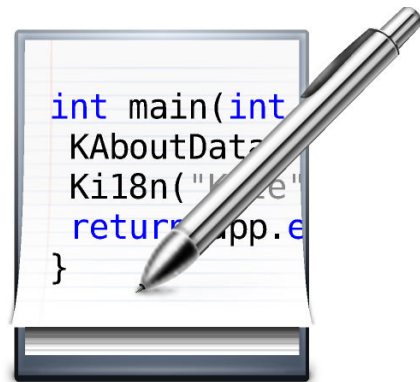


# The Kate Handbook

Anders Lund  
Seth Rothberg  
Dominik Haumann  
T.C. Hollingsworth



# The Kate Handbook

# Contents

<b>1</b>	<b>Introduction</b>	<b>10</b>
<b>2</b>	<b>The Fundamentals</b>	<b>11</b>
2.1	Starting Kate . . . . .	11
2.1.1	From the Menu . . . . .	11
2.1.2	From the Command Line . . . . .	11
2.1.2.1	Command Line Options . . . . .	12
2.1.3	Drag and Drop . . . . .	13
2.2	Working with Kate . . . . .	13
2.2.1	Quick Start . . . . .	13
2.2.2	Shortcuts . . . . .	13
2.3	Working With the KateMDI . . . . .	14
2.3.1	Overview . . . . .	14
2.3.1.1	The Main Window . . . . .	14
2.3.2	The Editor area . . . . .	14
2.4	Using Sessions . . . . .	15
2.5	Quick Open . . . . .	15
2.5.1	Using Quick Open . . . . .	15
2.5.2	Configuring Quick Open . . . . .	16
2.6	Getting Help . . . . .	16
2.6.1	With Kate . . . . .	16
2.6.2	With Your Text Files . . . . .	17
2.6.3	Articles on Kate . . . . .	17
<b>3</b>	<b>Working with the Kate Editor</b>	<b>18</b>
<b>4</b>	<b>Working with Plugins</b>	<b>19</b>
4.1	Kate Application Plugins . . . . .	19
4.2	External Tools . . . . .	20
4.2.1	Configuring External Tools . . . . .	21
4.2.2	Variable Expansion . . . . .	22
4.2.3	List of Default Tools . . . . .	24
4.3	Backtrace Browser Plugin . . . . .	26

## The Kate Handbook

4.3.1	Using the Backtrace Browser Plugin . . . . .	26
4.3.2	Configuration . . . . .	27
4.4	Build Plugin . . . . .	27
4.4.1	Introduction . . . . .	27
4.4.2	Using the Build Plugin . . . . .	27
4.4.2.1	Target Settings tab . . . . .	28
4.4.2.2	Output tab . . . . .	29
4.4.3	Menu Structure . . . . .	29
4.4.4	Thanks and Acknowledgments . . . . .	30
4.5	Close Except/Like Plugin . . . . .	30
4.5.1	Introduction . . . . .	30
4.5.2	Using the Close Except/Like Plugin . . . . .	30
4.5.3	Menu Structure . . . . .	30
4.6	Color Picker Plugin . . . . .	31
4.6.1	Introduction . . . . .	31
4.6.2	Configuration . . . . .	31
4.7	Colored Brackets . . . . .	31
4.7.1	Introduction . . . . .	31
4.7.2	Configuration . . . . .	31
4.8	CTags Plugin . . . . .	31
4.8.1	Introduction . . . . .	31
4.8.2	Configuration . . . . .	32
4.8.2.1	Common Index . . . . .	32
4.8.2.2	Session Index . . . . .	32
4.8.3	Using the CTags Plugin . . . . .	33
4.8.4	Menu Structure . . . . .	33
4.9	Document Preview Plugin . . . . .	34
4.9.1	Introduction . . . . .	34
4.9.2	Menu Structure . . . . .	34
4.9.3	Interface . . . . .	34
4.10	Document Switcher Plugin . . . . .	34
4.10.1	Menu Structure . . . . .	34
4.11	File System Browser . . . . .	35
4.11.1	Menu Structure . . . . .	35
4.11.2	Interface . . . . .	35
4.11.3	Configuration . . . . .	36
4.12	The Documents List . . . . .	37
4.12.1	Introduction . . . . .	37
4.12.2	Menu Structure . . . . .	38
4.12.3	Configuration . . . . .	38
4.13	GDB Plugin . . . . .	38
4.13.1	Introduction . . . . .	38

## The Kate Handbook

4.13.2	Menu and Toolbar Structure . . . . .	39
4.13.3	Debug View . . . . .	40
4.13.4	Call Stack and Locals . . . . .	41
4.13.5	Thanks and Acknowledgments . . . . .	42
4.13.6	Configuration . . . . .	42
4.13.6.1	Execution environment setup . . . . .	44
4.14	Project Plugin . . . . .	45
4.14.1	Introduction . . . . .	45
4.14.2	Structured View of the Files . . . . .	45
4.14.3	Switching Projects . . . . .	46
4.14.4	Search and Replace in Projects . . . . .	46
4.14.5	Simple Auto Completion . . . . .	46
4.14.6	Support for Building the Project . . . . .	47
4.14.7	Creating Projects . . . . .	47
4.14.7.1	Loading Projects Automatically . . . . .	47
4.14.7.2	Creating Projects Manually . . . . .	48
4.14.8	Current Project . . . . .	49
4.14.9	The Projects Menu . . . . .	50
4.15	LSP Client Plugin . . . . .	50
4.15.1	Menu Structure . . . . .	50
4.15.2	Goto Symbol support . . . . .	52
4.15.2.1	Configuring LSP Client Symbol Outline . . . . .	52
4.15.2.2	Global Goto symbol support . . . . .	52
4.15.3	Other Features . . . . .	52
4.15.4	Configuration . . . . .	53
4.15.4.1	LSP Server Configuration . . . . .	55
4.15.4.2	LSP Server Format On Save . . . . .	56
4.15.4.3	LSP Server Diagnostic Suppression . . . . .	56
4.15.4.4	LSP Server Troubleshooting . . . . .	57
4.15.4.5	Execution environment setup . . . . .	57
4.16	Search & Replace . . . . .	59
4.16.1	Introduction . . . . .	59
4.16.2	Interface . . . . .	60
4.16.2.1	Search Query . . . . .	60
4.16.2.2	Search in Folder Options . . . . .	60
4.16.2.3	Search Results . . . . .	62
4.16.3	Menu Structure . . . . .	62
4.17	Kate Scripts & Snippets . . . . .	62
4.17.1	Introduction . . . . .	62
4.17.2	Menu Structure . . . . .	63
4.17.3	Snippets panel . . . . .	63

## The Kate Handbook

4.17.3.1	Loading Snippet Repository Files . . . . .	63
4.17.3.2	Creating and Editing Repositories . . . . .	63
4.17.3.3	Creating and Editing Snippets . . . . .	65
4.17.4	Using Snippets . . . . .	68
4.17.5	Thanks and Acknowledgments . . . . .	68
4.18	Keyboard Macros Plugin . . . . .	68
4.18.1	Introduction . . . . .	68
4.18.2	Basic usage . . . . .	68
4.18.2.1	To start recording a keyboard macro: . . . . .	68
4.18.2.2	To end recording: . . . . .	69
4.18.2.3	To cancel recording: . . . . .	69
4.18.2.4	To play the current macro: . . . . .	69
4.18.3	Named macros . . . . .	69
4.18.3.1	To save the current macro: . . . . .	69
4.18.3.2	To load a saved macro as the current one: . . . . .	69
4.18.3.3	To play a saved macro without loading it: . . . . .	69
4.18.3.4	To wipe (i.e., delete) a saved macro: . . . . .	70
4.18.3.5	Tips for commands: . . . . .	70
4.18.4	Limitations . . . . .	70
4.19	SQL Plugin . . . . .	70
4.19.1	Introduction . . . . .	70
4.19.2	Connecting to a Database . . . . .	70
4.19.3	Running Queries . . . . .	71
4.19.3.1	INSERT/DELETE/UPDATE . . . . .	71
4.19.3.2	SELECT . . . . .	71
4.19.4	Browsing . . . . .	72
4.19.5	Menu Structure . . . . .	72
4.19.6	Thanks and Acknowledgments . . . . .	72
4.20	Symbol Viewer Plugin . . . . .	72
4.20.1	Using the Close Except/Like Plugin . . . . .	72
4.20.2	Menu Structure . . . . .	73
4.20.3	Configuration . . . . .	73
4.21	Terminal Tool View Plugin . . . . .	73
4.21.1	Menu Structure . . . . .	73
4.21.2	Configuration . . . . .	74
4.22	Text Filter Plugin . . . . .	74
4.22.1	Using the Text Filter Plugin . . . . .	74
4.22.2	Menu Structure . . . . .	75
4.23	XML Validation . . . . .	75
4.23.1	Menu Structure . . . . .	76
4.23.2	Thanks and Acknowledgments . . . . .	76

## The Kate Handbook

4.24 XML Completion . . . . .	76
4.24.1 How to Use . . . . .	76
4.24.2 Features and Limitations . . . . .	77
4.24.3 Menu Structure . . . . .	77
4.24.4 Thanks and Acknowledgments . . . . .	77
4.25 Compiler Explorer Plugin . . . . .	77
4.25.1 Usage . . . . .	78
4.26 Formatting Plugin . . . . .	78
4.26.1 Usage . . . . .	78
4.26.2 Supported languages and formatters . . . . .	78
4.26.3 Configuring . . . . .	79
4.26.4 Temporarily disable format on save . . . . .	80
<b>5 Advanced Editing Tools</b>	<b>81</b>
<b>6 Extending Kate</b>	<b>82</b>
6.1 Introduction . . . . .	82
6.2 Working with Syntax Highlighting . . . . .	82
6.3 Scripting with JavaScript . . . . .	82
6.4 Kate (C++) Application Plugins . . . . .	82
<b>7 The VI Input Mode</b>	<b>83</b>
<b>8 The Menu Entries</b>	<b>84</b>
8.1 The File Menu . . . . .	84
8.2 The Edit Menu . . . . .	85
8.3 The View Menu . . . . .	88
8.4 The Bookmarks Menu . . . . .	91
8.5 The Sessions Menu . . . . .	92
8.6 The Tools Menu . . . . .	92
8.7 The Settings and Help Menu . . . . .	93
<b>9 Configuring Kate</b>	<b>95</b>
9.1 Overview . . . . .	95
9.2 The Main Configuration Dialog . . . . .	96
9.3 The Kate Application Configuration . . . . .	96
9.3.1 General . . . . .	96
9.3.2 Session . . . . .	97
9.3.3 Plugins . . . . .	98
9.3.4 The Editor Component Configuration . . . . .	98
9.3.5 Configuring With Document Variables . . . . .	99
<b>10 Credits and License</b>	<b>100</b>
<b>A Regular Expressions</b>	<b>102</b>
<b>B Index</b>	<b>103</b>

# List of Tables

4.1	Some available KParts plugins . . . . .	34
-----	---	----

## **Abstract**

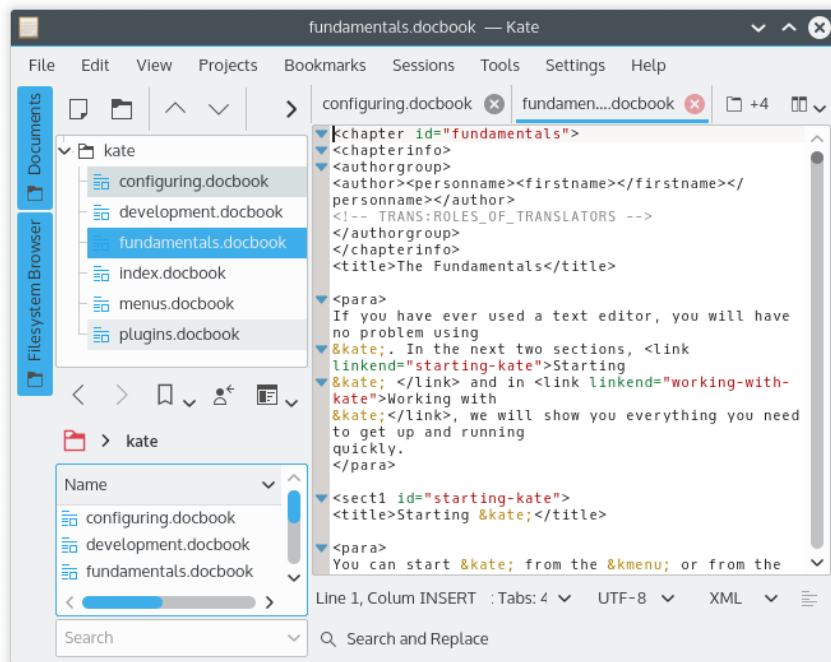
Kate is a programmer's text editor by KDE.  
This handbook documents Kate Version 22.08

# Chapter 1

## Introduction

Welcome to Kate, a programmer's text editor by KDE. Some of Kate's many features include configurable syntax highlighting for languages ranging from C and C++ to HTML to bash scripts, the ability to create and maintain projects, a multiple document interface (MDI), and a self-contained terminal emulator.

But Kate is more than a programmer's editor. Its ability to open several files at once makes it ideal for editing UNIX<sup>®</sup>'s many configuration files. This document was written in Kate.



*Editing this manual...*

## Chapter 2

# The Fundamentals

If you have ever used a text editor, you will have no problem using Kate. In the next two sections, [Starting Kate](#) and in [Working with Kate](#), we will show you everything you need to get up and running quickly.

### 2.1 Starting Kate

You can start Kate from the application launcher or from the command line.

#### 2.1.1 From the Menu

Open the KDE program menu by clicking on the application launcher icon on the toolbar at the bottom left of your screen. This will raise a menu. Move your cursor up the menu to the **Applications** → **Utilities** → **Advanced Text Editor Kate** menu item.

#### 2.1.2 From the Command Line

You can start Kate by typing its name on the command line. If you give it a file name, as in the example below, it will open or create that file.

```
%kate myfile.txt
```

If you have an active connection, and permission, you can take advantage of KDE's network transparency to open files on the internet.

```
%kate ftp://ftp.kde.org/pub/kde/README
```

To change the directory for temporary files, which defaults to `/tmp` set the `TMPDIR` environment variable before starting Kate, e.g.

```
%mkdir /tmp/kate -p && export TMPDIR=/tmp/kate && kate
```

### 2.1.2.1 Command Line Options

Kate accepts following command line options:

**kate --help**

This lists the options available at the command line.

**kate -s --start name**

Starts Kate with the session *name*. The session is created if it does not exist already. If a Kate instance running the specified session exists, the specified files are loaded in that instance.

**kate -p --pid PID**

Only reuses an instance with the specified PID (Process ID).

**kate -e --encoding encoding URL**

Uses the specified encoding for the document.

**kate -l --line line URL**

Navigates to the specified line after opening the document.

**kate -c --column column URL**

Navigates to the specified column after opening the document.

**kate -i --stdin**

Reads the document content from STDIN. This is similar to the common option `-` used in many command line programs, and allows you to pipe command output into Kate.

**kate --startanon**

Start Kate with a new anonymous session, implies `-n`.

**kate -n --new**

Force start of a new Kate instance (is ignored if `start` is used and another Kate instance already has the given session opened), forced if no parameters and no URLs are given at all.

**kate -b --block**

If using an already running Kate instance, block until it exits, if URLs given to open.

You can use Kate with this option as editor for typing in commit messages for version control systems like Git or Subversion. These systems expect to block the editor till you have entered your message, because they then open the temporary file, which would be empty if Kate immediately returned to the caller.

This option is also needed with KIO (KDE Input/Output), if you open a remote file (which has been downloaded to a temporary) and should be reuploaded, after you saved it.

**kate --tempfile**

When used, the specified files are treated as temporary files and deleted (if they are local files and you have sufficient permissions) when closed, unless they were modified since they were opened.

**kate --desktopfile filename**

The base file name of the desktop entry for this application.

This is in particular useful for wrapper applications and applications having in general multiple desktop files. Thus each desktop file can have its own command line for the `Exec` entry.

**kate --author**

Lists Kate's authors in the terminal window.

**kate -v --version**

Lists version information for Kate.

**kate --license**

Shows license information.

### 2.1.3 Drag and Drop

Kate uses the KDE Drag and Drop protocol. Files may be dragged and dropped onto Kate from the Desktop, the filemanager Dolphin or some remote FTP site opened in one of Dolphin's windows.

## 2.2 Working with Kate

[Quick Start](#) will show you how to toggle four simple options that will let you configure some of Kate's more powerful features right away. [Shortcuts](#) lays out some of the default keystroke shortcuts for those who can't or don't want to use a mouse.

### 2.2.1 Quick Start

This section will describe some of the items on the **View** menu so that you can quickly configure Kate to work the way you want it.

When you start Kate for the first time you will see two windows with white backgrounds. Above the two windows is a toolbar with the usual labeled icons. And above that, a menubar.

The left-hand window is a side bar. It combines the **Documents** and **Filesystem Browser** windows. Switch between the two by clicking on the tabs to the left of the window.

If you've started Kate with a file, the right-hand window will show the file you are editing and the **Documents** on the side bar will show the name of the file. Use the **Filesystem Browser** window to open files.

You can toggle all sidebar windows on and off in **View** → **Tool Views** menu or use **Ctrl+Alt+Shift+F**. This menu offers you your first glimpse into Kate's power and flexibility.

In **Tool Views** you have a list of all enabled plugins. Click the checkbox in front of each item or click with the left mouse button on the tool buttons in the sidebar to toggle this tool view on and off.

### 2.2.2 Shortcuts

Many of Kate's keystroke commands (shortcuts) are configurable by way of the [Settings](#) menu. By default Kate honors the following key bindings.

<b>F1</b>	Help
<b>Shift+F1</b>	<a href="#">What's this?</a>
<b>Ctrl+N</b>	New document
<b>Ctrl+L</b>	Save All
<b>Ctrl+O</b>	<a href="#">Open a document</a>
<b>Ctrl+Alt+O</b>	Quick Open
<b>Ctrl+Shift+F</b>	Full Screen Mode
<b>Ctrl+Shift+,</b>	Configure Kate
<b>Ctrl+W / Ctrl+Esc</b>	<a href="#">Close</a>
<b>Ctrl+Q</b>	Quit - close active copy of editor
<b>Ctrl+Alt+Shift+F</b>	Show Sidebars
<b>Ctrl+Shift+T</b>	Split Horizontal
<b>Ctrl+Shift+L</b>	Split Vertical
<b>F8</b>	Next Split View

<b>Shift+F8 / Ctrl+Esc</b>	Previous Split View
<b>Ctrl+Shift+R</b>	Close Current View
<b>Alt+Right</b>	Next Tab
<b>Alt+Left</b>	Previous Tab
<b>Ctrl+Shift+T</b>	Reopen Latest Closed Document(s)

Additionally you can use the shortcuts provided by the [KatePart](#) component and by all activated [Kate plugins](#).

## 2.3 Working With the KateMDI

### 2.3.1 Overview

Window, View, Document, Frame, Editor... What are they all in the terminology of Kate, and how do you get the most out of it? This chapter will explain all of that, and even more.

#### 2.3.1.1 The Main Window

The Kate Main Window is a standard KDE application window, with the addition of side bars containing tool views. It has a menubar with all the common menus, and some more, and a toolbar providing access to commonly used commands.

The most important part of the window is the editing area, by default displaying a single text editor component, in which you can work with your documents.

The docking capabilities of the window is used for the tool windows of any plugin enabled in the settings dialog.

Tool views can be positioned in any sidebar, to move a tool right click its sidebar button and select from the right mouse button menu

A tool view can be marked as *persistent* in the right mouse button menu for its sidebar button. The sidebar can contain more tools at one time so that when a tool is persistent other tools can be shown simultaneously.

If a plugin has configuration options you can use the first item in the context menu to open the corresponding page in Kate's settings dialog.

### 2.3.2 The Editor area

Kate is capable of having more than one document open at the same time, and also of splitting the editing area into any number of frames, similar to how for example Konqueror or the popular emacs text editor works. This way you can view several documents at the same time, or more instances of the same document, handy for example if your document contains definitions in the top that you want to see often for reference. Or you could view a program source header in one frame, while editing the implementation file in another.

When a document is available in more than one editor, changes made in one editor will immediately be reflected in the others as well. This includes changing the text as well as selecting text. Search operations or cursor movement is only reflected in the current editor.

It is currently not possible to have more instances of the same document open in the sense that one instance will be edited while the other will not.

When splitting an editor into two frames, it is divided into two equally sized frames, both displaying the current document of that editor. The new frame will be at the bottom (in the case of a horizontal split) or at the right (for a vertical split). The new frame gets the focus, which is visualized by the blinking cursor bar in the focused frame.

## 2.4 Using Sessions

Sessions is how Kate lets you keep more than one list of files and GUI configuration around. You can have as many named sessions as you want, and you can use unnamed or anonymous sessions for files you want to use only once. Currently Kate can save the list of open files, and the general window configuration in the session; future versions of Kate may add more features that can be saved in sessions. With the introduction of sessions, Kate also allows you to open any number of instances of the application instead of just one as it used to do as the default behavior.

Sessions are supported in three areas:

- *Command line options* that lets you select and start sessions when launching Kate from the command line.
- *The Sessions menu* that lets you switch, save, start and manage your sessions.
- *Configuration options* that lets you decide how sessions generally should behave.

When starting a new session, the GUI configuration of **Default Session** is loaded. To save window configuration in the default session, you need to enable saving window configuration in the sessions configuration page of the configuration dialog and then load the default session, set up the window as desired and save the session again.

When a named session is loaded, Kate will display the session name at the start of the window title, which then has the form *'Session Name: Document name or URL - Kate'*.

When opening files on the command line with `--start name` or if a session is selected using the session chooser, the specified session is loaded prior to the files specified on the command line. To open files from the command line in a new, unnamed session, configure Kate to start a new session as default in the session page of the configuration dialog or use `--start` with an empty string: `''`.

Since Kate 2.5.1 the PID of the current instance is exported to the environment variable `KATE_PID`. When opening files from the built in terminal Kate will automatically select the current instance if nothing else is indicated on the command line.

## 2.5 Quick Open

To be able to quickly open/switch between files, Kate comes with a built-in quick open dialog. You can open it with **Ctrl+Alt+O**.

Quick open can show all open documents in Kate as well as all files in open projects. To be able to see project files, you need to enable [Project Plugin](#).

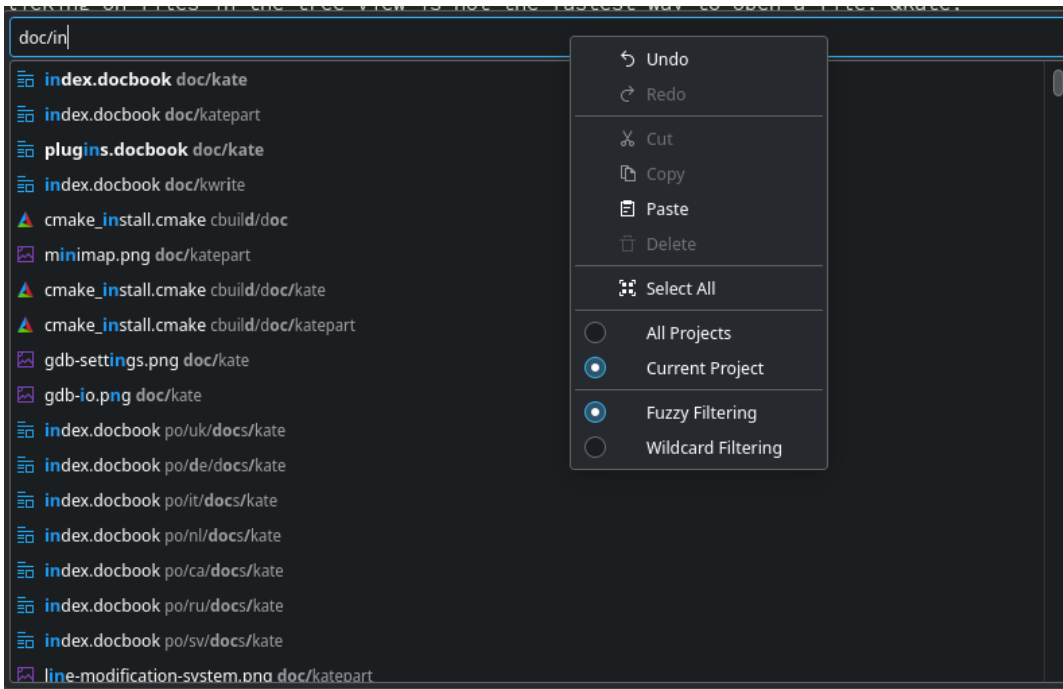
### 2.5.1 Using Quick Open

Using quick open is very simple. Once you open it, just type the name or portions of the name of the file you want to open and quick open will filter the list based on what you typed. Hitting **Enter** opens the selected file, while **Esc** hides the quick open.

