

Introducing Envato Video. Create a professional video to share your story.

Discover our app



Free Preview

Sign In



Categories ▾

Learning Guides ▾

Forum

PYTHON

How to Create a Sublime Text 2 Plugin

by Will Bond 14 Nov 2011

52 Comments



12



Sublime Text 2 is a highly customizable text editor that has been increasingly capturing the attention of coders looking for a tool that is powerful, fast and modern.

Today, we're going to [recreate my popular Sublime plugin](#) that sends CSS through the [Nettuts+ Prefixr](#) API for easy cross-browser CSS.

When finished, you'll have a solid understanding of how the [Sublime Prefixr plugin](#) is written, and be equipped to start writing your own plugins for the editor!

Preface: Terminology and Reference Material

The extension model for [Sublime Text 2](#) is fairly full-featured.

The extension model for [Sublime Text 2](#) is fairly full-featured. There are ways to change the syntax highlighting, the actual chrome of the editor and all of the menus. Additionally, it is possible to create new build systems, auto-completions, language definitions, snippets, macros, key bindings, mouse bindings and plugins. All of these different types of modifications are implemented via files which are organized into packages.

A package is a folder that is stored in your `Packages` directory. You can access your Packages directory by clicking on the *Preferences > Browse Packages...* menu entry. It is also possible to bundle a package into a single file by creating a zip file and changing the extension to `.sublime-package`. We'll discuss packaging

a bit more further on in this tutorial.

Sublime comes bundled with quite a number of different packages. Most of the bundled packages are language specific. These contain language definitions, auto-completions and build systems. In addition to the language packages, there are two other packages: `Default` and `User`. The

`Default` package contains all of the standard key bindings, menu definitions, file settings and a whole bunch of plugins written in Python. The `User` package is special in that it is always loaded last. This allows users to override defaults by customizing files in their `User` package.

During the process of writing a plugin, the [Sublime Text 2 API reference](#) will be essential.

During the process of writing a plugin, the [Sublime Text 2 API reference](#) will be essential. In addition, the `Default` package acts as a good reference for figuring out how to do things and what is possible. Much of the functionality of the editor is exposed via *commands*. Any operation other than typing characters is accomplished via commands. By viewing the *Preferences > Key Bindings - Default* menu entry, it is possible to find a treasure trove of built-in functionality.

Now that the distinction between a plugin and package is clear, let's begin writing

our plugin.

Step 1 - Starting a Plugin

Sublime comes with functionality that generates a skeleton of Python code needed to write a simple plugin. Select the *Tools > New Plugin...* menu entry, and a new buffer will be opened with this boilerplate.

```
1 | import sublime, sublime_plugin
2 |
3 | class ExampleCommand(sublime_plugin.TextCommand):
4 |     def run(self, edit):
5 |         self.view.insert(edit, 0, "Hello, World!")
```

Here you can see the two Sublime Python modules are imported to allow for use of the API and a new command class is created. Before editing this and starting to create our own plugin, let's save the file and trigger the built in functionality.

When we save the file we are going to create a new package to store it in. Press **ctrl+s** (Windows/Linux) or **cmd+s** (OS X) to save the file. The save dialog will open to the `User` package. Don't save the file there, but instead browse up a folder and create a new folder named `Prefixr`.

```
01 Packages/  
02 ...  
03 - OCaml/  
04 - Perl/  
05 - PHP/  
06 - Prefixr/  
07 - Python/  
08 - R/  
09 - Rails/  
10 ...
```

Now save the file inside of the Prefixr folder as `Prefixr.py`. It doesn't actually matter what the filename is, just that it ends in `.py`. However, by convention we will use the name of the plugin for the filename.

Now that the plugin is saved, let's try it out. Open the Sublime console by pressing **ctrl+`**. This is a Python console that has access to the API. Enter the following Python to test out the new plugin:

```
1 | view.run_command('example')
```

You should see `Hello world` inserted into the beginning of the plugin file. Be sure to undo this change before we continue.

Step 2 - Command Types and Naming

For plugins, Sublime provides three different types of commands.

- *Text commands* provide access to the contents of the selected file/buffer via a `View` object
- *Window commands* provide references to the current window via a `Window` object
- *Application commands* do not have a reference to any specific window or file/buffer and are more rarely used

Since we will be manipulating the content of a CSS file/buffer with this plugin, we are going to use the `sublime_plugin.TextCommand` class as the basis of our custom Prefixr command. This brings us to the topic of naming command classes.

In the plugin skeleton provided by Sublime, you'll notice the class:

```
1 | class ExampleCommand(sublime_plugin.TextCommand):
```

When we wanted to run the command, we executed the following code in the console:

```
1 | view.run_command('example')
```

Sublime will take any class that extends one of the `sublime_plugin` classes (`TextCommand`, `WindowCommand` or `ApplicationCommand`), remove the suffix `Command` and then convert the `CamelCase` into `underscore_notation` for the command name.

Thus, to create a command with the name `prefixr`, the class needs to be `PrefixrCommand`.

```
1 | class PrefixrCommand(sublime_plugin.TextCommand):
```

Step 3 - Selecting Text

One of the most useful features of Sublime is the ability to have multiple selections.

