VOEvent-parse Documentation

Release 0.8.0

Tim Staley

May 07, 2015

Version 0.8.0

Welcome to voevent-parse's documentation. If you're new here, I recommend you start with the *introduction* and then take a look at the *usage examples*.

CHAPTER 1

Contents

1.1 Introduction

1.1.1 What is voevent-parse?

A lightweight library for parsing, manipulating, and generating VOEvent XML packets, built atop lxml.objectify.

voevent-parse provides convenience routines to take care of many common tasks, so that accessing those vital data elements is as simple as:

```
import voeventparse
with open(xml_filename) as f:
    v = voeventparse.load(f)
print "AuthorIVORN:", v.Who.AuthorIVORN  #Prints e.g. ivo://nasa.gsfc.tan/gcn
v.Who.AuthorIVORN = 'ivo://i.heart.python/lxml' #Alters the XML value.
```

Note: As of version 0.8, voevent-parse is Python 3 compatible (tested with Python versions 2.7 & 3.4).

1.1.2 Rationale

voevent-parse aims to make dealing with VOEvent packets easy, while remaining small, flexible, and stable enough to be suitable for use as a dependency in a range of larger projects. To achieve this, we add a user-friendly layer on top of lxml.objectify which attempts to hide the messy details of working with the sometimes lengthy VOEvent schema, and also take care of some rather obscure lxml namespace handling. However, since the objects created are just regular lxml classes, the user is free to utilise the full power of the lxml library when required.

1.1.3 Installation

voevent-parse is pip installable, try running:

```
pip install voevent-parse
```

Note that voevent-parse depends upon lxml, and pip will attempt to install lxml first if not already present. lxml may be installed as a system package if the version distributed with your package manager is sufficiently up-to-date (version >= 2.3). If you're working with pip / virtualenv and not making use of system packages, then note that lxml has some prerequisites for compilation that can cause a standard `pip install` to fail with somewhat cryptic errors. On Ubuntu you can satisfy those requirements using:

```
sudo apt-get install libxml2-dev libxslt-dev
```

1.1.4 Documentation

Reference documentation can be found at http://voevent-parse.readthedocs.org, or generated directly from the repository using Sphinx.

1.1.5 Source, Issues, Development etc.

I intend to mark any updates by bumping the version number accordingly. That said, if you find yourself using voevent-parse in any serious context, do drop me an email so I can keep you informed of any updates or critical bugs.

Bug reports (or even better, pull requests) are welcomed. The source code and issue tracker may be found at https://github.com/timstaley/voevent-parse.

voevent-parse also has a suite of unit-tests which may be run in the usual manner, typically using nose from the repository root directory.

1.1.6 lxml.objectify 'gotchas'

The objectify library has a few syntactic quirks which can trip up new users. Firstly, you should be aware that the line `root.foo` actually returns an object that acts like a *list* of all the children of the `root` element with the name *foo*. What's confusing is that objectify has syntactic sugar applied so that `root.foo` is a shortcut alias for the more explicit `root.foo[0]`. This can be very confusing to the uninitiated, since it overrides some attributes of the the actual element values. To get around this, you should be aware of the accessor to the text representation of the value; `.text`, e.g.:

```
import lxml.objectify
root = lxml.objectify.Element('root')
root.foo = 'sometext' # Adds a child called 'foo' with value 'sometext'
print root.foo # 'sometext'
print len(root.foo) #  1. Wait, what?
# The string value clearly does not have length==1,
# the list of children called 'foo' does.
print root.foo.text # 'sometext'
print len(root.foo.text) # 8. Sanity prevails!
```

Another 'gotcha' is that *creating* multiple child elements of the same name is a bit unintuitive. Essentially, `objectify` works implicitly if each element has only one child:

```
from lxml import objectify, etree
simple_root = objectify.Element('simple_root')
simple_root.layer1 = None
simple_root.layer1.layer2 = 5
print etree.tostring(simple_root, pretty_print=True)
```

But if there are multiple children then each child must be explicitly declared as an lxml `Element` in order to co-exist with its siblings:

```
from lxml import objectify, etree
import math
siblings_root = objectify.Element('siblings')
siblings_root.bars = None
siblings_root.bars.append(objectify.Element('bar'))
siblings_root.bars.append(objectify.Element('bar'))
siblings_root.bars.bar[0] = math.pi
siblings_root.bars.bar[1] = 42
print etree.tostring(siblings_root, pretty_print=True)
```

... which is another reason to use voevent-parse as a user-friendly interface for common operations.

