# CONTENTS

1 README 3  
1.1 Purpose 3  
1.2 Dependencies 3  
1.3 Examples 3  
1.4 TPM Citation 5  
1.5 See Also 5  

2 `coords.position.Position` 7  
2.1 Utility Functions 9  

3 `coords.position.Coord` 11  
3.1 Subclasses 11  

4 `coords.angsep` 13  

5 `coords.astrodate` 15  
5.1 Global Variables 17  

6 `coords.pytpm_wrapper` 19  
6.1 Global Variables 19  

7 Indices and tables 23  

Python Module Index 25  
Python Module Index 27  
Index 29
Contents:
1.1 Purpose

This package aims to provide much of the IDL “astron” functionality that pertains to coordinate manipulations in an OO framework. Our target user is a typical astronomer who needs to analyze data, work with catalogs, prepare observing proposals, and prepare for observing runs.

The initial version will provide simple functionality for working with positions in the same reference frame, without having to worry about units.

1.2 Dependencies

numpy

pytpm – a Python wrapper for the TPM library graciously contributed by Jeff Percival

1.3 Examples

```python
>>> import astrolib.coords as C
>>> print C.__version__
0.39

Unit conversions

```python
>>> ob = C.Position('12:34:45.34 -23:42:32.60')
>>> ob.hmsdms()
'12:34:45.340 -23:42:32.600'
>>> ob.dd()  
(188.68891666666667, -23.709055555555555)
>>> ob.rad() 
(3.2932428578545374, -0.41380108198269777)
``` 

Angular separations

```python
>>> p1 = C.Position("01:23:45.30 +65:43:31.24")
>>> p2 = C.Position("01:23:45.62 +65:43:31.20")
>>> p1.angsep(p2)
0.000548 degrees
>>> delta = p1.angsep(p2)
>>> delta.arcsec()
```
>>> p1.within(p2, 3.0, units='arcsec')
True
>>> epsilon = C.AngSep(5.0)
>>> epsilon
5.000000 arcsec
>>> delta > epsilon
False

Astronomical Date specifications

>>> d = C.AstroDate('1997.3')  # Defaults to Julian year; J or B prefix also ok
>>> d.year
1997.3
>>> d.jd
2450558.8250000002
>>> d.mjd
50558.325000000186
>>> d2 = C.AstroDate('MJD50658.25')  # JD also ok for plain Julian Date
>>> d2.year
1997.5735797399041
>>> d2 < d
False

Coordinate conversions

>>> ob.j2000()
(188.68891666666667, -23.709055555555555)
>>> ob.b1950()
(188.03056480942405, -23.433637283819877)
>>> ob.galactic()
(298.01638938748795, 39.003358150874568)
>>> ob.ecliptic(timetag=C.AstroDate('J2000'))
(197.5848634558852, -18.293964120804738)
>>> p3.j2000()
(20.9375, -65.72261111111107)
>>> p4.j2000()
(21.356870704681981, -65.462921080444147)
>>> p3.angsep(p4)
0.312199 degrees
>>> p5 = C.Position((0.0,0.0),system='galactic')
>>> p5.j2000()
(266.40499571858879, -28.936169261309555)

Specify position in hmsdms

>>> polaris = C.Position("02:31:49.08 +89:15:50.8")
>>> polaris.dd()
(37.954500000000003, 89.26411111111106)
>>> polaris.hmsdms()
'02:31:49.080 +89:15:50.800'
>>> print polaris.details()
System: celestial
Equinox: j2000

Specify position in decimal degrees
>>> ob = C.Position((52.9860209, -27.7510006))
>>> ob.hmsdms()
'03:31:56.645 -27:45:03.602'
>>> ob.dd()
(52.9860209, -27.751000600000001)

Use as calculator without saving the intermediate object

>>> C.Position("12:34:45.4 -22:21:45.4").dd()
(188.68916666666667, -22.36261111111111)

1.4 TPM Citation

Investigators using this software for their research are requested to explicitly acknowledge “use of the TPM software library, by Jeffrey W. Percival” in any appropriate publications.

1.5 See Also

http://www.scipy.org/AstroLibCoordsHome
Position object to manage coordinate transformations.

