enki Documentation

Release 15.11.1

Andrei Kopats, Filipe Azevedo

January 11, 2016

Contents

Enki home page.

This is API documentation. It lists all modules, which may be used by Enki plugins.

To create your own plugins, you will probably also need:

- Hacking guide
- Plugin tutorial
- Qutepart documentation

CHAPTER 1

enki.core

Core creates basic user interface, provides basic functionality for user, contains plugin API.

1.1 core — Instances of core classes, initialize and terminate the system

Module initializes system at startup, terminates it, when Enki closed, and used for get core instances, such as main window, workspace, etc.

class enki.core.core.**Core**
Bases: QObject

Core object initializes system at startup and terminates when closing.

It creates instances of other core modules and holds references to it

restoreSession
restoreSession()

Signal for session plugin. Emitted, when initialization has been finished and all files, listed in the command line has been opened. Only if user hadn't passed –no-session key

aboutToTerminate
aboutToTerminate()

Signal emitted, before closing all files and terminating Enki

settingsDialogAccepted
settingsDialogAccepted()

Signal emitted, when settings dialog had been accepted

init(*profiler*, *cmdLine*)
Initialize core.

Called only by main()

term()
Terminate plugins and core modules

Called only by main()

mainWindow()
Get *enki.core.mainwindow.MainWindow* instance

actionManager()
Get main window :*enki.core.actionmanager.ActionManager* instance

workspace()
Get *enki.core.workspace.Workspace* instance

config()
Get *enki.core.config.Config* instance

loadedPlugins()
Get list of curretly loaded plugins (:`enki.core.Plugin` instances)

fileFilter()
Negative file filter

See :*enki.core.filefilter*

locator()
:*enki.core.locator.Locator* instance

Widget, which appears on Ctrl+L. Allows to execute textual commands Extendable with new commands

uiSettingsManager()
:*enki.core.uisettings.UISettingsManager* instance

Settings dialogue (Edit -> Settings) manager. Use it for adding own settings to the dialogue

commandLineArgs()
Dictionary of command line arguments, passed on Enki start

project()
Project support core module

:`enki.core.project.Project`

`enki.core.core.`**core = <enki.core.core.Core object>**
Core instance. It is accessible as:

```
from enki.core.core import core
core.anyMethod()
```

1.2 mainwindow — Main window of the UI. Fills main menu.

Module contains *enki.core.mainwindow.MainWindow* implementation

class `enki.core.mainwindow.`**MainWindow**
Bases: `QMainWindow`

Main UI window

Class creates window elements, fills main menu with items.

If you need to access to some existing menu items - check action path in the class constructor, than use next code:

```
core.actionManager().action("mFile/aOpen").setEnabled(True)
core.actionManager().action("mFile/aOpen").triggered.connect(self.myCoolMethod)
```

MainWindow instance is accessible as:

```
from enki.core.core import core
core.mainwindow()
```

Created by the core

hideAllWindows
hideAllWindows()

Signal emitted, when user toggled "Hide all" . Dock widgets are closed automatically, but other widgets, i.e. search widget, must catch this signal and close themselves.

directoryDropt
directoryDropt()

Signal emitted, when user drag-n-dropt directory to main windowd. FileBrowser shows directory

del_()
Explicitly called destructor

menuBar()
Reference to menuBar

topToolBar()
Top tool bar. Contains main menu, position indicator, etc

statusBar()
Return main window status bar. It is located on the top tool bar

setWorkspace(*workspace*)
Set central widget of the main window. Normally called only by core when initializing system

defaultTitle()
Default title. Contains name and version

centralLayout()
Layout of the central widget. Contains Workspace and search widget

appendMessage(*text*, *timeoutMs=10000*)
Append message to the queue. It will be shown as non-modal at the bottom of the window. Use such notifications, which are too long or too important for status bar but, not so important, to interrupt an user with QMessageBox

closeEvent(*event*)
NOT A PUBLIC API Close event handler. Shows save files dialog. Cancels close, if dialog was rejected

loadState()
Restore window state from main_window_state.bin and config. Called by the core after all plugins had been initialized

sizeHint()

dragEnterEvent(*event*)
QMainWindow method reimplementation. Say, that we are ready to accept dragged urls

dropEvent(*event*)
QMainWindow method reimplementation. Open dropt files

addDockWidget(*area*, *dock*)

removeDockWidget(*dock*)

restoreState(*state*)