For some more examples, you might also try: http://www.saltycrane.com/blog/2011/07/example-parsing-xml-lxml-objectify/.

1.1.7 See also

Alternative parsing libraries

voevent-parse was preceded by VOEventLib, which has similar aims but a different stylistic approach (see http://lib.skyalert.org/VOEventLib/VOEventLib/doc/index.html).

Brokers

In order to receive VOEvent packets, you will require a utility capable of connecting to the VOEvent backbone. Two such tools are Comet and Dakota.

Associated utility routines

Depending on what you want to use your VOEvents for, you may be interested in pysovo, a collection of routines for responding to VOEvents, and/or fourpiskytools, which provides basic routines for submitting VOEvents to a broker for publication.

1.1.8 Acknowledgement

If you make use of voevent-parse in work leading to a publication, we ask that you cite the ASCL entry.

1.2 Usage examples

1.2.1 Basic data access and manipulation

You can also `download` this example or view it on Github.

```
#!/usr/bin/python
"""A quick usage example.

Once voeventparse is installed, this should tell you most of what you need to know
in order to start doing things with VOEvent packets.

The attributes are built from the structure of the XML file,
so the best way to understand where the variable names come from is to simply
open the XML packet in your favourite web browser and dig around.

See also:
* lxml documentation at http://lxml.de/objectify.html
* VOEvent standard at http://www.ivoa.net/documents/VOEvent/
* VOEvent schema file at http://www.ivoa.net/xml/VOEvent/VOEvent-v2.0.xsd
"""
from __future__ import print_function
import copy
import voeventparse
from voeventparse.tests.resources.datapaths import swift_bat_grb_pos_v2

with open(swift_bat_grb_pos_v2, 'rb') as f:
    v = voeventparse.load(f)

#Basic attribute access
print("Ivorn:", v.attrib['ivorn'])
print("Role:", v.attrib['role'])
print( "AuthorIVORN:", v.Who.AuthorIVORN)
print( "Short name:", v.Who.Author.shortName)
print( "Contact:", v.Who.Author.contactEmail)

#Copying by value, and validation:
print( "Original valid as v2.0? ", voeventparse.valid_as_v2_0(v))
v_copy = copy.copy(v)
print( "Copy valid? ", voeventparse.valid_as_v2_0(v_copy))

#Changing values:
v_copy.Who.Author.shortName = 'BillyBob'
v_copy.attrib['role'] = voeventparse.definitions.roles.test
print( "Changes valid? ", voeventparse.valid_as_v2_0(v_copy))

v_copy.attrib['role'] = 'flying circus'
print( "How about now? ", voeventparse.valid_as_v2_0(v_copy))
print( "But the original is ok, because we copied? ", voeventparse.valid_as_v2_0(v))

v.Who.BadPath = "This new attribute certainly won't conform with the schema."
assert voeventparse.valid_as_v2_0(v) == False
del v.Who.BadPath
assert voeventparse.valid_as_v2_0(v) == True
#######################################################
# And now, SCIENCE
#######################################################
c = voeventparse.pull_astro_coords(v)
print( "Coords:", c)
```