```python
class astrolib.coords.position.Position(input, equinox='J2000', system='celestial', units='degrees')
```

The basic class in the coords library. The Position class is designed to permit users to define a position and then access many representations of it.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>The input used to create the Position.</td>
</tr>
<tr>
<td>units</td>
<td>Unit in which the coords were specified.</td>
</tr>
<tr>
<td>equinox</td>
<td>Equinox at which the coordinates were specified.</td>
</tr>
<tr>
<td>system</td>
<td>System in which the coordinates were specified.</td>
</tr>
<tr>
<td>_tpmstate</td>
<td>The TPM state of the position.</td>
</tr>
<tr>
<td>coord</td>
<td>A &quot;smart&quot; representation of the position.</td>
</tr>
<tr>
<td>_internal</td>
<td>The internal representation of the position (decimal degrees).</td>
</tr>
</tbody>
</table>

**Parameters**

- **input**: string (hh:mm:ss.ss +dd:mm:ss.sss) or tuple of numbers (dd.dddd, dd.dddd)

  Coordinates of the position.

- **equinox**: string

  Equinox in which the coords are specified.

- **system**: string

  Celestial, galactic, ecliptic, etc.

- **units**: {degrees, radians}

  Angular separation.

- **angsep**(other)

  Computes the angular separation (great circle distance) between two Positions.

  **Parameters**

  - **other**: another Position

  **Returns**

  - **ans**: AngSep

  Angular separation.

- **b1950**(timetag=None)

  Return the position in Mean FK4 B1950 coordinates.

  **Parameters**

  - **timetag**: AstroDate
Timetag of returned coordinate.

**Returns r, d :** (float, float)
   Tuple of RA and DEC in decimal degrees.

**dd()**

**Returns _internal :** (float, float)
   Position in decimal degrees.

**details()**

**Returns ans :** string
   Formatted system and equinox for printing.

**ecliptic (timetag=None)**
   Return the position in IAU 1980 Ecliptic coordinates.
   **Parameters timetag :** AstroDate
   Timetag of returned coordinate.
   **Returns r, d :** (float, float)
   Tuple of (le,be) in decimal degrees.

**galactic (timetag=None)**
   Return the position in IAU 1958 Galactic coordinates.
   **Parameters timetag :** AstroDate
   Timetag of returned coordinate.
   **Returns l, b :** (float, float)
   Tuple in decimal degrees.

**hmsdms()**

**Returns value :** string
   Position in hms dms.

**j2000 (timetag=None)**
   Return the position in Mean FK5 J2000 coordinates.
   **Parameters timetag :** AstroDate
   Timetag of returned coordinate.
   **Returns r, d :** (float, float)
   Tuple of RA and DEC in decimal degrees.

**rad()**

**Returns r1, r2 :** (float, float)
   Position in radians.

**tpmstate (endstate, epoch=None, equinox=None, timetag=None)**
   This method allows the expert user to call the blackbox routine of the TPM library directly, for access to more state transitions than are supported in this interface. Little documentation is provided here because it is assumed you know what you are doing if you need this routine.
   **Parameters endstate :** integer
As defined by the TPM state machine.

**epoch**: float

Epoch in Julian date; default is J2000.

**equinox**: float

Equinox in Julian date; default is _tpmequinox.

**timetag**: AstroDate

Timetag of returned coordinate.

**Returns x2, y2**: (float, float)

Transformed coordinates in decimal degrees.

**within**(other, epsilon, units='arcsec')

Check if other is within self.

**Parameters**

- **other**: another Position
- **epsilon**: AngSep or number
  Angular separation.
- **units**: {'arcsec', 'degrees'}
  Unit of epsilon, if it is specified as a number.

**Returns ans**: boolean

True if other is within epsilon of self.

### 2.1 Utility Functions

**astrolib.coords.position.dms**(number)

Convert from decimal to sexagesimal degrees, minutes, seconds.

**Parameters**

- **number**: number

**Returns**

- **sign, dd, mm, ss**: (string, int, int, float)
  Sign, degrees, minutes, and seconds.

**astrolib.coords.position.gcdist**(vec1, vec2)

Input (ra, dec) vectors in radians; output great circle distance in radians.

**Parameters**

- **vec1, vec2**: number
  Position in radians.

**Returns**

- **ans**: number
  Great circle distance in radians.

### References

- [http://wiki.astrogrid.org/bin/view/Astrogrid/CelestialCoordinates](http://wiki.astrogrid.org/bin/view/Astrogrid/CelestialCoordinates)

**astrolib.coords.position.hav**(theta)

Haversine function, units = radians. Used in calculation of great circle distance.
Parameters \texttt{theta} : number

Angle in radians.

Returns \texttt{ans} : number

\texttt{astrolib.coords.position.ahav} (x)

archaversine function, units = radians. Used in calculation of great circle distance.

Parameters \texttt{x} : number

Returns \texttt{ans} : number

Angle in radians.
class `astrolib.coords.position.Coord`

General class for subclasses.

A Coord is distinct from a Position by being intrinsically expressed in a particular set of units.

Each Coord subclass knows how to parse its own input, and convert itself into the internal representation (decimal degrees) used by the package.

### 3.1 Subclasses

**class `astrolib.coords.position.Degrees`** *(input)*

Decimal degrees coord.

**Attributes**

| a1, a2 | float | Longitude and latitude in decimal degrees. |

**Parameters input** : (float, float)

Coordinates in decimal degrees.

**class `astrolib.coords.position.Hmsdms`** *(input)*

Sexagesimal coord: longitude in hours of time (enforced).

**Attributes**

| a1, a2 | Numpy[int,int,float] | Longitude and latitude in hours, minutes, seconds |

**Parameters input** : string

Coordinates as hh:mm:ss +dd:mm:ss.sss (sign optional).

**class `astrolib.coords.position.Radians`** *(input)*

Radians coord.
Attributes

| a1, a2 | float | Longitude and latitude in radians. |

**Parameters input**: (float, float)

Coordinates in radians.
Angular separation between two Position. Can be defined by hand, or produced by “subtracting” two positions. Like Position, it will have an internal representation, and a variety of user-selected representations.