1.3 actionmanager — Provides text path based access to main menu actions

Use this module for adding own actions to the main menu

Shortcuts are configured by appshortcuts plugin

class `enki.core.actionmanager.`**`ActionMenuBar`**(*parent*, *actionManager*)
Bases: `QMenuBar`

Menu bar implementation. Contains actions, managed by ActionManager. Instance is created by MainWindow

class `enki.core.actionmanager.`**`ActionManager`**(*parent=None*)
Bases: `QObject`

Class provides text path based access to main menu actions

`actionInserted`
actionInserted(action)

Signal emitted, when new action has been inserted to the menu

`actionChanged`
actionChanged(action)

Signal emitted, when some action datahas been changed

`actionRemoved`
actionRemoved(action)

Signal emitted, when action has been removed from the menu

`del_`()

`action`(*path*)
Get action by its path. i.e. actionManager.action("mFile/mClose/aAll")

`menu`(*path*)
Get action by its path. i.e. actionManager.action("mFile/mClose/aAll")

`path`(*action*)
Get action path by reference to action

`allActions`()
Reqursive list of existing actions

`addAction`(*path*, *action*, *icon=<QIcon object>*, *shortcut=None*)
Add new action to the menu. Returns created QAction object. `action` might be string text or QAction instance.

`removeAction`(*pathOrAction*, *removeEmptyPath=False*)
Remove action from the menu

`addMenu`(*path*, *text*, *icon=<QIcon object>*)
Add menu to the main menu or submenu of main menu

`removeMenu`(*action*, *removeEmptyPath=False*)
Remove menu. If removeEmptyPath is True - remove also empty parent menus

`parentAction`(*action*)
Parent action of the action

`children`(*action*)
List of children of action

defaultShortcut (*action*)
Get actions default shortcut

setDefaultShortcut (*action*, *shortcut*)
Set actions default shortcut

1.4 workspace — Open documents and manage it

Terminology:

Workspace - main working area, where documents are placed.

Document - opened file, widget on workspace. *enki.core.document.Document* instance

enki.core.workspace.Workspace

class enki.core.workspace.**Workspace** (*mainWindow*)
Bases: QStackedWidget

Class manages set of opened documents, allows to open new file

instance is accessible as:

```
from enki.core.core import core
core.workspace()
```

documentOpened
documentOpened(*enki.core.document.Document*)

Signal emitted, when document has been created, i.e. textual file opened, or some other document added to workspace

documentClosed
documentClosed(*enki.core.document.Document*)

Signal emitted, when document was closed

currentDocumentChanged
currentDocumentChanged(*enki.core.document.Document* old, *enki.core.document.Document* current)

Signal emitted, when current document changed, i.e. user selected another document, new document opened, current closed

modificationChanged
modificationChanged(document, modified)

Signal emitted, when modified state of a document had been changed (file edited, or saved) Bool parameter contains new value

Convenience signal, which retransmits original signal, sent by the document

cursorPositionChanged
cursorPositionChanged(document)

Signal emitted, when cursor position has been changed

Convenience signal, which retransmits original signal, sent by the document

textChanged
textChanged(document)

Signal emitted, when text has been chagned

Convenience signal, which retransmits original signal, sent by the document

languageChanged
languageChanged(document, language)

Signal emitted, when highlighting (programming) language of a file has been changed

Convenience signal, which retransmits original signal, sent by **'Qutepart<qutepart.rtfd.org>'_**

indentWidthChanged
indentWidthChanged(document, width)

Signal emitted, when indentation with has been changed

Convenience signal, which retransmits original signal, sent by the document

indentUseTabsChanged
indentUseTabsChanged(document, use)

Signal emitted, when indentation mode has been changed

Convenience signal, which retransmits original signal, sent by the document

eolChanged
eolChanged(document, eol)

Signal emitted, when EOL mode has been changed

Convenience signal, which retransmits original signal, sent by the document

escPressed
escPressed()

Signal emitted, when Esc pressed in the editor. Search widget closes themselves on this signal

del_()
Terminate workspace. Called by the core to clear actions

eventFilter(*obj*, *event*)

keyPressEvent(*event*)

currentDocument()
Returns currently active (focused) document. None, if no documents are opened

setCurrentDocument(*document*)
Select active (focused and visible) document form list of opened documents

activateNextDocument()
Activate next document in the list

activatePreviousDocument()
Activate previous document in the list

focusCurrentDocument()
Set focus (cursor) to current document.

