Draft version May 31, 2024
Typeset using LaTeX default style in AASTeX631

The OTTER Catalog

Kate Alexander,[1] Noah Franz,[1] Suvi Gezari,[2,3] Sebastian Gomez,[2] Mitchell Karmen,[3] and V. Ashley Villar[4]

[1] *Department of Astronomy/Steward Observatory, 933 North Cherry Avenue, Tucson, AZ 85721-0065, USA*
[2] *Space Telescope Science Institute, 3700 San Martin Drive, Baltimore, MD 21218, USA*
[3] *Department of Physics and Astronomy, Johns Hopkins University, 3400 North Charles Street, Baltimore, MD 21218, USA*
[4] *Center for Astrophysics | Harvard & Smithsonian, 60 Garden Street, Cambridge, MA 02138-1516, USA*

Contents

1. SCHEMA

This document outlines the schema that the OTTER will use for the `json` files that contain all the relevant information for transients in the catalog. An Entity Relationship (ER) diagram of this schema is also available at https://dbdiagram.io/d/The-New-OSC-Schema-65249d45ffbf5169f05afd19. The GitHub organization that contains the database, schema, and frontend is available at https://github.com/astro-otter.

1.1. *Definitions*

The data for each transient object will be stored in individual `json` files with a uniform structure, where the name of each file will be the same as the name of the transient object. Here, we describe the schema for the database, as well as the different types of data that can be stored in the `json` files.

- **object**: each astronomical transient object (TDE, SN, FRB, etc.) is referred to as an "object". The default will be the IAU name of the object, followed by the shortest name alias.
- **property**: An overarching property of the object can be anything from "photometry", a "date", a "classification", a "host", etc. Can contain elements or subproperties.
- **subproperty**: any additional property within a property, useful when a large number of elements will share an additional keyword.
- **keyword**: the name of an item within a category, property, subproperty, or element in the `json` dictionary. These keywords are dependent on the property or subproperty that they belong to. Some commonly used examples are:
  1. value (§1.3.1): the value of the element (e.g. "17.2" for a magnitude).
  2. reference (§1.3.2): the origin of the element, such as a paper, broker, catalog, etc.
  3. units (§1.3.3): units of the value written in astropy format (e.g. `"km / s"`)
  4. type (§1.3.4): string, float, bool, etc.
  5. flag (§1.3.5): integer to reference known flags.
  6. computed (§1.3.6): a Boolean that specifies whether the value of the element was computed by the database (True), as opposed to an external reference (False). If not present it is assumed to be False.
  7. uui (§1.3.7): universally unique identifier (UUI) for the element to be referenced elsewhere.
  8. default (§1.3.8): a Boolean used for when there are multiple entries of the element.

1.2. *Best practices*

- Use `snake_case` for naming variables, a naming convention where each word is in lower case and is separated by underscores.
- Use prefixes when properties or elements should be categorized together.
- Refrain from using plurals or capital letters when naming variables.
- Refrain from using keywords already used in the schema as names of properties or elements.
- keywords formatted as strings will have empty strings for missing data.
- keywords formatted as floats will have `NaN` for missing data.
- keywords formatted as Boolean will have `null` for missing data.

1.3. *Common Keywords*

Here we provide a more detailed description and how to handle each of the optional keywords in an element, introduced in §1.1.

1.3.1. *value*

The most critical keyword in the database, which stores the value of any element, be that a parameter, measurement, or anything else. Can be a string, float, integer, or Boolean.

1.3.2. *reference*

The reference source for the value in the element. Whenever possible, the reference should be specified in the format for this is a 19-character ADS Bibcode (e.g. `2019ApJ...871..102N`). In the event that a Bibcode is not available, a string or common source can also be input here (e.g. "Smith et al.", "SOUSA", "WISeREP"). If `computed = True`, the value of this keyword can be set to the uui of the original element used to compute the value, or multiple values (e.g. magnitude and redshift). In the json file, these can be either a string or a list of strings (in case there are multiple references for one value). A complete list of acceptable references is below:

1. An ADS bibcode
2. A uui corresponding to another item in the `json` file.
3. "TNS"
4. "Pan-STARRS"
5. "GaiaAlerts"
6. "ATLAS"
7. "ZTF"
8. "ASAS-SN"
9. "WISeREP"
10. "SOUSA"

1.3.3. *units*

The units of the value. These should be formatted in astropy units format. For a description of this method see the Astropy documentation. For a full list of acceptable units see the list from Astropy. Examples might include things like `"km /s"`, `"AB"` (for AB magnitudes), or `"erg / ( s cm2 Hz)"`.

