ipyvuetify

Release 1.2.2

Feb 22, 2021

Introduction

CHAPTER 1

Introduction

Ipyvuetify is a widget library for making modern looking GUI's in Jupyter notebooks (classic and lab) and dashboards (Voilà). It's based on the Google material design philosophy best known from the Android user interface.

A large set of widgets is provided with many widgets having multiple variants. a few of which are displayed below.

There is support for responsive layouts, which are not that useful in a notebook because of the fixed width, but come in handy when making dashboards with Voilà, making your dashboard usable on tablets and phones.

When comparing ipyvuetify to ipywidgets, the standard widget library of Jupyter, ipyvuetify has a lot more widgets which are also more customizable and composable at the expense of a bit more verbosity in the source code.

Ipyvuetify uses the machinery of ipywidgets as a base, but has different conventions for the API. This is mainly due to the fact the Python API is generated from the JavaScript library it is based on: Vuetify. This exposes the full power of Vuetify and allows us to rely on the extensive documentation and examples of it. Generating code and relying on documentation from the underlying library allowed us to expose a lot of widgets to Jupyter in a relatively short amount of time.

In *Usage* all concepts and how they differ from ipywidgets will be explained and supported by examples.

To explore which widgets are available and how to use them we defer to the Vuetify documentation. You can browse examples on the left-hand side and see the source code by clicking on '< >' on the top right-hand side of the example. By reading *Usage* you will be able to translate the examples to ipyvuetify.

CHAPTER 2

Installation

2.1 Using pip

```
$ pip install ipyvuetify
```

2.2 Using conda

```
(myenv) $ conda install -c conda-forge ipyvuetify
```

2.3 Jupyter Lab

If you're using only the classic notebook, you're done. If you're using Jupyter Lab, the extension has to be installed in Jupyter Lab:

```
$ jupyter labextension install jupyter-vuetify
```

Note: ipyvuetify depends on ipywidgets being installed in Jupyter Lab, see the ipywidgets documentation on how to do that.

CHAPTER 3

Usage

This page shows how to use ipyvuetify and explains how it is different from other widget libraries you may know such as ipywidgets. It also explains how to use the Vuetify documentation. Most examples display real widgets which have animations and behavior.

3.1 Create an ipyvuetify widget

Below you see how to create an ipyvuetify widget.

```
import ipyvuetify as v

my_select = v.Select(
    label='Fruits',
    items=['Apple', 'Pear', 'Cherry'])
my_select
```

Attributes can be changed at a later time by:

```
my_select.items = [*my_select.items, 'Banana']
```

Note: A new List is created to change the items. In-place mutations of List and Dict, e.g. `my_select.append('Banana')`, are not detected by ipywidgets.

What widgets are available and how they look can be found in the Vuetify documentation. Browse the side bar on the left hand side and select a widget, then click <> on the right hand side on an example to see the source code for it. The HTML code may seem unfamiliar at first, but this documentation will guide you through it. For starters to translate the Vuetify widget names, which are starting with `v-`, to ipyvuetify, remove the `v-` prefix and CamelCase the remaining name. E.g `v-select` becomes `Select` and `v-list-item` becomes `ListItem`.

Equivalent Vuetify syntax of the example above:

```
<v-select label="Fruits" :items="['Apple', 'Pear', 'Cherry']" />
```

3.2 Setting Attributes

When translating from Vuetify HTML to Python, some attributes have to be treated different.

Python uses snake_case to separate words in attributes, while Vuetify uses kebab-case. For example the attribute `append-icon` becomes `append_icon`:

Vuetify:

```
<v-select append-icon="mdi-gamepad-down" label="Fruits" />
```

ipyvuetify:

```
v.Select(append_icon='mdi-gamepad-down', label='Fruits')
```

In HTML attributes don't have to have values, just defining the attribute is enough to use it as a boolean. In Python we have to set the value to True. For example `clearable` becomes `clearable=True`:

Vuetify:

```
<v-select clearable label="Fruits" :items="['Apple', 'Pear', 'Cherry']" value="Apple"␣
↪/>
```

ipyvuetify:

```
v.Select(clearable=True, label='Fruits', items=['Apple', 'Pear', 'Cherry'], value=
↪'Apple')
```

Some attribute have naming conflicts with Python or ipywidgets. These are `for`, `open`, `class` and `style` and must be suffixed with an underscore. For example `style` becomes `style_`

Vuetify:

```
<v-select style="width: 75px" label="Fruits" />
```

ipyvuetify:

```
v.Select(style_='width: 75px', label='Fruits')
```

In the Vuetify HTML examples you'll see attributes prefixed with a colon `:`. This means the attribute is bound to a variable or it is evaluated as an expression. If it is bound to a variable you'll see that variable being used in other parts of the example. In ipyvuetify we use `jslink()` to link these attributes. In the next section you'll see an example of this. To look at how that variable is initialized you select the 'script' tab on a Vuetify example.

