# lcdata

***Release 1.1.1***

**Kyle Boone**

**Sep 17, 2021**

# CONTENTS

# CHAPTER ONE

# ABOUT

lcdata is a package for manipulating large datasets of astronomical time series. lcdata is designed to handle very large datasets: it uses a compact internal representation to be able to keep many light curves in memory at the same time. For datasets that are too large to fit in memory, it offers the option of reading them from disks in chunks. lcdata also contains tools to download different publicly available releases of astronomical time series.

## 1.1 Installation

lcdata requires Python 3.6+ and depends on the following Python packages:

- astropy
- h5py
- numpy
- pytables
- pyyaml
- requests
- tqdm

### 1.1.1 Install using pip (recommended)

lcdata is available on PyPI. To install the latest release:

```
pip install lcdata
```

### 1.1.2 Install development version

The lcdata source code can be found on github.

To install it:

```
git clone git://github.com/kboone/lcdata
cd lcdata
pip install -e .
```

## 1.2 Usage

### 1.2.1 Overview

lcdata is designed to handle large datasets of light curves. Light curves are represented as tables in ~astropy.table.Table format, and are very similar to the ones used in sncosmo. A dataset can be created in several ways. For example, we can create a dataset from a list of sncosmo-like light curves:

```
>>> import lcdata
>>> import sncosmo
>>> light_curves = [sncosmo.load_example_data() for i in range(5)]
>>> dataset = lcdata.from_light_curves(light_curves)
```

The individual light curves in this dataset can be accessed as dataset.light_curves.

### 1.2.2 Metadata

The metadata associated with all of the light curves can be accessed from a common ~astropy.table.Table as dataset.meta:

```
>>> dataset.meta
      object_id        ra  dec   type   redshift  x1   c          x0            t0
--------------------- --- --- ------- -------- --- --- ------------------ -------
lcdata_xbdwhv_0000000 nan nan Unknown      0.5 0.5 0.2 1.20482820761e-05 55100.0
lcdata_xbdwhv_0000001 nan nan Unknown      0.5 0.5 0.2 1.20482820761e-05 55100.0
lcdata_xbdwhv_0000002 nan nan Unknown      0.5 0.5 0.2 1.20482820761e-05 55100.0
lcdata_xbdwhv_0000003 nan nan Unknown      0.5 0.5 0.2 1.20482820761e-05 55100.0
lcdata_xbdwhv_0000004 nan nan Unknown      0.5 0.5 0.2 1.20482820761e-05 55100.0
```

lcdata enforces a consistent metadata format. All light curves are guaranteed to have the following keys in their metadata.

- object_id: A unique identifer. Default: randomly assigned string
- ra: The right ascension. Default: nan
- dec: The declination. Default: nan
- type: A string representing the type of the light curve. Default: Unknown
- redshift: The redshift. Default: nan

Astronomical data comes in many different formats, and keyword usage is not standardized. lcdata will try to find all of these keys in the metadata using a list of known aliases.

```
>>> light_curve = sncosmo.load_example_data()
>>> light_curve.meta = {
...     'id': 'example_id',
...     'right_ascension': 1.,
...     'decl': 2.,
...     'class': 'Type Ia',
...     'other_var': 5.
... }
```

```
>>> dataset = lcdata.from_light_curves([light_curve])
>>> print(dataset.meta)
object_id   ra dec   type  redshift other_var
---------- --- --- ------- -------- ---------
example_id 1.0 2.0 Type Ia      nan       5.0
```

### 1.2.3 Light Curves

lcdata will standardize the format of light curves, similarly to how the metadata is standardized. Each light curve is guaranteed to have the following keys:

- time: times at which the light curve was sampled. Converted to a 64-bit float.
- flux: The flux at each point on the light curve. Converted to a 32-bit float.
- fluxerr: The uncertainty on the flux. Converted to a 32-bit float.
- band: A string representing bandpass that the light curve was observed in. We recommend using the sncosmo bandpass names here. Converted to a binary string.

Additional columns are left as is. If the light curve columns have different labels, lcdata will try to infer which ones are which using a set of aliases.

```
>>> light_curve = astropy.table.Table({
...     'bandpass': ['lsstu', 'lsstb', 'lsstr'],
...     'flux': [1., 2., 10.],
...     'mjd': [59000., 59010., 59020.],
...     'fluxerr': [1., 0.5, 3.],
...     'myvar': [1., 2., 5.],
... })
>>> print(dataset.light_curves[0])
  time  flux fluxerr  band myvar
------- ---- ------- ----- -----
59000.0  1.0     1.0 lsstu   1.0
59010.0  2.0     0.5 lsstb   2.0
59020.0 10.0     3.0 lsstr   5.0
```

### 1.2.4 Dataset Manipulation

Datasets can be manipulated in various ways.