Now that we have our plugin named properly, we can begin the process of grabbing CSS from the current buffer and sending it to the Prefixr API. One of the most useful features of Sublime is the ability to have multiple selections. As we are

grabbing the selected text, we need to write our plug into handle not just the first selection, but all of them.

Since we are writing a text command, we have access to the current view via `self.view`. The `sel()` method of the `View` object returns an iterable `RegionSet` of the current selections. We start by scanning through these for curly braces. If curly braces are not present we can expand the selection to the surrounding braces to ensure the whole block is prefixed. Whether or not our selection included curly braces will also be useful later to know if we can tweak the whitespace and formatting on the result we getback from the Prefixr API.

```
1 | braces = False
2 | sels = self.view.sel()
3 | for sel in sels:
4 |     if self.view.substr(sel).find('{') != -1:
5 |         braces = True
```

This code replaces the content of the skeleton `run()` method.

If we did not find any curly braces we loop through each selection and adjust the selections to the closest closing curly brace. Next, we use the built-in command `expand_selection` with the `to` arg set to `brackets` to ensure we have the complete contents of each CSS block selected.


```
1 | if not braces:
2 |     new_sels = []
3 |     for sel in sels:
4 |         new_sels.append(self.view.find('\}', sel.end()))
5 |     sels.clear()
6 |     for sel in new_sels:
7 |         sels.add(sel)
8 |     self.view.run_command("expand_selection", {"to": "brackets"})
```

If you would like to double check your work so far, please compare the source to the file `Prefixr-1.py` in the source code zip file.

Step 4 - Threading

To prevent a poor connection from interrupting other work, we need to make sure that the Prefixr API calls are happening in the background.

At this point, the selections have been expanded to grab the full contents of each CSS block. Now, we need to send them to the Prefixr API. This is a simple HTTP request, which we are going to use the `urllib` and `urllib2` modules for.

However, before we start firing off web requests, we need to think about how a potentially laggy web request could affect the performance of the editor. If, for some

reason, the user is on a high-latency, or slow connection, the requests to the Prefixr API could easily take a couple of seconds or more.

To prevent a poor connection from interrupting other work, we need to make sure that the Prefixr API calls are happening in the background. If you don't know anything about threading, a very basic explanation is that threads are a way for a program to schedule multiple sets of code to run seemingly at the same time. It is essential in our case because it lets the code that is sending data to, and waiting for a response from, the Prefixr API from preventing the rest of the Sublime user interface from freezing.

Step 5 - Creating Threads

We will be using the Python `threading` module to create threads. To use the threading module, we create a new class that extends `threading.Thread` called `PrefixrApiCall`. Classes that extend `threading.Thread` include a `run()` method that contains all code to be executed in the thread.

```
01 class PrefixrApiCall(threading.Thread):
02     def __init__(self, sel, string, timeout):
03         self.sel = sel
```

```

04         self.original = string
05         self.timeout = timeout
06         self.result = None
07         threading.Thread.__init__(self)
08
09     def run(self):
10         try:
11             data = urllib.urlencode({'css': self.original})
12             request = urllib2.Request('http://prefixr.com/api/index>
13                                     headers={'User-Agent': 'Sublime Prefixr'})
14             http_file = urllib2.urlopen(request, timeout=self.timeout)
15             self.result = http_file.read()
16             return
17
18         except (urllib2.HTTPError) as (e):
19             err = '%s: HTTP error %s contacting API% (__name__, st
20         except (urllib2.URLError) as (e):
21             err = '%s: URL error %s contacting API% (__name__, str
22
23         sublime.error_message(err)
24         self.result = False

```

Here we use the thread `__init__()` method to set all of the values that will be needed during the web request. The `run()` method contains the code to set up and execute the HTTP request for the Prefixr API. Since threads operate concurrently with other code, it is not possible to directly return values. Instead we set `self.result` to the result of the call.

Since we just started using some more modules in our plugin, we must add them to

the import statements at the top of the script.

```
1 import urllib
2 import urllib2
3 import threading
```

Now that we have a threaded class to perform the HTTP calls, we need to create a thread for each selection. To do this we jump back into the `run()` method of our `PrefixrCommand` class and use the following loop:

```
1 threads = []
2 for sel in sels:
3     string = self.view.substr(sel)
4     thread = PrefixrApiCall(sel, string,5)
5     threads.append(thread)
6     thread.start()
```

We keep track of each thread we create and then call the `start()` method to start each.

If you would like to double check your work so far, please compare the source to the file `Prefixr-2.py` in the source code zip file.

Step 6 - Preparing for Results

Now that we've begun the actual Prefixr API requests we need to set up a few last details before handling the responses.

