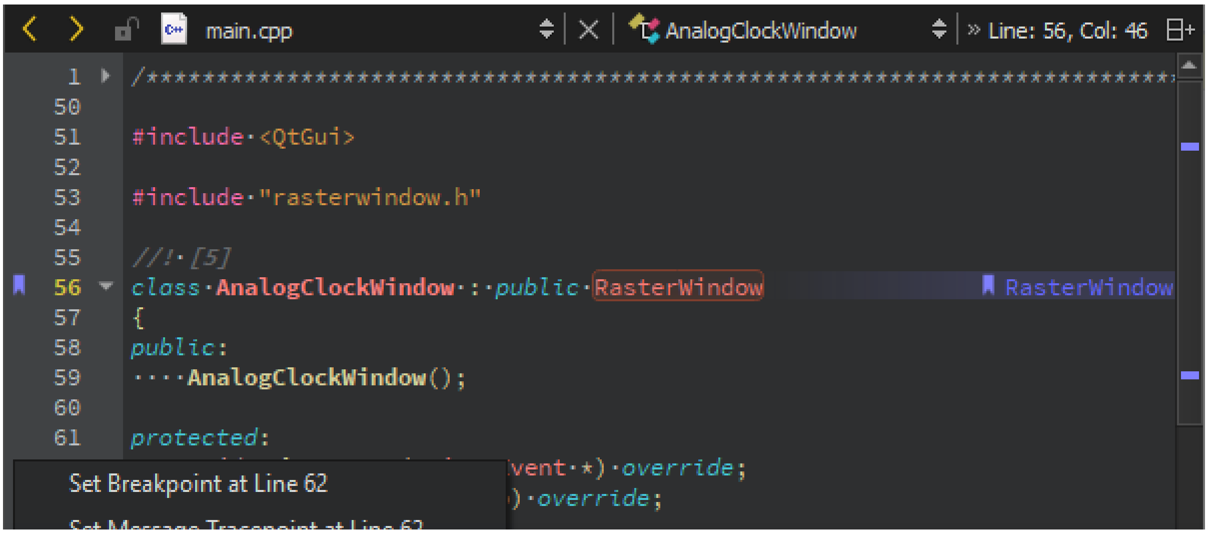
To move between split views and detached editor windows, select **Window** > **Go to Next Split or Window** or press **Ctrl+E, O**.

To remove a split view, place the cursor within the view you want to remove and select **Window** > **Remove Current Split**, press **Ctrl+E, 0**, or select the (**Remove Split**) button. To remove all but the currently selected split view, select **Window** > **Remove All Splits** or press **Ctrl+E, 1**.

Using Bookmarks

To insert or delete a bookmark in the **Edit** mode:

- Right-click the line number and select **Toggle Bookmark**.
- Press **Shift** and click the left margin at a line.
- Press **Ctrl+M** when the cursor is on a line.



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

#include <QtGui>

#include "rasterwindow.h"

//! [5]
class AnalogClockWindow : public RasterWindow
{
public:
    AnalogClockWindow();