By default, only the file name is matched while filtering. If you want to match path, you need to have a `"/"` in the typed text. For example: `"doc/index"` will match all the files that contain `"index"` inside the folder `"doc"`.

The documents which are already open are highlighted in bold and are listed at the top when the dialog opens. Also, when quick open shows up the previous open document is already selected so you can just press **Enter** and it will take you to that document.

## 2.5.2 Configuring Quick Open



Quick open provides a couple of config options. To access these options, right-click in the input line edit.

Currently available options are:

- Current Project** - Show files from current project only
- All Projects** - Show files from all open projects
- Fuzzy Filtering** - Use fuzzy matching algorithm to filter files
- Wildcard Filtering** - Use wildcard matching to filter files

## 2.6 Getting Help

### 2.6.1 With Kate

#### This manual

Offers detailed documentation on all menu commands, configuration options, tools, dialogs, plugins etc. as well as descriptions of the Kate window, the editor and various concepts used in the application.

Press **F1** or use the **Help** → **Kate Handbook** menu topic to view this manual.

#### What's This Help

What's This help offers immediate help with single elements of graphical windows, such as buttons or other window areas.

We strive to provide What's This help for any elements for which it makes sense. It is available throughout the configuration dialog, and in many other dialogs as well.

To employ What's This help, press **Shift+F1** or use the **Help** → **What's This** menu item to enable What's This mode. The cursor will turn into an arrow with a question mark, and

you can now click any element in the window to read the What's This help for that element, if it is available.

### **Help Buttons in Dialogs**

Some dialogs have a **Help** Button. Pressing it will start the KHelpCenter and open the relevant documentation.

## **2.6.2 With Your Text Files**

Kate does not (yet!) provide any means for reading document related documentation. Depending on the file you are editing, you may find the [Built in Terminal Emulator](#) helpful for viewing related UNIX<sup>®</sup> manual pages or info documentation, or you can use Konqueror.

## **2.6.3 Articles on Kate**

Kate's homepage provides some [Articles and Howtos](#) with further information beyond the scope of this handbook.

## Chapter 3

# Working with the Kate Editor

For information about the basics of working with the editor component underlying Kate, see the [Working with the KatePart Editor](#) chapter of the [KatePart Handbook](#).

## Chapter 4

# Working with Plugins

Anders Lund

You can enable the individual plugins in the [configuration dialog](#), which also provides access to additional configuration options for plugins that require it.

### 4.1 Kate Application Plugins

Kate plugins are additional functions for the Kate editor. They can add extra menus and shortcuts, and extend Kate's features. You can install as many or as few as you like, from within Kate. Open Kate's configuration dialog with **Settings** → **Configure Kate...** Select **Application** → **Plugins** to choose the wanted plugins.

The available application plugins are:

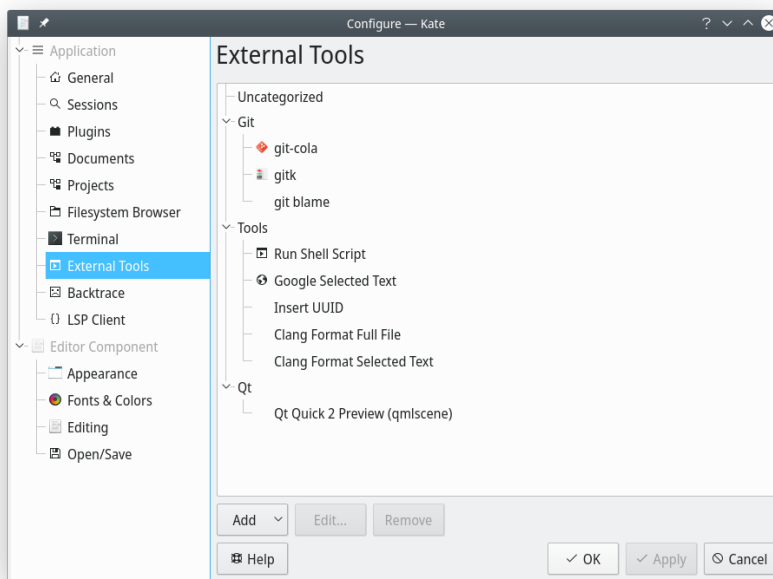
- [External Tools](#) - Run external tools and applications
- [Backtrace Browser](#) - C/C++ Backtrace navigation tool view
- [Build Plugin](#) - Compile or Make and parse error messages
- [Close Except/Like](#) - Close group of documents based on a common path or file extension
- [Color Picker](#) - Show preview for known color names
- [Colored Brackets](#) - Colored brackets for readability
- [CTags](#) - Look up definitions/declarations with CTags
- [Document preview](#) - Preview the document in the target format.
- [Document switcher](#) - Quick document switching with **Alt+Tab** behavior
- [File System Browser](#) - File system browser tool view
- [Document Tree View](#) - Displays the open files in a file tree
- [GDB](#) - Provides a simple GDB frontend
- [Project Plugin](#) - Integration with Git and other source control systems
- [Replicode](#) - Constructivist AI language and runtime
- [LSP Client](#) - LSP client providing code navigation and code completion for many languages

## The Kate Handbook

- [Search & Replace](#) - Search and replace in documents, folders, or projects
- [Scripts & Snippets tool view](#) - Create or download code snippets and editor scripts
- [Keyboard Macros](#) - Record and play keyboard macros (i.e., keyboard action sequences)
- [SQL Plugin](#) - Execute query on SQL databases
- [Symbol Viewer](#) - Extract and show reference symbols from source
- [Terminal tool view](#) - Have a terminal at the ready, using KDE's Konsole widget
- [Text Filter](#) - Process text using terminal commands
- [XML Completion](#) - Lists XML elements, attributes, attribute values and entities allowed by DTD
- [XML Validation](#)- Validates XML files using xmllint
- [Compiler Explorer](#)- Interface to compiler explorer tool
- [Formatting](#)- Code formatting plugin

## 4.2 External Tools

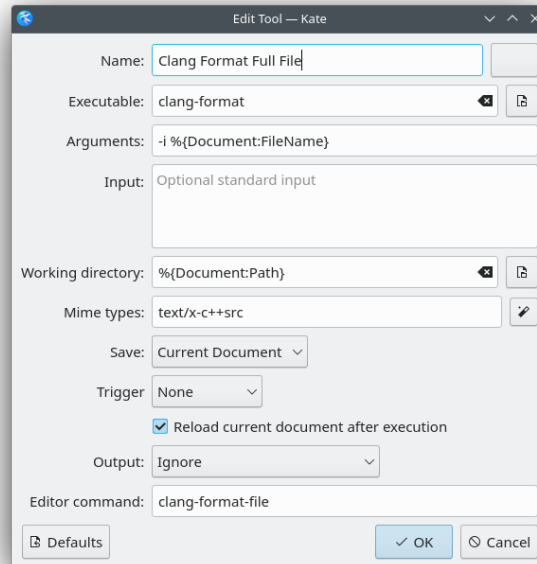
The **External Tools** plugin allows to invoke external applications with data related to the current document, for example its URL, directory, text or selection. Once enabled, a config page appears as depicted below that allows to change or remove existing tools. Similarly, new tools can be added to your liking. The tools will then appear in the **External Tools** submenu of the **Tools** menu of the application.



The config page allows to add new external tools by clicking on the **Add** button. In this case, a popup menu appears where one can either add a new external tool, add an existing tool from a predefined list, or add a new category to organize the external tools into categories. Similarly, the existing tools can be modified either by double-click or by invoking **Edit...**, and **Remove** removes the selected tools.

## 4.2.1 Configuring External Tools

Editing a tool opens a config dialog that allows fine-grained configuration of the tool:



As can be seen, many details can be defined, namely:

**Name**, the name of the tool, which will later appear in the menu.

**Icon**, optional icon that is visible in the menu.

**Executable**, executable including either a full path, or your executable must be in the PATH environment variable.

**Arguments**, optional arguments that are passed to the executable.

**Input**, optional input that is passed to the process via stdin.

**Working directory**, the working directory the tool will be started in. If empty, the working directory is set to the current document's path.

**Mime types**, if set, the tool is active only if the current document's mime type matches.

**Save**, when invoked, saves none, the current document, or all documents.

**Trigger**, a trigger to execute this tool. A trigger will only affect the currently active document and will only execute if the mimetype of current active document matches the mimetype of the external tool.

Following triggers are available:

**None**, this is the default, it means the tool has no trigger.

**Before Save**, this trigger will execute right before saving the document.

**After Save**, this trigger will execute the tool after the document was saved.

**Reload current document after execution**, useful when the current file is modified on disk.

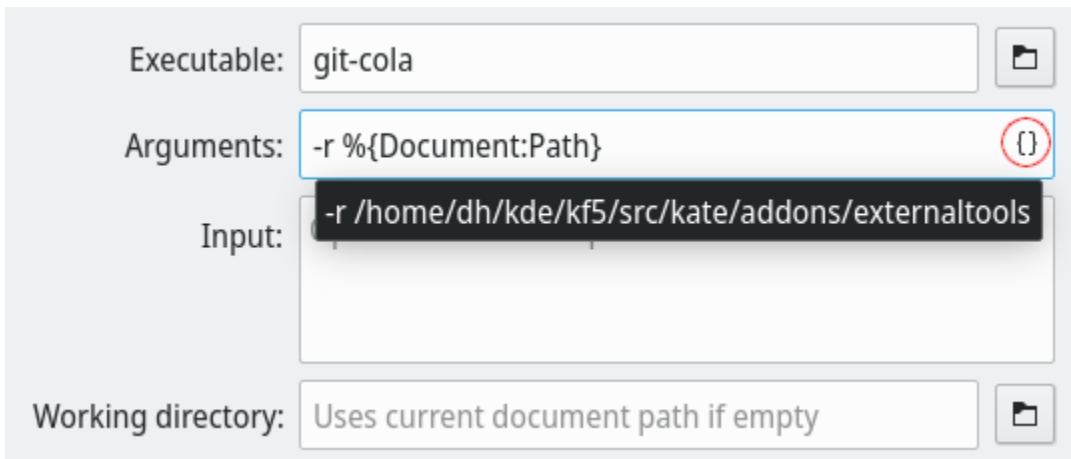
**Output**, the output defines the target of stdout. It is either set to **Ignored**, **Insert at Cursor Position**, **Replace Selected Text**, **Replace Current Document**, **Append to Current Document**, **Insert in New Document**, **Copy to Clipboard**, or **Display in Pane**.

**Editor command**, optional command that can be used to invoke the external tool via the built-in [command line](#).

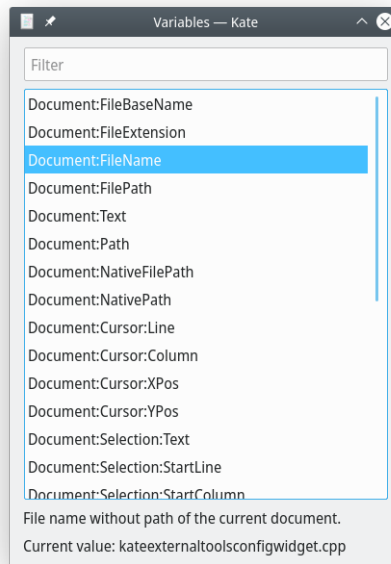
The button **Defaults** is visible only for tools that are shipped with Kate. When clicked, all tool's settings reverted to default (aka factory) values.

### 4.2.2 Variable Expansion

Some editing fields such as the **Executable**, the **Arguments**, the **Input** and the **Working Directory** support variables that are expanded on tool invocation. This is indicated by the icon {} that appears once one of these text input fields has focus (see red circle):



Hovering over one of these text input fields also shows a tooltip with the current expanded text. Further, clicking on the {} action will open a dialog that lists all available variables:



This feature provides a lot of flexibility when defining an external tool since all variables of the form `%{...}` are expanded when the tool gets invoked. There are two kind of variables supported:

- `%{variable-name}`

- `%{variable-name:<value>}`

The first form `%{variable-name}` simply replaces the variable with its contents. For instance, the variable `%{Document:FileName}` is replaced by the current document's filename without its path. The second form `%{variable-name:<value>}` gets the `<value>` as contents. For example, this can be used to expand an environment variable with `%{ENV:HOME}`, or one can obtain the current date in the preferred format like `%{Date:yyyy-MM-dd}`.

#### Supported variables include:

**Document:FileName:** File base name without path and suffix of the current document.

**Document:FileExtension:** File extension of the current document.

**Document:FileName:** File name without path of the current document.

**Document:FilePath:** Full path of the current document including the file name

**Document:Text:** Contents of the current document.

**Document:Path:** Full path of the current document excluding the file name.

**Document:NativeFilePath:** Full document path including file name, with native path separator (backslash on Windows).

**Document:NativePath:** Full document path excluding file name, with native path separator (backslash on Windows).

**Document:Cursor:Line:** Line number of the text cursor position in current document (starts with 0).

**Document:Cursor:Column:** Column number of the text cursor position in current document (starts with 0).

**Document:Cursor:XPos:** X component in global screen coordinates of the cursor position.

**Document:Cursor:YPos:** Y component in global screen coordinates of the cursor position.

**Document:Selection:Text:** Text selection of the current document.

**Document:Selection:StartLine:** Start line of selected text of the current document.

**Document:Selection:StartColumn:** Start column of selected text of the current document.

**Document:Selection:EndLine:** End line of selected text of the current document.

**Document:Selection:EndColumn:** End column of selected text of the current document.

**Document:RowCount:** Number of rows of the current document.

**Document:Variable:<variable>:** Expand arbitrary [document variables](#).

**Date:Locale:** The current date in current locale format.

**Date:ISO:** The current date (ISO).

**Date:<value>:** The current date ([QDate formatstring](#)).

**Time:Locale:** The current time in current locale format.

**Time:ISO:** The current time (ISO).

**Time:<value>:** The current time ([QTime formatstring](#)).

**ENV:<value>:** Access to environment variables.

**JS:<expression>:** Evaluate simple JavaScript statements.

**PercentEncoded:<text>:** Percent encoded text.

**UUID:** Generate a new UUID.

### 4.2.3 List of Default Tools

Several tools are shipped by default. However, if you have more useful tools please contribute those to [our GitLab project](#) so that we can add them to this list. All default tools are visible in the list view by default. However, all tools can be changed to your liking, including the category or even deleting tools. Deleted tools can be added back again by clicking on the **Add** button in the config page as described above.

GIT-COLA

**git-cola is a graphical git client that enables you to easily stage and commit changes. If installed, it is available also through the command line by typing `git-cola`**

**Name:** git-cola  
**Icon:** git-cola  
**Executable:** git-cola  
**Arguments:** -r %`{Document:Path}`  
**Editor command:** git-cola

GITK

**gitk is a git client as well that allows to nicely visualize the git history.**

**Name:** gitk  
**Icon:** git-gui  
**Executable:** gitk  
**Working directory:** %`{Document:Path}`  
**Editor command:** gitk

GIT BLAME

**Starts git blame to easily follow git changes in the current file.**

**Name:** git blame  
**Executable:** git  
**Arguments:** gui blame %`{Document:FileName}`  
**Save:** Current Document  
**Working directory:** %`{Document:Path}`  
**Editor command:** git-blame

RUN SHELL SCRIPT

**Starts an external konsole in which the current document is executed. The script needs to state the interpreter in the first line via a shebang `#!/path/interpreter`.**

**Name:** Run Shell Script  
**Icon:** system-run  
**Executable:** konsole  
**Arguments:** -e sh -c "cd %`{Document:Path}` && pwd && chmod -vc a+x %`{Document:FileName}` && ./%`{Document:FileName}` ; echo Press any key to continue. && read -n 1"  
**Save:** Current Document  
**Working directory:** %`{Document:Path}`  
**Editor command:** run-script

## The Kate Handbook

### GOOGLE SELECTED TEXT

**Search in google for the selected text.**

**Name:** Google Selected Text

**Icon:** globe

**Executable:** xdg-open

**Arguments:** "https://www.google.com/search?q=%{Document:Selection:Text}"

**Editor command:** google

### INSERT UUID

**Inserts a new UUID each time this action is invoked.**

**Name:** Insert UUID

**Executable:** echo

**Arguments:** %{UUID}

**Output:** Insert at Cursor Position

**Editor command:** uuid

### CLANG FORMAT FULL FILE

**Runs clang-format on the current file on disk. The document is reloaded afterwards.**

**Name:** Clang Format Full File

**Executable:** clang-format

**Arguments:** -i %{Document:FileName}

**Working directory:** %{Document:Path}

**Save:** Current Document

**Reload:** Yes

**Editor command:** clang-format-file

### CLANG FORMAT SELECTED TEXT

**Runs clang-format just on the selected text in the current document.**

**Name:** Clang Format Selected Text

**Executable:** clang-format

**Arguments:** -assume-filename: %{Document:FileName}

**Working directory:** %{Document:Path}

**Input:** %{Document:Selection:Text}

**Output:** Replace Selected Text

**Editor command:** clang-format-selection

### QT QUICK 2 PREVIEW (QMLSCENE)

**Previews the current qml file in qmlscene.**

**Name:** Qt Quick 2 Preview (qmlscene)

**Executable:** qmlscene

**Arguments:** %{Document:FileName}

**Save:** Current Document

**Working directory:** %{Document:Path}

**Editor command:** qml-preview

## JSON FORMAT FULL FILE

### Format the entire JSON file.

**Name:** JSON Format Full File  
**Icon:** application-json  
**Executable:** jq  
**Arguments:** %{Document:FileName}  
**Save:** Current Document  
**Working directory:** %{Document:Path}  
**Output:** Replace Current Document  
**Editor command:** json-format-file

## XML FORMAT FULL FILE

### Format the entire XML file.

**Name:** XML Format Full File  
**Icon:** application-xml  
**Executable:** xmllint  
**Arguments:** --format %{Document:FileName}  
**Save:** Current Document  
**Working directory:** %{Document:Path}  
**Output:** Replace Current Document  
**Editor command:** xml-format-file

## 4.3 Backtrace Browser Plugin

### 4.3.1 Using the Backtrace Browser Plugin

This plugin is meant for developers and probably of little use for users. It shows a backtrace delivered by gdb in a listview in a Kate toolview. Clicking on an item opens the selected file and jumps to the correct line number. It works for backtraces generated on your own machine, but it will also work for backtraces from other people, i.e. with `/home/dummy/qt-copy/.../qwidget.cpp` will still be found on other machines. For that to work, you have to index the directories where the source code is located.

Sometimes there are several files with the same name, e.g.

```
kdegraphics/okular/generators/dvi/config.h  
kdepim-runtime/resources/gmail/saslplugin/config.h
```

To pick the right choice, the plugin picks the last two parts of the URL, in this case this would be

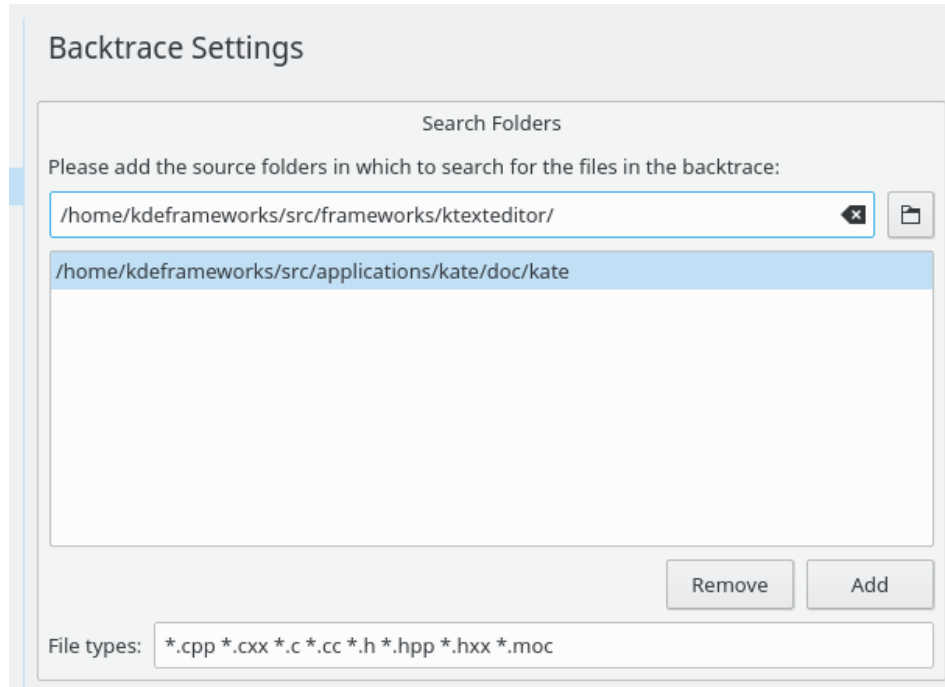
```
dvi/config.h  
saslplugin/config.h
```

And then usually the plugin finds the correct one.

Indexing master and a branches of course will lead to a clash.

## 4.3.2 Configuration

On the configuration page add the directories containing the source code.



*Configure Paths in Backtrace browser tool view*

Clicking **OK** will start indexing. When indexing is finished, open the toolview **Backtrace Browser**.

Now you can load a backtrace from the clipboard (e.g., when you clicked **Copy to Clipboard** in DrKonqi) or from a file.

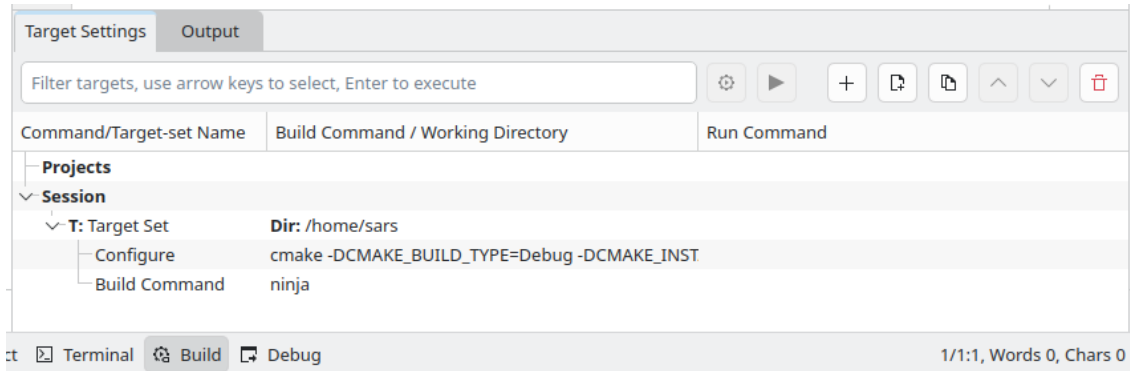
## 4.4 Build Plugin

### 4.4.1 Introduction

The Build plugin allows you to run actions like build, clean and compile on a project. You can also automatically run the generated applications. The plugin basically gives you a way to configure sets of commands to run and it can parse the output for links to files and specific lines and columns in those files. So even if the plugin is originally created for compiling C/C++ and mostly tested with it, the plugin can also be useful for other purposes and languages.

### 4.4.2 Using the Build Plugin

The Build plugin adds a **Build** tool view at the bottom and a **Build** menu on the menu bar. The tool view can be used to select and configure build targets, while the menu and its shortcuts can be used to select and execute the configured shell commands.



The **Build** tool view has two static tabs:

- **Target Settings**
- **Output**

#### 4.4.2.1 Target Settings tab

The target settings tab can be used to configure various build targets and define target sets.

A target-set is a group of commands that can be run in a specified working directory. Target sets can be added under either **Projects** or **Session**. The custom target-sets added under **Projects** are stored in a `.kateproject.build` file in the project root directory and are restored when the project is re-opened. The target-sets added under **Session** are stored in the current Kate session configuration.