1.3.4. *type*

The python data type of the value, can be one of `str`, `float`, `int`, or `bool`.

1.3.5. *flag*

For situations that are too nuanced or rare to store in their own keywords, there is the option to store this information in the form of a flag. This can be an integer value for a flag associated with the value or element, or a comma separated list of flags. For a list of flags and their definitions see §1.4.

1.3.6. *computed*

A Boolean that defines whether the value is original from a reference (False), or if the value was computed using the database (True). The version of the database used to compute the values with `computed == True` is stored in the `version` element of the metadata.

1.3.7. *uui*

A universally unique identifier (UUI) used to cross-reference different elements with each other. These can be created with the `uuid` Python package and look like this: `c2ee78a2-acc0-4c01-b878-543130587e9a`.

1.3.8. *default*

A Boolean value used in the event there are multiple competing entries for an important property such as the name of a transient, distance, or coordinates. There can only be one element with `default == True` for each element in a property.

1.3.9. *comment*

An optional simple string with a comment pertaining to the individual element.

1.4. *Flags*

There might be some situations that are relatively common, but either too complex to store as a keyword in an element, too long, or not necessary. In that case, there is the option to specify a flag. Additional flags can be added without having to re-define the entire schema. Here we provide the full list of flags and their definition. Multiple flags should be comma separated.

0 : No flags associated with the element.

1 : Reference for this value exists but is not a published source (ie. just someone submitting a value without publishing it first)

2 : Reference of the value is not known.

3 : A raccoon reduced the data, so you shouldn't trust this.

1.5. *Required Keywords*

This is a list of the required keywords for each type of data. Any other keywords listed in this can be assumed to be optional.

1.5.1. *Objects*

At a minimum, each object is required to have the name, coordinate, and reference_alias keywords. We require a reference for everything to ensure a high quality of data.

1.5.2. *Properties*

The required values for all properties are:

- name: Default name of the object. This is always required.
- coordinate: equitorial is always required with a reference. galactic and ecliptic are optional but must contain a reference.
- distance: Nothing required, ideally redshift will be provided. If one is provided, a reference is required!
- epoch: Nothing required, will be derived from photometry if provided. If one is provided, a reference is required!
- classification: Nothing required, TNS will be queried. If provided, a reference is required!

1.5.3. *Measurements*

Each photometry point must have at least the following:

- reference
- raw
- filter
- raw_units

Additionally, if any corrections were applied the values *must* be supplied as well! Similarly, if the magnitude is only an upper limit please set `upperlimit=True`.

FILL IN OTHER MEASUREMENTS LATER

2. PROPERTIES

2.1. *name*

The name information of the object.

2.1.1. *default_name*

The default name of the object. If available, this will be the IAU name without any prefixed (e.g. "2019qiz"). If the object does not have an IAU name, the shortest alias will be adopted as the default name.

2.1.2. *alias*

All known names and aliases of the object are stored in this list, the default name will be selected from this list and stored in the name keyword.

```
"name": "2019qiz",
"alias": [{
          "value": "AT2019qiz",
          "reference": "TNS"
          },
          {
          "value": "Melisandre",
          "reference": "2021ApJ...908....4V"
          },
          {
          "value": "ZTF19abzrhgq",
          "reference": "ZTF"
          }]
```

2.2. *coordinate*

List that contains the coordinates of the object in the same format as the original reference. The default will be equatorial coordinates in degrees, this is the value that will be read by the pipeline, or computed if not available. There is the option to have multiple element entries, if for example there are different coordinates from different observatories. The default coordinates used for calculations will be identified with the keyword `default = True`. This is different (but can be derived from) the optional coordinates values stored in for example, individual photometry measurements.

The type of coordinate is stored in the coord_type element and can be equatorial, galactic, or ecliptic. Examples of how an object's coordinates and associated keywords can be stored are shown below. The simplest coordinate example with RA and DEC in degrees, and an associated reference:

```
{
  "ra": 32.12178,
  "dec": 45.21694,
  "epoch": "J2000",
  "frame": "ICRS",
  "coord_type": "equatorial",
  "ra_units":"deg",
  "dec_units":"deg",
  "reference": "2019ApJ...871..102N",
}
```