First, we clear all of the selections because we modified them earlier. Later we will set them back to a reasonable state.

```
1 | self.view.sel().clear()
```

In addition we start a new `Edit` object. This groups operations for undo and redo. We specify that we are creating a group for the `prefixr` command.

```
1 | edit = self.view.begin_edit('prefixr')
```

As the final step, we call a method we will write next that will handle the result of the API requests.

```
1 | self.handle_threads(edit, threads, braces)
```

Step 7 - Handling Threads

At this point our threads are running, or possibly even completed. Next, we need to implement the `handle_threads()` method we just referenced. This method is going to loop through the list of threads and look for threads that are no longer running.

```
01 def handle_threads(self, edit, threads, braces, offset=0, i=0, dir=1):
02     next_threads = []
03     for thread in threads:
04         if thread.is_alive():
05             next_threads.append(thread)
06             continue
07         if thread.result == False:
08             continue
09         offset = self.replace(edit, thread, braces, offset)
10     threads = next_threads
```

If a thread is still alive, we add it to the list of threads to check again later. If the result was a failure, we ignore it, however for good results we call a new `replace()` method that we'll be writing soon.

If there are any threads that are still alive, we need to check those again shortly. In addition, it is a nice user interface enhancement to provide an activity indicator to show that our plugin is still running.

```

01  if len(threads):
02      # This animates a little activity indicator in the status area
03      before = i % 8
04      after = (7) - before
05      if not after:
06          dir = -1
07      if not before:
08          dir = 1
09      i += dir
10      self.view.set_status(prefixr, 'Prefixr [%s=%s]' % \
11                          (' ' * before, ' ' * after))
12
13      sublime.set_timeout(lambda: self.handle_threads(edit, threads,
14              braces, offset, i, dir), 100)
15      return

```

The first section of code uses a simple integer value stored in the variable `i` to move an `=` back and forth between two brackets. The last part is the most important though. This tells Sublime to run the `handle_threads()` method again, with new values, in another 100 milliseconds. This is just like the `setTimeout()` function in JavaScript.

The `lambda` keyword is a feature of Python that allows us to create a new unnamed, or anonymous, function.

The `sublime.set_timeout()` method requires a function or method and the

number of milliseconds until it should be executed. Without `lambda` we could tell it we wanted to run `handle_threads()`, but we would not be able to specify the parameters.

If all of the threads have completed, we don't need to set another timeout, but instead we finish our undo group and update the user interface to let the user know everything is done.

```
1 self.view.end_edit(edit)
2
3 self.view.erase_status('prefixr')
4 selections = len(self.view.sel())
5 sublime.status_message('Prefixr successfully run on %s selection%s%'
6     (selections, " if selections == 1 else s'))
```

If you would like to double check your work so far, please compare the source to the file `Prefixr-3.py` in the source code zip file.

Step 8 - Performing Replacements

With our threads handled, we now just need to write the code that replaces the original CSS with the result from the Prefixr API. As we referenced earlier, we are

going to write a method called `replace()`.

This method accepts a number of parameters, including the `Edit` object for undo, the thread that grabbed the result from the Prefixr API, if the original selection included braces, and finally the selection offset.

```
1 def replace(self, edit, thread, braces, offset):
2     sel = thread.sel
3     original = thread.original
4     result = thread.result
5
6     # Here we adjust each selection for any text we have already i
7     if offset:
8         sel = sublime.Region(sel.begin() + offset,
9                             sel.end() + offset)
```

The offset is necessary when dealing with multiple selections. When we replace a block of CSS with the prefixed CSS, the length of that block will increase. The offset ensures we are replacing the correct content for subsequent selections since the text positions all shift upon each replacement.

The next step is to prepare the result from the Prefixr API to be dropped in as replacement CSS. This includes converting line endings and indentation to match the current document and original selection.

```
1 result = self.normalize_line_endings(result)
2 (prefix, main, suffix)= self.fix_whitespace(original, result, sel,
3     braces)
4 self.view.replace(edit, sel, prefix+ main + suffix)
```

As a final step we set the user's selection to include the end of the last line of the new CSS we inserted, and then return the adjusted offset to use for any further selections.

```
1 end_point = sel.begin() + len(prefix) + len(main)
2 self.view.sel().add(sublime.Region(end_point, end_point))
3
4 return offset + len(prefix + main + suffix) - len(original)
```

If you would like to double check your work so far, please compare the source to the file `Prefixr-4.py` in the source code zip file.

Step 9 - Whitespace Manipulation

We used two custom methods during the replacement process to prepare the new CSS for the document. These methods take the result of Prefixr and modify it to match the current document.

`normalize_line_endings()` takes the string and makes sure it matches the line endings of the current file. We use the `Settings` class from the Sublime API to get the proper line endings.

```
1 def normalize_line_endings(self, string):
2     string = string.replace('\r\n', '\n').replace('\r', '\n')
3     line_endings = self.view.settings().get(default_line_ending)
4     if line_endings == 'windows':
5         string = string.replace('\n', '\r\n')
6     elif line_endings == 'mac':
7         string = string.replace('\n', '\r')
8     return string
```

The `fix_whitespace()` method is a little more complicated, but does the same kind of manipulation, just for the indentation and whitespace in the CSS block. This manipulation only really works with a single block of CSS, so we exit if one or more braces was included in the original selection.

```
1 def fix_whitespace(self, original, prefixed, sel, braces):
2     # If braces are present we can do all of the whitespace magic
3     if braces:
4         return ('', prefixed, '')
```