```python
class astrolib.coords.angsep.AngSep(value, units='arcsec')
```

The AngSep class lets the user compare the angular separation between different Position without having to think too much about units.

Each AngSep object is created with a particular length in a particular set of units. These are then converted into an internal representation which is used for all math and comparisons.

All unit checks are performed by checking the first few letters of the unit string, to provide more flexibility for the user. E.g., “arc”, “arcs”, “arcsec”, and “arcseconds” will all evaluate to arcseconds.

All math and comparisons can be done either between two AngSep objects, or between an AngSep object and a number. In the latter case, the number is assumed to have the same units as the AngSep object.

### Examples

Default units are arcsec

```python
>>> a = angsep.AngSep(5)
>>> a
5.000000 arcsec
```

The usual arithmetic works

```python
>>> b = angsep.AngSep(3)
>>> a+b
8.000000 arcsec
>>> a*3
15.000000 arcsec
```

Use AngSep together with Position

```python
>>> p1 = P.Position('12:34:45.23 45:43:21.12')
>>> p2 = P.Position('12:34:47.34 45:43:23.0')
>>> sep = p1.angsep(p2)
>>> eps = angsep.AngSep(30,units='arcsec')
>>> p1.within(p2,eps)
True
>>> p2.within(p1,20)
False
```
Note: Angular Separations are inherently positive: negative separations have no physical meaning, and are forbidden.

Attributes

<table>
<thead>
<tr>
<th>value</th>
<th>number</th>
<th>Magnitude of the angular separation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>units</td>
<td>string</td>
<td>Units in which the magnitude is expressed (arcsec, degrees, or radians).</td>
</tr>
<tr>
<td>_internal</td>
<td>number (degrees)</td>
<td>Internal representation of the separation.</td>
</tr>
</tbody>
</table>

Parameters value : number
Magnitude of the angular separation.

units : string
Arcsec or degrees.

Raises ValueError :
If value < 0. Negative separations are physically meaningless and thus forbidden.

approx (other, epsilon)
True if self and other are equal to within epsilon, which is considered to have the same units as self.

Note: This is not implemented as ‘abs(self-other)<epsilon’ because of the prohibition on negative separations.

Parameters other, epsilon : AngSep or number (units of self)

Returns ans : boolean

arcsec ()

Returns value : float
Separation in arcsec.

degrees ()

Returns value : float
Separation in degrees.

radians ()

Returns value : float
Separation in radians.

setunits (units)
Sets the units of the public representation, and converts the publically visible value to those units

Parameters units : {'radians', 'arcsec', 'degrees'}
CHAPTER
FIVE

COORDS.ASTRODATE

For more information about astronomical date specifications, consult a reference source such as this page provided by the US Naval Observatory.

Constants and formulae in this module were taken from the times.h include file of the tpm package by Jeff Percival, to ensure compatibility.

**class** astrolib.coords.astrodate.BesselDate(datespec)

**Attributes**

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>float</td>
<td>Decimal Besselian year</td>
</tr>
<tr>
<td>jd</td>
<td>float</td>
<td>Julian date</td>
</tr>
<tr>
<td>mjd</td>
<td>float</td>
<td>Modified Julian Date</td>
</tr>
<tr>
<td>datespec</td>
<td></td>
<td>Date specification as entered by the user</td>
</tr>
</tbody>
</table>

**jyear()**

Return the julian year using the already-converted julian date.

**Returns ans**: float

Decimal Julian year

**class** astrolib.coords.astrodate.JulianDate(datespec)

**Attributes**

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>float</td>
<td>Decimal Julian year</td>
</tr>
<tr>
<td>jd</td>
<td>float</td>
<td>Julian date</td>