The first row, in a target-set, contains a name for the set in column one, and in column two we have the directory where the commands should be executed. On each following row, we have a name for the command in column one, a build command in column two and a run command in column three. To edit one of these, double-click on the entry or use the edit shortcut (often **F2**).

#### Working Directory

The second column of the first row in a target-set, is used to configure the working directory used for compiling and running commands. If the project plugin is enabled, the working directory string can also contain placeholders for the project base directory path: `%B` and name: `%b`







#### Build Command

The second column in the "non-first rows", contains the shell command to run in the working directory. Note the word shell. Almost any shell command will do. The build command can contain placeholders. `%f` for the current file, `%d` for directory of the current file and `%n` for the base-name of the current file (file name without suffix).

#### Run Command

The third column in the "non-first rows", can contain a shell command, for execution in an actual terminal in the target-set working directory. The terminal is opened as a tab. The plugin will try to reuse the terminal tab, if the same command is executed and the previous application has exited.

On the top of the **Target Settings** tab, we have a toolbar with a target filter and the following buttons:

-  Build the selected target
- ▶  Build and run the selected target
-  Add a new build target
-  Create a new set of targets
-  Copy a command or target set
-  Delete the current command or target set

#### 4.4.2.2 Output tab

The **Output** tab shows the console output generated by the running (build) command. If a line contains a file location, that line will be clickable. If the line in the output also shows an error or warning, the line will have a different color.

If the option **Add errors and warnings to Diagnostics** is enabled in the plugin's settings page in Kate settings, the errors and warnings will also be added to the diagnostics view. To navigate to the previous error in the diagnostics view, press **Alt+Shift+Left**. To navigate to the next error, press **Alt+Shift+Right**.

#### 4.4.3 Menu Structure

##### **Build** → **Select Target...**

The target selection filter is focused in the Target Settings tab. Typing a name will filter out targets that do not match the typed string. It is also possible to use the arrow keys to navigate the tree of targets. When the wanted target is selected, pressing **Return** or **Enter** will execute and run the selected target.

##### **Build** → **Build Selected Target**

Builds the last selected target. If none has been selected, it works as **Select Target...**

##### **Build** → **Build and Run Selected Target**

Builds the last selected target and run the run command after the build command has finished successfully.

##### **Build** → **Compile Current File**

Try to compile the current file by searching for a command in a possible `compile_commands.json`.

##### **Build** → **Stop**

Stop building a target.

##### **Build** → **Focus Next Tab to the Left**

Focus the next build plugin tab to the left. Or open the build plugin tool-view if it is hidden.

##### **Build** → **Focus Next Tab to the Right**

Focus the next build plugin tab to the right. Or open the build plugin tool-view if it is hidden.

##### **Build** → **Load targets from CMake Build Dir**

Opens the file dialog and let's the user select a **CMakeCache.txt**. When a file is selected, the plugin generates cmake build command targets that can be executed in the build directory for the CMake based project.

## 4.4.4 Thanks and Acknowledgments

The Kate Build Plugin was written by Kåre Särs.

Special thanks to Google Code-In 2011 participant Salma Sultana for writing much of this section.

## 4.5 Close Except/Like Plugin

### 4.5.1 Introduction

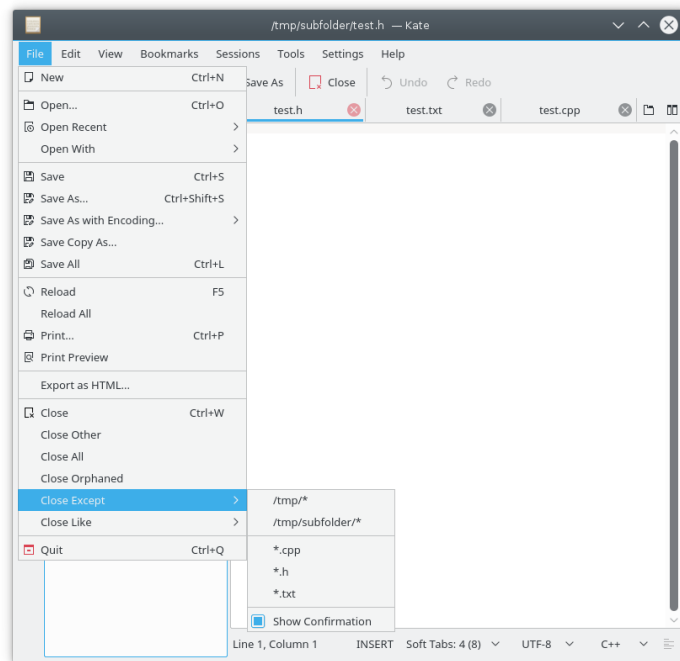
This plugin allows you to close a group of documents based on their extension and path.

### 4.5.2 Using the Close Except/Like Plugin

Assumed you have these documents opened in Kate:

```
/tmp/subfolder/test.h  
/tmp/test.cpp  
/tmp/test.txt
```

Then you have the following options to close documents as displayed in the screenshot:



Use the checkbox in the last item of the list to enable or disable a confirmation dialog. The selected option will be applied to both close actions.

### 4.5.3 Menu Structure

#### File → Close Except

Close all open documents, *except* those which match the path or file extension selected from the submenu.

**File** → **Close Like**

Close all open documents which match the path or file extension selected from the sub-menu.

## 4.6 Color Picker Plugin

### 4.6.1 Introduction

This plugin adds an inline color preview/picker to colors in the text (e.g., #FFFFFF, white).

To load this plugin open Kate's configuration dialog under **Settings** → **Configure Kate...** Then select **Color Picker** and close the dialog.

### 4.6.2 Configuration

On the Color Picker settings page in Kate's configuration, you can configure the following options of the plugin behavior.

**Show preview for known color names**

Whether to show the color picker for known color names (e.g., skyblue). See [this page](#) for the list of colors.

**Place preview after text color**

Whether to place the inline preview after text color in the text.

**Hex color matching**

Here, you can choose the best matching option for the colors used in your code.

## 4.7 Colored Brackets

### 4.7.1 Introduction

Colored brackets plugin colors matching bracket pairs in different colors to enhance readability. However, not all brackets get colored. A bracket whose matching opener or closer is not visible will be ignored. Similarly, a bracket pair that is the sole bracket pair on a line will not get colored.

### 4.7.2 Configuration

The plugin doesn't provide any configurations.

## 4.8 CTags Plugin

### 4.8.1 Introduction

**CTags** generates an index (or tag) file of language objects found in source files that allows these items to be quickly and easily located using this plugin in Kate.

A tag signifies a language object for which an index entry is available (or, alternatively, the index entry created for that object).

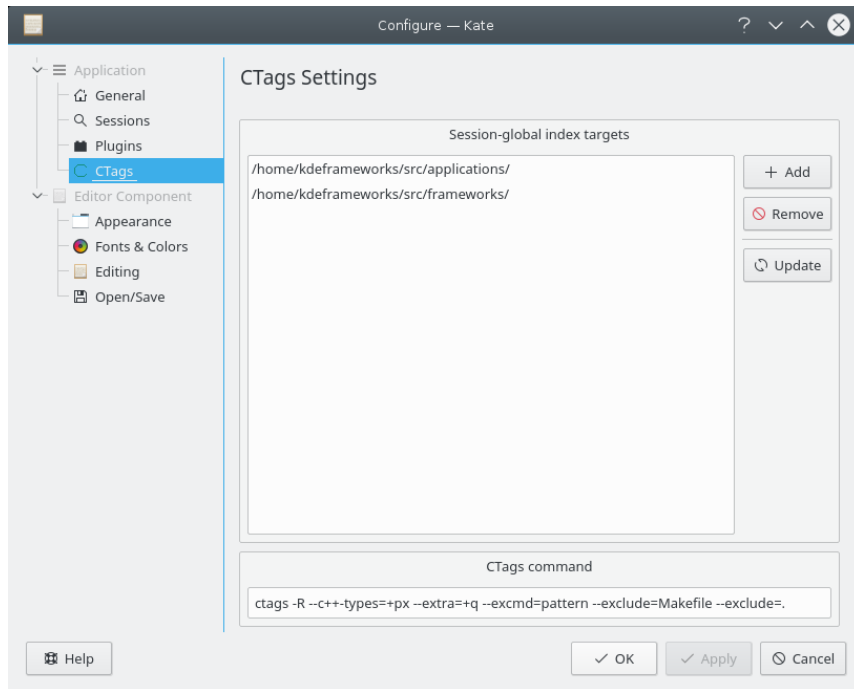
Tag generation is supported for these [programming languages](#).

## 4.8.2 Configuration

The CTags plugin uses two different database files for the index.

On the CTags settings page in Kate's configuration you can add or remove directories containing the source code and regenerate the common CTags database.

### 4.8.2.1 Common Index



#### *Configure CTags Global Database*

At the bottom of the settings page you can adapt the **CTags command**.

For more information about all available options please read the CTags man page. This man page is available in KHelpCenter and you can also enter the URL *man:/ctags* directly into Konqueror

Clicking **Update** will start indexing. When indexing is finished, close the dialog.

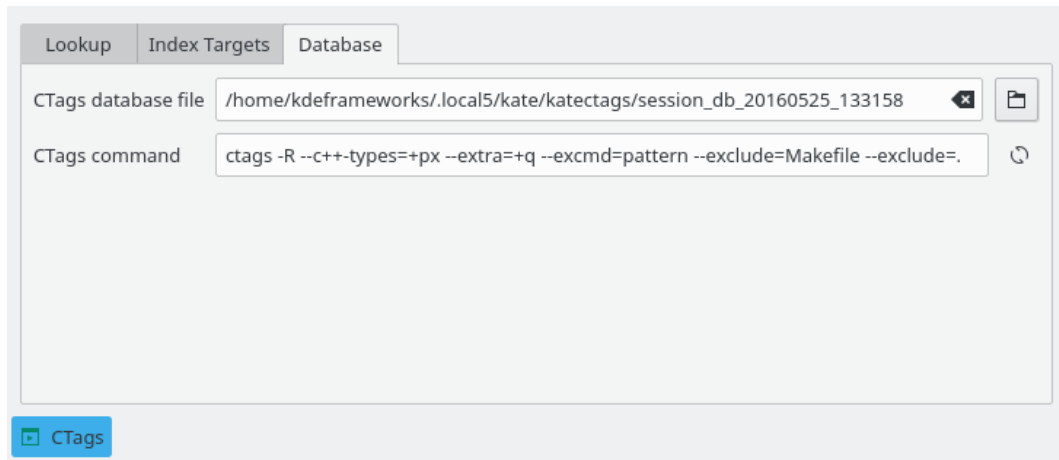
### 4.8.2.2 Session Index

To configure the session index open the **CTags** view.

#### **Index Targets**

On this tab you can add or remove directories containing the source code and manually regenerate the session specific CTags database.

#### **Database**



#### Configure CTags Session Database

Select another CTags database file, configure the CTags command or revert to the default command.

### 4.8.3 Using the CTags Plugin

You place the mouse cursor on the language object like function, symbol etc. that you are interested in and then select one of the actions in the **CTags** menu to jump to the line and file where the object is defined or declared.

By default the actions in the **CTags** menu have no shortcuts assigned. Use the [keyboard shortcut editor](#) to configure your own shortcuts.

Alternatively use the search field on the **Lookup** tab of the CTags view.

Entering characters into the search field will start the search and display matching names of language objects like functions, classes, symbols etc. together with type and filename.

Select an item in the list to jump to the corresponding line in the source file.

### 4.8.4 Menu Structure

#### **CTags** → **Jump back one step**

Navigate back in the history to the last visited tag.

#### **CTags** → **Lookup Current Text**

Opens the **Lookup** tab of the CTags view and displays all language objects matching the current text selection in the list.

#### **CTags** → **Go to Declaration**

If the cursor is in a definition object this will open the document containing the corresponding declaration if needed, activate its view and place the cursor at the start of the declaration.

#### **CTags** → **Go to Definition**

If the cursor is in a declaration object this will open the document containing the corresponding definition if needed, activate its view and place the cursor at the start of the definition.

## 4.9 Document Preview Plugin

### 4.9.1 Introduction

The plugin enables a live preview of the currently edited text document in the final format in the sidebar. So when editing e.g. a Markdown text or an SVG image, the result is instantly visible next to the source text.

For the display the plugin uses that KParts plugin which is currently selected as the preferred one for the MIME type of the document. If there is no KParts plugin for that type, no preview is possible.

To change the preferred plugin open the **File Associations** module in the System Settings and edit the **Services Preference Order** on the **Embedding** tab.

MIME type	KParts plugin
Markdown text	KMarkdownWebViewPart or OkularPart
SVG image	SVGPart
Qt™ UI files	KUIViewerPart
DOT graph files	KGraphviewerPart

Table 4.1: Some available KParts plugins

### 4.9.2 Menu Structure

**View** → **Tool Views** → **Show Preview**

Toggle the display of Kate's Document preview in a sidebar.

### 4.9.3 Interface

The buttons at the top of the preview window provide these actions:

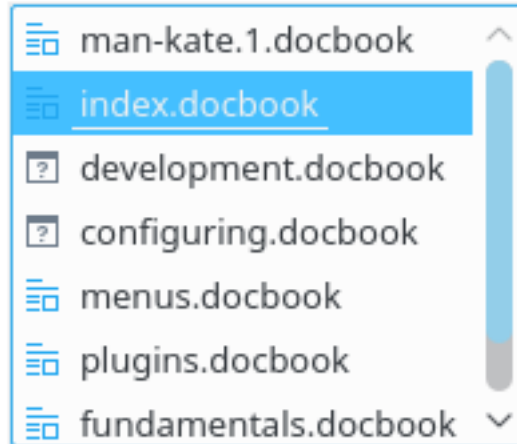
- Lock the preview to a given document. Selecting this option ensures that if switching the focus to the view of another document in the same Kate window, the preview will not follow to that document, but keep previewing this document.
- Enable or disable updates of the preview of the current document content
- Manually update the preview of the current document content
- A dropdown menu with actions from the KParts plugin

## 4.10 Document Switcher Plugin

### 4.10.1 Menu Structure

**View → Last Used Views (Ctrl+Tab), View → Last Used Views (Reverse) (Ctrl+Shift+Tab)**

Opens a list with the last viewed documents:



Keep the **Ctrl** key pressed and use the **Tab** key to cycle forward through the list. Additionally press the **Shift** key to reverse the direction.

Keep the shortcut **Ctrl+Tab** pressed and can use the **Up**, **Down**, **Home** or **End** keys to navigate in the list. Pressing a char key consecutively will cycle through all items with the first matching in the list. If you release the shortcut keys the view will switch to the selected document in the list.

## 4.11 File System Browser

The File System Browser is a folder viewer, allowing you to open files from a displayed folder in the current frame.

### 4.11.1 Menu Structure

**View → Tool Views → Show Filesystem Browser**

Toggle the display of Kate's Filesystem Browser.

### 4.11.2 Interface

From the top down, the Filesystem Browser consists of the following elements:

#### A Toolbar

This contains standard navigations tool buttons:

- ◀ **Back**  
Causes the folder view to **cd** to the previously displayed folder in the history. This button is disabled, if there is no previous item.
- ▶ **Forward**  
Causes the folder view to **cd** to the next folder in the history. This button is disabled, if there is no next folder.



### Bookmarks

Opens a submenu to edit or add bookmarks and to add a new bookmark folder.



### Current Document Folder

This button will cause the folder view to **cd** to the folder of the currently active document if possible. This button is disabled, if the active document is a new, unsaved file, or the folder in which it resides can not be decided.



### Options

#### Short View

Displays only the filenames.

#### Detailed View

Displays **Name**, **Date** and **Size** of the files.

#### Tree View

Like Short View, but folders can be expanded to view their contents.

#### Detailed Tree View

This also allows folders to be expanded, but displays the additional columns available in Detailed View.

#### Show Hidden Files

Displays files normally hidden by your operating system.

#### Automatically synchronize with current document

When this option is enabled the filesystem browser will automatically **cd** to the folder of the document currently open in the editing area every time it changes.

## A Location Entry

This displays a breadcrumb navigation to the currently open folder, similarly to Dolphin. You can click on a any folder to browse to it, or click on one of the arrows to the left of a folder to select any folders beneath it. You may also select from your list of Places by clicking the leftmost icon in the breadcrumb navigation, which displays an icon that represents your current Place.

You can also click to the right of the breadcrumbs to change them to a text box where you can type the path of a folder to browse. The URL entry maintains a list of previously typed paths. To choose one, use the arrow button to the right of the entry.

#### TIP

The URL entry has folder auto-completion. The completion method can be set using the right mouse button menu of the text entry.

## A Folder View

This is a standard KDE folder view.

## A Filter Entry

The Filter entry allows you to enter a filter for the files displayed in the folder view. The filter uses standard globs; patterns must be separated by white space. Example: **\*.cpp \*.h \*.moc**

To display all files, enter a single asterisk **\***.

The filter entry saves the last 10 filters entered between sessions. To use one, press the arrow button on the right of the entry and select the desired filter string. You can disable the filter by pressing the **Clear text** button to the left of the autocompletion arrow button.

## 4.11.3 Configuration

This plugin can be configured on the **Filesystem Browser** page of [Kate's configuration](#).

## Toolbar







Configure the buttons on the Filesystem Browser toolbar by moving the ones you want enabled to the **Selected Actions** list, and order them using the arrow buttons at the side of the list.

## 4.12 The Documents List

### 4.12.1 Introduction

The documents list displays a list of all documents currently open in Kate. Modified files will have a small **floppy disk** icon on their left to indicate that state.

On the top the Documents list has a toolbar with the following buttons:

-  Create new document
-  Open an existing document
-  Previous Document
-  Next Document
-  Save the current document
-  Save the current document under a new name

By default, the Documents list appears in **Tree Mode**, which displays the folder structure surrounding all currently open documents. Also available is **List Mode**, which displays a simple list of all open documents. You can switch modes by right-clicking on the list and selecting from the **View Mode** menu.

If two or more files with the same name (located in different folders) are open in **List Mode**, the names of the second will be prepended '(2)' and so on. The tool-tip for the file will display its full name including the path, allowing you to choose the desired one.

To display a document in the currently active frame, click the document name in the list.

The context menu has some common actions from the **File** menu.

Additionally there are filemanager actions to rename or delete the file. With **Copy Location** you can copy the full path of the document to the clipboard.

You can sort the list in a few different ways by right clicking the list and selecting from the **Sort By** menu. The options are:

#### Document Name

Lists the documents alphabetically by their name.

#### Document Path

Lists the documents alphabetically by the path to them.

#### Opening Order

Lists the documents in the order of opening.

The document list will per default visualize your history by shading the entries for the most recent documents with a background color. If the document was edited, an extra color is blended in. The most recent document has the strongest color, so that you can easily find the documents you are working on. This feature can be disabled in the [Documents](#) page of the configuration dialog.

The default location of the document list in the Kate window is to the left of the editing area.

## 4.12.2 Menu Structure

### **View → Previous Document (Alt+Up)**

Opens the document displayed above the currently open document in the Documents list.

### **View → Next Document (Alt+Down)**

Opens the document displayed below the currently open document in the Documents list.

### **View → Show Active**

Displays the currently open document in the Documents list.

## 4.12.3 Configuration

### **Background Shading**

This section allows you to enable or disable the background shading visualization of your recent activity, and choose which colors to use if enabled.

### **Sort By**

Set how you want the document list sorted. This can be set from the right mouse button menu in the document list as well.

### **View Mode**

This provides two options that effect the display of the Documents tool view. The **Tree View** option will display the documents in a tree underneath the folders they are in, while the **List View** option will display a flat list of documents.

### **Show Full Path**

When Tree View and this option are enabled, the folder entries displayed in the Documents tool view will display the full filesystem path to the folder in addition to the name of the folder. It has no effect in List View.

### **Show Toolbar**

When Tree View and this option are enabled, a toolbar with actions like **Save** is displayed above the list of documents. Uncheck this option if the toolbar should be hidden.

### **Show Close Button**

When this option is enabled, Kate will show a close button for opened documents on hover.

## 4.13 GDB Plugin

### 4.13.1 Introduction

Kate's GDB plugin provides a simple frontend to any debugger that supports the [Debugger Adapter Protocol](#). In particular, that includes the GNU Project Debugger, aka GDB, as [outlined here](#).

#### **IMPORTANT**

Previous experience with GDB is strongly recommended. For more information on using GDB, visit [the GDB website](#).

You can enable the GDB plugin in [the Plugins section of Kate's configuration](#).

**TIP**

If you compile using **gcc/g++** you might want to use the **-ggdb** command line argument.

After these preparations are made, open the source file in Kate, select the "debugger profile", enter the path to the executable in the **Settings** tab of the **Debug View** tool view, and select **Debug** → **Start Debugging** from the menu to get started.

The "debugger profile" selects the DAP server to use (e.g. GDB) and the way in which to launch this server. A typical case is to have the server launch a process as specified above, but it may also attach to a running process (in which case a PID will have to be specified rather than an executable). There may also be other modes which are specific to the language and DAP server. See also later for additional background and configuration details on this.

### 4.13.2 Menu and Toolbar Structure

All of these options are available in Kate's menus, and many are available on the Debug toolbar as well.

**View** → **Tool View** → **Show Debug View**

Shows a tool view containing GDB output, the GDB command line used, and other settings.

**View** → **Tool View** → **Show Locals and Stack**

Shows a list of all currently loaded variables and their values and a GDB backtrace.

**Debug** → **Targets**

A submenu containing a list of targets (executables).

**Debug** → **Start Debugging**

Starts GDB with a target.

**Debug** → **Kill / Stop Debugging**

Stops GDB.

**Debug** → **Restart Debugging**

Restarts GDB.

**Debug** → **Toggle Breakpoint / Break**

Set a breakpoint at the current cursor position.

**Debug** → **Step In**

Execute the present statement (function call will be debugged).

**Debug** → **Step Over**

Execute the present statement (function call will not be debugged).

**Debug** → **Step Out**

Resumes execution until the program that is executing terminates.

**Debug** → **Move PC**

Move program counter (next execution).

**Debug** → **Run To Cursor**

Runs the program until it reaches current cursor position.

**Debug → Continue**

Ignores any breakpoints and executes program until it terminates (successfully or not).

**Debug → Print Value**

Prints the value of the variable that the cursor is currently pointing to.

**Settings → Toolbars Shown → GDB Plugin**

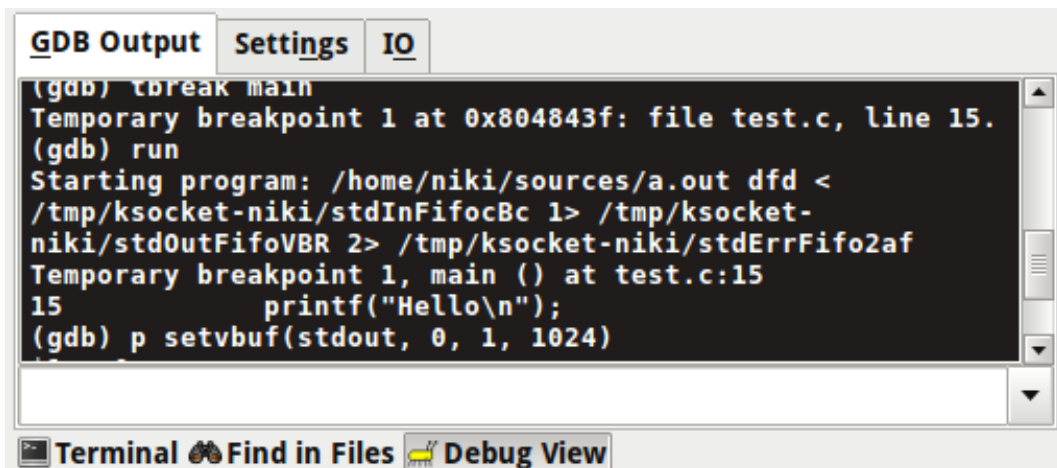
Display the debugging toolbar.

### 4.13.3 Debug View

The **Debug View** tool view consists of several tabs:

**GDB Output**

Contains output from GDB and a GDB command line.



*The **Output** tab displaying the output from a debugging session.*

**Settings**

**Executable**

Path to the target (executable) for debugging.

