pandoc-source-exec examples

Sebastian Höffner

March 2017

Preamble

Compile this document as follows:

```
pandoc --filter pandoc-source-exec \
       --highlight-style tango \
       -o example.pdf example.md
```

Usage

To execute code, add the class `exec` to your code:

````
```{ .python .exec }
print('Hello World')
```
````

This results in:

```
print('Hello World')
```

Output:

```
Hello World
```

You can also supply the interpreter keys in the `runas` argument:

````
```{ .python .exec runas=python2 }
print 'Hello World'
```
````

Or you can simply make up your own command:

````
```{ .exec cmd='/usr/bin/env python2 -c' }
print 'Hello World'
```
````

Examples

No execution

```
a = 3 + 5
print(a)
```

Simple execution

Using: `{ .python .exec }`

```
print('Hello World')
```

Output:

```
Hello World
```

Advanced execution

Known interpreter `{ .python .exec runas=python2 }`:

```
print 'Hello World'
```

Output:

```
Hello World
```

Custom interpreter `{ .exec cmd='/usr/bin/env python2 -c' }`:

```
print 'Hello World'
```

Output:

```
Hello World
```

Or ruby `{ .exec cmd='/usr/bin/env ruby -e' }`:

```
puts 'Hello World!'
```

Output:

```
Hello World!
```

Errors

`stderr` is piped to `stdout`, so that errors can also be shown.

Using: `{ .python .exec }`

```
print('Hello
```

Output:

```
  File "<string>", line 1
    print('Hello
                ^
SyntaxError: EOL while scanning string literal
```

File execution

Using: `{ .python .exec file='example.py' }`

File: `example.py`

```
import math

a = 3
b = 4
c = math.sqrt(a ** 2 + b ** 2)

print(c)
```

Output:

```
5.0
```

File without execution

Using: `{ .python file='example.py' }`

File: `example.py`

```
import math

a = 3
b = 4
c = math.sqrt(a ** 2 + b ** 2)

print(c)
```

Interactive execution

Using: `{ .python .exec .interactive }`

Interactive code will also be detected if the code block starts with `>>>`.

Note: This only works with python code so far, a custom command is not possible.

Note: The REPLWrapper changed, so this does only provide very limited support. In particular, only single-line-statements can be executed.

```
>>> a = 5 + 4
>>> 9 == a
True
>>> print(a)
9
```

API

The following keywords (classes denoted by a prefixed `.`, attributes with a following `=`) exist:

`.caption` and `caption=` Mutually exclusive. If `.caption` is used, instead of printing `File: ...` above the code, a caption is created above (using the LaTeX package `caption`) the listing and in the compiled LaTeX document the `\listofcodelistings` macro becomes available. To specify a custom caption, use `caption="My caption"`. If a filename was specified, this would render to "My caption (path/to/file.py)".

`.capbelow` Places the captions below the listing.

`shortcaption=` A short caption to be used in the list of code listings.

`cmd=` Allows to specify a custom interpreter command to execute the code. For example, to run ruby code one could use `cmd='ruby -e'`.

`.exec` Executes the following code cell according to the specified language. By default, it is only `echo`ed.

`file=` Replaces the code cell with content from the specified file. This searches recursively for files matching the pattern, so if you use `file=code.py` but your code is in fact in `src/code.py` it will still be found. Specify the full path to avoid ambiguities.

`.interactive` Executes the code as if it was inserted into an interactive session, returns results inline into the original code block. Only works for python code so far.

`runas=` Executes code with the specified executor, e.g. `python2` to distinguish it from `python` which defaults to `python3`. Can be overwritten by specifying `cmd=`.

`.hideimports` Hides import statements in output. Currently only supported for Python.

`pathlength=` Limits the number of path elements for a filename. If a path is e.g. `a/b/c/code.py` and `pathlength=2`, only `c/code.py` is shown. This is only useful using `file=`.

Supported languages

To be used with `runas=`, if it does not already match the language identifier:

- default
- perl
- php
- python
- python2
- python3
- ruby

default

```
default
```

Output:

```
default
```

perl

```
print 'perl';
```

Output:

```
perl
```

php

```
echo 'php';
```

Output:

```
php
```

python

```
print('python')
```

Output:

```
python
```

python2

```
print 'runas=python2'
```

Output:

```
runas=python2
```

python3

```
print('runas=python3')
```

Output:

```
runas=python3
```

ruby

```
puts 'ruby'
```

Output:

```
ruby
```

Removing imports

Removing imports affects only the final code rendering, not the execution.

````
```{ .python .exec .hideimports }
import statistics


print(statistics.mean([1, 2, 3])
```
````

Results in

```
print(statistics.mean([1, 2, 3]))
```

Output:

```
2
```

Plotting matplotlib

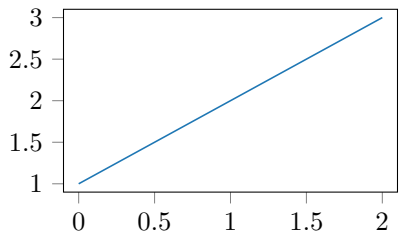
````
```{ .python .exec .plt }
import matplotlib.pyplot as plt

plt.plot([1, 2, 3])
```
````

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3])
```

Output:



Additionally `width=6cm` and `height=5cm` can be used. As a shortcut, one can instead use `plt=6cm,5cm`.

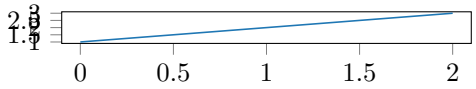
````
```{ .python .exec .plt width=7cm height=2cm }
import matplotlib.pyplot as plt

plt.plot([1, 2, 3])
```
````

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3])
```

Output:



Captions

Captions make proper "listing" environments, which are floating. They are set to `[htbp]`.

A normal “captionized” file

This is Code Listing 1.

````
```{ .python .caption file='example.py' }
```
````

Code Listing 1: `example.py`

```python
import math

a = 3
b = 4
c = math.sqrt(a ** 2 + b ** 2)

print(c)
```

A custom caption

This is Code Listing 2.

````
```{ .python caption="Custom caption" file='example.py' }
```
````

Code Listing 2: Custom caption (`example.py`)

```python
import math

a = 3
b = 4
c = math.sqrt(a ** 2 + b ** 2)

print(c)
```

Caption for a normal code block

This is Code Listing 3.

````
```{ .python caption="Caption for a normal code block" }
print('Hello World!')
```
````

Code Listing 3: Caption for a normal code block

```
print('Hello World!')
```

Empty caption

This is Code Listing 4. Note that empty captions are not included in the list of code listings (see below).

````
```{ .python .caption }
print('Hello World!')
```
````

Code Listing 4

```
print('Hello World!')
```

Short captions

Sometimes, long captions are too much for the list of code listings, thus you can provide a short caption:

````
```{ .python caption="Long caption" shortcaption="Short" }
print('Hello World!')
```
````

Code Listing 5: Long caption

```
print('Hello World!')
```

Placing a caption below

By default, captions are placed above the code block. By using the class `capbelow`, this can be changed:

````
```{ .python caption="Caption below" .capbelow }
print('Hello World!')
```
````

Caption with execution does also work

This is Code Listing 5 with an executed code block.

```
print('Hello World!')
```

Code Listing 6: Caption below

````
```{ .python .exec caption="Simple 'Hello World'" }
print('Hello World!')
```
````

Code Listing 7: Simple ‘Hello World’

```
print('Hello World!')
```

Output:

```
Hello World!
```

List of Code Listings

```
\listofcodelistings
```

List of Code Listings