Otherwise, we start by determining the indent level of the original CSS. This is done by searching for whitespace at the beginning of the selection.

```

1 | (row, col) = self.view.rowcol(sel.begin())
2 | indent_region = self.view.find('^\\s+', self.view.text_point(row, 0))
3 | if self.view.rowcol(indent_region.begin()) == row:
4 |     indent = self.view.substr(indent_region)
5 | else:
6 |     indent = ''

```

Next we trim the whitespace from the prefixed CSS and use the current view settings to indent the trimmed CSS to the original level using either tabs or spaces depending on the current editor settings.

```

01 | prefixed = prefixed.strip()
02 | prefixed = re.sub(re.compile('^\\s+', re.M), '', prefixed)
03 |
04 | settings = self.view.settings()
05 | use_spaces = settings.get('translate_tabs_to_spaces')
06 | tab_size = int(settings.get('tab_size', 8))
07 | indent_characters = '\\t'
08 | if use_spaces:
09 |     indent_characters = ' ' * tab_size
10 | prefixed = prefixed.replace('\\n', '\\n' + indent + indent_characters)

```

We finish the method up by using the original beginning and trailing white space to ensure the new prefixed CSS fits exactly in place of the original.

```
1 match = re.search('^(\\s*)', original)
2 prefix = match.groups()[0]
3 match = re.search('(\\s*)\\Z', original)
4 suffix = match.groups()[0]
5
6 return (prefix, prefixed, suffix)
```

With the `fix_whitespace()` method we used the Python regular expression (re)module, so we need to add it to the list of imports at the top of the script.

```
1 import re
```

And with this, we've completed the process of writing the `prefixr` command. The next step is to make the command easy to run by providing a keyboard shortcut and a menu entry.

Step 10 - Key Bindings

Most of the settings and modifications that can be made to Sublime are done via JSON files, and this is true for key bindings. Key bindings are usually OS-specific, which means that three key bindings files will need to be created for your plugin. The files should be named `Default (Windows).sublime-keymap` `Default`

(Linux).sublime-keymap and Default (OSX).sublime-keymap

```
1 Prefixr/  
2 ...  
3 - Default (Linux).sublime-keymap  
4 - Default (OSX).sublime-keymap  
5 - Default (Windows).sublime-keymap  
6 - Prefixr.py
```

The `.sublime-keymap` files contain a JSON array that contains JSON objects to specify the key bindings. The JSON objects must contain a `keys` and `command` key, and may also contain a `args` key if the command requires arguments. The hardest part about picking a key binding is to ensure the key binding is not already used. This can be done by going to the *Preferences > Key Bindings – Default* menu entry and searching for the keybinding you wish to use. Once you've found a suitably unused binding, add it to your `.sublime-keymap` files.

```
1 [  
2   {  
3     "keys": ["ctrl+alt+x"], "command": "prefixr"  
4   }  
5 ]
```

Normally the Linux and Windows key bindings are the same. The cmd key on OS X is specified by the string `super` in the `.sublime-keymap` files. When porting a key

binding across OSes, it is common for the `ctrl` key on Windows and Linux to be swapped out for `super` on OS X. This may not, however, always be the most natural hand movement, so if possible try and test your keybindings out on a real keyboard.

Step 11 - Menu Entries

One of the cooler things about extending Sublime is that it is possible to add items to the menu structure by creating `.sublime-menu` files. Menufiles must be named specific names to indicate what menu they affect:

- `Main.sublime-menu` controls the main program menu
- `Side Bar.sublime-menu` controls the right-click menu on a file or folder in the sidebar
- `Context.sublime-menu` controls the right-click menu on a file being edited

There are a whole handful of other menu files that affect various other menus throughout the interface. Browsing through the *Default* package is the easiest way to learn about all of these.

For Prefixr we want to add a menu item to the *Edit* menu and some entries to the

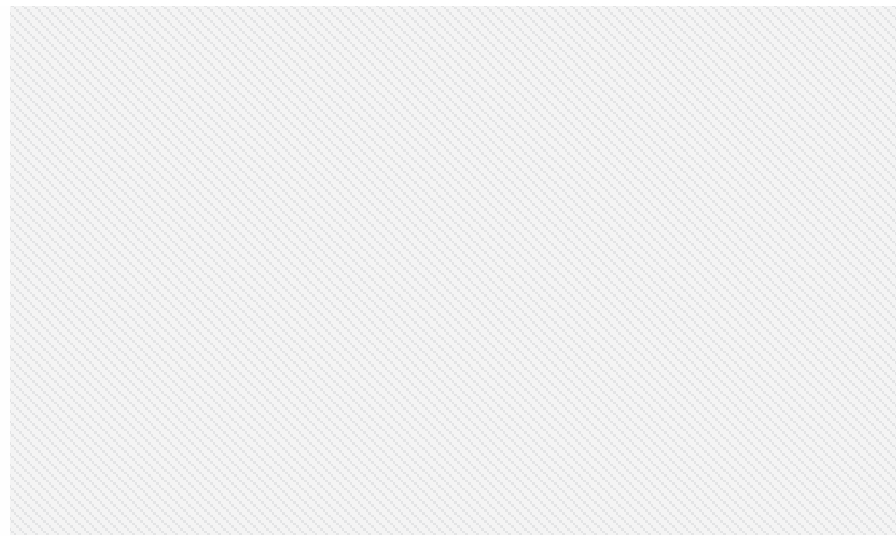
Preferences menu for settings. The following example is the JSON structure for the *Edit* menu entry. I've omitted the entries for the *Preferences* menu since they are fairly verbose being nested a few levels deep.

```
01  [
02  {
03      "id": "edit",
04      "children":
05      [
06          {"id": "wrap"},
07          { "command": "prefixr" }
08      ]
09  }
10  ]
```