**Working Directory**

The current working directory provided to the target.

**Arguments**

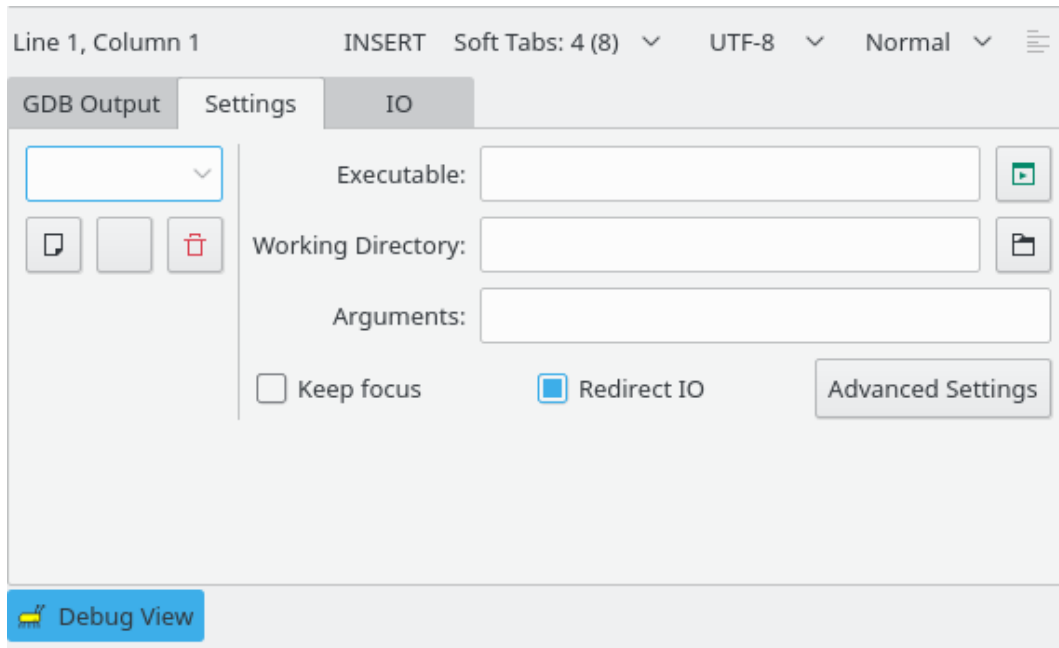
Arguments passed to the program.

**Keep focus**

Keeps focus on the GDB command line.

**Redirect IO**

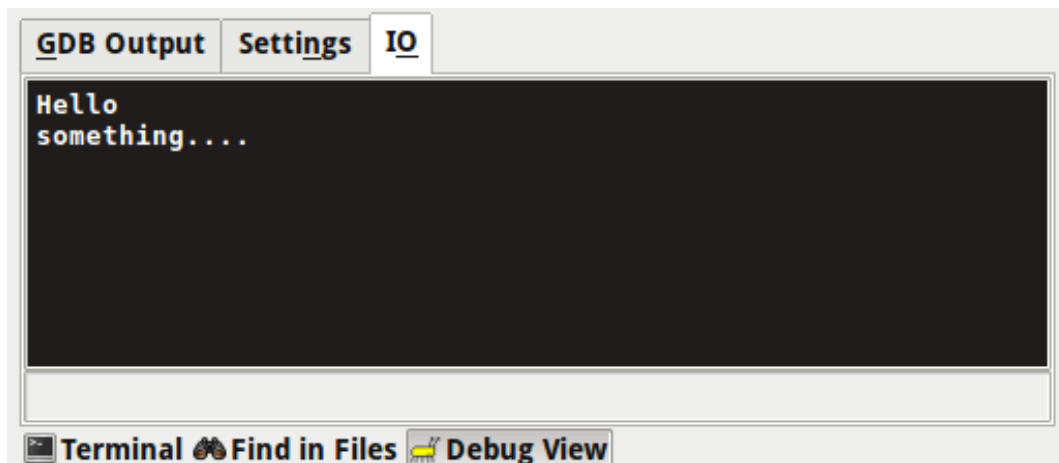
Opens a new **IO** tab in the **Debug View** where you can view output and provide input to the running program.



The *Settings* dialog displaying the configuration of a debugging session.

## IO

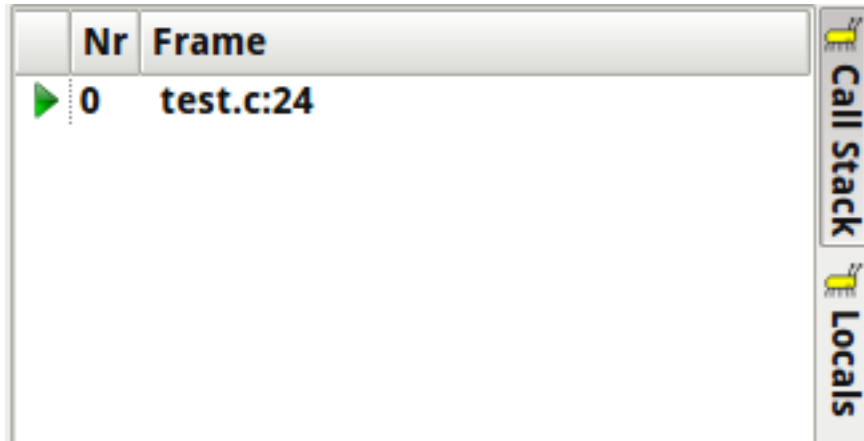
Contains an area that displays output from the running program and a command line where you may provide input to it.



The *IO* tab displaying output from a simple test program.

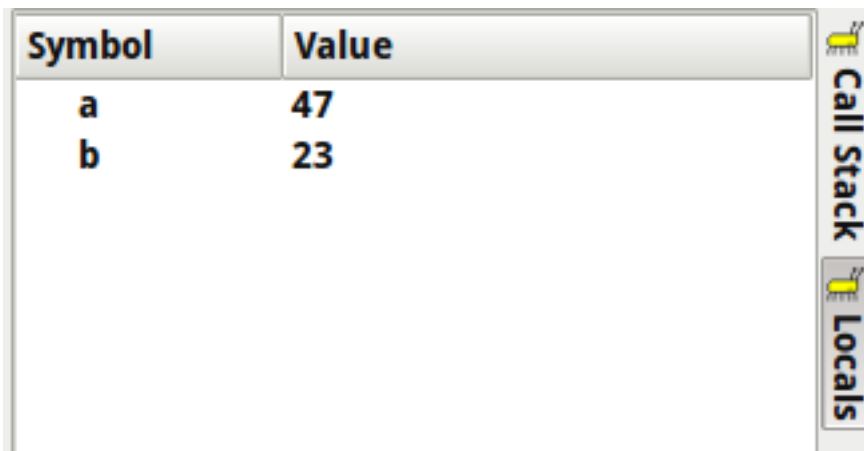
### 4.13.4 Call Stack and Locals

The **Call Stack** tool view contains a list of the formatted backtrace returned from GDB.



*The GDB Plugin's Call Stack tool view.*

The **Locals** tool view contains a list of all currently loaded variables from the program and their corresponding values.



*The GDB Plugin's Locals tool view.*

### 4.13.5 Thanks and Acknowledgments

Special thanks to Google Code-In 2011 participant Martin Gergov for writing much of this section.

### 4.13.6 Configuration

The plugin's configuration page defines the "debugger profiles" which can be selected. The default configuration (JSON) is shown there, and it can be "overlaid" by a user provided of similar form. An example excerpt is as follows:

```
{
  "dap": {
    "debugpy": {
      "url": "https://github.com/microsoft/debugpy",
      "run": {
        "command": ["python", "-m", "debugpy", "--listen", "${#run. ←
port}", "--wait-for-client"],
```

## The Kate Handbook

```
    "port": 0,
    "supportsSourceRequest": false
  },
  "configurations": {
    "launch": {
      "commandArgs": ["${file}", "${args|list}"],
      "request": {
        "command": "attach",
        "stopOnEntry": true,
        "redirectOutput": true
      }
    },
    "attach": {
      "commandArgs": ["--pid", "${pid}"],
      "request": {
        "command": "attach",
        "stopOnEntry": true,
        "redirectOutput": true
      }
    }
  },
  "gdb": {
    "url": "gdb",
    "run": {
      "command": [
        "gdb",
        "-i",
        "dap"
      ],
      "redirectStderr": true,
      "redirectStdout": true,
      "supportsSourceRequest": true
    }
  },
  "configurations": {
    "launch (debug)": {
      "request": {
        "command": "launch",
        "mode": "debug",
        "program": "${file}",
        "args": "${args|list}",
        "cwd": "${workdir}"
      }
    }
  }
}
```

Each of the entries in `configurations` is combined with the `run` data and forms a “profile”. This specifies the DAP server to launch along with its arguments, where the latter are specific to the profile (`commandArgs`). The other parts specify the DAP protocol request (`launch` or `attach`), along with DAP specific extensions.

Of course, the specified server should be installed (and typically also in `PATH` for proper execution).

Various stages of override/merge are applied; user configuration (loaded from file) overrides (internal) default configuration, and the “dap” entry in `.kateproject` project configuration in turn overrides.

#### 4.13.6.1 Execution environment setup

More background and specifics can be found in the [LSP Client Execution environment](#) section below. But suffice it to say here it may be needed to run the debuggee process (and the DAP server) in a “special environment”, whether merely defined by environment variables or some container (providing the required dependencies and circumstances for proper execution).

Similar to the example in the referenced section, the following configuration may be provided in a `.kateproject`.

```
{
  // this may also be an array of objects
  "exec": {
    "hostname": "foobar"
    // the command could also be an array of string
    "prefix": "podman exec -i foobarcontainer",
    "mapRemoteRoot": true,
    "pathMappings": [
      // either of the following forms are possible
      // a more automagic alternative exists as well, see referenced ↔
      // section
      [ "/dir/on/host", "/mounted/in/container" ]
      { "localRoot": "/local/dir", "remoteRoot": "/remote/dir" }
    ]
  },
  "dap": {
    "debuggy": {
      "run": {
        // in this section, it applies to all configurations
        // this will match/join with the above object
        "exec": { "hostname": "foobar" },
        // if server is connected to,
        // optionally specify explicit port (which is suitable ↔
        // published/forwarded)
        "port": 5678,
        // the server may then also have to accept more than ↔
        // localhost
        "host": "0.0.0.0"
      }
    }
  }
}
```

The referenced section should be consulted for details, but in essence the `prefix` will be prepended before the DAP server command line specified elsewhere. The effect is that the server is run within the specified container and then in turn also the launched process. The `pathMapping` arranges for transformation of filepaths between editor’s view and DAP server (container) view, e.g. when dealing with setting of breakpoints or handling of reported backtraces. Note that such mapping is optional and may or may not be useful. When dealing with C/C++ code that is compiled on “host”, the symbol info references source files on host which do not exist at all in the other environment. However, in other scripted (e.g. python) circumstances, actual runtime files are referenced (on the other environment).

The following must evidently be kept in mind.

- The DAP server must be present in the environment/container, which must be configured to support proper debugger operation (so, if needed, privileged, capabilities).
- Communication between editor and DAP must be possible. In case of a container, the latter should either use host networking, or provide a suitable mapped/published port along with

corresponding config snippet as in above example (as an auto-selected one in case of port 0 would not make it through).

- A specified executable/PID should be in “container” perspective, as well as any (debuggee executable) arguments.

Also, as in the LSP case, some environment variables are set; `KATE_EXEC_PLUGIN` is set to `dap`, `KATE_EXEC_SERVER` is set to the debugger/language type (e.g. `python`) and `KATE_EXEC_PROFILE` is set to the configuration entry (e.g. `launch`).

## 4.14 Project Plugin

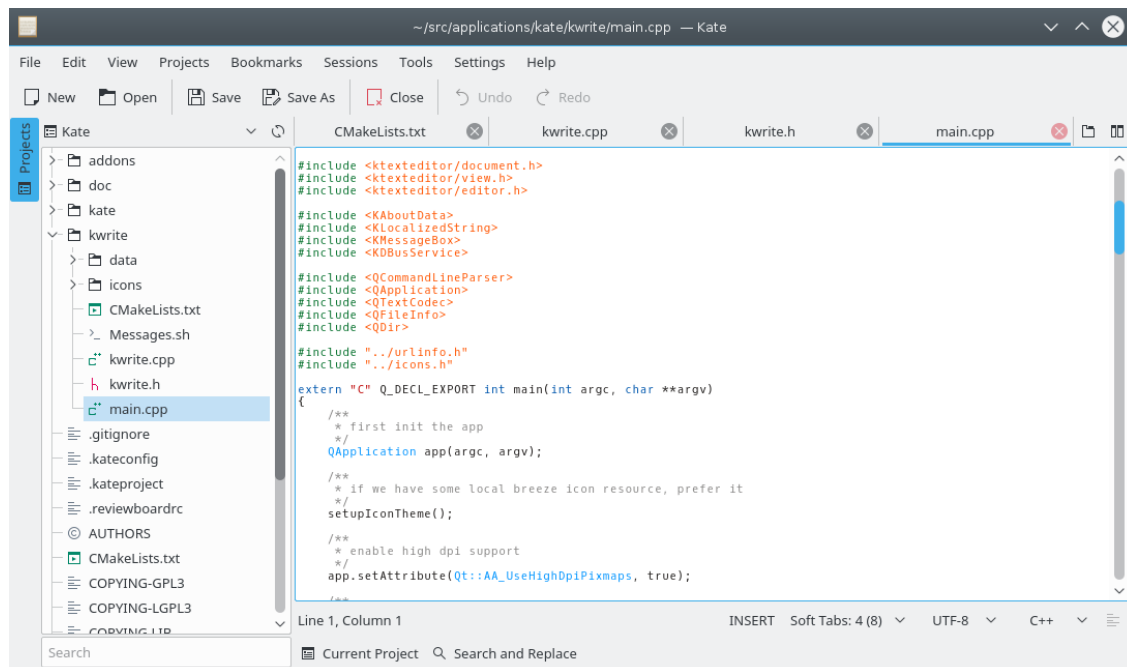
### 4.14.1 Introduction

The basic idea of the Project plugin is to have a structured list of files belonging to the project with the following properties:

1. Provide a structured view of the files
2. Make it easy and very fast to open and switch projects
3. Support search and replace for a project
4. Provide simple auto completion
5. Make it simple to quickly open files in the project
6. Support for building the project

### 4.14.2 Structured View of the Files

Once the Project plugin is loaded in the Kate configuration page, open a file in a project and a sidebar appears that lists all projects as well as the project files as follows:



As you can see, the currently active project is 'Kate', and its contents is listed in the tree view. Clicking on files in the tree view opens the file in the editor. Further, a context menu is provided with which you can open files with other applications, such as a .ui file with Qt Designer.

You can filter the items by typing parts of the file name you are looking for into the search bar at the bottom of the list.

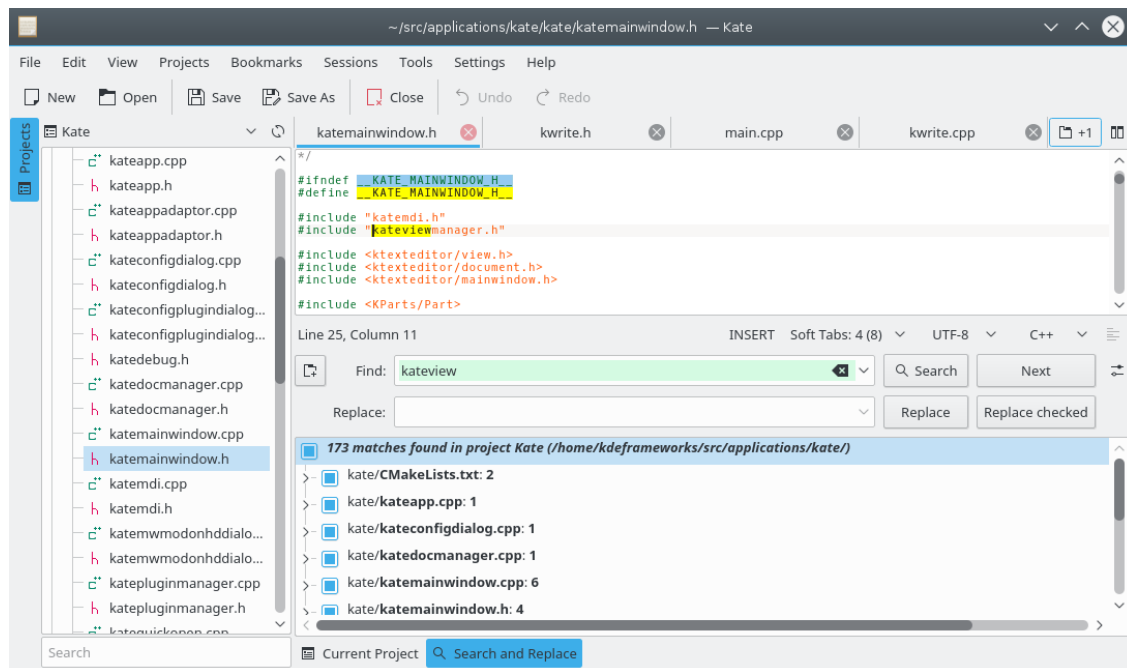
### 4.14.3 Switching Projects

The idea is that you never have to open a project manually, this is even not supported at all. Hence, what happens if you open a file, the Project plugin quickly scans the folder and its parent folders for a .kateproject file. If found, the project is automatically loaded.

Furthermore, if you open another document in Kate, that belongs to another project, the Project plugin automatically switches the current project. So intuitively, always the correct project is active. Of course, you can also switch the currently active project using the combo box.

### 4.14.4 Search and Replace in Projects

Kate has a Search and Replace plugin that shows up in the bottom sidebar. If a project is loaded, open the Search and Replace sidebar, and switch to the mode to search and replace in the current project:

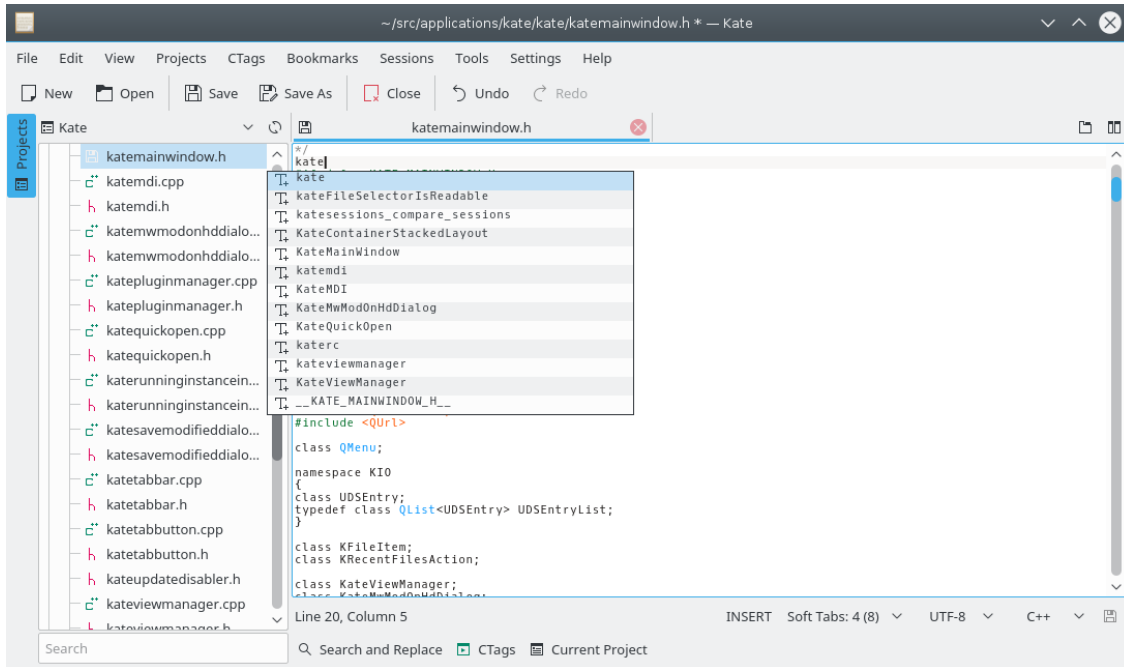


### 4.14.5 Simple Auto Completion

With the knowledge of all files belonging to a project, the Project plugin provides simple auto completion facilities based on CTags. If a project is initially opened, CTags parses all project files in a background thread and saves the CTags information to /tmp. This file then is used to populate the auto completion popup in Kate.

In contrast, without this auto completion, Kate is only capable of showing auto completion items based on the words in the current file. So the auto completion provided by the Project plugin is much more powerful.

## The Kate Handbook



If CTags is missing, a passive popup warns you about this issue. It is also noteworthy, that the CTags file in `/tmp` is cleaned up when Kate exits, so the plugin does not pollute any folder with unwanted files.

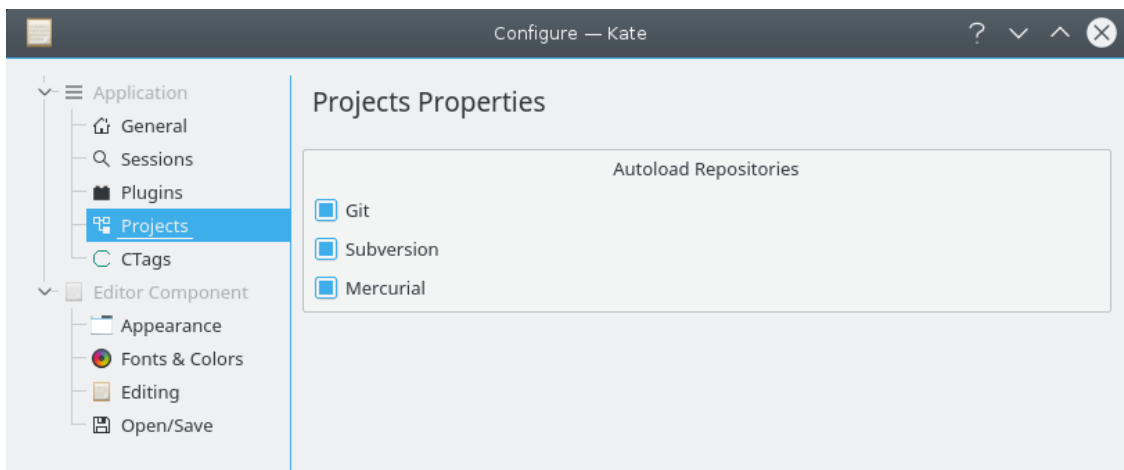
### 4.14.6 Support for Building the Project

Another feature is to have support for the [Build Plugin](#), so that it automatically is configured correctly.

### 4.14.7 Creating Projects

#### 4.14.7.1 Loading Projects Automatically

The Project plugin has an auto-loading feature. You can read the file list from the version control system. To this end, auto-loading for the respective version control system needs to be enabled in the settings (enabled by default):



#### 4.14.7.2 Creating Projects Manually

You just have to create a `.kateproject` file in the root folder of the project. For instance, the 'Kate' `.kateproject` file looks like this:

```
{
  "name": "Kate",
  "files": [
    {
      "git": 1
    }
  ]
}
```

The file content is written in JSON syntax. The project name is 'Kate', and the files contained in should be read from Git.

Also supported instead of `git` is subversion through `svn` and mercurial through `hg`. If you do not want to read files from a version control system, you can just invoke `kate` from command line as:

```
kate /path/to/folder
```

or you can tell it to recursively load files from directories as follows:

```
{
  "name": "Kate",
  "files": [
    {
      "directory": "kate",
      "filters": [
        "*.cpp",
        "*.h",
        "*.ui",
        "CMakeLists.txt",
        "Find*.cmake"
      ],
      "recursive": 1,
      "hidden": 1
    }
  ],
  "exclude_patterns" : [
    "^build/.*"
  ]
}
```

Here, subfolders and filters define what's part of the project. You can also mix version control and files based on filters. Hidden files will not be retrieved if the option

"hidden" is 0.

"exclude\_patterns" is a list of regex patterns that can be used to exclude folders and files from the project tree. In this example, All files and folders in a directory `build` from the root will be excluded.