If it's an expression it's mostly used to set a List or a Dict, as is done with `items` in the examples above. This can be the same in ipyvuetify.

3.3 Reading the value

Now we want to be able to read out the selected value. In ipywidgets this would be done by reading the `value` attribute. In Vue this is done with the `v-model` directive, which is translated to Python as `v_model` (note the '_' instead of '-'). The `v_model` attribute has to be explicitly set when creating the widget.

Vuetify:

```
<v-container>
    <v-select
        v-model="colorVariable"
        label="Colors"
        items="['red', 'green', 'blue']" />
    <v-chip :color="colorVariable"><v-chip>
</v-container>
```

ipyvuetify:

```
from ipywidgets import jslink

color_select = v.Select(
    v_model='green',
    label='Colors',
    items=['red', 'green', 'blue'])

color_display = v.Chip()

jslink((color_select, 'v_model'), (color_display, 'color'))

v.Container(children=[
    color_select,
    color_display
])
```

Note: ipyvuetify widgets have a `value` attribute, but that's only used for setting the value, it will not change on interactions with the widget.

3.4 The children attribute

Because ipyvuetify is based on HTML, which represents a GUI as a tree of elements, all widgets have an attribute `children` which is a list of widgets or strings. This way the same tree can be represented in Python. Sometimes something you would expect to be specified as an attribute, must be specified as an item in `children`, e.g. in ipywidgets the text of a button is set with the attribute `description` while in ipyvuetify the text is set with setting an item in the `children` list:

Vuetify:

```
<v-container>
    <v-btn color="primary">Click me</v-btn>
</v-container>
```

ipyvuetify

```
v.Container(children=[
    v.Btn(color='primary', children=['Click me'])
])
```

This has the benefit of composability, e.g. the button can, in addition to text, also contain an icon:

Vuetify:

```
<v-container>
    <v-btn color="primary">
        <v-icon left>
            mdi-email-edit-outline
        </v-icon>
        Click me
    </v-btn>
</v-container>
```

ipyvuetify:

```
v.Container(children=[
    v.Btn(color='primary', children=[
        v.Icon(left=True, children=[
            'mdi-email-edit-outline'
        ]),
        'Click me'
    ])
])
```

3.5 Events

Events are specified with `.on_event(event_name, callback_fn)` instead of setting an attribute like in ipywidgets.

```
btn = v.Btn(color='primary', children=['Click me'])
count = 0

def on_click(widget, event, data):
    global count
    btn.children=[f'Click me {count}']
    count += 1

btn.on_event('click', on_click)

v.Container(children=[
    btn
])

# The output of this example is intentionally left out, because
# it will not work without an active kernel.
```

The three arguments in the callback function are:

- widget: the widget the event originates from. This is useful when using the same callback for multiple widgets.
- event: the event name. This is useful when using the same callback for multiple events.
- data: data for the event. For e.g. `click` of `Btn` this contains which modifier keys are pressed and some information on the position of the mouse.

All HTML events can be used. The `on` prefix must be omitted.

Widgets can have custom events, to find out which, the Vuetify API explorer can be used. Search for a component and on the left-hand side of list of attributes you will find a tab for the events.

In Vuetify events are defined as attributes with an `@` prefix. The equivalent Vuetify syntax of the example above is:

```
<v-container>
    <v-btn color="primary" @click="on_click">
        Click me {{ count }}
    </v-btn>
</v-container>
```

The on_click method would be in the 'script' tab of an example and is not shown here.

3.6 Regular HTML tags

Sometimes some regular HTML tags are needed. For this the Html widget can be used. The attributes of the HTML tag can be accessed through the `attributes` trait.

Vuetify:

```
<v-container>
    <h1 title="a title">My heading</h1>
</v-container>
```

ipyvuetify

```
v.Container(children=[
    v.Html(
        tag='h1',
        attributes={'title': 'a title'},
        children=['My heading']
    )
])
```

3.7 Styling

To visually customize widgets, the underlying CSS facilities of Vuetify are exposed. With the `style_` attribute CSS properties can be set. Multiple CSS properties can be set by separating them with a semicolon `;`.

```
v.Select(label='Fruit', style_='width: 75px; opacity: 0.7')
```

With the `class_` attribute predefined Vuetify styles can be set. Predefined styles of note are spacing and *colors <https://vuetifyjs.com/styles/colors/>*. More can be found in the section 'Styles and animations' of the Vuetify documentation. Multiple classes can be applied by separating them with a space.

