

---

# **nukecontexts Documentation**

***Release 0.2.0***

**Florian Einfalt**

**Jun 25, 2018**



---

## Contents

---

|          |                            |           |
|----------|----------------------------|-----------|
| <b>1</b> | <b>Installation</b>        | <b>3</b>  |
| <b>2</b> | <b>Getting Started</b>     | <b>5</b>  |
| <b>3</b> | <b>Advanced Use</b>        | <b>7</b>  |
| 3.1      | Logging . . . . .          | 7         |
| 3.2      | Sentry support . . . . .   | 7         |
| <b>4</b> | <b>API Documentation</b>   | <b>9</b>  |
| <b>5</b> | <b>Indices and tables</b>  | <b>11</b> |
|          | <b>Python Module Index</b> | <b>13</b> |



Contents:



# CHAPTER 1

---

## Installation

---

To install `nukecontexts`, type:

```
$ pip install nukecontexts
```

Open Nuke's `init.py` file and add:

```
nuke.pluginAddPath('/path/to/your/local/python/site-packages')
```



## CHAPTER 2

---

### Getting Started

---

`nukecontexts` is a library of composable context managers for Nuke to manage the state of complex compositing scripts in code.

The most common use case for `nukecontexts` is automated rendering of multiple states of a compositing script. For example two different output formats, `jpg` and `png`.

```
import nuke
import contextlib
from nukecontexts import ctx

render_node = nuke.toNode('Write1')
with ctx.set_attr(render_node, 'file_type', 'jpeg'):
    nuke.execute(render_node.name(), 1, 1, 1)
with ctx.set_attr(render_node, 'file_type', 'png'):
    nuke.execute(render_node.name(), 1, 1, 1)
```

The power of `nukecontexts` comes with composable contexts, using `contextlib.nested()`. Arbitrarily complex, varying states of the compositing script can be defined and used to automatically generate different results.

```
merge_node = nuke.toNode('Merge1')
grade_node = nuke.toNode('Grade1')
switch_node = nuke.toNode('Switch1')

ctx1 = ctx.enable([merge_node, grade_node])
ctx2 = ctx.set_attr(grade_node, 'white', 2.0)
ctx3 = ctx.set_attr(switch_node, 'which', 0)
ctx4 = ctx.disable(merge_node)

with contextlib.nested(ctx1, ctx2, ctx3):
    """Render with the merge_node and grade_node enabled, the
    grade_node's white attribute set to 2.0 and the switch_node's switch
    position set to 0."""
    nuke.execute(render_node.name(), 1, 1, 1)
```

(continues on next page)

(continued from previous page)

```
with contextlib.nested(ctx3, ctx4):  
    """Render with the switch_node's switch position set to 0 and the  
    merge node disabled; the grade_node's gain value remains at the  
    original value."""  
    nuke.execute(render_node.name(), 1, 1, 1)
```

### 3.1 Logging

`nukecontexts` creates its own logger on import that is used by all context managers to indicate when contexts are entered and exited. The standard logger logs to `stdout`.

Should your pipeline have more advanced logging needs, simply pass your custom logger to each context manager, using the `log` keyword argument.

### 3.2 Sentry support

`nukecontexts` offers optional support for the [Sentry](#) error tracking service. To use [Sentry](#) with `nukecontexts`, install [Raven](#) into your environment, set the `SENTRY_DSN` environment variable before importing `nukecontexts` and you're good to go.



**exception** `nukecontexts.ctx.NukeContextError` (*message*, \**args*)

`nukecontexts.ctx.disabled` (\**args*, \*\**kws*)

Given a list of nodes (Node), disable on entry and restore to original value on exit.

**Parameters**

- **nodes** (*list*) – Nodes
- **log** (*logging.Logger*) – Logger

`nukecontexts.ctx.enabled` (\**args*, \*\**kws*)

Given a list of nodes (Node), enable on entry and restore to original value on exit.

**Parameters**

- **nodes** (*list*) – Nodes
- **log** (*logging.Logger*) – Logger

`nukecontexts.ctx.inventory` (\**args*, \*\**kws*)

Given a variable name, create a node inventory on entry and a separate node inventory on exit and save any new nodes into the newly created variable.

Beware that the new variable is created in `__builtins__` and is therefore accessible even after the context manager has exited.

**Use with namespace in mind!**

**Parameters** **var** (*str*) – Variable name

`nukecontexts.ctx.multiple_contexts` (\**args*, \*\**kws*)

Given a list of contextmanagers, sequentially enter all contextmanagers, raise `Exception` in case errors occur in contexts.

**Deprecated.** Use `contextlib.nested(*contexts)()`.

**Parameters** **contexts** (*list*) – List of contextmanagers

`nukecontexts.ctx.set_attr(*args, **kws)`

Given a list of nodes (Node), set a given attr to value on entry and restore to original value on exit.

#### Parameters

- **nodes** (*list*) – Nodes
- **attr** (*str*) – Attribute
- **value** (*str, int, float, bool*) – Value
- **log** (*logging.Logger*) – Logger

**class** `nukecontexts.ctx.Progress` (*iterable, name='nukecontexts', output=<open file '<stdout>', mode 'w'>*)

Convenience wrapper class around `tqdm.tqdm()` for easy progress bars

Usage:

```
>>> with Progress(iterable) as progress:
>>>     for item in progress:
>>>         #do something
```

`__init__` (*iterable, name='nukecontexts', output=<open file '<stdout>', mode 'w'>*)

#### Parameters

- **iterable** (*iter*) – Iterable to generate progress bar for
- **name** (*str*) – Progress bar label (default: 'nukecontexts')
- **output** (*io.TextIOWrapper or io.StringIO*) – Output stream (default: `sys.stdout`)

**class** `nukecontexts.ctx.AttributeSetter` (*nodes, attr, value, log=<logging.RootLogger object>*)

`__init__` (*nodes, attr, value, log=<logging.RootLogger object>*)

Given a list of nodes (Node), set a given attr to value on entry and restore to original value on exit.

#### Parameters

- **nodes** (*list*) – Nodes
- **attr** (*str*) – Attribute
- **value** (*str, int, float, bool*) – Value
- **log** (*logging.Logger*) – Logger

## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### n

`nukecontexts.ctx`, 9



## Symbols

`__init__()` (nukecontexts.ctx.AttributeSetter method), [10](#)

`__init__()` (nukecontexts.ctx.Progress method), [10](#)

## A

AttributeSetter (class in nukecontexts.ctx), [10](#)

## D

`disabled()` (in module nukecontexts.ctx), [9](#)

## E

`enabled()` (in module nukecontexts.ctx), [9](#)

## I

`inventory()` (in module nukecontexts.ctx), [9](#)

## M

`multiple_contexts()` (in module nukecontexts.ctx), [9](#)

## N

NukeContextError, [9](#)

nukecontexts.ctx (module), [9](#)

## P

Progress (class in nukecontexts.ctx), [10](#)

## S

`set_attr()` (in module nukecontexts.ctx), [9](#)