A more complex example with equatorial coordinates in hour angle, and computed galactic coordinates. In this case the set of coordinates with a reference is used as opposed to the one without a reference. Then the galactic coordinates are calculated based on the `default = True` value, and the `uui` associated is added.

```
"coordinate": [
    {
    "ra": "04:46:37.880",
    "dec": "-10:13:34.90",
    "ra_units": "hourangle",
    "dec_units": "deg",
    "coord_type": "equatorial",
    "reference": "TNS",
    "computed": false,
    "uui": "c2ee78a2-acc0-4c01-b878-543130587e9a",
    "default": true
```

```
},
{
"ra": "04:46:37.77867",
"dec": "-10:13:34.6800",
"ra_units": "hourangle",
"dec_units": "deg",
"coord_type": "equatorial",
"reference": "Fake",
"flag": "2,3"
},
{
"l": 207.876557,
"b": -32.322864,
"l_units": "deg",
"b_units": "deg",
"coord_type": "galactic",
"computed": true,
"reference": "c2ee78a2-acc0-4c01-b878-543130587e9a"
}]
```


The keywords that can be used within a coordinate element are:

- ra: right ascension (str, int, float)
- dec: declination (str, int, float)
- l: galactic longitude (str, int, float)
- b: galactic latitude (str, int, float)
- lon: longitude (str, int, float)
- lat: latitude (str, int, float)
- _units: suffix added to any of ra, dec, l, b, lon, lat to specify the units of the coordinate.
- _error: suffix added to any of ra, dec, l, b, lon, lat to specify the uncertaintiy in the coordinate.
- epoch: epoch (e.g. J2000, B1950)
- frame: coordinate frame (e.g., ICRS, FK5)
- default: Boolean. If multiple entries, use this value as the default (True)

Each element in the coordinate property can only have one pair of coordinates in the same frame, but multiple elements can be added.

2.3. *distance*

This property stores different values realted to the distance to an object and can be anything like a redshift, dispersion measure, luminosity distance, etc. These can be computed or measured, some examples listed below:


```
"distance": [
    {
        "value": 1.1 ,
        "reference": "2019ApJ...871..102N" ,
        "computed": False,
        "default": True,
        "uuid": "c2ee78a2-acc0-4c01-b878-543130587e9b",
        "distance_type": "redshift"
    },
    {
```

```
        "value": 0.9 ,
        "error": 0.1
        "reference": "2019ApJ...871..102N" ,
        "computed": False,
        "distance_type": "redshift"
    },
    {
        "value": 1 ,
        "unit": " pc " ,
        "cosmology": "Planck18",
        "reference": "c2ee78a2-acc0-4c01-b878-543130587e9b",
        "computed": True
        "distance_type": "luminosity"
    },
    {
        "value": 0.1,
        "reference": "2019ApJ...871..102N",
        "computed": False,
        "distance_type": "dispersion_measure"
    }]
```

Where the keys can be the following.

- value: The luminosity distance (float)
- unit: The units on the luminosity distance (str, astropy units)
- error: The error on the luminosity distance (float, optional)
- reference: The reference alias (integer)
- cosmology: Which cosmology was used to calculate the distance (str).
- computed: True if the value was computed, False otherwise (boolean)
- distance_type: The type of distance measure. Can be "redshift", "luminosity", "dispersion_measure", etc.

2.4. *date_reference*

These are mostly computed, even if it is something very simple like "first data point". But if it is an explosion time determined from a complex model in a paper, that is not computed and a reference can be added. If a value is computed, the reference should be a uuid pointing to another epoch that it was computed from. Example:

```
"date_reference": [
{
    "value": 56123.2 ,
    "date_format": "MJD",
    "date_type": "explosion",
    "reference":  "c2ee78a2-acc0-4c02-b878-543130587e9b",
    "computed": True
},
{
    "value": 56356.5,
    "date_format": "MJD"
    "reference": "c2ee78a2-acc0-4c02-b878-543130587e9b",
    "computed": True,
    "date_type": "peak"
},
{
    "value": "10/19/2023 14:36:43",
    "date_format": "MM/DD/YYYY HH:MM:SS",
    "reference": "2019ApJ...871..102N" ,
```

```
        "computed": False,
        "date_type": "discovery"
        "uuid": "c2ee78a2-acc0-4c02-b878-543130587e9b"
    },
    {
        "value": "123456",
        "date_format": "MJD",
        "reference": "c2ee78a2-acc0-4c02-b878-543130587e9b",
        "computed": True,
        "date_type": "discovery"
    }
    ]
```

Each element can have the following keys:

- value: The date of discovery
- date_format: The format of the date
- date_type: The type of date this is. Examples are "explosion", "peak", "discovery".
- reference: The alias for the reference
- computed: If it was computed (bool)

2.5. *classification*

The elements in the classification property have no names, they are simply entries in the classification property. In addition to the keywords available in §1.3, classification elements can have: object_class, confidence, and class_type keywords. An object can have multiple classifications.