Buttons without spacing:

```
v.Container(children=[
    v.Btn(children=[f'Button {i}']) for i in range(3)
])
```

With 2 units of margin in the x direction:

```
v.Container(children=[
    v.Btn(class_='mx-2', children=[f'Button {i}']) for i in range(3)
])
```

And colors:

```
v.Container(children=[
    v.Btn(class_=f'mx-2 indigo lighten-{i+1}', children=[f'Button {i}']) for i in␣
↪range(3)
])
```

3.8 Layout (HBox/VBox alternative)

In ipywidgets you would layout a grid of widgets with HBox and VBox.

```
import ipywidgets as widgets

widgets.HBox([
    widgets.VBox([
        widgets.Button(description="top left"),
        widgets.Button(description="bottom left"),
    ]),
    widgets.VBox([
        widgets.Button(description="top right"),
        widgets.Button(description="bottom right"),
    ]),
])
```

This can be done in ipyvuetify with the help of some classes described in flex helpers.

```
v.Html(tag='div', class_='d-flex flex-row', children=[
    v.Html(tag='div', class_='d-flex flex-column', children=[
        v.Btn(class_='ma-2', children=['top left']),
        v.Btn(class_='ma-2', children=['bottom left'])
    ]),
    v.Html(tag='div', class_='d-flex flex-column', children=[
        v.Btn(class_='ma-2', children=['top right']),
        v.Btn(class_='ma-2', children=['bottom right'])
    ]),
])
```

3.9 Icons

Icons can be displayed with the Icon widget:

```
v.Icon(children=['mdi-thumb-up'])
```

In some widgets icons are specified by setting an attribute:

```
v.Select(prepend_icon='mdi-thumb-up')
```

See materialdesignicons.com/4.5.95 for a list of available icons.

3.10 Themes

To enable the dark theme:

```
v.theme.dark = True
```

To customize the themes:

```
v.theme.themes.light.primary = '#b71c1c'

v.theme.themes.dark.primary = '#a71c1c'
```

Also, the pre-defined material colors are supported:

```
v.theme.themes.light.primary = 'colors.teal'

v.theme.themes.light.secondary = 'colors.teal.lighten4'
```

Available theme properties:

- primary
- secondary
- accent
- error
- info
- success
- warning
- anchor

3.11 Summary

Below you will find a summary of all concepts of Vuetify and how they translate to ipyvuetify to help with the translation from Vuetify examples to ipyvuetify.

- Component names convert to CamelCase and the v- prefix is stripped

Vuetify	`<v-list-tile .../>`
ipyvuetify	`ListTitle(...)`

- Attributes
  - convert to snake_case

Vuetify	`<v-menu offset-y ...`
ipyvuetify	`Menu(offset_y=True ...`

  - must have a value

Vuetify	`<v-btn round ...`
ipyvuetify	`Btn(round=True ...`

  - with naming conflicts, `style`, `class`, `open` and `for`, are suffixed with an _

Vuetify	`<v-btn class="mr-3" style="..." >`
ipyvuetify	`Btn(class_='mr-3', style_='...')`

- v-model (value in ipywidgets) contains the value directly

Vuetify	`<v-slider v-model="some_property" ...`
ipyvuetify	`myslider = Slider(v_model=25...` `jslink((myslider, 'v_model'), (..., ...))`

- Child components and text are defined in the children attribute

Vuetify	`<v-btn>text <v-icon>...</icon></v-btn>`
ipyvuetify	`Btn(children=['text', Icon(...)])`

- Event listeners are defined with on_event

Vuetify	`<v-btn @click='someMethod()' ...`
ipyvuetify	`def some_method(widget, event, data):` `button.on_event('click', some_method)`

- Regular HTML tags can made with the Html widget

Vuetify	`<div>...</div>`
ipyvuetify	`Html(tag='div', children=[...])`

CHAPTER 4

Advanced Usage

4.1 (Scoped) Slots

Slots are used to add content at a certain location in a widget. You can find out what slots a widget supports by using the Vuetify documentation. If you want to know what slots `Select` has, search for `v-select` on the Vuetify API explorer or for this example use the direct link. On the left-hand side of list of attributes you will find a tab 'slots'.

An example for using the slot 'no-data', which changes what the Select widget shows when it has no items:

Vuetify:

```
<v-select>
  <template v-slot:no-data>
    <v-list-item>
      <v-list-item-title>
        My custom no data message
      </v-list-item-title>
    </v-list-item>
  </template>
</v-select>
```

ipyvuetify:

```
v.Select(v_slots=[{
    'name': 'no-data',
    'children': [
        v.ListItem(children=[
            v.ListItemTitle(children=['My custom no data message'])])]
}])
```

Scoped slots are used if the parent widget needs to share its scope with the content. In the example below the events of the parent widget are used in the slot content.

