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.

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.

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.

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.

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.

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.

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", "--ba
            "commandDebug": ["clangd", "-log=verbose"
            "url": "https://clang.llvm.org/extra/clan
            "highlightingModeRegex": "^(C|ANSI C89|Ob
        },
        "cpp": {
            "use": "c",
            "highlightingModeRegex": "^(C\\+\\+|ISO C
        },
        "d": {
            "command": ["dls", "--stdio"],
            "url": "https://github.com/d-language-ser
```

```
            "highlightingModeRegex": "^D$"
        },
        "fortran": {
            "command": ["fortls"],
            "rootIndicationFileNames": [".fortls"],
            "url": "https://github.com/hansec/fortran
            "highlightingModeRegex": "^Fortran.*$"
        },
        "javascript": {
            "command": ["typescript-language-server",
            "rootIndicationFileNames": ["package.json
            "url": "https://github.com/theia-ide/type
            "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
            "highlightingModeRegex": "^Go$"
        },
        "python": {
            "command": ["python3", "-m", "pyls", "--c
            "url": "https://github.com/palantir/pytho
            "highlightingModeRegex": "^Python$"
        },
        "rust": {
            "command": ["rls"],
            "path": ["%{ENV:HOME}/.cargo/bin", "%{ENV
            "rootIndicationFileNames": ["Cargo.lock",
            "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.

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 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 as 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"],
                        "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 virtua
# (i.e. with virtualenv variables setup)
# so source the virtualenv
source XYZ
# server mileage or arguments may vary
exec myserver
```

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.

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", "
            }
        }
    }
}
```

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.

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 `QT_LOGGING_RULES=katelspclientplugin=true` suitably `export`'ed.