Used if user has finished work with some dialog, and, probably, want's to edit text

goTo(*filePath*, *absPos=None*, *line=None*, *column=None*, *selectionLength=None*)
Open file, activate it, and go to specified position. Select text after position, if necessary.

selectionLength specifies, how much characters should be selected

openFile(*filePath*)
Open file.

Return document, if opened, None otherwise

Open modal message box, if failed to open the file

openFiles (*filePaths*)
Open files.

Open modal message box and stop opening files, if failed to open any file

findDocumentForPath (*filePath*)
Try to find document for path. Fimilar to open(), but doesn't open file, if it is not opened On Unix may return file, for which path is not equal, if soft or hards links are used Return None, if not found

createEmptyNotSavedDocument (*filePath=None*)
Create empty not saved document. Used on startup, if no file was specified, and after File->New file has been triggered

documents ()
Get list of opened documents (*enki.core.document.Document* instances)

closeDocument (*document*)
Close opened file, remove document from workspace and delete the widget.

Ask for confirmation with dialog, if modified.

askToCloseAll ()
If have unsaved documents, ask user to save it and close all. Will save documents, checked by user.

Returns True, if user hasn't pressed Cancel Close

closeAllDocuments ()
Close all documents

If there are not saved documents, dialog will be shown.

Handler of File->Close->All

Returns True, if all files had been closed, and False, if save dialog rejected

If hideMainWindow is True, main window will be hidden, if user hadn't pressed "Cancel Close"

forceCloseAllDocuments ()
Close all documents without asking user to save

1.5 document — Opened file representation

class enki.core.document.**Document** (*parentObject*, *filePath*, *createNew=False*)
Bases: QWidget

Document is a opened file representation.

It contains file management methods and uses Qutepart as an editor widget. Qutepart is available as `qutepart` attribute.

documentDataChanged
documentDataChanged()

Signal emitted, when document icon or toolTip has changed (i.e. document has been modified externally)

del_ ()
Explicytly called destructor

isExternallyModified()
Check if document's file has been modified externally.

This method does not do any file system access, but only returns cached info

isExternallyRemoved()
Check if document's file has been deleted externally.

This method does not do any file system access, but only returns cached info

isNeverSaved()
Check if document has been created, but never has been saved on disk

filePath()
Return the document file absolute path.

`None` if not set (new document)

fileName()
Document file name without a path.

`None` if not set (new document)

setFilePath(*newPath*)
Change document file path.

Used when saving first time, or on Save As action

saveFile()
Save the file to file system.

Show QFileDialog if file name is not known. Return False, if user cancelled QFileDialog, True otherwise

saveFileAs()
Ask for new file name with dialog. Save file

reload()
Reload the file from the disk

If child class reimplemented this method, it MUST call method of the parent class for update internal bookkeeping

modelToolTip()
Tool tip for the opened files model

modelIcon()
Icon for the opened files model

invokeGoTo()
Show GUI dialog, go to line, if user accepted it

printFile()
Print file

1.6 config — Load and save settings

class enki.core.config.**Config**(*enableWriting*, *filePath*)
Settings storage.

This class stores core settings. Plugins are also allowed to store its settings here.

Instance is accessible as:

```
from enki.core.core import core
core.config()
```

Created by *enki.core.core.Core*

Use this object as a dictionary for read and write options. Example:

```
font = core.config()["Qutepart"]["DefaultFont"]  # read option
core.config()["Qutepart"]["DefaultFont"] = font  # write option
```

See also get() and set() methods

You SHOULD flush config, when writing changed settings is finished. Example:

```
core.config().flush()
```

Usually flushing is done by *enki.core.uisettings.UISettingsManager*

reload()
Reload config from the disk

get(*name*)
Get option by slash-separated path. i.e.

```
font = core.config().get("Editor/DefaultFont")
```

Raises KeyError if not found

set(*name*, *value*)
Set option by slash-separated path. i.e.

```
core.config().get("Editor/DefaultFont") = font
```

clear()
Clear the config

flush()
Flush config to the disk. Does nothing, if *enableWriting* is *False* (probably default config is opened)

1.7 uisettings — Settings dialogue

Module provides GUI to edit settings. This GUI may be used by other core modules and by plugins.