protected:
                                        vent *) override;
                              ) override;
```

Adding Notes to Bookmarks

To add a note to a bookmark:

- Select **Tools** > **Bookmarks** > **Edit Bookmark**.
- Press **Ctrl+Shift+M**.
- Right-click a bookmark and select **Edit Bookmark** in the context menu.

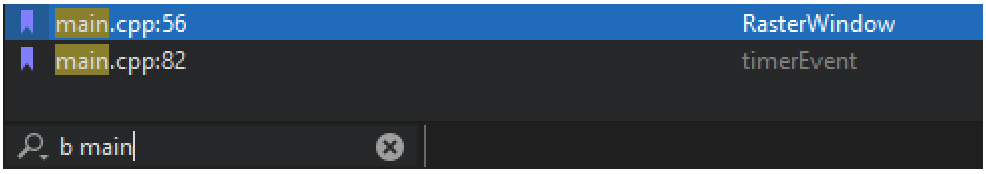
To view the note, move the mouse pointer over the bookmark or open the **Bookmarks** view in the sidebar.

Navigating Bookmarks

To go to the previous bookmark in the current session, select **Tools** > **Bookmarks** > **Previous Bookmark** or press **Ctrl+,**.

To go to the next bookmark in the current session, select **Tools** > **Bookmarks** > **Previous Bookmark** or press **Ctrl+.**.

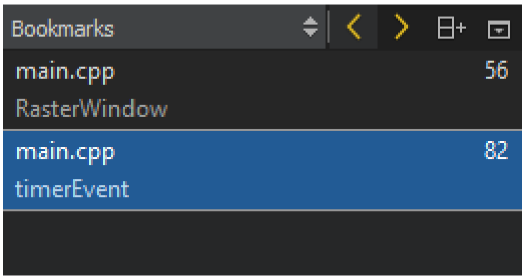
To use the locator to go to a bookmark, press **Ctrl+K** (or **Cmd+K** on macOS) to open the locator. Enter *b* and a space to display a list of bookmarks. To filter the bookmarks by line number or a text string, enter the number or string after the space. Double-click a bookmark in the list to go to it in the editor.



main.cpp:56 RasterWindow
main.cpp:82 timerEvent
b main

Viewing Bookmarks

Bookmarks are listed in the **Bookmarks** view in the sidebar. To move between bookmarks, select the **Previous Bookmark** or **Next Bookmark** button or use the keyboard shortcuts.



main.cpp 56
RasterWindow
main.cpp 82
timerEvent

Moving to Symbol Definition or Declaration

You can move directly to the definition or the declaration of a symbol in the **Edit** mode by holding the **Ctrl** key and clicking the symbol. If you have multiple splits opened, you can open the link in the next split by holding **Ctrl** and **Alt** while clicking the symbol.

To enable this moving function, select **Edit** > **Preferences** > **Text Editor** > **Behavior** > **Enable mouse navigation**.

There are several additional ways of moving between symbol definitions and declarations. All the functions described below are also available from the **Tools** > **C++** menu. The functions supported for QML and JavaScript code are available from the **Tools** > **QML/JS** menu.

Split. Following symbols is supported for namespaces, classes, functions, variables, include statements, and macros.

To switch between the definition and declaration of a function, place the cursor on either and press **Shift+F2** or right-click and select **Switch Between Function Declaration/Definition** or **Open Function Declaration/Definition in Next Split**. For example, this allows you to navigate from anywhere within a function body directly to the function declaration.

Links are opened in the same split by default. To open links in the next split, prepend **Ctrl+E** to the shortcut. For example, press **Ctrl+E,F2** to follow the symbol in the next split. If necessary, the view is automatically split. To change the default behavior, select **Edit** > **Preferences** > **Text Editor** > **Display** > **Always open links in another split**. Additional symbols are displayed and switching between definition and declaration is done in another split. If you change the default behavior, the shortcuts for opening link targets in the next split are used to open them in the current split.

To switch between C++ header and source files, right-click anywhere in a file and select **Switch Header/Source** or **Open Corresponding Header/Source in Next Split**. You can also press **F4** or **Ctrl+E,F4**, respectively.

Reparsing Externally Changed Files

If source files are modified from outside Qt Creator, the opened files will be reparsed automatically. For all other files, you can use **Tools** > **C++** > **Reparse Externally Changed Files** to update the code model.

Inspecting the Code Model

When you report a bug that is related to the C++ code model, the Qt Creator developers might ask you to write information about the internal state of the code model into a log file and to deliver the file to them for inspection.

To view information about the C++ code model in the **C++ Code Model Inspector** dialog and write it to a log file, select **Tools** > **Debug Qt Creator** > **Inspect C++ Code Model** or press **Ctrl+Shift+F12**.

Qt Creator generates the code model inspection log file in a temporary folder.

Qt Creator underlines semantic errors in olive in the C++ code editor. To check the correct paths for includes that are not resolved or that are resolved to the wrong file, select **Project Parts** > **Header Paths**.

‹ Writing Code Semantic Highlighting ›