If you want to add support for the building, you can write a `.kateproject` like this:

```
{
  "name": "Kate",
  "files": [
    {
      "git": 1
```

```

    }
  ],
  "build": {
    "directory": "build",
    "build": "make all",
    "clean": "make clean",
    "install": "make install",
    "targets": [
      {
        "name": "all",
        "build_cmd": "ninja"
        "run_cmd": "./bin/kate"
      },
      {
        "name": "kate",
        "build_cmd": "ninja kate-bin"
      }
    ]
  }
}

```

The targets specified above will then appear in the [Build Plugin](#) under **“Project Plugin Targets”**. If the

“targets” array is specified then

“build” ,

“clean” and

“install” are ignored. Each element in the array specifies a target.

“name” is the name of the target,

“build\_cmd” will be used to build the target,

“run\_cmd” will be used to run the target. Most important of all is

“directory” , this is where the commands will be executed.

In case you have a `.kateproject` file tracked by a control version system, but you need to tweak the configuration for a particular workspace, you can save those changes to a separated file named `.kateproject.local`. The content of this file will take precedence over `.kateproject`.

#### 4.14.8 Current Project

Using **Projects** → **Go To (Alt+1)** you can open the **Current Project** view at the bottom of the editor window with four tabs:

Line 4, Column 1      INSERT   Soft Tabs: 4   UTF-8   XML

Terminal   Code Index   Code Analysis   Notes

Analysis finished.

File	Line	Severity	Message
btfileindexer.h	32	style	Class 'BtFileIndexer' has a constructor with 1 argument that is not ex...
katebacktracebrowser.h	144	style	Class 'KateBtConfigDialog' has a constructor with 1 argument that is ...
close_except_plugin.h	122	style	Class 'CloseExceptPlugin' has a constructor with 1 argument that is n...
katefilebrowserconfig.cpp	123	style	The scope of the variable 'aitem' can be reduced.
katefilebrowserconfig.cpp	160	style	The scope of the variable 'lb' can be reduced.

Start Analysis...

### Terminal Panel

A [Terminal emulator](#) starting in the root folder of the project.

### Code Index

Entering characters into the search bar will start the search and display matching names of functions, classes, symbols etc. together with kind, filename and line number.

Select an item in the list to jump to the corresponding line in the source file.

### Code Analysis

Click **Start Analysis** to run a static code analysis for the C and C++ using **cppcheck** and to generate a report showing filename, line number, severity (style, warning etc.) and the issue found.

Select an item in the list to jump to the corresponding line in the source file.

### Notes

Text entered in this tab will be saved in the file `.kateproject.notes`.

## 4.14.9 The Projects Menu

The **Projects** menu allows you to switch between currently open projects. It is displayed by the Project plugin.

### Projects → Back (Ctrl+Alt+Left)

Switch to the previous project.

### Projects → Forward (Ctrl+Alt+Right)

Switch to the next project.

### Projects → Go To (Alt+1)

Open the **Current Project** view at the bottom of the editor window.

## 4.15 LSP Client Plugin

The LSP Client plugin provides many language features such as code completion, code navigation or finding references based on the [Language Server Protocol](#).

Once you have enabled the LSP Client in the plugin page, a new page called LSP Client will appear in your Kate configuration dialog.

### 4.15.1 Menu Structure

If appropriate, a corresponding LSP command is also mentioned in the explanation below, the documentation of which may then provide additional background and interpretation, though it may vary depending on the actual language. The phrase 'current symbol' refers to the symbol corresponding to the current cursor position, as so determined by the language and server implementation.

#### LSP Client → Go to Definition

[textDocument/definition] Go to current symbol definition.

#### LSP Client → Go to Declaration

[textDocument/declaration] Go to current symbol declaration.

**LSP Client → Go to Type Definition**

[textDocument/typeDefinition] Go to current symbol type definition.

**LSP Client → Find References**

[textDocument/references] Find references to current symbol.

**LSP Client → Find Implementations**

[textDocument/implementation] Find implementations of current symbol.

**LSP Client → Highlight**

[textDocument/documentHighlight] Highlight current symbol references in current document.

**LSP Client → Hover**

[textDocument/hover] Hover info for current symbol.

**LSP Client → Format**

[textDocument/formatting] [textDocument/rangeFormatting] Format the current document or current selection.

**LSP Client → Rename**

[textDocument/rename] Rename current symbol.

**LSP Client → Quick Fix**

[textDocument/codeAction, workspace/executeCommand] Computes and applies a quick fix for a diagnostic on current position (or line).

**LSP Client → Show selected completion documentation**

Show documentation for a selected item in the completion list.

**LSP Client → Enable signature help with auto completion**

Also show signature help in the completion list.

**LSP Client → Include declaration in references**

Request to include a symbol's declaration when requesting references.

**LSP Client → Add parentheses upon function completion**

Automatically add a pair of parentheses after completion of a function.

**LSP Client → Show hover information**

Show hover information upon (mouse cursor) hover. Regardless of this setting, the request can always be manually initiated.

**LSP Client → Format on typing**

[document/onTypeFormatting] Format parts of document when typing certain trigger characters. For example, this might apply indentation upon newline, or as otherwise determined by LSP Server. Note that editor indentation scripts might be trying to do the same (depending on the mode) and so it may not be advisable to have both enabled at the same time.

**LSP Client → Incremental document synchronization**

Send partial document edits to update the server rather than whole document text (if supported).

**LSP Client → Highlight goto location**

Provide a transient visual cue after performing a goto to a location (of definition, declaration, etc).

**LSP Client → Show diagnostics notifications**

[textDocument/publishDiagnostics] Process and show diagnostics notifications sent by server.

**LSP Client → Show diagnostics highlights**

Add text highlights for ranges indicated in diagnostics.

**LSP Client → Show diagnostics marks**

Add document marks for lines indicated in diagnostics.

**LSP Client → Switch to diagnostic tab**

Switch to the diagnostic tab in the plugin toolview.

**LSP Client → Close all non-diagnostics tabs**

Close all non-diagnostics (e.g. references) tabs in plugin toolview.

**LSP Client → Restart LSP Server**

Restart current document's LSP Server.

**LSP Client → Restart all LSP Servers**

Stop all LSP Servers which will then be (re)started as needed.

## 4.15.2 Goto Symbol support

LSP Client can help you jump to any symbol in your project or current file. To jump to any symbol in the file, use the toolview "LSP Client Symbol Outline" on the right border of kate. This toolview lists all symbols found by the server in current document.

### 4.15.2.1 Configuring LSP Client Symbol Outline

By default the symbols are sorted by their occurrence in the document but you can change the sort to be alphabetical. To do so, right click in the toolview and check "Sort Alphabetically".

The toolview shows the symbols in tree mode by default, however you can change it to a list using the context menu.

### 4.15.2.2 Global Goto symbol support

To jump to any symbol in your project, you can open the goto symbol dialog using **Ctrl+Alt+p**. The dialog is empty when it opens but as soon as you type something the dialog will start showing you matching symbols. The quality of matches as well as filtering capabilities depend upon the server that you use. For example, clangd supports fuzzy filtering but some other server may not.

## 4.15.3 Other Features

Clangd switch source header command is supported. To switch source header in a C or C++ project either use the "Switch Source Header" option from the context menu or the shortcut **F12**.

You can jump to a symbol quickly by putting your mouse over the symbol and then pressing **Ctrl** + left mouse button.

#### 4.15.4 Configuration

The plugin's configuration page mostly allows for persistent configuration of some of the above menu items. However, there is one additional entry to specify the Server Configuration file. This is a JSON file that can be used to specify the LSP server to start (and then to communicate with over stdin/stdout). For convenience, some default configuration is included, which can be inspected in the plugin's configuration page. To aid in the explanation below, an excerpt of that configuration is given here:

```
{
  "servers": {
    "bibtex": {
      "use": "latex",
      "highlightingModeRegex": "^BibTeX$"
    },
    "c": {
      "command": ["clangd", "-log=error", "--background-index"],
      "commandDebug": ["clangd", "-log=verbose", "--background-index ↵
        ↵"],
      "url": "https://clang.llvm.org/extra/clangd/",
      "highlightingModeRegex": "^(C|ANSI C89|Objective-C)$"
    },
    "cpp": {
      "use": "c",
      "highlightingModeRegex": "^(C\\+\\+|ISO C\\+\\+|Objective-C ↵
        ↵\\+\\+)$"
    },
    "d": {
      "command": ["dls", "--stdio"],
      "url": "https://github.com/d-language-server/dls",
      "highlightingModeRegex": "^D$"
    },
    "fortran": {
      "command": ["fortls"],
      "rootIndicationFileNames": [".fortls"],
      "url": "https://github.com/hansec/fortran-language-server",
      "highlightingModeRegex": "^Fortran.*$"
    },
    "javascript": {
      "command": ["typescript-language-server", "--stdio"],
      "rootIndicationFileNames": ["package.json", "package-lock.json ↵
        ↵"],
      "url": "https://github.com/theia-ide/typescript-language-server ↵
        ↵",
      "highlightingModeRegex": "^JavaScript.*$",
      "documentLanguageId": false
    },
    "latex": {
      "command": ["texlab"],
      "url": "https://texlab.netlify.com/",
      "highlightingModeRegex": "^LaTeX$"
    },
    "go": {
      "command": ["go-langserver"],
      "commandDebug": ["go-langserver", "-trace"],
      "url": "https://github.com/sourcegraph/go-langserver",
      "highlightingModeRegex": "^Go$"
    },
    "python": {
```

## The Kate Handbook

```
    "command": ["python3", "-m", "pyls", "--check-parent-process"],
    "url": "https://github.com/palantir/python-language-server",
    "highlightingModeRegex": "^Python$"
  },
  "rust": {
    "command": ["rls"],
    "path": ["%{ENV:HOME}/.cargo/bin", "%{ENV:USERPROFILE}/.cargo/ ↵
      bin"],
    "rootIndicationFileNames": ["Cargo.lock", "Cargo.toml"],
    "url": "https://github.com/rust-lang/rls",
    "highlightingModeRegex": "^Rust$"
  },
  "ocaml": {
    "command": ["ocamlmerlin-lsp"],
    "url": "https://github.com/ocaml/merlin",
    "highlightingModeRegex": "^Objective Caml.*$"
  }
}
```

Note that each “command” may be an array or a string (in which case it is split into an array). Also, a top-level “global” entry (next to “server”) is considered as well (see further below). The specified binary is searched for in the usual way, e.g. using `PATH`. If it is installed in some custom location, then the latter may have to be extended. Or alternatively, a (sym)link or wrapper script may be used in a location that is within the usual `PATH`. As illustrated above, one may also specify a “path” that will be searched for after the standard locations.

All of the entries in “command”, “root” and “path” are subject to variable expansion.

The “highlightingModeRegex” is used to map the highlighting mode as used by Kate to the language id of the server. If no regular expression is given, the language id itself is used. If a “documentLanguageId” entry is set to false, then no language id is provided to the server when opening the document. This may have better results for some servers that are more precise in determining the document type than doing so based on a kate mode.

From the above example, the gist is presumably clear. In addition, each server entry object may also have an “initializationOptions” entry, which is passed along to the server as part of the ‘initialize’ method. If present, a “settings” entry is passed to the server by means of the ‘workspace/didChangeConfiguration’ notification. Either of “completionTriggerCharacters” or “signatureTriggerCharacters” may be specified as a JSON object with string members “exclude” and/or “include”. These will be used to respectively exclude or add some characters to the respective trigger set as provided by the server.

Various stages of override/merge are applied;

- user configuration (loaded from file) overrides (internal) default configuration
- “lspclient” entry in `.kateproject` project configuration overrides the above
- the resulting “global” entry is used to supplement (not override) any server entry

One server instance is used per (root, servertype) combination. If “root” is specified as an absolute path, then it is used as-is, otherwise it is relative to the ‘projectBase’ (as determined by the [Project plugin](#)) if applicable, or otherwise relative to the document’s directory. If not specified and “rootIndicationFileNames” is an array of filenames, then a parent directory of current document containing such a file is selected. Alternatively, if “root” is not specified and “rootIndicationFilePatterns” is an array of file patterns, then a parent directory of the current document matching the file pattern is selected. As a last fallback, the home directory is selected as “root”. For any document, the resulting “root” then determines whether or not a separate instance is needed. If so, the “root” is passed as `rootUri/rootPath`.

In general, it is recommended to leave root unspecified, as it is not that important for a server (your mileage may vary though). Fewer server instances are obviously more efficient, and they also have a 'wider' view than the view of many separate instances.

As mentioned above, several entries are subject to variable expansion. A suitable application of that combined with "wrapper script" approaches allows for customization to a great many circumstances. For example, consider a python development scenario that consists of multiple projects (e.g. git repos), each with its own virtualenv setup. Using the default configuration, the python language server will not be aware of the virtual env. However, that can be remedied with the following approach. First, the following fragment can be entered in LSPClient plugin's "User Server Settings":

```
{
  "servers":
  {
    "python":
    {
      "command": ["pylsp_in_env", "%{Project:NativePath ↵
        }"],
      "root": "."
    }
  }
}
```

The root entry above is relative to the project directory and ensures that a separate language server is started for each project, which is necessary in this case as each has a distinct virtual environment.

pylsp\_in\_env is a small "wrapper script" that should be placed in PATH with the following (to-be-adjusted) content:

```
#!/bin/bash
cd $1
# run the server (python-lsp-server) within the virtualenv
# (i.e. with virtualenv variables setup)
# so source the virtualenv
source XYZ
# server mileage or arguments may vary
exec myserver
```

This is but one example of a more general pattern which may be handled a bit more comfortably as outlined in the [Execution environment](#) section below.

#### 4.15.4.1 LSP Server Configuration

Each particular LSP server has its own way of customization and may use language/tool specific means for configuration, e.g. tox.ini (a.o. for python), .clang-format for C++ style format. Such configuration may then also be used by other (non-LSP) tools (such as then tox or clang-format). On top of that, some LSP servers also load configuration from custom files (e.g. .ccls). Furthermore, custom server configuration can also be passed through LSP (protocol), see the aforementioned "initializationOptions" and "settings" entries in server configuration.

Since various level of override/merge are applied, the following example of user specified client configuration tweaks some python-language-server configuration.

```
{
  "servers": {
    "python": {
      "settings": {
```

```
        "pyls": {
          "plugins": {
            "pylint": {
              "enable": true
            }
          }
        }
      }
    }
  }
}
```

Unfortunately, LSP server configuration/customization is often not so well documented, in ways that only examining the source code shows configuration approaches and the set of available configuration options. In particular, the above example's server supports many more options in "settings". See [another LSP client's documentation](#) for various other language server examples and corresponding settings, which can easily and readily be transformed to the JSON configuration that is used here and outlined above.

#### 4.15.4.2 LSP Server Format On Save

You can enable "format on save" from LSP settings in configure dialog.

#### 4.15.4.3 LSP Server Diagnostic Suppression

It may happen that diagnostics are reported which are not quite useful. This can be quite cumbersome, especially if there are many (often of the same kind). In some cases, this may be tweaked by language (server) specific means. For example, the [clangd configuration mechanism](#) allows tweaking of some diagnostics aspects. In general, however, it may not always be evident how to do so, or it may not even be possible at all in desired ways due to server limitations or bug.

As such, the plugin supports diagnostics suppression similar to e.g. valgrind suppressions. The most fine-grained configuration can be supplied in a "suppressions" key in the (merged) JSON configuration.

```
{
  "servers": {
    "c": {
      "suppressions": {
        "rulename": ["filename", "foo"],
        "clang_pointer": ["", "clang-tidy", "clear_pointer"],
      }
    }
  }
}
```

Each (valid) rule has an arbitrary name and is defined by an array of length 2 or 3 which provides a regex to match against the (full) filename, a regex to match against the diagnostic (text) and an optional regex matched against the (source code range of) text to which the diagnostic applies.

In addition to the above fine-grained configuration, the context menu in the diagnostics tab also supports add/remove of suppressions that match a particular diagnostic (text) exactly, either globally (any file) or locally (the specific file in question). These suppression are stored in and loaded from session config.

#### 4.15.4.4 LSP Server Troubleshooting

It is one thing to describe how to configure a (custom) LSP server for any particular language, it is another to end up with the server running smoothly. Usually, the latter is fortunately the case. Sometimes, however, problems may arise due to either some “silly” misconfiguration or a more fundamental problem with the server itself. The latter might typically manifest itself as a couple of attempts at starting the server, as so reported in Kate Output tab. The latter, however, is only meant to convey high-level messages or progress rather than to provide detailed diagnostics, and even less so for what is in fact another process (the LSP server).

The usual way to diagnose this is to add some flag(s) to the startup command (of the language server) that enables (additional) logging (to some file or standard error), in as far as it does not do so by default. If Kate is then started on the command line, then one might be able to obtain more (in)sight in what might be going wrong.

It may also be informative to examine the protocol exchange between Kate’s LSP client and the LSP server. Again, the latter usually has ways to trace that. The LSP client also provides additional debug tracing (to stderr) when Kate is invoked with the following `LSPCLIENT_DEBUG=1` suitably export’ed.

#### 4.15.4.5 Execution environment setup

The python virtualenv example above is but one example of an “execution environment” that operates in a distinct and separate way from the usual host environment. This could be achieved by different variable settings (e.g. virtualenv), or a (s)chroot setup (switching to another dir as new root), a container (e.g. podman, docker), or an ssh session to another host. In each case, the “other environment” is defined by an “execution prefix”. That is, some program can be invoked/run in the other environment by means of a “prefix” (a program and arguments) appended with the intended invocation. For example, `podman exec -i containername` or `ssh user@host`.

In particular, as in the previous virtualenv example, one may choose/need to run an LSP server in such a separate environment (e.g. a container with all dependencies needed by some project). The “manual” approach outlined above has as disadvantage that it replaces the standard LSP command-line, which then has to be specified and duplicated again in the wrapper script. Also, in some of the other examples mentioned above the “path namespace” of host (as viewed by the editor) may be different from that of the environment. To address these matters in a more systematic way (than a “custom” approach), some additional configuration can be specified.

For example, the following can be specified in a `.kateproject` configuration. Obviously, the “fake” comments should not be included.

```
{
  // this may also be an array of objects
  "exec": {
    "hostname": "foobar"
    // the command could also be an array of string
    "prefix": "podman exec -i foobarcontainer",
    "mapRemoteRoot": true,
    "pathMappings": [
      // either of the following forms are possible
      // a more automagic alternative exists as well, see later/below
      [ "/dir/on/host", "/mounted/in/container" ]
      { "localRoot": "/local/dir", "remoteRoot": "/remote/dir" }
    ]
  },
  "lspclient": {
    "servers": {
      "python": {
        // this will match/join with the above object

```

## The Kate Handbook

```
    "exec": { "hostname": "foobar" },
    // confine this server to this project root,
    // so it is not used for other projects that may be opened
    // (other servers may already employ specific roots, but ↔
    // python generally not)
    "root": "."
  },
  "c": {
    // as above
    "exec": { "hostname": "foobar" },
    "root": "."
  }
}
}
```

So, what happens as a result of the above? As mentioned, the `lspclient` part of the above is merged onto the global config, hence an `exec` section is found (for specified languages). A search is performed for another object (that specifies matching `hostname`) in either `exec` or `lspclient` section, and a matching one is as a basis for a merge. As a result, an LSP server (for C and python) will have its commandline appended to the specified (variable substituted) prefix, and will therefore be started within the given container. Of course, the container must have been created, in a proper started state and equipped with proper LSP servers. One might have been tempted to use the `global` section (within `lspclient`.) That could also work, but then *all* LSP servers would be started with that prefix, including those for e.g. Markdown, Bash script or JSON. It is more likely that the usual host will still supply these (if in use). So, as always, it depends on your particular setup.

However, the LSP server may now observe different (remote) paths than the (local) ones that are seen and used by the editor. The specified `pathMappings` are used to translate back and forth between either within the communication with the LSP server. That is, of course, as much as possible. Clearly, not all local paths have a remote representation, but the missing ones are also not seen (by the server) and do not pose a problem. Conversely, however, the (remote) server may now see and provide references in the “remote root” which are not evidently/easily represented in the local system. The enabled `mapRemoteRoot` implicitly maps the remote root onto a “local URL” `exec://foobar/`, which is then handled by a plain-and-simple KIO protocol. The latter essentially uses (e.g.) `podman exec -i foobarcontainer cat somefile` for copy from remote to local (and other such variations using tools from the `coreutils` suite). Suffice it to say it is not meant for general use and claims no performance whatsoever, but it does suffice to get a referenced file quickly and easily loaded into editor.

It is now easily seen that a simplified version of above configuration (without `hostname` or `pathMappings`) could be used to handle the `virtualenv` without having to duplicate server commandline (in wrapper script).

The following may not be so easily seen, so it is mentioned here explicitly.

- Both a `hostname` and the actual `prefix` define the execution environment (the former by name, the latter by content). Within an editor process instance, the same `hostname` should not be associated with different `prefix`, as such leads to undefined behavior (with no diagnostic required).
- Both `prefix` and `pathMappings` are subject to (editor) variable expansion (but do mind the foregoing item).
- At runtime, some environment variables are also set, which may be used to (subtly) adjust the “prefix launcher”’s behavior (though again mind the first item). In particular, `KATE_EXEC_PLUGIN` is set to `lspclient` and `KATE_EXEC_SERVER` is set to the server’s id (e.g. `python`).

- In particular, also `KATE_EXEC_INSPECT` is set to 1. This notifies the “prefix launcher” that the receiver (Kate plugin) accepts some out-of-band/protocol data. This allows the launcher to use some means to determine path mappings (e.g. `podman inspect`) and to communicate this in suitable format. This will then be removed from the otherwise LSP protocol conforming stream and used to extend any defined path mapping. This may serve as an alternative to specifying e.g. `bind` mounts explicitly (again, duplicating the container’s definition). Concretely, the above snippet could then be used instead;

```
{
  // ...
  "exec": {
    "hostname": "foobar"
    // unfortunate repetition of name, but the helper script is plain ↔
    -and-simple
    "prefix": "exec_inspect.sh foobarcontainer podman exec -i ↔
      foobarcontainer",
    "mapRemoteRoot": true,
    "pathMappings": []
  }
  // ...
}
```

Last but not least, what if the “fallback KIO” approach is not considered adequate? It is in practice often possible to “mount” the remote root into the local filesystem and then specify the latter in a `pathMappings`. For starters, the well-known `sshfs` (fuse) filesystem can mount a (really) remote system into the local one. For a container (podman/docker), most (filesystem driver) setups support the following trick;

```
$ rootdir=/proc/`podman inspect --format '{{.State.Pid}}` containername '/' ↔
  root
# some symlinks may cause issues, see also alternative below
$ sudo mount --bind $rootdir /somewhere/containername
```

If that fails, then one could resort to `sshfs`, or in fact a subset thereof to mount the remote/container root;

```
# see respective man-pages; sftp-server suffices for actual (unencrypted) ↔
  file ops protocol
$ socat 'exec:podman exec -i containername /usr/lib/openssh/sftp-server'
  'exec:sshfs -o transform_symlinks -o passive \:/ /somewhere/ ↔
  containername'
# ...
$ fusermount -u /somewhere/containername
```

Note that such mount of remote root into host filesystem is likely to be specified manually and explicitly in a `pathMappings` section (barring a very intelligent prefix launcher that arranges all such automatically).

## 4.16 Search & Replace

### 4.16.1 Introduction

Kate’s Search & Replace plugin allows you to search for text or [regular expressions](#) in many different files at once. You can search all open files, all the files in one directory and optionally its subdirectories, or in the active file. You can even filter by filename, for instance searching only files that end with a particular file extension.

## 4.16.2 Interface

### 4.16.2.1 Search Query

The following options are always displayed at the top of the Search in Files tool view:



You can have as many searches as you want open at the same time. Simply click the new tab button at the top-left corner of the Search tool view and a new results tab will open permitting you to perform another search.



The button in the top right-corner of the Search in Files tool view will toggle the bottom half of the tool view between displaying additional options for the Search in Folder mode and the results of your search.

#### Find

This is where you type in what you want to find. You may enter standard text, or a regular expression if enabled.

#### Replace (text box)

Replacement text that will be added to file(s) in place of the text in the **Find** text box.

