Widget Events

In this lecture we will discuss widget events, such as button clicks!

Special events

The `Button` is not used to represent a data type. Instead the button widget is used to handle mouse clicks. The `on_click` method of the `Button` can be used to register a function to be called when the button is clicked. The docstring of the `on_click` can be seen below.

In [ ]:
```
import ipywidgets as widgets

print(widgets.Button.on_click.__doc__)
```

Example #1 - on_click

Since button clicks are stateless, they are transmitted from the front-end to the back-end using custom messages. By using the `on_click` method, a button that prints a message when it has been clicked is shown below.

In [ ]:
```
from IPython.display import display
button = widgets.Button(description="Click Me!")
display(button)

def on_button_clicked(b):
    print("Button clicked.")

button.on_click(on_button_clicked)
```

Example #2 - on_submit

The `Text` widget also has a special `on_submit` event. The `on_submit` event fires when the user hits `enter`.

In [ ]:
```
text = widgets.Text()
display(text)

def handle_submit(sender):
    print(text.value)

text.on_submit(handle_submit)
```

Traitlet events

Widget properties are IPython traitlets and traitlets are eventful. To handle changes, the `observe` method of the widget can be used to register a callback. The docstring for `observe` can be seen below.

In [ ]:
```
print(widgets.Widget.observe.__doc__)
```

Signatures

Mentioned in the docstring, the callback registered must have the signature `handler(change)` where `change` is a dictionary holding the information about the change.

Using this method, an example of how to output an `IntSlider`'s value as it is changed can be seen below.

In [ ]:
```
int_range = widgets.IntSlider()
display(int_range)

def on_value_change(change):
    print(change['new'])

int_range.observe(on_value_change, names='value')
```

Linking Widgets

Often, you may want to simply link widget attributes together. Synchronization of attributes can be done in a simpler way than by using bare traitlets events.

Linking traitlets attributes in the kernel¶

The first method is to use the `link` and `dlink` functions from the `traitlets` module. This only works if we are interacting with a live kernel.

In [ ]:
```
import traitlets
```

In [ ]:
```
# Create Caption
caption = widgets.Label(value = 'The values of slider1 and slider2 are synchronized')

# Create IntSliders
slider1 = widgets.IntSlider(description='Slider 1')
slider2 =  widgets.IntSlider(description='Slider 2')

# Use trailets to link
l = traitlets.link((slider1, 'value'), (slider2, 'value'))

# Display!
display(caption, slider1, slider2)
```

In [ ]:
```
# Create Caption
caption = widgets.Label(value='Changes in source values are reflected in target1')

# Create Sliders
source = widgets.IntSlider(description='Source')
target1 = widgets.IntSlider(description='Target 1')

# Use dlink
dl = traitlets.dlink((source, 'value'), (target1, 'value'))
display(caption, source, target1)
```

Function `traitlets.link` and `traitlets.dlink` return a `Link` or `DLink` object. The link can be broken by calling the `unlink` method.

In [ ]:
```
# May get an error depending on order of cells being run!
l.unlink()
dl.unlink()
```

Registering callbacks to trait changes in the kernel

Since attributes of widgets on the Python side are traitlets, you can register handlers to the change events whenever the model gets updates from the front-end.

The handler passed to observe will be called with one change argument. The change object holds at least a `type` key and a `name` key, corresponding respectively to the type of notification and the name of the attribute that triggered the notification.

Other keys may be passed depending on the value of `type`. In the case where type is `change`, we also have the following keys:

- `owner` : the HasTraits instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.

In [ ]:
```
caption = widgets.Label(value='The values of range1 and range2 are synchronized')
slider = widgets.IntSlider(min=-5, max=5, value=1, description='Slider')

def handle_slider_change(change):
    caption.value = 'The slider value is ' + (
        'negative' if change.new < 0 else 'nonnegative'
    )

slider.observe(handle_slider_change, names='value')

display(caption, slider)
```

Linking widgets attributes from the client side

When synchronizing traitlets attributes, you may experience a lag because of the latency due to the roundtrip to the server side. You can also directly link widget attributes in the browser using the link widgets, in either a unidirectional or a bidirectional fashion.

Javascript links persist when embedding widgets in html web pages without a kernel.

In [ ]:
```
# NO LAG VERSION
caption = widgets.Label(value = 'The values of range1 and range2 are synchronized')

range1 = widgets.IntSlider(description='Range 1')
range2 = widgets.IntSlider(description='Range 2')

l = widgets.jslink((range1, 'value'), (range2, 'value'))
display(caption, range1, range2)
```

In [ ]:
```
# NO LAG VERSION
caption = widgets.Label(value = 'Changes in source_range values are reflected in target_range

source_range = widgets.IntSlider(description='Source range')
target_range = widgets.IntSlider(description='Target range')

dl = widgets.jsdlink((source_range, 'value'), (target_range, 'value'))
display(caption, source_range, target_range)
```

Function `widgets.jslink` returns a `Link` widget. The link can be broken by calling the `unlink` method.

In [ ]:
```
l.unlink()
dl.unlink()
```

The difference between linking in the kernel and linking in the client

Linking in the kernel means linking via python. If two sliders are linked in the kernel, when one slider is changed the browser sends a message to the kernel (python in this case) updating the changed slider, the link widget in the kernel then propagates the change to the other slider object in the kernel, and then the other slider's kernel object sends a message to the browser to update the other slider's views in the browser. If the kernel is not running (as in a static web page), then the controls will not be linked.

Linking using jslink (i.e., on the browser side) means contructing the link in Javascript. When one slider is changed, Javascript running in the browser changes the value of the other slider in the browser, without needing to communicate with the kernel at all. If the sliders are attached to kernel objects, each slider will update their kernel-side objects independently.

To see the difference between the two, go to the ipywidgets documentation and try out the sliders near the bottom. The ones linked in the kernel with `link` and `dlink` are no longer linked, but the ones linked in the browser with `jslink` and `jsdlink` are still linked.

Continuous updates

Some widgets offer a choice with their `continuous_update` attribute between continually updating values or only updating values when a user submits the value (for example, by pressing Enter or navigating away from the control). In the next example, we see the "Delayed" controls only transmit their value after the user finishes dragging the slider or submitting the textbox. The "Continuous" controls continually transmit their values as they are changed. Try typing a two-digit number into each of the text boxes, or dragging each of the sliders, to see the difference.

In [ ]:
```
import traitlets
a = widgets.IntSlider(description="Delayed", continuous_update=False)
b = widgets.IntText(description="Delayed", continuous_update=False)
c = widgets.IntSlider(description="Continuous", continuous_update=True)
d = widgets.IntText(description="Continuous", continuous_update=True)

traitlets.link((a, 'value'), (b, 'value'))
traitlets.link((a, 'value'), (c, 'value'))
traitlets.link((a, 'value'), (d, 'value'))
widgets.VBox([a,b,c,d])
```

Sliders, `Text`, and `Textarea` controls default to `continuous_update=True`. `IntText` and other text boxes for entering integer or float numbers default to `continuous_update=False` (since often you'll want to type an entire number before submitting the value by pressing enter or navigating out of the box).

Conclusion

You should now feel comfortable linking Widget events!