Vuetify:

```
<v-tooltip>
    <template v-slot:activator="tooltip">
        <v-btn v-on="tooltip.on" color="primary">
            button with tooltip
        </v-btn>
    </template>
    Insert tooltip text here
</v-tooltip>
```

ipyvuetify:

```
v.Container(children=[
    v.Tooltip(bottom=True, v_slots=[{
        'name': 'activator',
        'variable': 'tooltip',
        'children': v.Btn(v_on='tooltip.on', color='primary', children=[
            'button with tooltip'
        ]),
    }], children=['Insert tooltip text here'])
])
```

In the Vuetify examples you will actually see:

```
...
<template v-slot:activator="{ on }">
    <v-btn v-on="on">
...
```

Instead of the functionally equivalent (like used in the example above):

```
...
<template v-slot:activator="tooltip">
    <v-btn v-on="tooltip.on">
...
```

The `{ on }` is JavaScript syntax for destructuring an object. It takes the 'on' attribute from an object and exposes it as the 'on' variable.

Note: The 'default' slot can be ignored, this is where the content defined in the `children` attribute goes.

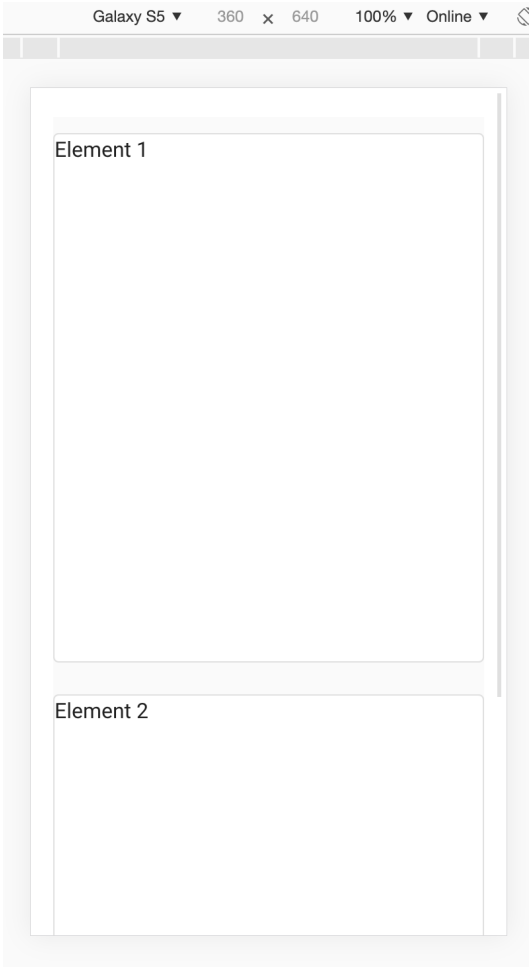
4.2 Responsive Layout

When making dashbords with Voilà you can change the layout depending on the users screen size. This is done with a grid system. For example on a laptop (breakpoint md) you could fit two elements next to each other while on a smartphone (defined with 'cols' as default) you would want one element to take up the full width:

```
v.Row(children=[
    v.Col(cols=12, md=6, children=[
        v.Card(outlined=True, style_='height: 400px', children=[f'Element {i}'])
    ]) for i in range (1,3)
])
```

Which displays on a laptop as:

On a phone as:



In the display section you will find CSS helper classes to do more customizations based on screen size.

4.3 Event modifiers

In Vue event modifiers can be used to change event behavior.

For example when you have two nested elements and want a different click handler for the inner and outer element, the `stop` event modifier can be used by appending `.stop` to the event name:

```
icon = v.Icon(right=True, children=['mdi-account-lock'])
btn = v.Btn(color='primary', children=[
    'button',
    icon
])

icon.on_event('click.stop', lambda *args: print('icon clicked'))
btn.on_event('click', lambda *args: print('btn clicked'))

v.Container(children=[
    btn
])

# Note: the event handlers won't work in this page because there is no active kernel.
```

4.4 .sync

When you see `.sync` appended to an attribute in Vuetify syntax, it means the attribute has a two-way binding (like `v-model`). This is shorthand in Vue that automatically listens to an event named `update:<attributeNameInCamelCase>`.

We can achieve the same manually in ipyvuetify by setting an event handler `<widget>.on_event('update:<attributeNameInCamelCase>', <function>)`

Vuetify:

```
<v-navigation-drawer :mini-variant.sync="someProperty" ...
```

ipyvuetify:

```
drawer = v.NavigationDrawer(mini_variant=True, ...)

def update_mini(widget, event, data):
    drawer.mini_variant = data

drawer.on_event('update:miniVariant', update_mini)
```