#### Search

When you've finished configuring everything, just press the **Search** button to perform your search; to perform the search in a new tab press and hold **Ctrl** before pressing the **Search** button. You may also press **Enter** in the **Find** text box to do the same (similarly pressing **Ctrl+Enter** will perform the search in a new tab).

#### Replace

When you've finished configuring everything, just press the **Replace** button to replace the text entered in the **Find** text box with that of the **Replace** text box. You may also press **Enter** in the **Replace** text box to do the same.


#### Next

Go to the next match of your search query, switching files if necessary.

#### Replace Checked

The same as **Replace**, but will only perform replacements in files that are checked in the pane below.

### 4.16.2.2 Search in Folder Options

These options are displayed below the aforementioned query options. If search results are being displayed instead, press the  button to display them.

#### Search in

This has three options. Select **Open Files** to search all files currently open in Kate. Select **Folder** to search inside a folder and optionally its subfolders. Select **Current File** to search only in the active file.

If the **Projects** plugin is loaded, you can also search in the **Current Project** or in **All Open Projects**.

### Match case

Restricts search results to only those that have the exact same combination of upper and lower case letters as your search query.

### Regular expressions

Permits you to use [regular expressions](#) instead of simple text as your search query.

### Expand results

Display all the results found in each file, instead of just a list of files that contain the search query.

### Folder

You may enter the path of the folder you wish to search. For instance, you might enter `~/development/kde/kate/` if you wished to search the Kate source code. This option is only available when using **in Folder** mode.

### Open file dialog

Press this button to locate the folder in your desktop's folder browser. This button only works when using **Folder** mode.



Press this button to change **Folder** to the parent of the currently selected folder. This button only works when using **Folder** mode.



This button will set the **Folder** entry to the folder in which the currently open document is located. This button only works when using **Folder** mode.

### Filter

This permits you to only search filenames that match a particular pattern. For instance, to only search files written in C++, change it to `*.cpp`. To search only files beginning with `kate`, change it to `kate*`. You can enter multiple filters separated with a comma (,). This option is not available when using **Open files** mode.

### Exclude

The opposite of **Filter**, this prevents Kate from searching files that match the specified patterns. As with **Filter**, you can enter multiple patterns separated with a comma (,). This option is not available when using **Open files** mode.

### Recursive

If this option is enabled, Kate will also search in all subfolders of the selected folder. This option is only available when using **Folder** mode.

### Include hidden

If this option is enabled, Kate will also search in files or folders that are typically hidden by your operating system. This option is only available when using **Folder** mode.

### Follow symbolic links

The Search in Files plugin typically does not follow [symbolic links](#). When this option is enabled, the plugin will follow them instead and search inside the files or folders they reference. This option is only available when using **Folder** mode.


#### WARNING

It's possible for symbolic links to reference a folder that is the parent of the folder currently being searched, or other folders that contain symbolic links to their parent. If there is such a link in the folder being searched and this option is enabled, Kate will repeatedly follow the link and search the folder, and the search will never complete.

### Include binary files

If enabled, Kate will also search in files that do not appear to be text files.

#### 4.16.2.3 Search Results

The results of your search are displayed below the query options. If options for Search in Folder mode are displayed, simply press the  button to display them. They will also automatically be displayed as soon as a search is performed.

The search results display a list of files that contains text that matches your search query, followed by the number of matches found in that file.

To see a list of matches in that file, simply click the expansion arrow to the left of the file name. (If you selected the **Expand results** option, this will already be done for you.) The line number each match is found on will be displayed, followed by the contents of that line, with your search query indicated in bold text.

To open the file your result was found in, simply double-click it. Kate will open the file if needed. You can also move the cursor to the location of a particular match by double-clicking on its listing instead of the file name.

### 4.16.3 Menu Structure

#### Edit → Search in Files (Ctrl+Alt+F)

Launches the Search and Replace tool view.

#### Edit → Go to Next Match

Go to the next match in a search performed by the Search and Replace plugin.

#### Edit → Go to Previous Match

Go to the previous match in a search performed by the Search and Replace plugin.

#### View → Tool Views → Show Search and Replace

Toggle the display of Kate's **Search and Replace** tool.

## 4.17 Kate Scripts & Snippets

### 4.17.1 Introduction

The Kate Scripts & Snippets plugin allows you to add, and manage your own bits of automation. This can range from static re-useable pieces of plain text, over sophisticated templates with custom automated and scriptable fields, to fully custom JavaScript scripts to modify the contents of your document.

For the most part, these types of content are managed in exactly the same way. Therefore, for ease of reading, the following discussion does not differentiate between static snippets, scripted snippets, and scripts. Rather, the term 'snippet' is used to refer to all of them.

The plugin also supports assigning keyboard shortcuts, and inserting snippets via code completion. Further, you can download collections of snippets (so-called repositories) from the internet.

## 4.17.2 Menu Structure

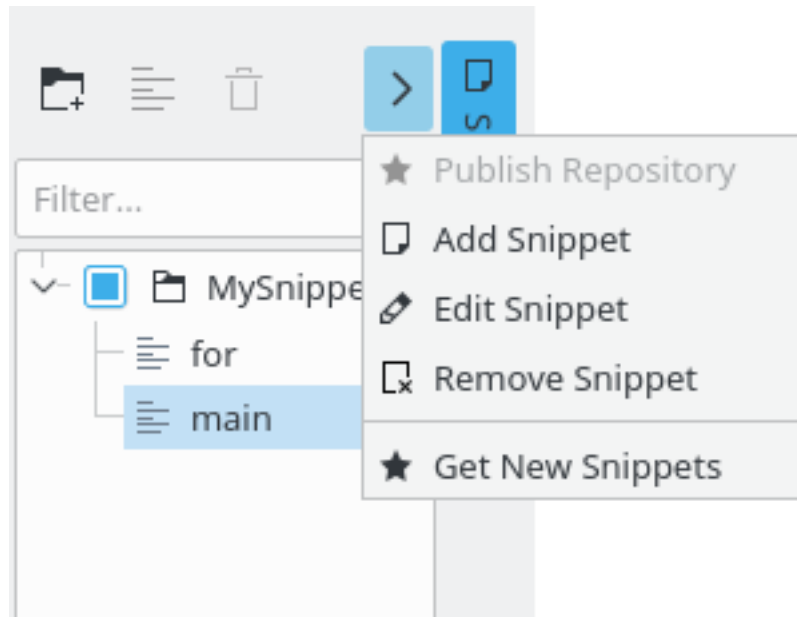
### View → Tool Views → Show Snippets

Shows snippets panel containing all snippets in your repository that are for the currently opened file type.

### Tools → Create Snippet

Create a new snippet, which is a reusable chunk of text you may insert in any part of any document, or a small script.

## 4.17.3 Snippets panel



*The panel for Kate Snippets.*

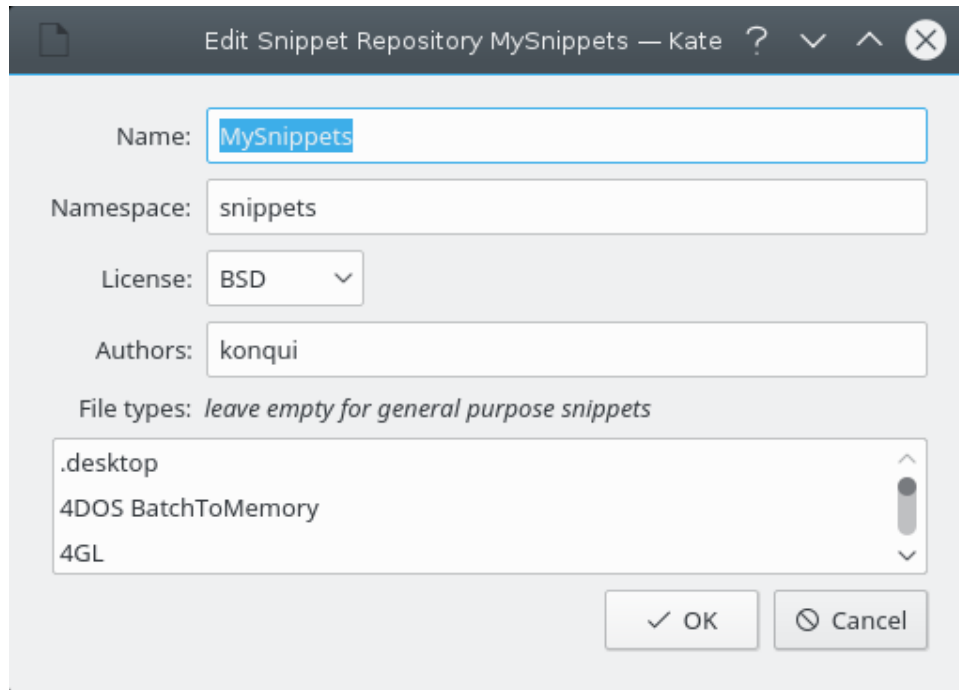
In the panel you should see a list of snippet repositories, along with options to create your own, get them from the Internet or load them from a local file. Each repository has a checkbox that can be used to activate or deactivate it. There are also buttons to edit and delete existing repositories.

### 4.17.3.1 Loading Snippet Repository Files

You can download snippet repositories from the Internet. Just click **Get New Snippets** and a window with a list of snippet repositories will open. After downloading the desired snippet, make sure that you have activated it.

### 4.17.3.2 Creating and Editing Repositories

To create a new snippet repository, click **Add Repository**. You should now see a dialog that asks for the name of the snippet file, license and author. After choosing the desired options, click **OK**.



*The repository editor interface.*

The snippet repository editor contains the following options:

**Name**

Appears in the list of snippets in the tool view and is also searched when using the code completion feature.

**Namespace**

Prefix used while using code completion.

**License**

Select the license for you snippet repository.

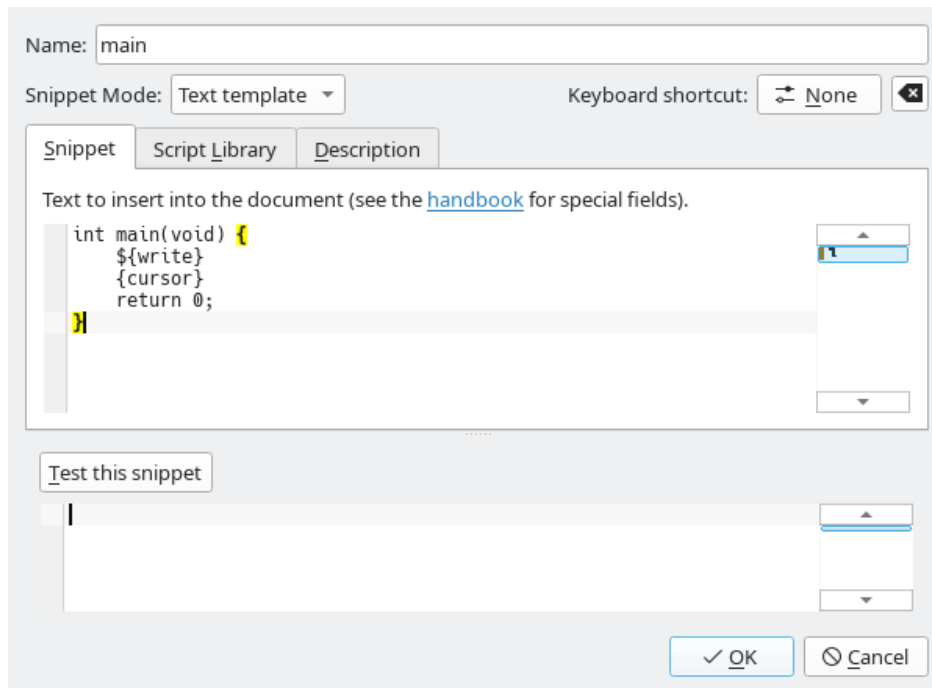
**Authors**

Enter the name(s) of the author(s) of the snippet file.

**File types**

Select the file type(s) you want the snippet repository to apply to. It is set to “ by default, so the repository applies to all files. You can change it to something like **C++** for instance, or select from a list by clicking on the items. You can specify more than one file type pressing the **Shift** while adding types.

### 4.17.3.3 Creating and Editing Snippets



*The snippet editor interface.*

#### Name

The name as shown in the list of snippets, and in the completion list (required).

#### Type

Snippets may either be defined as text templates or scripts. Both types allow to use JavaScript functions (see below for details), and can thus be used to similar effect. However, as a rule of thumb, text templates will be more suitable if you mostly want to *insert* text, while scripts are often an easier solution if you want to *modify* text).

#### NOTE

Kate needs to be compiled against KTextEditor version 6.15.0 or higher, for this feature to be available. For earlier versions, only text template snippets can be used.

#### Shortcut

Pressing this shortcut will insert (or run) the snippet into the current document.

#### Snippet (type **Text Template**)

The text your snippet will insert into the document.

A snippet can contain editable fields. They can be cycled by pressing **Tab**. The following expressions can be used in the template text to create fields:

**`\${field\_name}** creates a simple, editable field. All subsequent occurrences of the same *field\_name* create fields which mirror the contents of the first during editing.

#### TIP

Field names may contain any character except for the closing curly bracket (}).

To insert **`\${text}** literally in a snippet, escape the \$ sign with a backslash: **`\${text}**. The text will then not be turned into a field. To insert literal backslashes before a field, escape each individual backslash: **`\${field}**, **`\${field}**, and so on.

`${func(other_field1, other_field2, ...)}` creates a field which evaluates a JavaScript function on each edit and is replaced by that function's return value. See the [Script Library](#) tab for more information.

`${field_name=default}` sets a default value for the field. *default* is a JavaScript expression. Use quotes (`${field_name="text"}`) to specify a fixed string as the default value.

**TIP**

When using default values (`${field_name=default}`), keep in mind that the default value is evaluated immediately when the snippet is inserted into the document, and it is not updated later when fields are changed.

You can reference other fields in default values if they are defined before the default value that is being evaluated. However, this will only give you access to the default value of these fields.

If no custom default value is defined, the default value is a field's name.

`${cursor}` marks the end position of the cursor after everything else was filled in. Inserting text at this position will finish editing. Plus after pressing **Esc** to finish editing, the cursor will jump to this position.

### Snippet (type **Script**)

The JavaScript code to evaluate for this snippet

If this code contains a **return** statement, the returned string will be inserted at the current cursor position. You can also use the [Kate scripting API](#) to modify the document directly.

Additionally, your code may use functions defined in the [Script Library](#) tab, which are shared between all snippets in a repository.

### Script Library

JavaScript helper functions to use in your snippets. These functions are shared between all snippets in a repository.

You can use the [Kate scripting API](#) to get the selected text, full text, file name, and more by using the appropriate methods of the **document** and **view** objects. Refer to the scripting API documentation for more information.

When a function is called in a field of a **Text Template** snippet, its *return* value is inserted into the text.

#### NOTE

Using functions in **Text Template** snippets:

- Functions can access the up-to-date contents of all fields through the **fields** object: use **fields.my\_field** or **fields["my\_field"]**. When using a function as the default value for a field, only fields defined in the text before are available.
- Fields can be passed as arguments to functions. The field name can be used directly if it is a valid JavaScript identifier and no variable with that name exists. Otherwise use the **fields** object: **`\${func(field)}`**, **`\${func(fields.document)}`**, **`\${func(fields["my\_field"])}`**
- Function fields (e.g. **`\${func()}`**) are re-evaluated every time any field's contents are changed. However, this does not apply to function calls in default values: they are only evaluated once when the snippet is inserted into the document.
- For more complex scripts it may be important to understand that *first*, the raw snippet is inserted into the document, and *then* functions are being evaluated. E.g., if a function retrieves the text on the line where the snippet is being inserted, that text will also contain **`\${functionCall()}`**. Where this causes complications, consider using a **Script**-type snippet, instead.
- To simply wrap the currently selected text into tags, use: **<strong>`\${view.selectedText()}`</strong>**
- Remember that only the *return value* of a function is inserted in a field.

#### Description

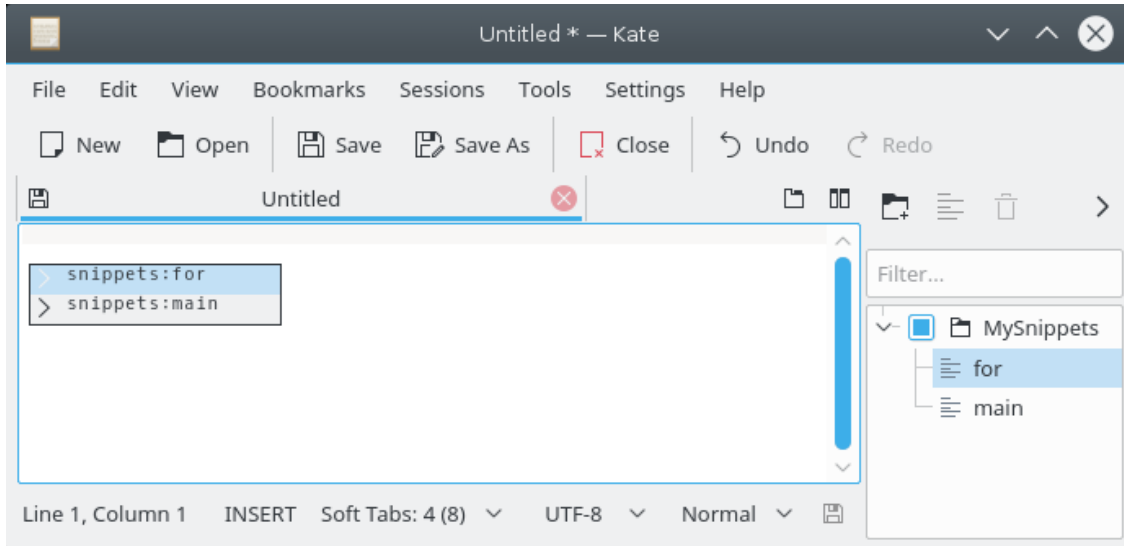
An optional description of what this snippet does. This will be shown in tooltips. The description may contain basic HTML formatting.

#### Usage example

The following example invokes a script that wraps the selected text, or - if there is no selection - a default text, into tags (Snippet type **Script**):

```
let range = view.hasSelection() ? view.selection() : new Range(view. ←
    cursorPosition(), view.cursorPosition());
let innertext = range.isEmpty() ? "Bold" : document.text(range);
document.removeText(range);
document.insertText(range.start, "<strong>" + innertext + "</strong>");
```

## 4.17.4 Using Snippets



*Selecting from a list of snippets.*

You can call snippets in several ways:

- By clicking on the snippet from the tool view.
- Using a keyboard shortcut, if you have assigned one.
- While writing, you can press **Ctrl+Space**, which will display all the snippets in a convenient window from which you can choose. This key combination provides functionality similar to code completion.

If the snippet contains variables (besides `${cursor}`) the cursor will automatically go to the first occurrence of a variable and will wait for you to write something. When you are done, you can press **Tab** to move to the next variable, and so on.

## 4.17.5 Thanks and Acknowledgments

Kate Snippets was written by Joseph Wenninger.

Special thanks to Google Code-In 2011 participant Martin Gergov for writing much of this section.

## 4.18 Keyboard Macros Plugin

### 4.18.1 Introduction

Record and play keyboard macros (i.e., keyboard action sequences).

### 4.18.2 Basic usage

#### 4.18.2.1 To start recording a keyboard macro:

**Tools** → **Keyboard Macros** → **Record Macro...** (**Ctrl+Shift+K**).

The plugin will record every key presses until you end recording.

#### 4.18.2.2 To end recording:

**Tools** → **Keyboard Macros** → **End Macro Recording (Ctrl+Shift+K)**.

The plugin will stop recording key presses and save the sequence as the current macro.

#### 4.18.2.3 To cancel recording:

**Tools** → **Keyboard Macros** → **Cancel Macro Recording (Ctrl+Alt+Shift+K)**.

The plugin will stop recording key presses but the current macro won't change.

#### 4.18.2.4 To play the current macro:

**Tools** → **Keyboard Macros** → **Play Macro (Ctrl+Alt+K)**.

The plugin will play the current macro.

The **kmplay** command without any arguments will also play the current macro.

### 4.18.3 Named macros

It is possible to save keyboard macros by giving them a name.

Named macros are persistent between Kate's sessions, they're saved in the `keyboardmacros.json` file in Kate's user data directory (usually `~/.local/share/kate/`).

#### 4.18.3.1 To save the current macro:

**Tools** → **Keyboard Macros** → **Save Current Macro (Alt+Shift+K)**.

The plugin will prompt you for a name and save the macro under it.

The **kmsave name** command will save the current macro under the name **name**.

#### 4.18.3.2 To load a saved macro as the current one:

**Tools** → **Keyboard Macros** → **Load Named Macro...**

The plugin lists saved macros as items in this submenu, activating an item will load the corresponding macro as the current one.

The **kmload name** command will load the macro saved under the name **name** as the current one.

#### 4.18.3.3 To play a saved macro without loading it:

**Tools** → **Keyboard Macros** → **Play Named Macro...**

The plugin lists saved macros as items in this submenu, activating an item will play the corresponding macro without loading it.

Note that each saved macros is an action that is part of the current action collection so that a custom shortcut can be assigned to it through the **Settings** → **Configure Keyboard Shortcuts...** interface.

The **kmplay name** command will play the macro saved under the name **name** without loading it.

#### 4.18.3.4 To wipe (i.e., delete) a saved macro:

**Tools** → **Keyboard Macros** → **Wipe Named Macro...**

The plugin lists saved macros as items in this submenu, activating an item will wipe (i.e., delete) the corresponding macro.

The **kmwipe name** command will wipe the macro saved under the name **name**.

#### 4.18.3.5 Tips for commands:

Note that after the **km** prefix, all these commands use a different letter so you can efficiently call them using tab-completion!

### 4.18.4 Limitations

As of now, keyboard macros fail to play properly if some types of GUI widgets are used: QMenu, QuickOpenLineEdit, or TabSwitcherTreeView, for example. I'm not sure why but my first guess would be that these widgets work in a non-standard way regarding keyboard events.

## 4.19 SQL Plugin

### 4.19.1 Introduction

The Structured Query Language (SQL) is a specialized language for updating, deleting, and requesting information from databases.

The Kate SQL Plugin allows you to:

- Create a database
- Connect to existing databases
- Insert and delete data in the database
- Execute queries
- Display results in a table

### 4.19.2 Connecting to a Database

Select **Add Connection** from the **SQL** menu or toolbar, and then select the Qt™ database driver you want to use (including QSQLITE, QMYSQL3, QMYSQL, QODBC3, QODBC, QPSQL7, and QPSQL). If you can't see the desired driver, you need to install it. Then, press **Next**.

If the database you selected uses a file, simply indicate the database's location and press the **Next** button. If it requires connecting to a server, you must enter the hostname of the server, your username and password, and any other information that particular driver may require. Then press **Next**.

Finally, give a name to your connection, and press **Finish**.

### 4.19.3 Running Queries

#### 4.19.3.1 INSERT/DELETE/UPDATE

You can insert, delete, and update data using the SQL plugin just as you would from the command line or from within a program. Simply enter a query and press the **Run query** button in the toolbar or use **SQL → Run query (Ctrl+E)**.

---

#### Example 4.1 Some Example Queries

---

##### INSERT

```
INSERT INTO table_name ("feature1", "feature2", "feature3", "feature4", ↵  
    "feature5")  
VALUES ("value1", "value2", "value3", "value4", "value5" )
```

##### DELETE

```
DELETE FROM table_name WHERE name = "text"
```

##### UPDATE

```
UPDATE table_name SET "feature1" = "text", "feature2" = "text", " ↵  
    feature3" = "text", "feature4" = "text", "feature5" = "text"
```

---

#### 4.19.3.2 SELECT

After running a **SELECT** query, you can view the results as a table that will appear in the **SQL Data Output** tool view at the bottom of Kate, or as text in the **SQL Text Output**. If there is an error, you can see it in the text output.

---

#### Example 4.2 Example SELECT Query

---

```
SELECT * FROM table_name
```

---

In the **SQL Data Output** tool view, there are several buttons:

##### Resize columns to contents

Changes the size of columns to fit their contents.

##### Resize rows to contents

Changes the size of rows to fit their contents.

##### Copy

Selects all of the table contents and copies it to the clipboard buffer.

##### Export

Exports all of the table contents to a file, the clipboard, or the current document in the Comma Separated Values format.