Class type can be a bibcode for the ad-hoc paper (e.g. TDE H+He

- object_class: the common object classes (e.g. SN, TDE, SN Ia, FRB, etc.)
- confidence: value between 0 and 1 with the confidence or probability of the given object class.
- class_type: is the classification photometric, spectroscopic, or machine learning predicted?

```
"classification": [
  {
    "object_class": "SN Ia",
    "confidence": 1.0,
    "class_type": "spectroscopic",
    "reference": "2018MNRAS.476..261B",
    "default": True
  },
  {
    "object_class": "SN",
    "confidence": 0.2,
    "class_type": "spectroscopic",
    "reference": "TNS",
    "default": False
  },
  {
    "object_class": "SN Ia",
    "confidence": 0.98,
    "class_type": "predicted",
    "reference": "2017ApJ...476...61C",
    "default": False
  }
]
```

Measurements are any values that were obtained from a real telescope, be that photometry, spectroscopy, polarization measurements, neutrinos, X-ray data, etc. There is no separate property for astrometry, as it is assumed any astrometric measurement will have an associated flux, and can added to the photometry property.

2.6. *photometry*

The photometry property will store all UV, optical, and IR photometry. This is what will likely be the most extensive element of the catalog. For some objects photometric measurements will have many repeated keywords such as photometric system, telescope, or whether or not they have some correction applied.

Groups of related photometry can be stored in individual arrays like in the example shown below.

```
"photometry": [
  {
    "telescope": "ZTF",
    "mag_system": "AB",
    "reference": "2019ApJ...871..102N",
    "flux": [5.0, 5.0, 5.0, 5.0],
    "filter": ['r', 'r', 'r', 'r']
  },
  {
    "telescope": "ASAS-SN",
    "mag_system": "Vega",
    "reference": "2018MNRAS.476..261B",
    "raw": [5.0, 5.0, 5.0],
    "raw_err": [1.0, 1.0, 1.0]
  },
  {
    "filter": "Clear",
    "raw": [1]
  }
]
```

The full list of possible keywords that can be associated with a photometric measurement are the same for the phot_N subproperty or the magnitude element, they are all listed in the following section (§**??**).

A measured magnitude is the most common element in the database, these can have a number of associated keywords. Each magnitude element can only have one flux-associated value (i.e. a magnitude element cannot have an AB magnitude and a flux). In addition to the default keywords listed for every element in §1.3, the keywords that can be associated with a magnitude element are:

- raw : raw flux value, can be magnitude (for optical, infrared, or UV), energy (for X-ray), or flux density (for radio). This should be the closest available value to that measured by the telescope, without corrections, usually as published in a paper.
- raw_err : error of raw flux value.
- raw_units: astropy units of the raw flux value.
- value : flux with corrections applied, can be a magnitude, energy, counts or flux density. If counts provided, must provide telescope_area.
- value_err: error on the corrected flux value.
- value_units: units of the corrected flux value, can be astropy units or magnitude system.
- epoch_zeropoint : if date is epoch, what is the offset.
- epoch_redshift : if date is epoch, what is the redshift.
- filter : name of telescope filter.
- filter_key : Name of filter in the format described in the metadata (§3.2)

- obs_type : The type of observation made. Can be ("UVOIR", "xray", or "radio")
- telescope_area: collecting area of the telescope. This must be provided if flux _units is "counts"
- date : time of measurement.
- date_format : format on the of measurement (e.g. MJD, JD, etc).
- date_err : uncertainty in the date value.
- ignore : was the data ignored.
- upperlimit : Boolean, is the mag an upper limit?
- sigma : significance of upper limit.
- sky : sky brightness in the same units as mag.
- telescope : telescope that took the data.
- instrument : instrument on telescope that took the data.
- phot_type : is the photometry PSF, Aperture, or synthetic.
- exptime : Exposure time.
- aperture : If aperture photometry, aperture diameter in arcseconds.
- observer : Person or group that observed the data.
- reducer : Person who reduced the data.
- pipeline : Pipeline used to reduce the data.
- corr_k : Boolean. Is the raw value k-corrected? Can be None which means that we are uncertain if it is k-corrected.
- corr_s : Boolean. Is the raw value s-corrected? Can be None which means that we are uncertain if it is s-corrected.
- corr_av : Boolean. Is the raw value Milky Way extinction corrected? Can be None which means that we are uncertain if it is av-corrected.
- corr_host : Boolean. Is the raw value host subtracted? Can be None which means that we are uncertain if it is host subtracted.
- corr_hostav : Boolean. Is the raw value corrected for intrinsic host extinction? Can be None which means that we are uncertain if it is host extinction corrected.
- val_k : Float. Value of the k-correction applied to mag.
- val_s : Float. Value of the s-corrected applied to mag.
- val_av : Float. Value of the Milky Way extinction correction applied to mag.
- val_host : Float. Value of the host contribution applied to mag.
- val_hostav : Float. Value of the intrinsic host extinction applied to mag.

