Tkinter 8.5 reference: a GUI for Python

NEW MEXICO TECH
Computer Center www.nmt.edu/tcc

54.6. Writing your handler: The `Event` class

The sections above tell you how to describe what events you want to handle, and how to bind them. Now let us turn to the writing of the handler that will be called when the event actually happens.

The handler will be passed an `Event` object that describes what happened. The handler can be either a function or a method. Here is the calling sequence for a regular function:

```
def handlerName(event):
```

And as a method:

```
    def handlerName(self, event):
```

The attributes of the `Event` object passed to the handler are described below. Some of these attributes are always set, but some are set only for certain types of events.

`.char`	If the event was related to a `KeyPress` or `KeyRelease` for a key that produces a regular ASCII character, this string will be set to that character. (For special keys like delete, see the `.keysym` attribute, below.)
`.delta`	For `MouseWheel` events, this attribute contains an integer whose sign is positive to scroll up, negative to scroll down. Under Windows, this value will be a multiple of 120; for example, 120 means scroll up one step, and -240 means scroll down two steps. Under MacOS, it will be a multiple of 1, so 1 means scroll up one step, and -2 means scroll down two steps. For Linux mouse wheel support, see the note on the `Button` event binding in Section 54.3, "Event types".
`.height`	If the event was a `Configure`, this attribute is set to the widget's new height in pixels.

`.keycode`	For `KeyPress` or `KeyRelease` events, this attribute is set to a numeric code that identifies the key. However, it does not identify which of the characters on that key were produced, so that “`x`” and “`X`” have the same `.keyCode` value. For the possible values of this field, see Section 54.5, “Key names”.
`.keysym`	For `KeyPress` or `KeyRelease` events involving a special key, this attribute is set to the key's string name, e.g., `'Prior'` for the PageUp key. See Section 54.5, “Key names” for a complete list of `.keysym` names.
`.keysym_num`	For `KeyPress` or `KeyRelease` events, this is set to a numeric version of the `.keysym` field. For regular keys that produce a single character, this field is set to the integer value of the key's ASCII code. For special keys, refer to Section 54.5, “Key names”.
`.num`	If the event was related to a mouse button, this attribute is set to the button number (1, 2, or 3). For mouse wheel support under Linux, bind `Button-4` and `Button-5` events; when the mouse wheel is scrolled up, this field will be 4, or 5 when scrolled down.
`.serial`	An integer serial number that is incremented every time the server processes a client request. You can use `.serial` values to find the exact time sequence of events: those with lower values happened sooner.
`.state`	An integer describing the state of all the modifier keys. See the table of modifier masks below for the interpretation of this value.
`.time`	This attribute is set to an integer which has no absolute meaning, but is incremented every millisecond. This allows your application to determine, for example, the length of time between two mouse clicks.
`.type`	A numeric code describing the type of event. For the interpretation of this code, see Section 54.3, “Event types”.
`.widget`	Always set to the widget that caused the event. For example, if the event was a mouse click that happened on a canvas, this attribute will be the actual `Canvas` widget.
`.width`	If the event was a `Configure`, this attribute is set to the widget's new width in pixels.
`.x`	The *x* coordinate of the mouse at the time of the event, relative to the upper left corner of the widget.
`.y`	The *y* coordinate of the mouse at the time of the event, relative to the upper left corner of the widget.
`.x_root`	The *x* coordinate of the mouse at the time of the event, relative to the upper left corner of the screen.
`.y_root`	The *y* coordinate of the mouse at the time of the event, relative to the upper left corner of the screen.

Use these masks to test the bits of the `.state` value to see what modifier keys and buttons were pressed during the event:

Mask	Modifier
`0x0001`	Shift.
`0x0002`	Caps Lock.
`0x0004`	Control.
`0x0008`	Left-hand Alt.
`0x0010`	Num Lock.
`0x0080`	Right-hand Alt.
`0x0100`	Mouse button 1.
`0x0200`	Mouse button 2.
`0x0400`	Mouse button 3.

Here's an example of an event handler. Under Section 54.1, "Levels of binding", above, there is an example showing how to bind mouse button 2 clicks on a canvas named `self.canv` to a handler called `self.__drawOrangeBlob()`. Here is that handler:

```
    def __drawOrangeBlob(self, event):
        '''Draws an orange blob in self.canv where the mouse is.
        '''
        r = 5   # Blob radius
        self.canv.create_oval(event.x-r, event.y-r,
            event.x+r, event.y+r, fill='orange')
```

When this handler is called, the current mouse position is `(event.x, event.y)`. The `.create_oval()` method draws a circle whose bounding box is square and centered on that position and has sides of length `2*r`.

Next: 54.7. The extra arguments trick
Contents: *Tkinter* 8.5 reference: a GUI for Python
Previous: 54.5. Key names
Home: About New Mexico Tech

John W. Shipman
Comments welcome: tcc-doc@nmt.edu

Last updated: 2013-12-31 17:59
URL: `http://www.nmt.edu/tcc/help/pubs/tkinter/web/event-handlers.html`