#### **Clear**

Removes everything from the table view.

You can now change the colors displayed in the table in the **SQL** section of **Settings** → **Configure Kate...**

### **4.19.4 Browsing**

You can browse your database using the **Database schema** browser tool view on the left. The information displayed varies depending on which database driver you are using.

To refresh this list, right-click anywhere in the tool view and select **Refresh**. To generate a query on any entry in the list, right-click on an entry, select **Generate**, and select the query type (**SELECT**, **UPDATE**, **INSERT**, or **DELETE**) from the submenu that appears.

### **4.19.5 Menu Structure**

#### **SQL → Add connection...**

Adds a new connection using any database driver.

#### **SQL → Remove connection**

Removes the selected connection.

#### **SQL → Edit connection...**

Edits the current connection's settings.

#### **Connections**

All database connections you have created are listed between the **Edit connection** and **Run query** menu items. Select one to run queries or make modifications to it.

#### **SQL → Run query (Ctrl+E)**

Runs your query.

### **4.19.6 Thanks and Acknowledgments**

The SQL Plugin was written by Marco Mentasti.

Special thanks to Google Code-In 2011 participant Ömer Faruk ORUÇ for writing much of this section.

## **4.20 Symbol Viewer Plugin**

### **4.20.1 Using the Close Except/Like Plugin**

It allows developers to view symbols (functions, macros and structures) from source code.

By clicking the parsed information you can easily browse the code.

At the moment the following languages are supported:

C/C++, Java™, Perl, PHP, Python, Ruby, XSLT, Tcl/Tk, Fortran

Feature list:

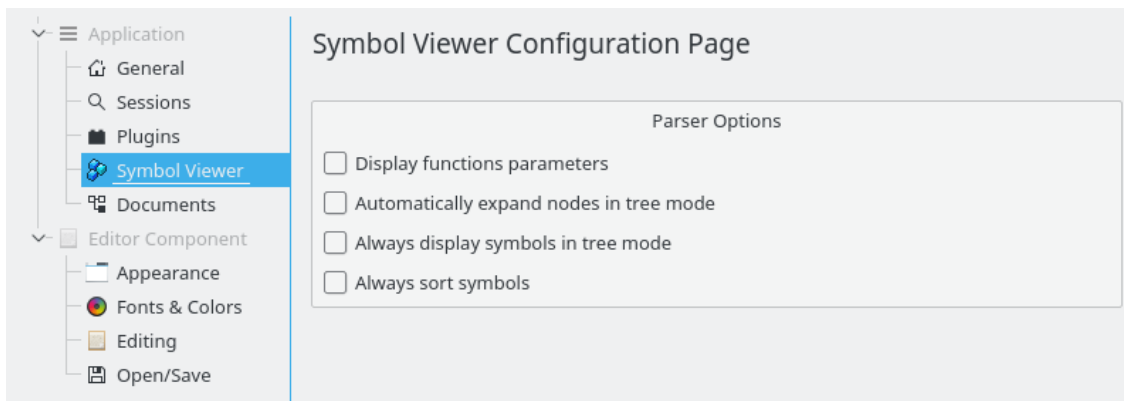
- List/Tree mode
- Enable/disable sorting
- Hide/Show Functions Parameters
- Expand/collapse tree mode
- Auto-update on document change
- Code parsing is based on the Syntax-Highlighting framework from KDE Frameworks

## 4.20.2 Menu Structure

**View** → **Tool Views** → **Show Symbol List (Ctrl+\\)**

Toggle the display of Kate's Symbol List displaying Functions, Macros and Structures of the source code in the active document.

## 4.20.3 Configuration



*Choose the default parser options*

## 4.21 Terminal Tool View Plugin

The built in Terminal Emulator is a copy of the KDE Konsole terminal application, for your convenience. It is available from the **View** → **Tool Views** → **Show Terminal Panel** menu item and will get the focus whenever displayed. Additionally, if the [Automatically synchronize the terminal with the current document when possible](#) option is enabled, it will change to the directory of the current document if possible when it is displayed, or when the current document changes.

The default location in the Kate window is at the bottom, below the editing area.

You can configure the Konsole using its right mouse button menu, for more information, see the [Konsole manual](#).

The built-in terminal emulator is provided by the Terminal Tool View plugin.

### 4.21.1 Menu Structure

**View** → **Tool Views** → **Show Terminal Panel**

Toggles the display of the built-in terminal emulator.

When activated for the first time, the terminal will be created.

When the terminal emulator is displayed, it will get the focus, so that you can start typing in commands immediately. If the [Automatically synchronize the terminal with the current document when possible](#) option is enabled in the **Terminal** page of the [Main configuration dialog](#) the shell session will change to the directory of the active document, if it is a local file.

#### **Tools → Pipe to Terminal**

Feed the currently selected text into the built-in terminal emulator. No newline is added after the text.

#### **Tools → Synchronize Terminal with Current Document**

This will cause the built-in Terminal to `cd` into the directory of the active document.

Alternatively, you can configure Kate to always keep the terminal in sync with the current document. See [Section 4.21.2](#) for more information.

#### **Tools → Run Current Document**

Run the current document in Konsole. Only the local documents can be run. Kate will show a warning before running document because this action can be a serious security threat.

#### **Tools → Focus/Defocus Terminal Panel**

Switch the focus from the current document to the terminal and vice versa.

## **4.21.2 Configuration**

You can configure the Terminal Tool View plugin on the **Terminal** page of the [configuration dialog](#).

The following options are available:

#### **Automatically synchronize the terminal with the current document when possible**

This will cause the built-in terminal to `cd` into the directory of the active document when launched and when a new document gets the focus. If not enabled, you have to do all your navigation in the terminal on your own.

#### **Set EDITOR environment variable to 'kate -b'**

This sets the `EDITOR` environment variable so programs run in the built-in terminal that automatically open a file in an editor will open them in Kate instead of the default editor configured in your shell. You will not be able to continue using the terminal until you have closed the file in Kate, so the calling program is aware you have finished editing the file.

#### **Hide Konsole on pressing 'Esc'**

This allows closing the built-in terminal by pressing the `Esc` key. May cause issues with terminal applications that use `Esc` key, e.g. `vim`. Add such applications in the text input box below. The items in the list should be separated with comma.

## **4.22 Text Filter Plugin**

### **4.22.1 Using the Text Filter Plugin**

You can use this plugin to process selected text using terminal commands. The selection will be used as input for the command, and the output will either replace the selection or be copied to the clipboard, depending on the user's preference.

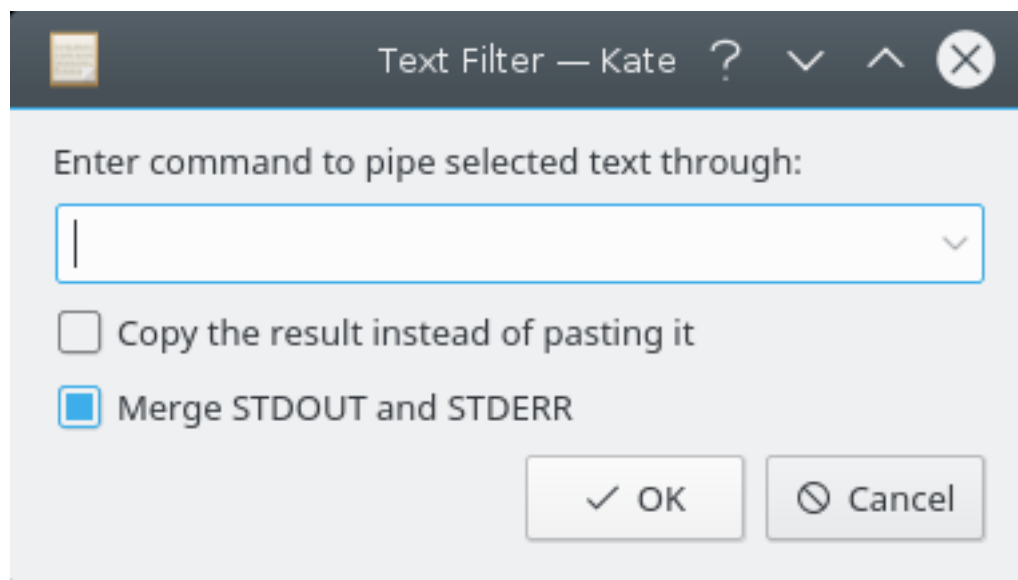
EXAMPLES:

- **less /etc/fstab** - paste the contents of this file or copy it to the clipboard
- **wc** - count lines, words and characters of the selection and paste this into the document or copy it to the clipboard
- **sort** - sort lines of the selection and paste the result into the document or copy it to the clipboard

## 4.22.2 Menu Structure

**Tools** → **Filter Text (Ctrl+\\)**

Opens the Text Filter dialog:



Enter the shell command into the combobox or select a previous command from the history.

### **Copy the result instead of pasting it**

Copy the result to clipboard leaving a document unchanged.

### **Merge STDOUT and STDERR**

If checked, an output from STDOUT and STDERR will be merged and no errors will be reported. Otherwise, STDERR will be displayed as a passive message.

## 4.23 XML Validation

This plugin checks XML files for validity and being well-formed.

This plugin checks the current file. A list of warnings and errors will appear at the bottom of Kate's main window. You can click on an error message to jump to the corresponding place in the file. If the file has a 'DOCTYPE' the DTD given with this doctype will be used to check the file for validity. The DTD is expected at a position relative to the current file, e.g. if the doctype refers to 'DTD/xhtml1-transitional.dtd' and the file is `/home/peter/test.xml` the DTD is expected to be located at `/home/peter/DTD/xhtml1-transitional.dtd`. However, remote DTDs specified via http are supported.

If the file has no doctype it will be checked for being well-formed.

To learn more about XML check out the [official W3C XML pages](#).

Internally this plugin calls the external command `xmllint`, which is part of `libxml2`. If this command is not correctly installed on your system, the plugin will not work.

To load this plugin open Kate's configuration dialog under **Settings** → **Configure Kate...** Then select **XML Validation** which will appear in the **Application / Plugins** section and close the dialog.

### 4.23.1 Menu Structure

**XML** → **Validate XML**

This will start the check, as described above.

### 4.23.2 Thanks and Acknowledgments

Kate Plugin 'XML Validation' copyright 2002 Daniel Naber [daniel.naber@t-online.de](mailto:daniel.naber@t-online.de).

Documentation copyright 2002 Daniel Naber

## 4.24 XML Completion

This plugin gives hints about what is allowed at a certain position in an XML file, according to the file's DTD. It will list possible elements, attributes, attribute values or entities, depending on the cursor position (e.g. all entities are listed if the character on the left of the cursor is '&'). It's also possible to close the nearest open tag on the left.

The DTD must exist in XML format, as produced by the Perl program `dtdparse`. We will call a DTD in this format 'meta DTD'. Some meta DTDs are supplied. They are installed in `katexmltools/` in `qtpaths --paths GenericDataLocation`, which is also the default folder when you choose **Assign Meta DTD...** To produce your own meta DTDs, get `dtdparse` from <http://dtdparse.sourceforge.net>.

### 4.24.1 How to Use

Start Kate and open the configuration dialog under **Settings** → **Configure Kate...** Then select **XML Completion** which will appear in the **Application** → **Plugins** page and close the dialog. After that, select **XML** → **Assign Meta DTD...** If your document contains no 'DOCTYPE' or the doctype is unknown, you will have to select a meta DTD from the file system. Otherwise the meta DTD that matches the current document's DOCTYPE will be loaded automatically.

You can now use the plugin while typing your text:

#### < (less than key)

This will trigger a list of possible elements unless the cursor is inside a tag already. Note that you currently cannot use this to insert the top level element (e.g. '<html>').

#### </(less than key + slash)

Entering these characters will offer to close the current element (nearest open one to the left of the cursor). Press **Enter** to accept the suggestion. Unlike the **Close Element** menu item, this works only with a DTD assigned.

#### " (quote key)

The quote key will trigger a list of possible attribute values (if there are any) if you are inside a tag.

**(space key)**

This key will trigger a list of possible attributes for the current element if you are inside a tag.

**& (ampersand key)**

This key will trigger a list of named entities.

## 4.24.2 Features and Limitations

You can test all functions and limitations by loading `katexmltools/testcases.xml` in `qtpaths --paths GenericDataLocation` into Kate and following the instructions.

## 4.24.3 Menu Structure

### XML → Insert Element... (Ctrl+Enter)

This will open a dialog that lets you insert an XML element. The `<`, `>` characters and the closing tag will be inserted automatically. If you have selected text when this menu item is selected, the selected text will be surrounded by the opening and the closing tag. The dialog also offers completion of all elements that may be inserted at the current cursor position if you have assigned a meta DTD by using **Assign Meta DTD...**

### XML → Close Element (Ctrl+<)

This will search your text for a tag that is not yet closed and will close it by inserting the corresponding closing tag. The search starts at the cursor position and goes left. If it cannot find an open tag nothing will happen.

### XML → Assign Meta DTD...

This will tell the plugin which meta DTD to use for the current document. Note that this assignment will not be saved. You will have to repeat it when you start Kate the next time.

## 4.24.4 Thanks and Acknowledgments

Kate Plugin 'XML Completion' copyright 2001,2002 Daniel Naber [daniel.naber@t-online.de](mailto:daniel.naber@t-online.de).

KDE SC 4 version copyright 2010 Tomáš Trnka

Documentation copyright 2001,2002 Daniel Naber

## 4.25 Compiler Explorer Plugin

The compiler explorer plugin allows you to use compiler explorer from within Kate. Compiler explorer is usually limited to 1 file only and it is difficult to use it to see the codegen of a file in your project. This plugin aims to simplify this process by automatically detecting compile commands for your file.

### 4.25.1 Usage

The first thing you need to do is to open a compiler explorer tab. Hit **Ctrl+Alt+I** to open the command bar and search for "Open Current File in Compiler Explorer" and hit enter. You can add this action to the toolbar or assign a shortcut to it for easier access.

Next you need to setup a local instance of compiler explorer or use the public <https://godbolt.org/> instance. See <https://github.com/compiler-explorer/compiler-explorer>. Once you have an instance, click on **Option** combobox in the compiler explorer tab you opened earlier and select **Change Url...** and then enter the url of the compiler explorer server.

Next you need a `compile_commands.json` if you need to see codegen for a file in your project. Otherwise you can skip this part. This can easily be obtained if you are using a modern build system like CMake or Meson. For CMake it is enough to pass

`-DCMAKE_EXPORT_COMPILE_COMMANDS=ON` when invoking `cmake`. The

`compile_commands.json` should be placed at the root of your project. Close and reopen the compiler explorer tab in Kate. After reopening the tab it should show the compile commands need to compile the file

If you are just using compiler explorer to see the codegen for something temporary, you can just write down the compile commands for your file and click the **Compile**. Note that auto recompilation on editing is not supported. You need to click **Compile** every time you want to recompile the file

Once the file is compiled you should see the codegen on the right hand side. You can right-click and select **Reveal linked code** to see the codegen for a particular line. Similarly on the right hand side, you can right click and select **Scroll to source** to show the source code for a given codegen line

## 4.26 Formatting Plugin

The formatting plugin allows one to format code easily. The plugin aims to preserve the document's undo history and user's cursor position when formatting the code so that the formatting of code doesn't disrupt user's work. This is especially important for automatic formatting on save.

### 4.26.1 Usage

It allows the user to format code in two ways:

- Manually using the "Format Document" action.
- Automatically on save

#### TIP

You can search for this action in the command bar and bind a shortcut to it for easier access

### 4.26.2 Supported languages and formatters

The current list of supported languages and formatters are as follows:

- C/C++/ObjectiveC/ObjectiveC++/Protobuf/GLSL/Java - **clang-format**

- Javascript/Typescript/JSX/TSX - **prettier**
- JSON
  - **prettier**
  - **clang-format**
  - **jq**
- Dart - **dartfmt**
- Rust - **rustfmt**
- Go - **gofmt**
- XML - **xmllint**
- Zig - **zigfmt**
- CMake - **cmake-format**
- D - **dfmt**
- Fish Shell - **fish\_indent**
- Bash - **shfmt**
- Nix - **nixfmt**
- QML - **qmlformat**
- HTML - **prettier**
- Swift - **swiftformat**
- Erlang - **erlfmt**
- Godot Script - **gdformat**
- Python
  - **autopep8**
  - **ruff**
- YAML
  - **yamlfmt**
  - **prettier**

### 4.26.3 Configuring

The plugin can be configured in two ways:

- Globally, from the Configure dialog
- On a per project basis using the `.kateproject` file

When reading the config, the plugin will first try to read the config from `.kateproject` file and then read the global config. Example:

## The Kate Handbook

```
{
  "formatOnSave": true,
  "formatterForJson": "jq",
  "cmake-format": {
    "formatOnSave": false
  },
  "autopep8": {
    "formatOnSave": false
  }
}
```

The above:

- enables “format on save” globally
- specifies “jq” as the formatter for JSON
- disables “format on save” for cmake-format and autopep8

To configure formatting for a project, first create a `.kateproject` file and then add a “formatting” object to it. In the “formatting” object you can specify your settings as shown in the previous example. Example:

### NOTE

You need to enable the [Project plugin](#) for this to work

```
{
  "name": "My Cool Project",
  "files": [
    {
      "git": 1
    }
  ],
  "formatting": {
    "formatterForJson": "clang-format",
    "autopep8": {
      "formatOnSave": false
    }
  }
}
```

### 4.26.4 Temporarily disable format on save

While working you will often need to disable the formatting for a short while due to any reason. You can do this from the menubar, **Tools** → **Format on Save**. You can also trigger this action from the **Command bar**.

## Chapter 5

# Advanced Editing Tools

For information about the advanced editing tools included with Kate, see the [Advanced Editing Tools](#) chapter of the [KatePart Handbook](#).

## Chapter 6

# Extending Kate

T.C. Hollingsworth

### 6.1 Introduction

Like any advanced text editor, Kate offers a variety of ways to extend its functionality. You can [write simple scripts to add functionality with JavaScript](#) or add even more functionality to the editor itself with [Kate Application Plugins written in C++](#). Finally, once you have extended Kate, you are welcome to [join us](#) and share your enhancements with the world!

### 6.2 Working with Syntax Highlighting

For information about adding or modifying syntax highlighting definitions, see the [Working with Syntax Highlighting](#) section of the [Development](#) chapter of the [KatePart Handbook](#).

### 6.3 Scripting with JavaScript

For information about scripting with JavaScript, see the [Scripting with JavaScript](#) section of the [Development](#) chapter of the [KatePart Handbook](#).

### 6.4 Kate (C++) Application Plugins

[Kate Application Plugins](#) extend the functionality of the Kate editor itself in any way you can imagine, using the same programming language Kate is written in, C++.

To get started, see the [Writing a Kate Plugin](#) tutorial on the [Kate website](#).

## Chapter 7

# The VI Input Mode

For information about Kate's VI input mode, see the [VI Input Mode chapter of the KatePart Handbook](#).

## Chapter 8

# The Menu Entries

### 8.1 The File Menu

**File** → **New (Ctrl+N)**

This command starts a new document in the editing window. In the **Documents** list on the left the new file is named *Untitled*.

**File** → **Open... (Ctrl+O)**

Displays a standard KDE **Open File** dialog. Use the file view to select the file you want to open, and click on **Open** to open it.

**File** → **Open Recent**

This is a shortcut to open recently saved documents. Clicking on this item opens a list to the side of the menu with several of the most recently saved files. Clicking on a specific file will open it in this application - if the file still resides at the same location.

**File** → **Open With**

This submenu presents a list of applications known to handle the MIME type of your current document. Activating an entry will open the current document with that application.

In addition, an entry **Other...** launches the open with dialog box that allows you to select another application to open the active file. Your file will still be open in Kate.

**File** → **Save (Ctrl+S)**

This saves the current document. If there has already been a save of the document then this will overwrite the previously saved file without asking for the user's consent. If it is the first save of a new document the save as dialog (described below) will be invoked.

**File** → **Save As... (Ctrl+Shift+S)**

This allows a document to be saved with a new file name. This is done by means of the file dialog box described above in the [Open](#) section of this help file.

**File** → **Save As with Encoding**

Save a document with a new file name in a different encoding.

**File** → **Save Copy As**

Save a copy of the document with a new file name and continue editing the original document.

**File** → **Save All (Ctrl+L)**

This command saves all modified open files.

**File → Reload (F5)**

Reloads the active file from disk. This command is useful if another program or process has changed the file while you have it open in this application.

**File → Reload All**

Reloads all opened files..

**File → Print... (Ctrl+P)**

Opens a simple print dialog allowing the user to specify what, where, and how to print.

**File → Export as HTML**

Save the currently open document as an HTML file, which will be formatted using the current syntax highlighting and color scheme settings.

**File → Close (Ctrl+W)**

Close the active file with this command. If you have made unsaved changes, you will be prompted to save the file before Kate closes it.

**File → Close Other**

Close other open documents.

**File → Close All**

This command closes all the files you have open in Kate.

**File → Close Orphaned**

Close all documents in the file list, which could not be reopened during startup, because they are not accessible anymore.

**File → Quit (Ctrl+Q)**

This command closes Kate and any files you were editing. If you have made unsaved changes to any of the files you were editing, you will be prompted to save them.

## 8.2 The Edit Menu

**Edit → Undo (Ctrl+Z)**

Undo the last editing command (typing, copying, cutting etc.)

**NOTE**

This may undo several editing commands of the same type, like typing in characters.

**Edit → Redo (Ctrl+Shift+Z)**

This will reverse the most recent change (if any) made using Undo.

**Edit → Cut (Ctrl+X)**

This command deletes the current selection and places it on the clipboard. The clipboard works invisibly and provides a way to transfer data between applications.

**Edit → Copy (Ctrl+C)**

This copies the currently selected text to the clipboard so that it may be pasted elsewhere. The clipboard works invisibly and provides a way to transfer data between applications.

**Edit → Paste (Ctrl+V)**

This will insert the first item in the clipboard at the cursor position. The clipboard works invisibly and provides a way to transfer data between applications.

**NOTE**

If Overwrite Selection is enabled, the pasted text will overwrite the selection, if any.

**Edit → Paste Selection (Ctrl+Shift+Ins)**

This will paste the [mouse selection](#) contents that were chosen previously. Mark some text with the mouse pointer to paste it in the currently open file using this menu item.

**Edit → Swap with clipboard contents**

This will swap the selected text with the [clipboard](#) contents.

**Edit → Clipboard History**

This submenu will display the beginning of portions of text recently copied to the clipboard. Select an item from this menu to paste it in the currently open file.

**Edit → Copy as HTML**

Copy the selection as HTML, formatted using the current syntax highlighting and color scheme settings.

**Edit → Select All (Ctrl+A)**

This will select the entire document. This could be very useful for copying the entire file to another application.

**Edit → Deselect (Ctrl+Shift+A)**

Deselects the selected text in the editor if any.

**Edit → Block Selection Mode (Ctrl+Shift+B)**

Toggles Selection Mode. When the Selection Mode is **BLOCK**, the status bar contains the string **[BLOCK]** and you can make vertical selections, e.g. select column 5 to 10 in lines 9 to 15.

**Edit → Input Modes**

Switch between a normal and a vi-like, modal editing mode. The vi input mode supports the most used commands and motions from vim's normal and visual mode and has an optional vi mode statusbar. This status bar shows commands while they are being entered, output from commands and the current mode. The behavior of this mode can be configured in the [Vi Input Mode](#) section of the **Editing** page in this application's settings dialog.



**Edit → Overwrite Mode (Ins)**

Toggles the Insert/Overwrite modes. When the mode is **INS**, you insert characters where the cursor is. When the mode is **OVR**, writing characters will replace the current characters if your cursor is positioned before any character. The status bar shows the current state of the Overwrite Mode, either **INS** or **OVR**.


**Edit → Find... (Ctrl+F)**


This opens the incremental search bar at the bottom of the editor window. On the left side of the bar is a button with an icon to close the bar, followed by a small text box for entering the search pattern.

When you start entering characters of your search pattern, the search starts immediately. If there is a match in the text this is highlighted and the background color of the entry field changes to light green. If the search pattern does not match any string in the text, this is indicated by a light red background color of the entry field.

Use the  or  button to jump to the next or previous match in the document.