1.7.1 Conception

Settings dialogue subsystem consists of 3 major entities:

- *UISettings.ui* GUI form. Contains of controls.
- *Option classes.

  Every object of the class links together control on GUI and option in the config file. It loads its option from config to GUI, and saves from GUI to config.

  config may be either *enki.core.config.Config* or python dictionary
- *enki.core.uisettings.UISettingsManager*. Invokes the dialogue. Emits signals when Plugins shall add own settings to the dialogue and when Plugins shall apply settings

Edit the diagramm

1.7.2 GUI dialog invocation workflow

1. Enki starts. Each plugin connects themselves to **UISettingsManager.aboutToExecute**
2. An user clicks *Settings->Settings*
3. UISettings.ui are created
4. *enki.core.uisettings.UISettingsManager* emits **aboutToExecute**
5. Each plugins adds own options to the dialogue
6. Each option loads its value from the *enki.core.config.Config*
7. The user edits settigns
8. The user clicks *OK*
9. Each option writes it's new value to the config
10. *enki.core.uisettings.UISettingsManager* emits **dialogAccepted**
11. Each plugin applies own settings

1.7.3 Adding new settings

If you need to add own settings to UISettings dialog, you should:

1. Connect to UISettingsManager.aboutToExecute and UISettingsManager.dialogAccepted
2. Add controls to the dialog. You may edit UISettings.ui or add your controls dynamically during dialog creation (connect to UISettingsManager.aboutToExecute for adding dynamically)
3. Create *Option class instance for every configurable option on UISettingsManager.aboutToExecute

1.7.4 Classes

Main classes:

- *enki.core.uisettings.UISettings* - settings dialogue. Created when shall be executed
- *enki.core.uisettings.UISettingsManager* - manager. Exists always

Option types:

- *enki.core.uisettings.CheckableOption* - bool option, CheckBox
- *enki.core.uisettings.TextOption* - string option, line edit
- *enki.core.uisettings.ListOnePerLineOption* - list of strings option, text edit
- *enki.core.uisettings.NumericOption* - numeric option, any numeric control
- *enki.core.uisettings.ColorOption* - color option, button
- *enki.core.uisettings.FontOption* - font option, button
- *enki.core.uisettings.ChoiseOption* - string from the set option, combo box

class enki.core.uisettings.**Option**(*dialog*, *config*, *optionName*, *control*)
Base class for all Options. Every class knows control on UISettings form, configuration option name, and can load/save the option

Do not create dirrectly, use *Option classes

load()
Load the value from config to GUI. To be implemented by child classes

save()
Save the value from GUI to config. To be implemented by child classes

class enki.core.uisettings.**CheckableOption**(*dialog*, *config*, *optionName*, *control*)
Bases: *enki.core.uisettings.Option*

Bool option.

Control may be QCheckBox, QGroupBox or other control, which has .isChecked() and .setChecked()

load()
Load the value from config to GUI

save()
Save the value from GUI to config

class enki.core.uisettings.**TextOption**(*dialog*, *config*, *optionName*, *control*)
Bases: *enki.core.uisettings.Option*

Text option

Control may be QLineEdit

load()
Load the value from config to GUI

save()
Save the value from GUI to config

class enki.core.uisettings.**ListOnePerLineOption**(*dialog*, *config*, *optionName*, *control*)
Bases: *enki.core.uisettings.Option*

List of strings. One item per line.

Control may be QPlainTextEdit

load()
Load the value from config to GUI

save()
Save the value from GUI to config

class enki.core.uisettings.**NumericOption**(*dialog*, *config*, *optionName*, *control*)
Bases: *enki.core.uisettings.Option*

Numeric option.

Control may be QSlider or other control, which has .value() and .setValue() methods

load()
Load the value from config to GUI

save()
Save the value from GUI to config

class enki.core.uisettings.**ColorOption**(*dialog*, *config*, *optionName*, *control*)
Bases: *enki.core.uisettings.Option*

Color option

Control must be enki.widgets.ColorButton

load()
Load the value from config to GUI

save()
Save the value from GUI to config

class enki.core.uisettings.**FontOption**(*dialog*, *config*, *familyOption*, *sizeOption*, *editControl*, *buttonControl*)
Bases: *enki.core.uisettings.Option*

Font option.

Option has 2 controls:

- QLineEdit is an example of font
- Button is used for open font dialogue.