</tr>
<tr>
<td>mjd</td>
<td>float</td>
<td>Modified Julian Date</td>
</tr>
<tr>
<td>datespec</td>
<td></td>
<td>Date specification as entered by the user</td>
</tr>
</tbody>
</table>

**byear()**

Return Besselian year based on previously calculated julian date.

**Returns ans**: float

Decimal Besselian year.

**astrolib.coords.astrodate.AstroDate(datespec=None)**

AstroDate can be used as a class for managing astronomical date specifications (despite the fact that it was implemented as a factory function) that returns either a BesselDate or a JulianDate, depending on the properties of the datespec.
AstroDate was originally conceived as a Helper class for the Position function for use with pytpm functionality, but also as a generally useful class for managing astronomical date specifications.

The philosophy is the same as Position: to enable the user to specify the date once and for all, and access it in a variety of styles.

**Todo**

1. Add math functions! Addition, subtraction.
2. Is there a need to support other date specifications? eg FITS-style dates?

**Parameters**

- **datespec**: string, float, integer, `datetime.datetime`, or `None`

**Returns**

- **value**: `JulianDate` or `BesselDate`

Date specification as entered by the user. Permissible specifications include:

- **Julian year**
  - Return a JulianDate

- **Besselian year**
  - Return a BesselDate

- **Julian date**
  - ‘JD2437241.81’, ‘2437241.81’, 2437241.81
  - Return a JulianDate

- **Modified Julian date**
  - ‘MJD37241.31’
  - Return a JulianDate

- **datetime.datetime object**
  - Assume input time is UTC
  - Return a JulianDate

- **None**
  - Return the current time as a JulianDate

**Raises** `ValueError`:

- Raises an exception if the date specification is a string, but begins with a letter that is not ‘B’, ‘J’, ‘JD’, or ‘MJD’ (case insensitive).

---

**astrobib.coords.astrodate.byear2jd(byear)**

- **Parameters**
  - **byear**: float

  Decimal Besselian year.

- **Returns**
  - **value**: float

  Julian date.
The documentation for `astrolib.coords.astrodate` includes the following functions and variables:

**JD to Year**

- **Function**: `jd2jyear(jd)`
- **Parameters**:
  - `jd`: float
  - Julian date.
- **Returns**:
  - `value`: float
  - Decimal Julian year.

**Year to JD**

- **Function**: `jyear2jd(jyear)`
- **Parameters**:
  - `jyear`: float
  - Decimal Julian year.
- **Returns**:
  - `value`: float
  - Julian date.

**UTC to JD**

- **Function**: `utc2jd(utc)`
  - Convert UTC to Julian date.
  - **Parameters**:
    - `utc`: `datetime.datetime` object
      - UTC (Universal Civil Time).
  - **Returns**:
    - `value`: float
      - Julian date to the nearest second.

### 5.1 Global Variables

**Global Constants**

- **Variables**:
  - `B1950 = 2433282.42345905`
  - `J2000 = 2451545.0`
  - `CB = 36524.21987817305`
  - `CJ = 36525.0`
  - `MJD_0 = 2400000.5`

- **Conversion**:
  - Convert a string or number to a floating point number, if possible.
This routine wraps the \texttt{pytpm.blackbox} routine in order to apply the longitude convention preferred in \texttt{coords}. All \texttt{astrolib.coords} routines should call \texttt{blackbox} instead of \texttt{pytpm.blackbox}.

Since \texttt{pytpm} is itself a wrapper for the TPM library, the change could have been made there; but the modulo operator in C only works on integers, so it was simpler to do it in python. Also, this leaves \texttt{pytpm} itself as a more transparent wrapper for TPM.

\texttt{astrolib.coords.pytpm_wrapper.blackbox}(x, y, \texttt{instate}, \texttt{outstate}, \texttt{epoch}, \texttt{equinox}, \texttt{timetag=None})

\textbf{Parameters} \texttt{x}, \texttt{y} : \texttt{float}

Position in decimal degrees.

