

---

# **uarray Documentation**

*Release 0.5.2+0.gac77bea.dirty*

**Quansight-Labs**

**Jul 28, 2019**



---

## Contents

---

<b>1</b>	<b>What's new in unumpy?</b>	<b>3</b>
<b>2</b>	<b>Relation to the NumPy duck-array ecosystem</b>	<b>5</b>
2.1	unumpy . . . . .	5
2.1.1	Writing Backends . . . . .	6
2.1.2	Meta-array support . . . . .	7
<b>3</b>	<b>Indices and tables</b>	<b>9</b>
	<b>Python Module Index</b>	<b>11</b>
	<b>Index</b>	<b>13</b>



**Note:** This page describes the overall philosophy behind *unumpy*. If you are interested in an augmentation for NEP-22, please see the *unumpy* page.

---

*unumpy* builds on top of *uarray*. It is an effort to specify the core NumPy API, and provide backends for the API.



# CHAPTER 1

---

## What's new in `unumpy`?

---

`unumpy` is the first approach to leverage `uarray` in order to build a generic backend system for (what we hope will be) the core NumPy API specification. It's possible to create the backend object, and use that to perform operations. In addition, it's possible to change the used backend via a context manager.





---

## Relation to the NumPy duck-array ecosystem

---

`uarray` is a backend/dispatch mechanism with a focus on array computing and the needs of the wider array community, by allowing a clean way to register an implementation for any Python object (functions, classes, class methods, properties, dtypes, ...), it also provides an important building block for [NEP-22](#). It is meant to address the shortcomings of [NEP-18](#) and [NEP-13](#); while still holding nothing in `uarray` itself that's specific to array computing or the NumPy API.

### 2.1 unumpy

---

**Note:** If you are interested in writing backends or multimethods for `unumpy`, please look at the documentation for `uarray`, which explains how to do this.

---

`unumpy` is meant for three groups of individuals:

- Those who write array-like objects, like developers of Dask, Xnd, PyData/Sparse, CuPy, and others.
- Library authors or programmers that hope to target multiple array backends, listed above.
- Users who wish to target their code to other backends.

For example, the following is currently possible:

```
>>> import uarray as ua
>>> import unumpy as np
>>> import unumpy.dask_backend as dask_backend
>>> import unumpy.sparse_backend as sp_backend
>>> import sparse, dask.array as da
>>> def main():
...     x = np.zeros(5)
...     return np.exp(x)
>>> with ua.set_backend(dask_backend):
...     isinstance(main(), da.core.Array)
```

(continues on next page)

(continued from previous page)

```
True
>>> with ua.set_backend(sp_backend):
...     isinstance(main(), sparse.SparseArray)
True
```

Now imagine some arbitrarily nested code, all for which the implementations can be switched out using a simple context manager.

`unumpy` is an in-progress mirror of the NumPy API which allows the user to dynamically switch out the backend that is used. It also allows auto-selection of the backend based on the arguments passed into a function. It does this by defining a collection of `uarray` multimethods that support dispatch. Although it currently provides a number of backends, the aspiration is that, with time, these back-ends will move into the respective libraries and it will be possible to use the library modules directly as backends.

Note that currently, only coverage is very incomplete. However, we have attempted to provide at least one of each kind of object in `unumpy` for reference. There are `ufunc`s and `ndarray`s, which are classes, methods on `ufunc` such as `__call__`, and `reduce` and also functions such as `sum`.

Where possible, we attempt to provide default implementations so that the whole API does not have to be reimplemented, however, it might be useful to gain speed or to re-implement it in terms of other functions which already exist in your library.

The idea is that once things are more mature, it will be possible to switch out your backend with a simple import statement switch:

```
import numpy as np # Old method
import unumpy as np # Once this project is mature
```

Currently, the following functions are supported:

- All NumPy universal functions.
  - `ufunc` reductions

For the full range of functions, use `dir(unumpy)`.

You can use the `uarray.set_backend` decorator to set a backend and use the desired backend. Note that not every backend supports every method. For example, PyTorch does not have an exact `ufunc` equivalent, so we dispatch to actual methods using a dictionary lookup. The following backends are supported:

- `numpy_backend`
- `torch_backend`
- `xnd_backend`
- `dask_backend`
- `cupy_backend`
- `sparse_backend`

### 2.1.1 Writing Backends

Since `unumpy` is based on `uarray`, all overrides are done via the `__ua_*__` protocols. We strongly recommend you read the [uarray documentation](#) for context.

All functions/methods in `unumpy` are `uarray` multimethods. This means you can override them using the `__ua_function__` protocol.

In addition, `unumpy` provides `numpy.ndarray`, `numpy.ufunc` and `numpy.dtype` as dispatchables that can be checked and converted via the `__ua_convert__` protocol.

## 2.1.2 Meta-array support

If meta-arrays and libraries depend on `unumpy` instead of NumPy, they can benefit from containerization and hold arbitrary arrays; not just `numpy.ndarray` objects.

Inside their `__ua_function__` implementation, they might need to do something like the following:

```
>>> class Backend: pass
>>> meta_backend = Backend()
>>> meta_backend.__ua_domain__ = "numpy"
>>> def ua_func(f, a, kw):
...     # We do this to avoid infinite recursion
...     with ua.skip_backend(meta_backend):
...         # Actual implementation here
...         pass
>>> meta_backend.__ua_function__ = ua_func
```

In this form, one could do something like the following to use the meta-backend:

```
>>> with ua.set_backend(sp_backend), ua.set_backend(meta_backend):
...     np.zeros(200)
```

And it would perform the correct thing.



## CHAPTER 3

---

### Indices and tables

---

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



**u**

unumpy, 5





## U

unumpy (*module*), 5