This option opens Font dialogue automatically, when button has been clicked

load()
Load the value from config to GUI

save()
Save the value from GUI to config

class enki.core.uisettings.**ChoiseOption**(*dialog*, *config*, *optionName*, *controlToValue*)
Bases: *enki.core.uisettings.Option*

This option allows to choose value from few possible.

It is presented as set of QRadioButton's

controlToValue dictionary contains mapping *checked radio button name: option value*

load()
Load the value from config to GUI

save()
Save the value from GUI to config

class enki.core.uisettings.**UISettings**(*parent*)
Bases: QDialog

Settings dialog widget

appendPage(*path*, *widget*, *icon=None*)
Append page to the tree. Called by a plugin for creating own page. Example:

```
widget = MitSchemeSettings(dialog)
dialog.appendPage(u"Modes/MIT Scheme", widget, QIcon(':/enkiicons/languages/scheme.png'))
```

appendOption(*option*)
Append *Option instance to the list of options

on_twMenu_itemSelectionChanged()

class enki.core.uisettings.**UISettingsManager**
Bases: QObject

Add to the main menu *Settings->Settings* action and execute settings dialogue

aboutToExecute
aboutToExecute(*enki.core.uisettings.UISettings*)

Signal emitted, when dialog is about to be executed. Plugins shall add own settings to the dialogue

dialogAccepted
accepted()

Signal emitted, when dialog has been accepted. Plugins shall save and apply settings

del_()

1.8 filefilter — Filter temporary files from FS views

File browser, Locator, and probably other functionality shall ignore temporaty files, such as *.bak, *.o for C++, *.pyc for Python.

This module provides regular expression, which tests, if file shall be ignored. Regexp is constructed from patterns, configurable in the settings

class enki.core.filefilter.**FileFilter**
Bases: QObject

Module implementation

regExpChanged
regExpChanged()

Signal emitted, when regExp has changed

regExp()
Get negative filer reg exp.

If file name matches it, ignore this file

1.9 locator — Locator dialog and functionality

Implements widget, which appears, when you press Ctrl+L and it's functionality

Contains definition of AbstractCommand and AbstractCompleter interfaces

exception enki.core.locator.**InvalidCmdArgs**
Bases: exceptions.UserWarning

class enki.core.locator.**AbstractCommand**
Bases: QObject

Base class for Locator commands.

Inherit it to create own commands. Than add your command with Locator.addCommandClass()

Public attributes:

- `command` - Command text (first word), i.e. `f` for Open and `s` for Save
- `signature` - Command signature. Shown in the Help. Example: `[f] PATH [LINE]`
- `description` - Command description. Shown in the Help. Example: `Open file. Globs are supported`
- `isDefaultCommand` - If True, command is executed if no other command matches. Must be `True` for only 1 command. Currently it is FuzzyOpen
- `isDefaultPathCommand` - If True, command is executed if no other command matches and text looks like a path. Must be `True` for only 1 command. Currently it is Open

- `isDefaultNumericCommand` - If True, command is executed if no other command matches and text looks like a number. Must be `True` for only 1 command. Currently it is GotoLine

command = NotImplemented

signature = NotImplemented

description = NotImplemented

isDefaultCommand = False

isDefaultPathCommand = False

isDefaultNumericCommand = False

updateCompleter
Signal is emitted by the command after completer has changed.

Locator will call `completer()` method again after this signal. Use this signal only for commands for which completer is changed dynamically, i.e. FuzzyOpen loads project files asyncronously. For the majority of commands it is enough to implement `completer()` method.

terminate()
Terminate the command if necessary.

Default implementation does nothing

static **isAvailable**()
Check if command is available now.

i.e. SaveAs command is not available, if no files are opened

setArgs()
Set command arguments.

This method can be called multiple times while the user edits the command. Raise `InvalidCmdArgs` if the arguments are invalid.

completer()
:*enki.core.locator.AbstractCompleter* instance for partially typed command.

Return `None`, if your command doesn't have completer, or if completion is not available now.

onCompleterLoaded(*completer*)
This method is called after `completer.load()` method has finished. Completer instance created in `completer()` is passed as parameter.

The command can use loaded data from the completer now.

onItemClicked(*fullText*)
Item in the completion tree has been clicked.

Update internal state. After this call Locator will execute the command if isReadyToExecute or update line edit text with lineEditText() otherwise.

lineEditText()
Get text for Locator dialog line edit.