The one piece to pay attention to is the `id` keys. By specifying the `id` of an existing menu entry, it is possible to append an entry without redefining the existing structure. If you open the `Main.sublime-menu` file from the `Default` package and browse around, you can determine what `id` you want to add your entry to.

At this point your Prefixr package should look almost identical to the [official version on GitHub](#).

Advertisement



Step 12 - Distributing Your Package

Now that you've taken the time to write a useful Sublime plugin, it is time to get into the hand of other users.

Sublime supports distributing a zip file of a package directory as a simple way to share packages. Simply zip your package folder and change the extension to `.sublime-package`. Other users may now place this into their Installed Packages directory and restart Sublime to install the package.

Along with easy availability to lots of users, having your package available via Package Control ensures users get upgraded automatically to your latest updates.

While this can certainly work, there is also a package manager for Sublime called [Package Control](#) that supports a master list of packages and automatic upgrades. To get your package added to the default channel, simply host it on [GitHub](#) or [BitBucket](#) and then fork the channel file ([on GitHub](#), or [BitBucket](#)), add your repository and send a pull request. Once the pull request is accepted, your package will be available to thousands of users using Sublime. Along with easy availability to lots of users, having your package available via Package Control ensures users get upgraded automatically to your latest updates.

If you don't want to host on GitHub or BitBucket, there is a custom JSON channel/repository system that can be used to host anywhere, while still providing the package to all users. It also provides advanced functionality like specifying the availability of packages by OS. See the [PackageControl page](#) for more details.

Go Write Some Plugins!

Now that we've covered the steps to write a Sublime plugin, it is time for you to dive

in! The Sublime plugin community is creating and publishing new functionality almost every day. With each release, Sublime becomes more and more powerful and versatile. The [Sublime Text Forum](#) is a great place to get help and talk with others about what you are building.

Advertisement

Difficulty:
Intermediate

Length:
Short

Categories:



Suggested Envato Tuts+ Course

[Getting Started With Django](#)

\$9

[Python](#)[Web Development](#)[Sublime Text](#)

Translations:

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

**Translate this
post**

Powered by



Download Attachment 

View Online Demo 

About Will Bond



I'm Will Bond. I built the open source Flourish PHP library and I work full time as the director of engineering at iMarc. In my spare time I hack on Noted and various open source and commercial software.

Related Tutorials



Add User Chat to an iOS Application With Magnet

[Code](#)



PostCSS Deep Dive: Create Your Own Plugin

[Web Design](#)



Create a Mobile Application Using WordPress, Ionic, and AngularJS

[Code](#)

What Would You Like to Learn?

[Suggest an idea](#) to the content editorial team at Envato Tuts+.

Advertisement

52 Comments

Nettuts+

♥ Recommend 4

🔗 Share



Join the discussion...



greg • 3 years ago

It would be great if this tutorial gets an update for ST3. I guess that Will Bond is busy with other packages but after things settle it would be great to see what changes particularly in

11 ^ | v • Reply • Share ›



chadananda • 3 years ago

Worst tutorial ever. "See how simple plugins are, let's write a simple one to do search and forget to cover multi-threading." Good lord.

22 ^ | v • Reply • Share ›



Deven • 4 years ago

Sublime is growing and this tutorial really help me out to understand the sublime editor.

2 ^ | v • Reply • Share ›



Vaisagh Viswanathan • 3 years ago

Hi, I was trying to create my own plugin and I'm kind of getting stuck in the part where I want to add a menu item. When I add the key binding to the default OSX keybinding file the key binding doesn't work. I simply create a Default (OSX).sublime-keymap in the folder this key binding is no longer missing some step?

1 ^ | v • Reply • Share ›



Andy Matthews • 4 years ago

It feels like you left parts out of your first section.

1) Using Sublime Text 2 1.4 (Windows), the plugin Boilerplate looks like this:

```
import sublime, sublimeplugin
class SampleCommand(sublimeplugin.TextCommand):
def run(self, view, args):
view.insert(view.size(), "Hello, World!\n")
```

ASP, CSS, Erlang, etc." or should it appear within the User directory? A disclaimer...when I created the plugin file my save path was:

C:\Users\AppData\Roaming\Sublime Text\Packages\Default\Options

So going up a single directory simply took me to the Default folder.

