jdTextEdit

Release 11.0

JakobDev

Oct 04, 2023

CONTENTS

W.I.P.

CHAPTER

ONE

GENERAL

1.1 Commandline Arguments

jdTextEdit has currently the following Commandline Arguments:

-p, --portable Run in portable mode

--data-dir DATADIR Sets the data directory

--disable-plugins Start without plugins

--no-session-restore Start without restoring the session

--disable-updater Start with disabled Updater

--distribution-file PATH Sets custom distribution.json

--language LANGCODE Starts jdTextEdit in the given language and ignore the system language or the user settings

1.2 Environment Variables

jdTextEdit has currently the following Environment Variables:

JDTEXTEDIT_DATA_PATH Set the data path of jdTextEdit.

JDTEXTEDIT_DISABLE_UPDATER Set this variable to disable the Updater of jdTextEdit

1.3 userChrome.css

jdTextEdit supports custom styling with CSS. Just place a file called userChrome.css in the data directory. For information about creating a style sheet take a look at the Qt Documentation.

CHAPTER

TWO

PLUGIN DEVELOPMENT

2.1 Getting Started

First create a new directory in the plugins directory with a __init__.py inside. The __init__.py must have the following functions:

```
def main(env):
    #Write your code here

def getID():
    return "myPlugin"

def getName():
    return "My Plugin"

def getVersion():
    return "1.0"

def getAuthor():
    return "John Doe"
```

2.2 PluginAPI

The plugin API contains the following functions:

```
addLanguage(language: LanguageBase)
```

Adds a language to jdTextEdit

```
getEditorSignals() -> EditorSignals:
```

Returns the Editor Signals.

```
getMainWindowSignals() -> MainWindowSignals:
```

Returns the Mainwindow Signals.

```
getApplicationSignals() -> ApplicationSignals
```

Returns the Application Signals.

```
addSettingsTab(tab: SettingsTabBase)
```

Adds a Settings Tab.

```
registerSetting(key: str,value: str)
```

Register a new Setting.

```
addTranslationDirectory(path: str)
```

Adds a directory which contains translations.

```
addBigFilesCheckBox(setting: str, text:str)
```

Adds a Checkbox to the Big files Settings Tab.

```
addTheme(theme: ThemeBase)
```

Adds a Theme.

```
addSidebarWidget(widget: SidebarWidgetBase)
```

Adds a Sidebar Widget.

```
addAction(action: QAction)
```

Adds a Action to the list in the settings menu.

2.3 Add Language

To add a new language you need to create a class that inherit from LanguageBase.

```
from jdTextEdit.api.LanguageBase import LanguageBase
from PyQt5.Qsci import QsciLexerPython, QsciAPIs

class MyLanguage(LanguageBase):
    def getLexer(self):
        return QsciLexerPython()

    def getName(self):
        return "My Language"

    def getID(self):
        return "myplugin.mylanguage"

    def getExtensions(self):
        return ["py"]

    def getStarttext(self):
        return ["#!/usr/bin/python"]

    def getAPI(self,lexer):
```

(continues on next page)

(continued from previous page)

```
        api = QsciAPIs(lexer)
        api.add("Hello")
        api.prepare()
```

Here's a description of all functions

```
getLexer()
```

Returns a QsciScintilla Lexer. You can use a existing one or write one by yourself.

```
getName()
```

Retruns the Name of your Language.

```
getID()
```

Returns the ID of your Language. The ID is used to identify your Lanmguage, so make sure it is used by nobody else.

The following functions are optional.

```
getExtensions()
```

Returns a list with all extension for you filetype. e.g. if the list contains "mylang" and the user open a file with the name "text.mylang" your language will be set.

```
getStarttext()
```

Some language starts every or the moist time with a special text. e.g. all XML files start with <?xml. This function returns a list which all known starttexts for your language.

```
getAPI(lexer)
```

Retruns the API for the language.

2.4 Editor Signals

editorInit Emited when a Editor Widget is created

Arguments

- CodeEdit: The Widget which is created

openFile Emited when a File is opened

Arguments

- CodeEdit: The Widget whre the file is opened
- str: The path of the file

linesChanged Emited when the a line is changed. Same as the QScintilla Signal.

Arguments

- CodeEdit: The Widget where the Line is changed

textChanged Emited when the Text is changed. Same as the QScintilla Signal.

Arguments

- CodeEdit: The Widget where the Text is changed

indicatorClicked Emited when a indicator is clicked is changed. Same as the QScintilla Signal.

Arguments

- CodeEdit: The Widget where the indicatir is clicked
- int: Line
- iint Index
- Qt::KeyboardModifiers: state

indicatorReleased Emited when a indicator is clicked is released. Same as the QScintilla Signal.

Arguments

- CodeEdit: The Widget where the indicatir is realeased
- int: Line
- iint Index
- Qt::KeyboardModifiers: state

contextMenu Emited when a the user try to open a context menu. Accept the event that given as arg to prevent the original menu to open, so you can create your own menu.

Arguments

- CodeEdit: The Widget where the menu is opened
- QEvent: The context menu event

settingsChanged Emited when a the settingsof a widget are changed.

Arguments

- CodeEdit: The Widget where the settings are changed
- Settings: The new settings

languageChanged Emited when a the Language is changed.

Arguments

- CodeEdit: The Widget where the Language is changed
- LanguageBase: The Language. None if the Language is removed.

saveSession Emiited when the session is saved.

Arguments

- CodeEdit: The Widget where the session is saved
- dict: A dict in which you can put your custom properties

restoreSession Emiited when the session is restored.

Arguments

- CodeEdit: The Widget where the session is restored
- dict: Contains all the session properties

2.5 TabWidget Signals

tabCreated Emited when a new tab is created

Arguments

- EditTabWidget: The Widget in which the new tab is opened
- EditContainer: The Widget of the new tab

tabClosed Emited when a tab is closed

Arguments

- EditTabWidget:; The Widget in which the tab is closed

2.6 Mainwindow Signals

windowInit Emited when a Mainwindow is created.

Arguments

- MainWindow: The Mainwindow which is created

2.7 Application Signals

settingsChanged Emited when the settings are changed.

Arguments

- Settings: The new settings

CHAPTER

THREE

DISTRIBUTION

3.1 Default Data

You can create a directory with the name default_data inside the instalation directory. If a user which has no data directory starts jdTextEdit the content of default_data will be copied into the users data directory.

3.2 Global Commands and Macros

You can add Global Commands and Macros. Just copy the commands.json and the macros.json from the data directory to the instalation directory and it will be aviable for all users.

3.3 distribution.json

distribution.json is a JSON file which is placed in the Program Directory. It has some options for people who are building packages. Currently the following options are available:

You don't need to set all options. Just set the options you need and remove the others from the file.

You can use enviroment variables in all paths.

```
{
    "enableUpdater": false,
    "_description_": "Enable or disable the updater",

    "dataDirectory": "~/.edit",
    "_description_": "Sets the Data Directory.",

    "aboutMessage": "Hello from John Doe",
    "_description_": "This message is shown in the About Window.",

    "templateDirectories": [],
    "_description_": "Add template directories",

    "enableTranslationWarning": false,
    "_description_": "Show a warning when no language is build"
}
```

CHAPTER

FOUR

GETTING INVOLVED

4.1 Translate jdTextEdit

You can translate jdTextEdit using Weblate.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search