This method is called after internal command state has been updated with onItemClicked()

isReadyToExecute()
Check if command is ready to execute.

It is ready, when it is complete (contains all mandatory arguments) and arguments are valid

execute()
Execute the command

class enki.core.locator.**AbstractCompleter**
Completer for Locator.

Provides:

- inline completion
- command(s) description
- status and any other information from command
- list of possible completions

If `mustBeLoaded` class attribute is True, `load()` method will be called in a thread.

mustBeLoaded = False

load(*stopEvent*)
Load necessary data in a thread. This method must often check `stopEvent` `threading.Event` and return if it is set.

rowCount()
Row count for TreeView

columnCount()
Column count for tree view. Default is 1

text(*row*, *column*)
Text for TreeView item

icon(*row*, *column*)
Icon for TreeView item. Default is None

isSelectable(*row*, *column*)
Check if item is selectable with arrow keys

inline()
Inline completion.

Shown after cursor. Appedned to the typed text, if Tab is pressed

getFullText(*row*)
Row had been clicked by mouse. Get inline completion, which will be inserted after cursor

autoSelectItem()
Item, which shall be auto-selected when completer is applied.

If not None, shall be `(row, column)`

Default implementation returns None

class enki.core.locator.**StatusCompleter**(*text*)
Bases: *enki.core.locator.AbstractCompleter*

AbstractCompleter implementation, which shows status message

rowCount()
AbstractCompleter method implementation

Return count of available commands

text(*row*, *column*)
AbstractCompleter method implementation

Return command description

enki.core.locator.**splitLine**(*text*)
Split text onto words Return list of (word, endIndex)

class enki.core.locator.**Locator**
Bases: QObject

del_()

addCommandClass(*commandClass*)
Add new command to the locator. Shall be called by plugins, which provide locator commands

removeCommandClass(*commandClass*)
Remove command from the locator. Shall be called by plugins when terminating it

1.10 json — Utility functions for loading and saving JSON

Enki uses JSON for storing settings.

This module is a wrapper around standard json module, which catches and shows exceptions when loading and saving JSON files

enki.core.json_wrapper.**load**(*filePath*, *dataName*, *defaultValue*)
Try to load data from JSON file. If something goes wrong - shows warnings to user. But, not if file not exists. dataName used in error messages. i.e. 'cursor positions', 'file browser settings' defaultValue is returned, if file not exists or if failed to load and parse it

enki.core.json_wrapper.**dump**(*filePath*, *dataName*, *data*, *showWarnings=True*)
Try to save data to JSON file. Show exceptions on main window and print it, if something goes wrong

CHAPTER 2

enki.lib

Code (but not widgets), which is not used by core, but, may be used by more than one plugin.

2.1 buffpopen — Buffered subprocess. Popen implementation

This implementation allows to read and write output without blocking

class enki.lib.buffpopen.**BufferedPopen**(*command*)

Bufferred version of Popen. Never locks, but uses unlimited buffers. May eat all the system memory, if something goes wrong.

start(*args*)

Start the process

stop()

Stop the process

isAlive()

Check if process is alive

write(*text*)

Write data to the subprocess

readOutput()

Read stdout data from the subprocess

2.2 htmldelegate — QStyledItemDelegate delegate. Draws HTML

enki.lib.htmldelegate.**htmlEscape**(*text*)

Replace special HTML symbols with escase sequences

class enki.lib.htmldelegate.**HTMLDelegate**(*parent=None*)

Bases: QStyledItemDelegate

QStyledItemDelegate implementation. Draws HTML

http://stackoverflow.com/questions/1956542/how-to-make-item-view-render-rich-html-text-in-qt/1956781#1956781

paint(*painter*, *option*, *index*)

QStyledItemDelegate.paint implementation

sizeHint(*option*, *index*)
QStyledItemDelegate.sizeHint implementation

2.3 pathcompleter — Path completer for Locator

enki.lib.pathcompleter.**makeSuitableCompleter**(*text*)
Returns PathCompleter if text is normal path or GlobCompleter for glob

class enki.lib.pathcompleter.**AbstractPathCompleter**(*text*)
Bases: *enki.core.locator.AbstractCompleter*