Matches in the document are highlighted even when you close the search bar. To clear this highlighting, press the **Esc** key.

You can choose whether the search should be case sensitive. Selecting  will limit finds to entries that match the case (upper or lower) of each of the characters in the search pattern.

Click on the  button at the right side of the incremental search bar to switch to the power search and replace bar.

**Edit → Find Variants → Find Next (F3)**

This repeats the last find operation, if any, without calling the incremental search bar, and searching forwards through the document starting from the cursor position.

**Edit → Find Variants → Find Previous (Shift+F3)**

This repeats the last find operation, if any, without calling the incremental search bar, and searching backwards instead of forwards through the document.

**Edit → Find Variants → Find Selected (Ctrl+H)**

Finds next occurrence of selected text.

**Edit → Find Variants → Find Selected Backwards (Ctrl+Shift+H)**



Finds previous occurrence of selected text.

**Edit → Replace... (Ctrl+R)**

This command opens the power search and replace bar. On the upper left side of the bar is a button with an icon to close the bar, followed by a small text box for entering the search pattern.


You can control the search mode by selecting **Plain text**, **Whole words**, **Escape sequences** or **Regular expression** from the drop down box.


If **Escape sequences** or **Regular expression** are selected, the **Add...** menuitem at the bottom of the context menu of the text boxes will be enabled and allows you to add escape sequences or regular expression items to the search or replace pattern from predefined lists.


Use the  or  button to jump to the next or previous match in the document.

Enter the text to replace with in the text box labeled **Replace** and click the **Replace** button to replace only the highlighted text or the **Replace All** button to replace the search text in the whole document.

You can modify the search and replace behavior by selecting different options at the bottom

of the bar. Selecting  will limit finds to entries that match the case (upper or lower)

of each of the characters in the search pattern.  will search and replace within the current selection only. The **Find All** button highlights all matches in the document and shows the number of found matches in a small popup.

Click on the  button at the right side of the power search and replace bar to switch to the incremental search bar.

**Edit → Go To → Go to Matching Bracket (Ctrl+6)**

Move the cursor to the associated opening or closing bracket.

**Edit → Go To → Select to Matching Bracket (Ctrl+Shift+6)**

Selects the text between associated opening and closing brackets.

**Edit → Go To → Go to Previous Modified Line**

Lines that were changed since opening the file are called modified lines. This action jumps the previous modified line.

**Edit → Go To → Go to Next Modified Line**

Lines that were changed since opening the file are called modified lines. This action jumps the next modified line.

**Edit → Go To → Go to Line... (Ctrl+G)**

This opens the goto line bar at the bottom of the window which is used to have the cursor jump to a particular line (specified by number) in the document. The line number may be entered directly into the text box or graphically by clicking on the up or down arrow spin controls at the side of the text box. The little up arrow will increase the line number and the down arrow decrease it. Close the bar with a click on the button with an icon on the left side of the bar.

## 8.3 The View Menu

**View → New Window**

Opens another instance of Kate. The new instance will be identical to your previous instance.

**View → Next Tab (Alt+Right)**

Activates the next tab in the tabbar.

**View → Previous Tab (Alt+Left)**


Activates the previous tab in the tabbar.

**View → Reopen Latest Closed Document (Ctrl+Shift+T)**

Reopens the latest closed document or documents.

**View → Quick Open (Ctrl+Alt+O)**

Show a search field and a list of opened files in the editor area. While entering text in the search field the document names and document URLs are searched for matching text. While entering text in the search field you can use the cursor keys **Up** and **Down** to navigate in the list view. Pressing the **Enter** key or double clicking on an item in the list switches the view to the document selected in the list view. This makes switching between documents easier, if there are a lot of them open.

This action is available also using the  icon at the top right of the editor window.

**View → Split View → Previous Split View (Shift+F8)**

Focus the previous document view, if you have split the editor area in more views.

**View → Split View → Next Split View (F8)**

Focus the next document view, if you have split the editor area in more views.

**View → Split View → Left Split View**

Focus the split view intuitively on the left, using the cursor position to disambiguate if necessary.

**View → Split View → Right Split View**

Focus the split view intuitively on the right, using the cursor position to disambiguate if necessary.

**View → Split View → Upward Split View**

Focus the split view intuitively upward, using the cursor position to disambiguate if necessary.

**View → Split View → Downward Split View**

Focus the split view intuitively downward, using the cursor position to disambiguate if necessary.

**View → Split View → Split Vertical (Ctrl+Shift+L)**

This will split the frame (which may be the main editing area) in two equally sized frames, the new one to the left of the current one. The new frame gets the focus, and will display the same document as the old one.

See also [Working with the Kate MDI](#).

**View → Split View → Split Horizontal (Ctrl+Shift+T)**

Splits the current frame (which may be the main editing area) in two equally sized frames, the new one below the current one. The new frame gets the focus, and displays the same document as the old one.

See also [Working with the Kate MDI](#)

**View → Split View → Move Document to New Vertical Split**

This will split the currently active view vertically into two views and move the currently active document to right view.

See also [Working with the Kate MDI](#).

**View → Split View → Move Document to New Horizontal Split**

This will split the currently active view horizontally into two views and move the currently active document to view below.

See also [Working with the Kate MDI](#)

**View → Split View → Toggle Orientation**

Switch between horizontal and vertical split.

**View → Split View → Close Current View (Ctrl+Shift+R)**

Closes the active frame, which can be identified as the one displaying a blinking cursor. This is disabled, if there is only one frame (the main editing area).

No documents get closed by closing a frame – they will still be available in the [View Menu](#) as well as in the File List.

See also [Working with the Kate MDI](#)

**View → Split View → Close Inactive Views**

Closes all frames except the active frame (the one with a blinking cursor). This is disabled, if there is only one frame (the main editing area).

No documents get closed by closing a frame – they will still be available in the [View Menu](#) as well as in the File List.

**View → Split View → Hide Inactive Views**

This hides all split views except the currently active one.

**View → Split View → Move Splitter Left**

When Split View is enabled, this will move the border between two vertically split documents further left.

**View → Split View → Move Splitter Right**

When Split View is enabled, this will move the border between two vertically split documents further right.


**View → Split View → Move Splitter Up**

When Split View is enabled, this will move the border between two horizontally split documents further up.

### View → Split View → Move Splitter Down

When Split View is enabled, this will move the border between two horizontally split documents further down.

#### NOTE

Some common actions in the **ViewSplit View** menu are available using the  button at the top right corner of the editor window

### View → Tool Views

#### View → Tool Views → Show Sidebars (Ctrl+Alt+Shift+F)

Toggles the display of the sidebar button rows. This command does not affect the display of the sidebar content widgets, any sidebar that is visible will stay visible, and if you assigned shortcuts to the commands below those will of course continue to work.

#### View → Tool Views → Show *Plugin*

A list of all enabled plugins. Use the checkbox in front of each item to toggle the display of the tool view.

### View → Switch to Command Line (F7)

This command will toggle the display of the [built-in command line](#).

### View → Enlarge Font (Ctrl++)

This increases the display font size.

### View → Shrink Font (Ctrl+-)

This decreases the display font size.

### View → Schema

This menu lists the available color schemes. You can change the schema for the current view here, to change the default schema you need to use the [Fonts & Colors](#) page of the config dialog.

### View → Word Wrap → Dynamic Word Wrap (F10)

Toggles dynamic word wrap in the current view. Dynamic word wrap makes all the text in a view visible without the need for horizontal scrolling by rendering one actual line on more visual lines as needed.

### View → Word Wrap → Dynamic Word Wrap Indicators

Choose when and how the dynamic word wrap indicators should be displayed. This is only available if the **Dynamic Word Wrap** option is checked.

### View → Word Wrap → Show Static Word Wrap Marker

Toggles the display of a vertical line indicating the position of the wrap width as configured in the [config dialog](#). This feature requires that you use a true fixed-width font.

### View → Borders → Show Icon Border (F6)

This is a toggle item. Setting it on checked will make the Icon Border visible in the left side of the active editor, and vice versa. The Icon Border indicates the positions of the marked lines in the editor.

### View → Borders → Show Line Numbers (F11)

This is a toggle Item. Setting it on checked will make a pane displaying the line numbers of the document visible in the left border of the active editor, and vice versa.

**View → Borders → Show Scrollbar Marks**

Toggles the visualization of bookmarks (and other marks) on the vertical scrollbar. When enabled, a mark is represented by a thin line in the mark color at the scrollbar, clicking the middle mouse button on the line will scroll the view to a position near the mark.

**View → Borders → Show Scrollbar Mini-Map**

This will replace the scrollbar with a visualization of the current document. For more information on the scrollbar minimap, see the [Scrollbar Minimap section of the KatePart Handbook](#).

**View → Code Folding**

These options pertain to [code folding](#):

**Show Folding Markers (F9)**

Toggles the display of the folding marker pane in the left side of the view.

**Fold Current Node**

Collapse the region that contains the cursor.

**Unfold Current Node**

Expand the region that contains the cursor.

**Fold Toplevel Nodes (Ctrl+Shift+-)**

Collapse all toplevel regions in the document. Click on the right pointing triangle to expand all toplevel regions.

**Unfold Toplevel Nodes (Ctrl+Shift++)**

Expand all toplevel regions in the document.

**Show Non-Printable Spaces**

Show/hide bounding box around non-printable spaces.

## 8.4 The Bookmarks Menu

Below the entries described here, one entry for each bookmark in the active document will be available. The text will be the first few words of the marked line. Choose an item to move the cursor to the start of that line. The editor will scroll as necessary to make that line visible.

**Bookmarks → Set Bookmark (Ctrl+B)**

Sets or removes a bookmark in the current line of the active document. (If it's there, it is removed, otherwise one is set.)

**Bookmarks → Clear All Bookmarks**

This command will remove all the markers from the document as well as the list of markers which is appended at the bottom of this menu item.

**Bookmarks → Previous (Alt+PgUp)**

This will move the cursor to beginning of the first above line with a bookmark. The menuitem text will include the line number and the first piece of text on the line. This item is only available when there is a bookmark in a line above the cursor.

**Bookmarks → Next (Alt+PgDn)**

This will move the cursor to beginning of the next line with a bookmark. The menuitem text will include the line number and the first piece of text on the line. This item is only available when there is a bookmark in a line below the cursor.

## 8.5 The Sessions Menu

This menu contains entries for using and managing Kate sessions. For more information, read [Using Sessions](#).

### Sessions → New

Creates a new empty session. All currently open files will be closed.

### Sessions → Open Session...

Open an existing session. The Session Chooser dialog is displayed to let you choose one.

### Sessions → Quick Open Session

This submenu lets you open an existing session.

### Sessions → Save Session

Save the current session. If the session is anonymous, you will be prompted for a session name.

### Sessions → Save Session As...

Save the current session under a new name. You are prompted for a name to use.

### Sessions → Manage Sessions...

Displays the Session Manager dialog which allows you to rename and delete sessions.

## 8.6 The Tools Menu

### Tools → Read Only Mode

Set the current document to Read Only mode. This prevents any text addition and any changes in the document formatting.

### Tools → Mode

Choose the filetype scheme you prefer for the active document. This overwrites the global [filetype](#) mode set in **Settings** → **Configure Kate...** in the Filetypes tab for your current document only.

### Tools → Highlighting

Choose the Highlighting scheme you prefer for the active document. This overwrites the global highlighting mode set in **Settings** → **Configure Editor...** for your current document only.

### Tools → Indentation

Choose the [style of indentation](#) you want for your active document. This overwrites the global indentation mode set in **Settings** → **Configure Editor...** for your current document only.

### Tools → Encoding

You can overwrite the default encoding set in **Settings** → **Configure Editor...** in the **Open/Save** page to set a different encoding for your current document. The encoding you set here will be only valid for your current document.

### Tools → End of Line

Choose your preferred end of line mode for your active document. This overwrites the global end of line mode set in **Settings** → **Configure Editor...** for your current document only.

**Tools → Add Byte Mark Order (BOM)**

Checking this action you can explicitly add a byte order mark for unicode encoded documents. The byte order mark (BOM) is a Unicode character used to signal the endianness (byte order) of a text file or stream, for more information see [Byte Order Mark](#).

**Tools → Scripts**

This submenu contains a list of all scripted actions. The list can easily be modified by [writing your own scripts](#). This way, Kate can be extended with user-defined tools.

There is a complete list of scripts in the [KatePart documentation](#).

**Tools → Invoke Code Completion (Ctrl+Space)**

Manually invoke command completion, usually by using a shortcut bound to this action.

**Tools → Word Completion**

**Reuse Word Below (Ctrl+9)** and **Reuse Word Above (Ctrl+8)** complete the currently typed text by searching for similar words backward or forward from the current cursor position. **Shell Completion** pops up a completion box with matching entries.

**Tools → Spelling → Automatic Spell Checking (Ctrl+Shift+O)**

When **Automatic Spell Checking** is enabled, wrongly spelled text is underlined in the document on-the-fly.

**Tools → Spelling → Spelling...**

This initiates the spellchecking program - a program designed to help the user catch and correct any spelling errors.

For more information on how to use the KDE spellchecking program, see the [Check Spelling section of the KDE Fundamentals documentation](#).

**Tools → Spelling → Spelling (from cursor)...**

This initiates the spellchecking program but it starts where your cursor is instead of at the beginning of the document.

**Tools → Spelling → Spellcheck Selection...**

Spellchecks the current selection.

**Tools → Spelling → Change Dictionary**

Displays a drop down box with all available dictionaries for spellchecking at the bottom of the editor window. This allows easy switching of the spellcheck dictionary e.g. for automatic spellcheck of text in different languages.

## 8.7 The Settings and Help Menu


Kate has the common KDE **Settings** and **Help** menu items, for more information read the sections about the [Settings Menu](#) and [Help Menu](#) of the KDE Fundamentals with these additional entries:

**Settings → Color Theme**

Use a different color scheme from the system's global color schemes described in the System Settings module [Colors](#).

**Settings → Show Tabs**

Tabs are moveable using the left mouse button and have actions in the context menu to close documents, copy the path to the clipboard or open the folder of the document in the filemanager. Using the **Quick Open** button makes switching between documents easy.

Click the  button with the left mouse button to open a menu with actions from the **ViewSplit View** menu.

**Settings → Show Path in Titlebar**

If enabled the full path of the active document is displayed, otherwise only the filename. This is useful if you edit several files with the same filename to distinguish them.

## Chapter 9

# Configuring Kate

Anders Lund

### 9.1 Overview

Kate offers several means of tweaking the application to behave as desired. The most important ones are:

#### **The Configuration Dialog**

The main configuration tool, allowing you to configure the Kate application, the editor component and the usage of plugins.

#### **The Settings Menu**

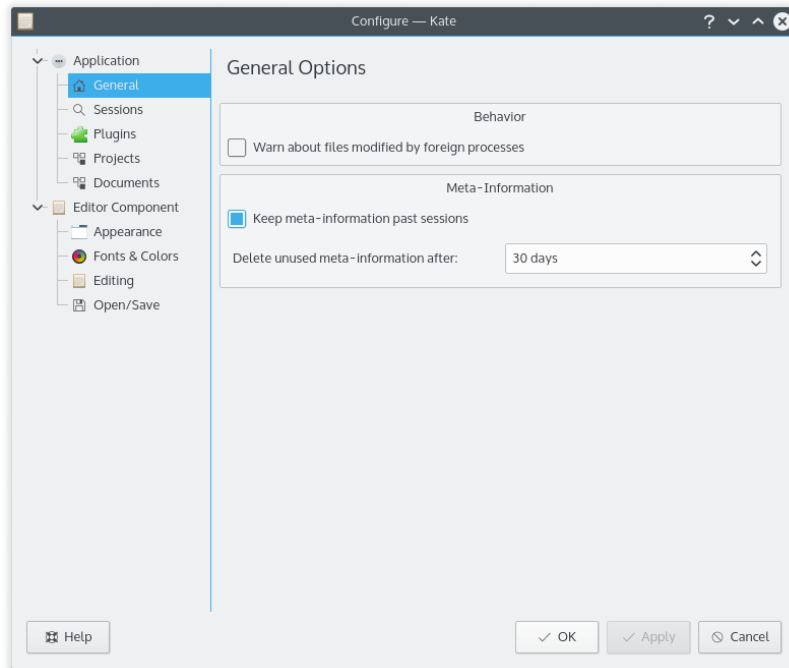
Allows you to change often used settings, and to launch the configuration dialogs.

#### **The View Menu**

Allows you to split the current frame, as well as to display the icons and line numbers pane for the currently edited document.

The embedded terminal uses the configuration defined in the System Settings, and may also be configured by clicking the right mouse button to display a context menu.

## 9.2 The Main Configuration Dialog



The Kate configuration dialog displays a tree of topics on the left, and a configuration page corresponding to the selected topic on the right.

The configuration is divided into two groups, namely:

- [Application configuration](#)
- [Editor component configuration](#)

## 9.3 The Kate Application Configuration

This group contains pages to configure the main Kate application

### 9.3.1 General

This section contains a few global options for Kate

#### Behavior

##### Open each document in its own window

If enabled, each document will be opened in its own window. If not enabled, each document will be opened in a new tab in the current window.

##### Switch to output view upon message type

This option allows configuring when Kate should show an output pane depending on the type of action output.

It is possible to choose between **Never**, **Error** (on error), **Warning** (on warning or above), **Info** (on info or above), and **Log** (on log or above).

### Use a separate dialog for handling externally modified files

When enabled, Kate will notify you with a modal dialog about all files modified from outside the application whenever the main window receives input focus. You will be able to deal with several modified files at once, you can reload, save or discard changed files in groups.

If not enabled, Kate will individually ask you what to do for each modified file only when that file's view receives focus.

## Quick Open

### Match Mode

Set the list mode for the [Quick Open](#) tool. The files can be matched by their name or by their path.

### List Mode

Set the list mode for the [Quick Open](#) tool. It is possible to choose from **Current Project Files** and **All Projects Files**.

## Tabs

### Limit number of tabs

Set the maximum number of tabs. Choose **Unlimited** if you do not want to restrict this number.

### Auto hide tabs

When checked tabs will be hidden if only one document is open.

### Show close button

When checked each tab will display a close button.

### Expand tabs

When checked tabs take as much size as possible.

### Double click opens a new document

When checked double click opens a new document.

### Middle click closes a document

When checked middle click closes a document.

### Allow tab scrolling

When checked this will allow scrolling in tab bar when number of tabs is large.

### Elide tab text

When checked tab text might be elided if its too long.

## Tabs

### Backward button pressed

Allows selecting the mouse back button action between the **Previous tab** and the **History back** items.

### Forward button pressed

Allows selecting the mouse forward button action between the **Next tab** and the **History forward** items.

## 9.3.2 Session

This section contains options related to [using sessions](#).

### Application Startup Behavior

Select how you want Kate to behave at startup. This setting can be overridden by specifying what to do on the [command line](#).

**Start new session**

With this option, Kate will start a new, unnamed session when you start the application.

**Load last-used session**

Kate will use the most recently opened session at startup. This is good if you want to use the same session always or switch rarely.

**Manually choose a session**

Kate will display a small dialog that lets you choose your preferred session, or load the default session if none have been saved. This is the default behavior. Nice if you use a lot of different sessions frequently.

**Application Startup/Shutdown Behavior**

Select how you want Kate to behave at shutdown. It is possible to define what Kate should **Automatically save and restore**.

**Newly-created unsaved files**

With this item checked, Kate will automatically save all newly-created unsaved files.

**Files with unsaved changes**

This item allows configuring Kate for automatically saving all files with unsaved changes on shutdown.

**Close Kate entirely when the last file is closed**

If enabled, Kate will shutdown when the last file being edited is closed, otherwise a blank page will open so that you can start a new file.

**Session Elements**

**Include window configuration**

If enabled, Kate will save the window configuration with each session.

**Keep meta-information past sessions**

When enabled, Kate will store meta data such as bookmarks and session configuration even when you close your documents. The data will be used if the document is unchanged when reopened.

**Delete unused meta-information after**

Set the maximum number of days to keep meta information for previously opened files. This helps keep the database of meta information reasonably sized.

Any changes to the session data (opened files and, if enabled, window configuration) will always be saved.

### 9.3.3 Plugins

This page provides a list of installed plugins for the Kate application. Each plugin is represented with its name and a short description. You can check the checkbox with an item to enable the plugin it represents.

If a plugin provides configuration options, a section to access those will appear as a child of this page.

For more information about the available plugins, see chapter 4.

### 9.3.4 The Editor Component Configuration

For information about this section of the configuration dialog, see the [Editor Component Configuration section of the KatePart Handbook](#).

### 9.3.5 Configuring With Document Variables

For information about using document variables with Kate, see the [Configuring with Document Variables](#) section of the [KatePart Handbook](#).

## Chapter 10

# Credits and License

Kate. Program copyright 2000, 2001, 2002 - 2005 by the Kate developer team.

THE KATE TEAM:

**Christoph Cullmann** [cullmann@kde.org](mailto:cullmann@kde.org)

Project Manager & Core Developer

**Anders Lund** [anders@alweb.dk](mailto:anders@alweb.dk)

Core Developer, Perl syntax highlighting, documentation

**Joseph Wenninger** [kde@jowenn.at](mailto:kde@jowenn.at)

Core Developer, syntax highlighting

**Michael Bartl** [michael.bartl1@chello.at](mailto:michael.bartl1@chello.at)

Core Developer

**Phlip** [phlip\\_cpp@my-deja.com](mailto:phlip_cpp@my-deja.com)

The project compiler

**Waldo Bastian** [bastian@kde.org](mailto:bastian@kde.org)

The cool buffer system

**Matt Newell** [newellm@proaxis.com](mailto:newellm@proaxis.com)

Testing...

**Michael McCallum** [gholam@xtra.co.nz](mailto:gholam@xtra.co.nz)

Core Developer

**Jochen Wilhemly** [digisnap@cs.tu-berlin.de](mailto:digisnap@cs.tu-berlin.de)

KWrite Author

**Michael Koch** [koch@kde.org](mailto:koch@kde.org)

KWrite port to KParts

**Christian Gebauer** [gebauer@bigfoot.com](mailto:gebauer@bigfoot.com)

Unspecified

**Simon Hausmann** [hausmann@kde.org](mailto:hausmann@kde.org)

Unspecified

## The Kate Handbook

**Glen Parker** [glenebob@nwlink.com](mailto:glenebob@nwlink.com)

KWrite Undo History, KSpell integration

**Scott Manson** [sdmanson@alltel.net](mailto:sdmanson@alltel.net)

KWrite XML syntax highlighting support

**John Firebaugh** [jfirebaugh@kde.org](mailto:jfirebaugh@kde.org)

Various Patches

**Dominik Haumann** [dhdev@gmx.de](mailto:dhdev@gmx.de)

Developer, Highlight wizard

MANY OTHER PEOPLE HAVE CONTRIBUTED:

**Matteo Merli** [merlim@libero.it](mailto:merlim@libero.it)

Highlighting for RPM Spec-Files, Diff and more

**Rocky Scaletta** [rocky@purdue.edu](mailto:rocky@purdue.edu)

Highlighting for VHDL

**Yury Lebedev**

Highlighting for SQL

**Chris Ross**

Highlighting for Ferite

**Nick Roux**

Highlighting for ILERPG

**John Firebaugh**

Highlighting for Java™, and much more

**Carsten Niehaus**

Highlighting for L<sup>A</sup>T<sub>E</sub>X

**Per Wigren**

Highlighting for Makefiles, Python

**Jan Fritz**

Highlighting for Python

**Daniel Naber**

Small bugfixes, XML plugin

Documentation copyright 2000,2001 Seth Rothberg [sethmr@bellatlantic.org](mailto:sethmr@bellatlantic.org)

Documentation copyright 2002, 2003, 2005 Anders Lund [anders@alweb.dk](mailto:anders@alweb.dk)

This documentation is licensed under the terms of the [GNU Free Documentation License](#).

This program is licensed under the terms of the [GNU General Public License](#).

## Appendix A

# Regular Expressions

For information about using regular expressions in Kate, see the [Regular Expressions appendix to the KatePart Handbook](#).

## Appendix B

# Index

### C

configure  
  settings  
    preferences, [95](#)

### D

Documents list, [37](#)

### E

Editing Area, [14](#)

### M

Main window, [14](#)

### T

Terminal emulator, [73](#)