3) Hitting CTRL+` indeed brought up the console but when entering "view.run_command" I got an error:

```
>>> view.run_command('example')
```

[see more](#)

1 ^ | v • Reply • Share ›



Carl Giesberts → Andy Matthews • 3 years ago

It was a bit hard to find, but this page has the answer: <http://docs.sublimetext.info/>

ExampleCommand becomes 'example' as the string argument to run_command
AnythingElseCommand likewise becomes 'anything_else'

1 ^ | v • Reply • Share ›



Keith David Bershatsky → Carl Giesberts • 3 years ago

I recently made a similar error by using an older syntax that has since been deprecated. I used "view.runCommand('example')]", but instead should be "view.run_command('example')"

1 ^ | v • Reply • Share ›



currybill • 4 years ago



2. So I downloaded it. I haven't a clue how to use it. As a front end graphic designer of so baffled. How for instance can I view my html in my browser? It would be brilliant if you we for idiots. I've just thought - maybe there's a Sublime Text for Dummies?

I did ask the forum for help but the instruction didn't work for me or I just didn't understand know exactly what ST2 is about but getting under the bonnet is bewildering - I used to be have you seen what's under the bonnet of a modern car? Best regards

1 ^ | v • Reply • Share ›



BrettJay → currybill • 4 years ago

Start writing your markup, and save the html file and preview it in a web browser. Typically working with an editor like Sublime Text (or other editors like TextMate) in the text editor and then switching to and refreshing the page in your browser to see Firebug (in Firefox), the Element Inspector in Webkit browsers, and Dragonfly (in debugging css layouts and Javascript, and should also help you learn HTML and

2 ^ | v • Reply • Share ›



kapil garg • 4 months ago

The last parts like key bindings and package distribution are more helpful then the plugin much..

^ | v • Reply • Share ›



Chris Sells • 10 months ago

Is there a way to pop up a new window via a sublime plugin that can communicate back? Imagine being able to invoke a command, pop up a separate GUI window, pass data to the pass data back to sublime to invoke commands, etc.?



gheni · a year ago

hello can i dev a plugin that can convert input text to utf8&Reverse and save with unrever

^ | v · Reply · Share ›



Alex Hunt · a year ago

Fantastic tutorial that really got me started on my first plugin :)

<https://github.com/huntie/supe...>

<http://hunt.ghost.io/improving...>

^ | v · Reply · Share ›



HuaHero · 3 years ago

Thanks for this good tutorial,especially for us who have a long way to make progress in p

^ | v · Reply · Share ›



Shawn McCool · 3 years ago

Ok, but this ignores all of the basics. How to analyse data structures, methods for debug

Let's say you're looping through selections.. Well.. what are the available fields / methods sublime text API either.

Entry seems to be a pain in the ass due to lack of documentation.

^ | v · Reply · Share ›



Amir Masoud · 3 years ago

How can I get all pressed key with API?

^ | v · Reply · Share ›



Chris Bumgardner → Amir Masoud · 3 years ago

Check out my plugin I started the other day to do just that: <https://github.com/cbur>

1 ^ | v · Reply · Share ›



Pablo · 3 years ago

Great tutorial, very usefull

^ | v · Reply · Share ›



Ilya · 3 years ago

how to make localization?

^ | v · Reply · Share ›



Kjell · 4 years ago

JSON is also verbose, but better than XML

<http://www.w3schools.com/json/...>

```
var JSONObject = {  
  "name":"John Johnson",  
  "street":"Oslo West 16",  
  "age":33,  
  "phone":"555 1234567"};
```

Syntactically this will also work when one creates a special DSL

```
SomeObjectName {  
  name John Johnson
```

```
street Oslo West 16
```

```
}
```

```
age 33
```

```
phone 555 1234567
```

```
}
```

The simplification with a special DSL vs. JSON is large.

^ | v • Reply • Share ›



Yongning Liang • 4 years ago

COOL, love will's sublime plugins, and this tutorial!

^ | v • Reply • Share ›



Gemma W. • 4 years ago

Thanks for the tutorial. However, I'm still looking for a tutorial on how to create your own scheme using ST2.

^ | v • Reply • Share ›



Cindy • 4 years ago

I don't know if it's relevant to this post but how do I get the Mac terminal output in the built really wonder is how to get the CoronaSDK terminal output in the ST2 Console?

^ | v • Reply • Share ›



Dan LaManna • 4 years ago

Great tutorial! Glad Sublime is so open to plugin developers, however I did have a question I've read the tutorial and the API reference a good amount and I'm having a hard time understanding which commands would work. I could only call prefixr for the run method, but no other methods

case, each keybinding would have to refer to a class, is there a way to extend every class for the plugin?

Thanks!

^ | v • Reply • Share ›



Joey • 4 years ago

" To prevent a poor connection from interrupting other work, we need to make sure that the work is happening in the background. "

If you really wanna to go to a whole lot of trouble to maintain a nice experience, you'd be better off using the ``view.add_regions`` API (et al) than managing offsets yourself.

If you aren't familiar with ``view.add_regions``, check it out, it's really useful for keeping track of the buffer will, or may, be modified.

