Blockext Documentation

Release 0.1

Connor Hudson, Tim Radvan

July 23, 2016

Contents

Blockext is the easiest way to write extensions for Scratch 2.0 and Snap! using the Python programming language.

Contents:

CHAPTER 1

Tutorial

> WARNING: this documentation is a work-in-progress, as is the library. So this document is incomplete and things will change. Sorry!

This tutorial shows you how to write extensions that are compatible with both Scratch 2.0 and Snap!.

It assumes familiarity with at least one of these programming languages. Don't worry if you've only used one – we'll explain the differences you need to know about.

It also assumes familiarity with Python, and that you've already installed blockext. See **install_** if not.

1.1 Example

Blockext is a Python module that makes writing extensions for these block-based programming languages fairly easy. Here's a quick example:

```
from blockext import *

class Tutorial:
    def __init__(self):
        self.light = False

    def do_toggle_light(self, times):
        for i in range(times):
            self.light = not self.light

    def is_light_on(self):
        return self.light

    def get_weather(self, day, city):
        import random
        return random.choice(["sunny", "cloudy", "windy"])

descriptor = Descriptor(
    name = "Tutorial Example",
    port = 5000,
    blocks = [
        Block('do_toggle_light', 'command', 'press light switch %n times',
            defaults=[1])
        Block('is_light_on', 'predicate', 'light is on?'),
        Block('get_weather', 'reporter', 'weather %m.day in %m.city'),
    ],
    menus = dict(
```

```
        day = ["today", "yesterday"],
        city = ["Barcelona", "Boston", "Bournemouth"],
    ),
)

extension = Extension(Tutorial, descriptor)

if __name__ == '__main__':
    extension.run_forever(debug=True)
```

Let's see it in action! Save and run the example, and then point your web browser to http://localhost:5000/. You'll then see a web page with the following options, above a list of the blocks you just defined:

- Download Scratch 2.0 extension
- Download Snap! blocks

Click each of them to save the blocks to your downloads folder.

1.2 In Scratch

Now, load up the Scratch 2.0 offline editor. Shift-click the "File" menu, and select "Import Experimental Extension". Select the `scratch_example.s2e` file to load the extension blocks into Scratch.

You can then select the purple "More Blocks" tab to see your blocks loaded into Scratch!

Try "say"-ing the "light is on?" and "weather forecast" blocks, and try using the "press switch" block to change the value of the light reporter.

1.3 In Snap!

Open http://snap.berkeley.edu/run in your browser, and drag the `snap_example.xml` file you just downloaded into the Snap! window.

If that doesn't work, use "Import" from the "File" menu instead.

1.4 Step-by-step

Now you've seen the example extension in action, let's break down the code. Here's the first line:

```
from blockext import *
```

This is just importing the entire contents of the `blockext` module. If you've done any Python, you might have seen this before. Next:

```
light = False

@command("press light switch %n times")
def toggle_light(times=1):
    global light
    for i in range(times):
        light = not light
```

The `@command` line is called a decorator. You don't need to know how it works, just that a line starting with an `@` symbol always goes before a function, and does something special to the function.

In this case, the `command` decorator is turning the function into a block definition for a command block. (Command blocks are the ones with the hole on the top and the puzzle-piece stub on the bottom.)

The string just after the `@command` part is the text that will get used on the block

The function's *name* is used internally so that Scratch/Snap! can recognise the block. This means that you can change the block's text without breaking existing projects that use your extension. As long as you don't change the function name, existing projects will still work.

Let's have a look at the rest of the blocks:

```
@predicate("light is on?")
def is_light_on():
    return light

@reporter("weather forecast for %m.city")
def forecast(city="Boston"):
    import random
    return random.choice(["windy", "snowy", "sunny"])
```

I skipped over this line:

```
menu("city", ["Barcelona", "Boston", "Brighton"])
```

This defines the options for the menu.

Now, the final line:

```
run("Tutorial Example", "example", 5000)
```

This starts the extension running on port 5000. We also specify its long name and (short) name. The long name is displayed to the user; the short name is used in the filenames. (NB. v0.2 will just have "name".)

- TODO: finish.
- TODO: rewrite for the new v0.2 interface.
- TODO: Doesn't crash if you throw an exception.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search