1.2.2 Author a new VOEvent packet

You can also download this example or view it on Github.

```
#!/usr/bin/python
from __future__ import print_function
import datetime
import os
import voeventparse as vp
from lxml import etree

# Set the basic packet ID and Author details

v = vp.Voevent(stream='astronomy.physics.science.org/super_exciting_events',
               stream_id=123, role=vp.definitions.roles.test)
```

```
vp.set_who(v, date=datetime.datetime.utcnow(),
           author_ivorn="voevent.4pisky.org")

vp.set_author(v, title="4PiSky Testing Node",
              shortName="Tim"
)


# Now create some Parameters for entry in the 'What' section.

# Strictly speaking, parameter values should be strings,
# with a manually specified dataType; one of
# `string` (default), `int` , or `float`.
# e.g.
int_flux = vp.Param(name='int_flux',
                    value="2.0e-3",
                    unit='Janskys',
                    ucd='em.radio.100-200MHz',
                    dataType='float',
                    ac=False)
int_flux.Description = 'Integrated Flux'

# But with ac=True (autoconvert) we switch on some magic to take care
# of this for us automatically.
# See ``Param`` docstring for details.
p_flux = vp.Param(name='peak_flux',
                  value=1.5e-3,
                  unit='Janskys',
                  ucd='em.radio.100-200MHz',
                  ac=True
)
p_flux.Description = 'Peak Flux'

v.What.append(vp.Group(params=[p_flux, int_flux], name='source_flux'))

#Note ac=True (autoconvert) is the default setting if dataType=None (the default)
amb_temp = vp.Param(name="amb_temp",
                    value=15.5,
                    unit='degrees',
                    ucd='phys.temperature')

amb_temp.Description = "Ambient temperature at telescope"
v.What.append(amb_temp)


# Now we set the sky location of our event:
vp.add_where_when(v,
                  coords=vp.Position2D(ra=123.5, dec=45, err=0.1,
                                       units='deg',
                                       system=vp.definitions.sky_coord_system.fk5),
                  obs_time=datetime.datetime(2013, 1, 31, 12, 5, 30),
                  observatory_location=vp.definitions.observatory_location.geosurface)

# Prettyprint some sections for desk-checking:
print( "\n***Here is your WhereWhen:***\n")
print( vp.prettystr(v.WhereWhen))

print( "\n***And your What:***\n")
```

```python
print( vp.prettystr(v.What))

# You would normally describe or reference your telescope / instrument here:
vp.add_how(v, descriptions='Discovered via 4PiSky',
           references=vp.Reference('http://4pisky.org'))

# The 'Why' section is optional, allows for speculation on probable
# astrophysical cause
vp.add_why(v, importance=0.5,
           inferences=vp.Inference(probability=0.1,
                                   relation='identified',
                                   name='GRB121212A',
                                   concept='process.variation.burst;em.radio')
)

# We can also cite earlier VOEvents:
vp.add_citations(v,
                 vp.Citation(
                     ivorn='ivo://astronomy.physics.science.org/super_exciting_events#101',
                     cite_type=vp.definitions.cite_types.followup))

# Check everything is schema compliant:
vp.assert_valid_as_v2_0(v)

output_filename = 'new_voevent_example.xml'
with open(output_filename, 'wb') as f:
    vp.dump(v, f)

print( "Wrote your voevent to ", os.path.abspath(output_filename))
```

1.3 voevent-parse API reference

Warning: Much of the content within assumes the reader has at least a summary understanding of the VOEvent specifications.

Note: The top-level `__init__.py` file imports key classes and subroutines into the top-level `voeventparse` namespace, for brevity.

1.3.1 `voeventparse.voevent` - Basic VOEvent packet manipulation

Routines for handling etrees representing VOEvent packets.

`voeventparse.voevent.`**`Voevent`**(*stream*, *stream_id*, *role*)

Create a new VOEvent element tree, with specified IVORN and role.

Parameters

- **stream** (*string*) – used to construct the IVORN like so:

```python
ivorn = 'ivo://' + stream + '#' + stream_id
```

(N.B. `stream_id` is converted to string if required.) So, e.g. we might set:

```
stream='voevent.soton.ac.uk/super_exciting_events'
stream_id=77
```

- **stream_id** (*string*) – See above.
- **role** (*string*) – role as defined in VOEvent spec. (See also `definitions.roles`)

Returns Root-node of the VOEvent, as represented by an lxml.objectify element tree ('etree'). See also http://lxml.de/objectify.html#the-lxml-objectify-api

`voeventparse.voevent.`**`loads`**`(`*s*`)`

Load VOEvent from bytes.

This parses a VOEvent XML packet string, taking care of some subtleties. For Python 3 users, `s` should be a bytes object - see also http://lxml.de/FAQ.html, "Why can't lxml parse my XML from unicode strings?" (Python 2 users can stick with old-school `str` type if preferred)

Currently pretty basic, but we can imagine using this function to homogenise or at least identify different VOEvent versions, etc.

Parameters **s** (*bytes*) – Bytes containing raw XML.

Returns *Voevent* etree.

Return type Root-node of the :class

`voeventparse.voevent.`**`load`**`(`*file*`)`

Load VOEvent from file object.

See also: `loads()` :param file: An open file object (binary mode preferred), see also :type file: file :param http://lxml.de/FAQ.html: :param "Can lxml parse from file objects opened in unicode/text mode?":

Returns *Voevent* etree.

Return type Root-node of the :class

`voeventparse.voevent.`**`dumps`**`(`*voevent*, *pretty_print=False*, *xml_declaration=True*, *encoding=u'UTF-8'*`)`

Converts voevent to string.