Addition:

```
>>> dataset = dataset1 + dataset2
```

Selecting a subset:

```
>>> dataset = dataset[5:10]
```

### 1.2.5 Saving a Dataset in HDF5 format

lcdata has an optimized HDF5 reader/writer that can be used to rapidly load very large light curve datasets.

Datasets can be read from and written out to disk in HDF5 format.

```
>>> dataset.write_hdf5('./dataset.h5')
```

```
>>> dataset = lcdata.read_hdf5('./dataset.h5')
```

A dataset on disk can be appended to:

```
>>> dataset_2.write_hdf5('./dataset.h5', append=True)
```

Some datasets are too large to fit in memory all at once. lcdata can load only the metadata of a dataset into memory, and then access the light curves themselves on demand.

```
>>> # Read only the metadata
>>> dataset = lcdata.read_hdf5('./dataset.h5', in_memory=False)
```

```
>>> # Read a specific light curve
>>> light_curve = dataset.light_curves[10]
```

```
>>> # Select a subset of the dataset and load all of its light curves into memory.
>>> subset = dataset[1000:2000].load()
```

A common use case for this functionality is to process all of the light curves in the dataset in chunks. lcdata provides a helper to do this:

```
>>> for chunk in dataset.iterate_chunks(chunk_size=1000):
...     # At each iteration, chunk is an lcdata Dataset with the next 1000
...     # light curves.
```

## 1.3 Built-in Datasets

lcdata contains scripts that can be used to download commonly-used light curve datasets and save them in an efficient and fast-to-read HDF5 format. The currently-supported datasets are:

| Dataset | Script | Reference |
|---|---|---|
| PLAsTiCC | `lcdata_download_plasticc` | PLAsTiCC |
| PanSTARRS-1 | `lcdata_download_ps1` | Villar et al. 2020 |

After installing lcdata, these scripts can be run from any directory on the command line to download the corresponding dataset(s). By default, these will be placed in `./data/`, although the location can be changed with the `--directory` flag.

### 1.3.1 Using downloaded datasets

A full description of how datasets are used in lcdata can be found on the *Usage* page.

To open a dataset that can fit in memory and read metadata/light curves from it:

```
>>> dataset = lcdata.read_hdf5('./data/ps1.h5')

>>> print(dataset.meta)
<Table length=5243>
object_id    ra       dec      type  redshift ...
   str9    float64  float64   str13  float64  ...
--------- -------- -------- ------- -------- ...
PS0909006 333.9503   1.1848    SNIa    0.284 ...
PS0909010  37.1182  -4.0789    SNIa     0.27 ...
PS0910012  52.4718 -28.0867   SNIax    0.079 ...
PS0910016  35.3073    -3.91    SNIa     0.23 ...
      ...      ...      ...     ...      ... ...

>>> print(dataset.light_curves[0])
<Table length=376>
  time    flux  fluxerr  band
float64 float32 float32 bytes6
------- ------- ------- ------
55029.6  -9.919  11.995 ps1::g
55074.4 211.372   33.03 ps1::g
55086.4 112.477  10.463 ps1::g
55089.4 102.993   10.58 ps1::g
    ...     ...     ...    ...
```

For large datasets that can't fit into memory, a common workflow is to process the dataset in smaller chunks that can fit in memory. To load a large dataset in chunks of 1000 light curves:

```
>>> dataset = lcdata.read_hdf5('./data/plasticc_test.h5', in_memory=False)

>>> for chunk in dataset.iterate_chunks(chunk_size=1000):
...     # At each iteration, chunk is an lcdata Dataset with the next 1000
...     # light curves.
```

## 1.4 Reference / API

### 1.4.1 Datasets

| | |
|---|---|
| `Dataset`(meta[, light_curves]) | A dataset of light curves. |
| `LightCurveMetadata`(meta_row) | Class to handle the metadata for a light curve in a dataset. |
| `HDF5Dataset`(path, meta) | Dataset corresponding to an HDF5 file on disk. |
| `HDF5LightCurves`(dataset) | Class to interface with light curves in an HDF5 file on disk. |

### lcdata.Dataset

**class** lcdata.**Dataset**(*meta*, *light_curves=None*)

A dataset of light curves.

**Parameters**

- **meta** (Table) – Metadata table.
- **light_curves** (List[Table], optional) – List of light curves where each light curve is represented by an astropy Table.

**__init__**(*meta*, *light_curves=None*)

#### Methods

| | |
|---|---|
| *__init__*(meta[, light_curves]) | |
| add_meta(meta[, suffix, warn_on_disagreement]) | Add additional metadata into the dataset. |
| get_lc([key]) | |
| get_sncosmo_lc(key, **kwargs) | Get a light curve in sncosmo format. |
| write_hdf5(path[, append, overwrite, ...]) | Write the dataset to an HDF5 file |

### lcdata.LightCurveMetadata

**class** lcdata.**LightCurveMetadata**(*meta_row*)

Class to handle the metadata for a light curve in a dataset.