Base class for PathCompleter and GlobCompleter

mustBeLoaded = True

rowCount()
Row count in the list of completions

text(*row*, *column*)
Item text in the list of completions

icon(*row*, *column*)
Item icon in the list of completions

isSelectable(*row*, *column*)

getFullText(*row*)
User clicked a row. Get inline completion for this row

class enki.lib.pathcompleter.**PathCompleter**(*text*)
Bases: *enki.lib.pathcompleter.AbstractPathCompleter*

Path completer for Locator. Supports globs

Used by Open command

load(*stopEvent*)

inline()
Inline completion. Displayed after the cursor

class enki.lib.pathcompleter.**GlobCompleter**(*text*)
Bases: *enki.lib.pathcompleter.AbstractPathCompleter*

Path completer for Locator. Supports globs, does not support inline completion

Used by Open command

load(*stopEvent*)

CHAPTER 3

enki.widgets

Set of reusable widgets.

3.1 dockwidget — Extended QDockWidget for Enki main window

This class adds next features to QDockWidget:

- has action for showing and focusing the widget
- closes themselves on Esc
- title bar contains QToolBar

class `enki.widgets.dockwidget.`**`DockWidget`**(*parentObject*, *windowTitle*, *windowIcon=<QIcon object>*, *shortcut=None*)

Bases: `QDockWidget`

Extended QDockWidget for Enki main window

`closed`
closed()

Signal emitted, when dock is closed

`shown`
shown()

Signal emitted, when dock is shown

`keyPressEvent`(*event*)
Catch Esc. Not using QShortcut, because dock shall be closed, only if child widgets haven't catched Esc event

`showAction`()
Action shows the widget and set focus on it.

Add this action to the main menu

`titleBarWidget`()
QToolBar on the title.

You may add own actions to this tool bar

`closeEvent`(*event*)
Widget was closed

showEvent (*event*)
Widget was shown

3.2 lineedit — Extended QLineEdit. Supports prompt and Clear button

This class:

- shows prompt text, which is visible only if line edit is empty
- shows Clear button, which is visible only when widget is not empty

Don't use this class if you need classical line edit

enki.widgets.lineedit.**tr**(*text*)

class enki.widgets.lineedit.**LineEdit**(*parent*)
Bases: QLineEdit

Extended QLineEdit. Supports prompt and Clear button

clearButtonClicked
clearButtonClicked()

Signal emitted, when Clear button has been clicked

promptText()
Current prompt text

setPromptText(*prompt*)
Set prompt text

paintEvent(*event*)
QLineEdit.paintEvent implementation. Draws prompt

resizeEvent(*event*)
QLineEdit.resizeEvent implementation Adjusts Clear button position

setClearButtonVisible(*visible*)
Set Clear button visible

3.3 colorbutton — Button, which is used for configuring colors

Button shows selected color as own icon and opens QColorDialog when clicked

enki.widgets.colorbutton.**tr**(*text*)
Dummy tr() implementation

class enki.widgets.colorbutton.**ColorButton**(*colorOrParent*, **args*)
Bases: QToolButton

Button, which is used for configuring colors

colorChanged
colorChanged(color)

Signal emitted, after current color has changed

color()
Currently selected color

setColor(*color*)
Set color. Update button icon

3.4 termwidget — Terminal emulator widget

Shows intput and output text. Allows to enter commands. Supports history.

This widget only provides GUI, but does not implement any system terminal or other functionality

class enki.widgets.termwidget.**TermWidget**(*font*, **args*)
Bases: QWidget

Widget wich represents terminal. It only displays text and allows to enter text. All highlevel logic should be implemented by client classes

terminate()

eventFilter(*obj*, *event*)

setLanguage(*language*)
Set highlighting language for input widget

execCommand(*text*)
Save current command in the history. Append it to the log. Execute child's method. Clear edit line.

childExecCommand(*text*)
Reimplement in the child classes to execute enterred commands

appendOutput(*text*)
Appent text to output widget

appendError(*text*)
Appent error text to output widget. Text is drawn with red background

appendHint(*text*)
Appent error text to output widget. Text is drawn with red background

clear()
Clear the widget

isCommandComplete(*text*)
Executed when Enter is pressed to check if widget should execute the command, or insert newline.

Implement this function in the child classes.

CHAPTER 4

enki.plugins

This package contains plugins, which extend the core with additional functionality.

Plugins do not export any public API and are not included to this docs. But code consists docstrings.

Every plugin is optional, therefore, no other modules are allowed to depend on plugin.

e

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W