Note: Default encoding is UTF-8, in line with VOE2.0 schema. Declaring the encoding can cause diffs with the original loaded VOEvent, but I think it's probably the right thing to do (and lxml doesn't really give you a choice anyway).

Parameters

- **voevent** (`Voevent`) – Root node of the VOevent etree.
- **pretty_print** (*bool*) – indent the output for improved human-legibility when possible. See also: http://lxml.de/FAQ.html#why-doesn-t-the-pretty-print-option-reformat-my-xml-output
- **xml_declaration** (*bool*) – Prepends a doctype tag to the string output, i.e. something like `<?xml version='1.0' encoding='UTF-8'?>`

Returns Bytes containing raw XML representation of VOEvent.

`voeventparse.voevent.`**`dump`**`(`*voevent*, *file*, *pretty_print=True*, *xml_declaration=True*`)`

Writes the voevent to the file object.

e.g.:

```
with open('/tmp/myvoevent.xml','wb') as f:
    voeventparse.dump(v, f)
```

Parameters

- **voevent** (`Voevent`) – Root node of the VOevent etree.
- **file** (*file*) – An open (binary mode) file object for writing.
- **pretty_print** (*bool*) –
- **pretty_print** – See `dumps()`
- **xml_declaration** (*bool*) – See `dumps()`

`voeventparse.voevent.`**`valid_as_v2_0`**`(`*voevent*`)`

Tests if a voevent conforms to the schema.

Parameters **voevent** (`Voevent`) – Root node of a VOEvent etree.

Returns: Bool (VOEvent is valid?)

`voeventparse.voevent.`**`assert_valid_as_v2_0`**`(`*voevent*`)`

Raises `lxml.etree.DocumentInvalid` if voevent is invalid.

Especially useful for debugging, since the stack trace contains a reason for the invalidation.

Parameters **voevent** (`Voevent`) – Root node of a VOEvent etree.

Returns: None. NB raises `lxml.etree.DocumentInvalid` if VOEvent does not conform to schema.

`voeventparse.voevent.`**`set_who`**`(`*voevent*, *date=None*, *author_ivorn=None*`)`

Sets the minimal 'Who' attributes: date of authoring, AuthorIVORN.

Parameters

- **voevent** (`Voevent`) – Root node of a VOEvent etree.
- **date** (*datetime.datetime*) – Date of authoring. NB Microseconds are ignored, as per the VOEvent spec.
- **author_ivorn** (*string*) – Short author identifier, e.g. `voevent.4pisky.org/ALARRM`. Note that the prefix `ivo://` will be prepended internally.

`voeventparse.voevent.`**`set_author`**`(`*voevent*, *title=None*, *shortName=None*, *logoURL=None*, *contactName=None*, *contactEmail=None*, *contactPhone=None*, *contributor=None*`)`

For setting fields in the detailed author description.

This can optionally be neglected if a well defined AuthorIVORN is supplied.

Note: Unusually for this library, the args here use CamelCase naming convention, since there's a direct mapping to the `Author.*` attributes to which they will be assigned.

Parameters **voevent** (`Voevent`) – Root node of a VOEvent etree. The rest of the arguments are strings corresponding to child elements.

`voeventparse.voevent.`**`add_where_when`**`(`*voevent*, *coords*, *obs_time*, *observatory_location*`)`

Add details of an observation to the WhereWhen section.

Parameters

- **voevent** (`Voevent`) – Root node of a VOEvent etree.
- **coords** (`Position2D`) – Sky co-ordinates of event.
- **obs_time** (*datetime.datetime*) – Nominal DateTime of the observation.
- **observatory_location** (*string*) – Telescope locale, e.g. 'La Palma'. May be a generic location as listed under `voeventparse.definitions.observatory_location`.

`voeventparse.voevent.`**`add_how`**`(`*voevent*, *descriptions=None*, *references=None*`)`

Add descriptions or references to the How section.

Parameters

- **voevent** (`Voevent`) – Root node of a VOEvent etree.
- **descriptions** (*string*) – Description string, or list of description strings.
- **references** (`voeventparse.misc.Reference`) – A reference element (or list thereof).

`voeventparse.voevent.`**`add_why`**`(`*voevent*, *importance=None*, *expires=None*, *inferences=None*`)`

Add Inferences, or set importance / expires attributes of the Why section.

Note: `importance` / `expires` are 'Why' attributes, therefore setting them will overwrite previous values. `inferences`, on the other hand, are appended to the list.