The dataset has a metadata table with rows for each light curve. This class is a view into the metadata table that behaves like a dict. Modifying it will update the underlying metadata table.

**Parameters** **meta_row** (Row) – Row in the metadata table corresponding to a single light curve.

**__init__**(*meta_row*)

#### Methods

| | |
|---|---|
| *__init__*(meta_row) | |
| clear() | |
| copy([use_cache, update_cache]) | |
| get(k[,d]) | |
| items() | |
| keys() | |
| pop(k[,d]) | If key is not found, d is returned if given, otherwise KeyError is raised. |

continues on next page

Table 3 – continued from previous page

| | |
|---|---|
| popitem() | as a 2-tuple; but raise KeyError if D is empty. |
| setdefault(k[,d]) | |
| update([E, ]**F) | If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v |
| values() | |

### lcdata.HDF5Dataset

**class** lcdata.**HDF5Dataset**(*path*, *meta*)

Dataset corresponding to an HDF5 file on disk.

This class only loads the metadata by default. Accessing a light curve in the dataset will read it from disk.

Typically you won't want to use this class directly. Instead, call *lcdata.read_hdf5* with `in_memory` set to False to read a file.

**Parameters**

- **path** (*str*) – Path to the file.
- **meta** (Table) – Metadata table.

**__init__**(*path*, *meta*)

#### Methods

| | |
|---|---|
| *__init__*(path, meta) | |
| add_meta(meta[, suffix, warn_on_disagreement]) | Add additional metadata into the dataset. |
| count_chunks(chunk_size) | Count the number of chunks that are in the dataset for a given chunk_size |
| get_chunk(chunk_idx, chunk_size) | Get a chunk from the dataset |
| get_lc([key]) | |
| get_sncosmo_lc(key, **kwargs) | Get a light curve in sncosmo format. |
| iterate_chunks(chunk_size) | Iterate through the dataset in chunks |
| load() | Load the current dataset into memory. |
| write_hdf5(path[, append, overwrite, ...]) | Write the dataset to an HDF5 file |

### lcdata.HDF5LightCurves

**class** lcdata.**HDF5LightCurves**(*dataset*)

Class to interface with light curves in an HDF5 file on disk.

The light curves are kept on disk, and only loaded into memory when explicitly asked for.

**Parameters** **dataset** (*HDF5Dataset*) – Dataset to handle the light curves for.

**__init__**(*dataset*)

#### Methods

| | |
|---|---|
| *__init__*(dataset) | |
| count(value) | |
| index(value, [start, [stop]]) | Raises ValueError if the value is not present. |
| load([start_idx, end_idx]) | Load a series of light curves into memory |

*Loading a dataset*

| | |
|---|---|
| *from_light_curves*(light_curves) | Load a dataset from a list of light curves. |
| *from_observations*(meta, observations) | Load a dataset from a table of all of the observations. |
| *from_avocado*(name, **kwargs) | Load a dataset from avocado. |
| *read_hdf5*(path[, in_memory]) | Read a dataset from an HDF5 file |

### lcdata.from_light_curves

lcdata.**from_light_curves**(*light_curves*)

Load a dataset from a list of light curves.

**Parameters** **light_curves** (List[Table]) – List of light curves

**Returns** Dataset containing all of these light curves.

**Return type** *Dataset*

### lcdata.from_observations

lcdata.**from_observations**(*meta*, *observations*)

Load a dataset from a table of all of the observations.

**Parameters**

- **meta** (Table) – Table containing the metadata with one row for each light curve.
- **observations** (Table) – Table containing all of the observations.

**Returns** A Dataset of light curves built from these tables.

**Return type** *Dataset*

### lcdata.from_avocado

`lcdata.`**`from_avocado`**(*name*, ***kwargs*)

Load a dataset from avocado.

**Parameters** **name** (*str*) – Name of the dataset to load.

**Returns** the loaded dataset in lcdata format.

**Return type** *Dataset*

### lcdata.read_hdf5

`lcdata.`**`read_hdf5`**(*path*, *in_memory=True*)

Read a dataset from an HDF5 file

**Parameters**

- **path** (*str*) – Path of the dataset
- **in_memory** (*bool*) –

*Manipulating light curves*

| | |
|---|---|
| *parse_light_curve*(light_curve[, parse_meta, ...]) | Parse a light curve and convert it to lcdata format. |
| *to_sncosmo*(light_curve) | Convert an lcdata light curve to sncosmo format. |
| *generate_object_id*() | Generate a random unique object ID for a light curve. |

### lcdata.parse_light_curve

`lcdata.`**`parse_light_curve`**(*light_curve*, *parse_meta=True*, *verbose=False*)

Parse a light curve and convert it to lcdata format.