Save z_phase and t0_phase for a phase time

Clearly most magnitude measurements will not have all of these parameters available, but the user has the option to include up to all of these, in addition to the filter-specific keywords listed in §3.2. Although it is recommended to store the filter-specific keywords in the metadata to avoid repetition. The `mag` and `raw` keywords do not necessarily have to contain magnitudes, they can contain flux values. This is acceptable as long as it is specified in the units field.

In the case of the magnitude element, the computed keyword does not apply. In this case `raw` is always assumed to be not computed, and `mag` is always assumed to be computed, where the computation can be as simple as applying a total correction of 0 mags.

2.7. *host*

Information about the host galaxy where the object is located.

- host_ra: The Right Ascension of the host galaxy
- host_dec: the Declination of the host galaxy
- host_z: Redshift of the host galaxy
- host_type: The classification of the host (ex. spiral, elliptical, dwarf, AGN, etc.)
- host_name: The name of the host galaxy
- reference: The reference for this host information

3. METADATA

Metadata is information that does not necessarily fit into any of the existing categories, but is instead used to support the database structure. Some examples of metadata are listed in this section.

3.1. *reference_details*

We plan to store the 19 digit ADS bibcode for every reference for ease of analysis. This will store a mapping between those bibcodes and human readable names for easy visualization.

```
"reference_alias": [
  {
    "name": "2019ApJ...871..102N",
    "human_readable_name": "somename et al. (2019)"
  },
  {
    "name": "2020ApJ...874...22N",
    "human_readable_name": "aname et al. (2020)"
  },
]
```

The two keywords used in this element are name and human_readable_name, which store the name of the reference and the corresponding alias integer, respectively.

3.2. *filter_alias*

The basic information about a filter can be stored in the metadata of the `json` file for quick reference, without needing to add it to each individual photometry measurement.

```
"filter_alias": [
  {
    "filter_key": "PAN-STARRS_PS1.r",
    "wave_eff": 6155.47,
    "wave_min": 5391.11,
    "wave_max": 7038.08,
    "zp": 3173.02,
    "wave_units": "AA",
    "zp_units": "Jy",
    "zp_system": "Vega",
  }
]
```

Individual measurements can just reference the `filter_key` instead of storing this repeated information in each element. The keywords supported in the `filter_reference` element are:

- filter_key: keyword shared between observations and this reference. The format is adopted from the SVO, but replacing the slash `/` with an underscore `_` character.

- wave_eff: effective wavelength of filter.
- wave_min: minimum wavelength of filter.
- wave_max: maximum wavelength of filter.
- freq_eff: effective frequency of the filters (for radio!).
- freq_min: minimum frequency of the filters (for radio!).
- freq_max: maximum frequency of the filters (for radio!).
- zp: filter zeropoint.
- wave_units: units of wavelengths.
- freq_units: units of frequency.
- zp_units: units of zeropoint value.
- zp_system: system of zeropoint value (e.g. "Vega").

3.3. *schema_version*

Version of the schema used.

```
"schema_version": {
  "value": 1.0,
  "comment": "cite us please"
}
```

4. TEMPLATE JSON FILE

A sample JSON file that could be used to fill your data into is available publicly at https://drive.google.com/file/d/10-llOOJGGgtCurJn8lemKdNpD07b4E4w/view?usp=sharing

5. POSSIBLY CONTROVERSIAL DECISIONS / FAQ

1. We use a document database to store individual object json files as separate documents instead of the more popular Relational Schema. The document schema just fits our data much easier and should not result in any slowdowns.
2. We decided to use full ads bibcodes for references throughout a single JSON file. This is just easier for us (and hopefully for everyone)! Note: this is different from the original OSC which used aliases.
3. We will store the original data format with a `computed = False` flag, always! But, we will store some other basic computations for quick queries with `computed = True` and the reference as the uuid. For example, we may get a date in MM/DD/YYYY format but convert it to MJD and then store both.