Parameters

- **voevent** (`Voevent`) – Root node of a VOEvent etree.
- **importance** (*float*) – Value from 0.0 to 1.0
- **expires** (*datetime.datetime*) – Expiration date given inferred reason (See voevent spec).
- **inferences** (`voeventparse.misc.Inference`) – Inference or list of inferences, denoting probable identifications or associations, etc.

`voeventparse.voevent.`**`add_citations`**`(`*voevent*, *citations*`)`

Add citations to other voevents.

The schema mandates that the 'Citations' section must either be entirely absent, or non-empty - hence we require this wrapper function for its creation prior to listing the first citation.

Parameters

- **voevent** (`Voevent`) – Root node of a VOEvent etree.
- **citation** (`voeventparse.misc.Citation`) – Citation or list of citations.

1.3.2 `voeventparse.misc` - Subtree-elements and other helpers

Routines for creating sub-elements of the VOEvent tree, and a few other helper classes.

class `voeventparse.misc.`**`Position2D`**

A namedtuple for simple representation of a 2D position as described by the VOEvent spec.

Parameters

- **ra** (*float*) – Right ascension.
- **dec** (*float*) – Declination

- **err** (*float*) – Error radius.
- **units** (`definitions.coord_units`) – Coordinate units e.g. degrees, radians.
- **system** (`definitions.sky_coord_system`) – Co-ordinate system e.g. UTC-FK5-GEO

`voeventparse.misc.`**`Param`**(*name*, *value=None*, *unit=None*, *ucd=None*, *dataType=None*, *utype=None*, *ac=True*)

'Parameter', used as a general purpose key-value entry in the 'What' section.

May be assembled into a `Group`.

NB `name` is not mandated by schema, but *is* mandated in full spec.

Parameters

- **value** (*string*) – String representing parameter value. Or, if `ac` is true, then 'autoconversion' is attempted, in which case `value` can also be an instance of one of the following:
  - `bool`
  - `int`
  - `float`
  - `datetime.datetime`

  This allows you to create Params without littering your code with string casts, or worrying if the passed value is a float or a string, etc. NB the value is always *stored* as a string representation, as per VO spec.
- **unit** (*string*) – e.g. 'deg' for degrees.
- **ucd** (*string*) – unified content descriptor. For a list of valid UCDs, see: http://vocabularies.referata.com/wiki/Category:IVOA_UCD.
- **dataType** (*string*) – Denotes type of `value`; restricted to 3 options: `string` (default), `int` , or `float`. (NB *not* to be confused with standard XML Datatypes, which have many more possible values.)
- **utype** (*string*) – See http://wiki.ivoa.net/twiki/bin/view/IVOA/Utypes
- **ac** (*bool*) – Attempt automatic conversion of passed `value` to string, and set `dataType` accordingly (only attempted if `dataType` is the default, i.e. `None`). (NB only supports types listed in _datatypes_autoconversion dict)

`voeventparse.misc.`**`Group`**(*params*, *name=None*, *type=None*)

Groups together Params for adding under the 'What' section.

Parameters

- **params** (list of `Param()`) – Parameter elements to go in this group.
- **name** (*string*) – Group name. NB `None` is valid, since the group may be best identified by its type.
- **type** (*string*) – Type of group, e.g. 'complex' (for real and imaginary).

`voeventparse.misc.`**`Reference`**(*uri*, *meaning=None*)

Represents external information, typically original obs data and metadata.

Parameters

- **uri** (*string*) – Uniform resource identifier for external data, e.g. FITS file.

- **meaning** (*string*) – The nature of the document referenced, e.g. what instrument and filter was used to create the data?

`voeventparse.misc.`**`Inference`**(*probability=None*, *relation=None*, *name=None*, *concept=None*)
Represents a probable cause / relation between this event and some prior.

Parameters

- **probability** (*float*) – Value 0.0 to 1.0.
- **relation** (*string*) – e.g. 'associated' or 'identified' (see Voevent spec)
- **name** (*string*) – e.g. name of identified progenitor.
- **concept** (*string*) – One of a 'formal UCD-like vocabulary of astronomical concepts', e.g. http://ivoat.ivoa.net/stars.supernova.Ia - see VOEvent spec.

`voeventparse.misc.`**`Citation`**(*ivorn*, *cite_type*)
Used to cite earlier VOEvents.