We currently assume that the light curve has a zeropoint of 25 in the AB magnitude system. The zeropoint/zpsys columns that are present in sncosmo-formatted light curves are ignored. This should be improved at some point.

**Parameters**

- **light_curve** (Table) – Input light curve in an arbitrary format
- **parse_meta** (*bool, optional*) – Whether to parse the metadata, by default True

**Returns** Light curve in lcdata format

**Return type** Table

### lcdata.to_sncosmo

`lcdata.`**`to_sncosmo`**(*light_curve*)

Convert an lcdata light curve to sncosmo format.

This adds the zp and zpsys keys that are required by sncosmo, and converts the band name to a string instead of the bytes type used internally.

**Parameters** **light_curve** (Table) – Light curve in lcdata format

**Returns** Light curve in sncosmo format

**Return type** Table

### lcdata.generate_object_id

lcdata.**generate_object_id**()

Generate a random unique object ID for a light curve.

We want to make this unique but readable. It is also important that if different datasets are generated with different runs of they program they have different IDs. To accomplish this, we use the format lcdata_[random session string]_[count]. The random_session_string will be consistent for all of the light curves generated in the same session. The count will start from zero and increase.

## 1.4.2 Schemas

| | |
|---|---|
| *schema.verify_schema*(schema) | Verify a schema |
| *schema.get_default_value*(schema, key[, count]) | Get the default value for a key in a schema. |
| *schema.find_alias*(names, aliases) | Given a list of names, find the one that matches a list of aliases. |
| *schema.format_table*(table, schema[, verbose]) | Format a table with a given schema. |

### lcdata.schema.verify_schema

lcdata.schema.**verify_schema**(*schema*)

Verify a schema

**Parameters** **schema** (*dict[dict]*) – Schema to verify. See `schema.py` for details.

**Raises** `ValueError` – For any noncomplient schemas. The error message will describe what part of the schema is invalid.

### lcdata.schema.get_default_value

lcdata.schema.**get_default_value**(*schema*, *key*, *count=None*)

Get the default value for a key in a schema.

**Parameters**

- **schema** (*dict[dict]*) – Schema to compare to
- **key** (*str*) – Key to look for in the schema.
- **count** (*int, optional*) – For default functions that return different values, the number of values to return. By default, only a single value is returned.

**Returns** The default value parsed from the schema.

**Return type** default_value

**Raises** `ValueError` – If a default value does not exist for this key in the schema.

#### lcdata.schema.find_alias

`lcdata.schema.find_alias(names, aliases)`

Given a list of names, find the one that matches a list of aliases.

Inspired by and very similar to `sncosmo.alias_map`.

**Parameters**

- **names** (*list[str]*) – List of names that are available
- **aliases** (*list[str]*) – List of aliases to search through. The first one that is available will be returned.

**Returns** **alias** – Matching alias is one was found, or None otherwise.

**Return type** str or None

#### lcdata.schema.format_table

`lcdata.schema.format_table(table, schema, verbose=False)`

Format a table with a given schema.

**Parameters**

- **table** (`Table`) – Table to format
- **schema** (*dict[dict]*) – Schema to use for formatting
- **verbose** (*bool, optional*) – Whether to print debugging messages, by default False

**Returns** Formatted table

**Return type** `Table`

**Raises** `ValueError` – If there are required keys in the schema that are missing in the table.

### 1.4.3 Utilities

| | |
|---|---|
| *utils.download_file*(url, path[, filesize]) | Download a file with a tqdm progress bar. |
| *utils.download_zenodo*(record, basedir) | Download a record from Zenodo. |

#### lcdata.utils.download_file

`lcdata.utils.download_file(url, path, filesize=None)`

Download a file with a tqdm progress bar.

This will check if the file already exists, and skip it if it does. If the filesize is known, this function verifies that the function on disk has the right size, and redownloads it if it doesn't.

**Parameters**

- **url** (*str*) – URL to download from
- **path** (*str*) – Path on disk to download the file to.
- **filesize** (*int, optional*) – Size of the file (if known), by default None

### lcdata.utils.download_zenodo

`lcdata.utils.`**`download_zenodo`**`(`*record*, *basedir*`)`

Download a record from Zenodo.

**Parameters**

- **`record`** (*`str`*) – Zenodo record number.
- **`basedir`** (*`str`*) – Directory to download the record to.

Source code: https://github.com/kboone/lcdata

# INDEX

## Symbols



## D



## F



## G



## H



## L



## P



## R



## T



## V