"Yeah, cool, I can make edits while I wait then huh? What if I actually remove one of the items? I say. (Leaving aside the fact that the grouped undo is by design)

There's an undocumented 3rd parameter (usually only discovered by the curious) to the ``region`` constructor. It's the meta property available via ``region.meta()``. It's just a humble integer, deterministically identify any region you put into ``view.add_regions`` when you retrieve it via ``region``.

eg. You feed in a length 4 list of regions indexed via `[0, 1, 2, 3]` and you get back a length 4 list of regions. Use the `region.meta()` property rather than the indexes)

(In general, you shouldn't mess with the meta of any regions you add to view self)

[see more](#)

• Reply • Share ›



Emmanuel · 4 years ago

Excellent tutorial !

It's going far beyond that what we usually see. I think that this kind of tutorial is what some (advanced users) want ! This is not the typical tutorial which scratch the surface. It's going deeper. There should be tutorials and that's what some users need (maybe not everyone but there's no offense that's the reality...).

Excellent initiative !

All my respect for the job done.

PS: sorry for my english, i'm french ;)

^ | v · Reply · Share ›



Gabriel Mateu's · 4 years ago

What Alexander Hultner said describes what I feel about this awesome editor:

"I heard about Sublime from Jeff's post and never looked back."

Thanks!

^ | v · Reply · Share ›



pit · 4 years ago

hi

thank for the information

| · Reply · Share ›



Liam · 4 years ago

Thanks Will for this great tutorial. I wish I had this two days again when I was making my nonetheless I persevered... (with the help of sublimetext.info and the forum's plugin devel

I hope more people catch on and write some tutorials on how to use some other core ST in the API docs, such as the autocomplete pop-up or the quick panel.

Now I just need to work on my python skills.

Thanks again.

^ | v · Reply · Share ›



Tresor Paris · 4 years ago

awesome! thanks for sharing!

^ | v · Reply · Share ›



dkbose · 4 years ago

I heard about Sublime from Jeff's post and never looked back. The best editor - thanks fo plugin. This will definitely rock it more.

^ | v · Reply · Share ›



John · 4 years ago

My friend just told me about Sublime Text. This is awesome, thank you.

^ | v · Reply · Share ›



Alexander Hultner · 4 years ago

This is a great tutorial. I've been considering testing out sublime 2 so I'll really have use o



the guide.

^ | v • Reply • Share ›



Johan Steen • 4 years ago

Brilliant! This is probably one of the coolest tutorials I've seen on net.tuts ever. Extremely chance to follow it up with a tutorial demonstrating of how to cook your own syntax highlight available in Sublime out of the box?

Keep it up! Cheers!

^ | v • Reply • Share ›



Fed03 ➔ Johan Steen • 4 years ago

ty for the awesome tutorial it will be extremely usefull^

i look forward to see an explanatory tutorial on highlight code too.

Just to understand how to color <?php and ?> tooks me days XD

^ | v • Reply • Share ›



adumpaul • 4 years ago

Great tutorial.Thank you for sharing.

^ | v • Reply • Share ›



Akkis • 4 years ago

Thank you for this tut. This is really helpful. I start using Sublime these days, but i use als want to make "All Open Documents" replaces. Does Sublime support it?

^ | v • Reply • Share ›



Will Bond → Akkis · 4 years ago

Yes, as of build 2139 Sublime supports replacing across open files, folders, or cc filename.

^ | v · Reply · Share ›



Alex John · 4 years ago

I just only knew that there is "Sublime Text" editor and when i was used this editor. I feel g thanks to that editor. i love it.

^ | v · Reply · Share ›



Brian Temecula · 4 years ago

I wish Sublime Text 2 was a little more fine tuned for PHP. I purchased a license, and the was a very irritating indentation of parenthesis. The indentation isn't part of javascript, onl nuts. Anyways, nice to see an article about the plugin creation.

^ | v · Reply · Share ›



Will Bond → Brian Temecula · 4 years ago

There have been a few situations where the built-in indentation rules don't exactly you are comfortable with editing regular expressions, you can go to Preferences : open the PHP folder. Inside there you'll find a file named "Indentation Rules.tmPre how indentation is increased and decreased.

^ | v · Reply · Share ›



Jeffrey Way → Brian Temecula · 4 years ago

Remember that it's beta software.

^ | v · Reply · Share ›



Brian Temecula → Jeffrey Way · 4 years ago

Oh, I know. I do think it's quite awesome, and have a sense of excitement is available. Gotta love the new code folding sidebar buttons in the last one

1 ^ | v · Reply · Share ›



Sergey · 4 years ago

Great article! And great approach by taking a real world plugin as an example!

^ | v · Reply · Share ›



Jake · 4 years ago

Good tutorial, but I think that if you are trying to reach an audience of people who haven't and maybe even people who don't know Python very well, it would have been a much better be MUCH more simple. Like 20 lines simple.

The best way to expose people to a plugin API like ST's is to have them create several small plugins that all do a variety of different (and maybe useless) things. Getting people comfortable where to look for what is the goal.

Half of the source code is dedicated to threading and http requests... So in the end, half of the time is spent trying to understand the threading and extra Python stuff instead of what you can do with it, which is the purpose of the tutorial in the first place.