Parameters

- **ivorn** (*string*) – It is assumed this will be copied verbatim from elsewhere, and so these should have any prefix (e.g. 'ivo://','http://') already in place - the function will not alter the value.
- **cite_type** (`definitions.cite_types`) – String conforming to one of the standard citation types.

1.3.3 `voeventparse.convenience` - Convenience routines

Convenience routines for common actions on VOEvent objects

`voeventparse.convenience.`**`pull_astro_coords`**(*voevent*, *index=0*)
Extracts the *AstroCoords* from a given *WhereWhen.ObsDataLocation*.

Note that a packet may include multiple 'ObsDataLocation' entries under the 'WhereWhen' section, for example giving locations of an object moving over time. Most packets will have only one, however, so the default is to just return co-ords extracted from the first.

Parameters

- **voevent** (`voeventparse.voevent.Voevent`) – Root node of the VOEvent etree.
- **index** (*int*) – Index of the ObsDataLocation to extract AstroCoords from.

Returns The sky position defined in the ObsDataLocation.

Return type `Position2D`

`voeventparse.convenience.`**`pull_isotime`**(*voevent*, *index=0*)
Extracts the event time from a given *WhereWhen.ObsDataLocation*.

Accesses a *WhereWhere.ObsDataLocation.ObservationLocation* element and returns the AstroCoords.Time.TimeInstant.ISOTime element, converted to a datetime.

Note that a packet may include multiple 'ObsDataLocation' entries under the 'WhereWhen' section, for example giving locations of an object moving over time. Most packets will have only one, however, so the default is to access the first.

Parameters

- **voevent** (`voeventparse.voevent.Voevent`) – Root node of the VOevent etree.
- **index** (*int*) – Index of the ObsDataLocation to extract an ISOtime from.

Returns Specifically, we return a standard library datetime object, i.e. one that is **timezone-naive** (that is, agnostic about its timezone, see python docs). This avoids an added dependency on pytz.

Return type `datetime.datetime`

The details of the reference system for time and space are provided in the AstroCoords object, but typically time reference is UTC.

`voeventparse.convenience.`**`pull_params`**`(`*voevent*`)`

Attempts to load the *What* section of a voevent as a nested dictionary.

Parameters **voevent** (`voeventparse.voevent.Voevent`) – Root node of the VOevent etree.

Returns

Mapping of `Group->Param->Attribs`. Access like so:

```
foo_param_val = what_dict['GroupName']['ParamName']['value']
```

Note: Parameters without a group are indexed under the key 'None' - otherwise, we might get name-clashes between *params* and *groups* (unlikely but possible) so for ungrouped Params you'll need something like:

```
what_dict[None]['ParamName']['value']
```

Return type Nested dict

`voeventparse.convenience.`**`prettystr`**`(`*subtree*`)`

Print an element tree with nice indentation.

Prettyprinting a whole VOEvent often doesn't seem to work, probably for issues relating to whitespace cf. http://lxml.de/FAQ.html#why-doesn-t-the-pretty-print-option-reformat-my-xml-output This function is a quick workaround for prettyprinting a subsection of a VOEvent, for easier desk-checking.

Parameters **subtree** (*lxml.etree*) – A node in the VOEvent element tree.

Returns Prettyprinted string representation of the raw XML.

Return type string

1.3.4 `voeventparse.definitions` - Standard or common string values

This module simply serves to store the XML schema, a 'skeleton' VOEvent xml document for creation of new instances, and various other minor definitions.

These values may be used in place of literal strings, to allow autocompletion and document the fact that they are 'standardized' values.

class `voeventparse.definitions.`**`roles`**

`observation` **= u'observation'**

`prediction` **= u'prediction'**

`utility` **= u'utility'**

`test` **= u'test'**

class `voeventparse.definitions.`**`sky_coord_system`**

Common coordinate system identifiers. See also `Position2D`.

`fk5` **= u'UTC-FK5-GEO'**

class `voeventparse.definitions.`**`observatory_location`**

Common generic values for the WhereWhen.ObservatoryLocation attribute.

`geosurface` **= u'GEOSURFACE'**

`geolunar` **= u'GEOLUN'**

class `voeventparse.definitions.`**`coord_units`**

Handy tags listing the unit names used by voeventparse.

`degrees` **= u'degrees'**

class `voeventparse.definitions.`**`cite_types`**

Possible types of `Citation()`

`followup` **= u'followup'**

`supersedes` **= u'supersedes'**

`retraction` **= u'retraction'**

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

Index