^ | v · Reply · Share ›



Will Bond → Jake · 4 years ago

Thanks for great the feedback!

I can see that there may be some desire for a more basic tutorial on writing a plugin

and are often appropriate, they tend to run out of useful material very quickly for the try to build something themselves.

The Sublime API reference does a pretty good job of covering what functionality is of bundled plugins that cover some basic examples of extending the editor. While a picture of putting all of the pieces together. You certainly can spend the time and effort and look at all of the different types of files to see what is going on, but with this tutorial survey of what is frequently used.

From my experience in writing a number of plugins I've found that most of them use subprocesses, so I felt it was important to at least provide an example of using them are necessary. This decision was made partially because of various comments at the Plugin Development board on the Sublime Forum (<http://www.sublimetext.com>

All that said, I do think that more basic, and advanced, tutorials could augment this

[see more](#)

1 ^ | v • Reply • Share ›



Charles Roper → Will Bond • 4 years ago

I really liked this tutorial too - thanks Will. I also love the idea of creating a series of example tutorials to demonstrate other aspects of plugin creation, syntax (and indenting) and so on.

^ | v • Reply • Share ›



Arturo → Will Bond • 4 years ago

This tutorial was excellent for me.

I don't know python but I am an experienced programmer in other languages. I consider this tutorial great since it gave me handles plugins and how to work with threads and http request in python. It's actually useful, and not just another "draw pink elephants".

Although I agree with Jake that a more basic tut will reach a more basic audience, there's no "One Tutorial to rule them all (devs)" :)

^ | v • Reply • Share ›



Rian Ariona • 4 years ago

awesome! thanks for sharing! :)

^ | v • Reply • Share ›



Muhammad Adnan • 4 years ago

Awesome, liked.

^ | v • Reply • Share ›

Load more comments

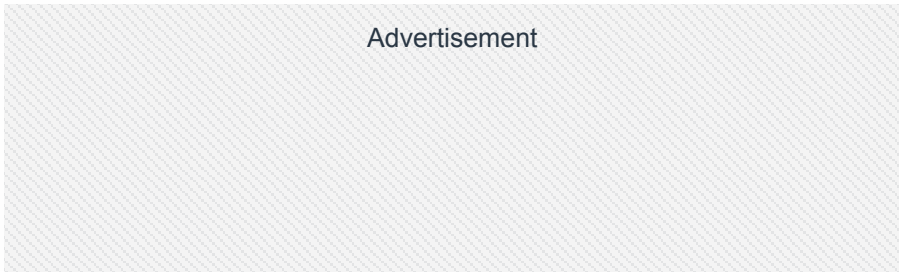


Subscribe

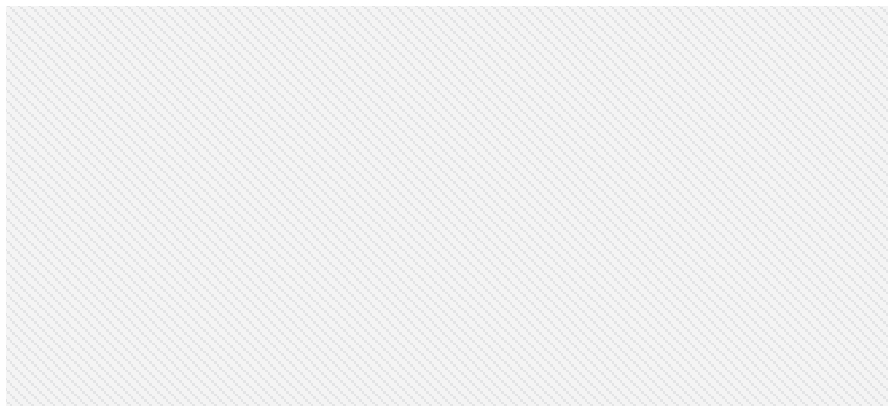


Add Disqus to your site

Privacy



Advertisement



Teaching skills to millions worldwide.

21,010 Tutorials

Video

685 Courses

Meet Envato

About Envato

Explore our Ecosystem

Careers

Join our Community

Teach at Envato Tuts+

Translate for Envato Tuts+

Forums

Blog

Community meetups

Help and Support

FAQ

Help Center

Terms of Use

About Envato Tuts+

Advertise

Email Newsletters

Get Envato Tuts+ updates, news, surveys & offers.

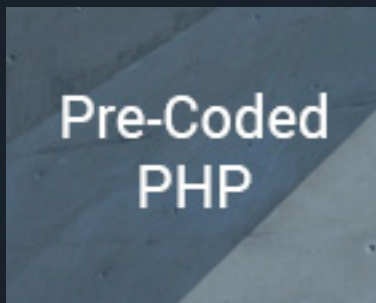
Subscribe

[Privacy Policy](#)



Custom digital services like logo design, WordPress installation, video production and more.

[Check out Envato Studio](#)



Build anything from social networks to file upload systems. Build faster with pre-coded PHP scripts.

[Browse PHP on CodeCanyon](#)

Follow Envato Tuts+



© 2015 Envato Pty Ltd. Trademarks and brands are the property of their respective owners.