\texttt{instate}, \texttt{outstate} : \texttt{int}

The TPM states of the position.

\texttt{epoch} : \texttt{float}

Epoch of the position.

\texttt{equinox} : \texttt{float}

Equinox of the position.

\texttt{timetag} : \texttt{AstroDate}

Timetag of returned coordinate.

\textbf{Returns} \texttt{r}, \texttt{d} : \texttt{float}

Converted coordinate.

\section{6.1 Global Variables}

\texttt{astrolib.coords.pytpm.\texttt{b1950} = 2433282.42345905} \texttt{float(x) -> floating point number}

Convert a string or number to a floating point number, if possible.

\texttt{astrolib.coords.pytpm.\texttt{j2000} = 2451545.0} \texttt{float(x) -> floating point number}

Convert a string or number to a floating point number, if possible.
float(x) -> floating point number
Convert a string or number to a floating point number, if possible.

float(x) -> floating point number
Convert a string or number to a floating point number, if possible.

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by ‘+’ or ‘-‘ and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>>
int('0b100', base=0) 4

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by ‘+’ or ‘-‘ and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>>
int('0b100', base=0) 4

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by ‘+’ or ‘-‘ and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>>
int('0b100', base=0) 4

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by ‘+’ or ‘-‘ and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>>
int('0b100', base=0) 4

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by ‘+’ or ‘-‘ and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>>
int('0b100', base=0) 4

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by ‘+’ or ‘-‘ and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>>
int('0b100', base=0) 4
to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>>
int(‘0b100’, base=0) 4

>>> astrolib.coords.pytpm.s06 = 6
>>> int(x=0) -> int or long int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the
conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal
in the given base. The literal can be preceded by ‘+’ or ‘-’ and be surrounded by whitespace. The base defaults
to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>>
int(‘0b100’, base=0) 4
INDICES AND TABLES

- genindex
- modindex
- search
PYTHON MODULE INDEX

a  
astrolib.coords.angsep, 13
astrolib.coords.astrodate, 15
astrolib.coords.pytpm_wrapper, 19
PYTHON MODULE INDEX

a
astrolib.coords.angsep, 13
astrolib.coords.astrodate, 15
astrolib.coords.pytpm_wrapper, 19
A
ahav() (in module astrolib.coords.position), 10
AngSep (class in astrolib.coords.angsep), 13
angsep() (astrolib.coords.position.Position method), 7
approx() (astrolib.coords.angsep.AngSep method), 14
arcsec() (astrolib.coords.angsep.AngSep method), 14
AstroDate() (in module astrolib.coords.astrodate), 15
astrolib.coords.angsep (module), 13
astrolib.coords.astrodate (module), 15
astrolib.coords.pytpm_wrapper (module), 19

B
B1950 (in module astrolib.coords.astrodate), 17
b1950 (in module astrolib.coords.pytpm), 19
b1950() (astrolib.coords.position.Position method), 7
BesselDate (class in astrolib.coords.astrodate), 15
blackbox() (in module astrolib.coords.pytpm_wrapper), 19
byear() (astrolib.coords.astrodate.JulianDate method), 15
byear2jd() (in module astrolib.coords.astrodate), 16

C
CB (in module astrolib.coords.astrodate), 17
CB (in module astrolib.coords.pytpm), 20
CJ (in module astrolib.coords.astrodate), 17
CJ (in module astrolib.coords.pytpm), 19
Coord (class in astrolib.coords.position), 11

D
dd() (astrolib.coords.position.Position method), 8
Degrees (class in astrolib.coords.position), 11
degrees() (astrolib.coords.angsep.AngSep method), 14
details() (astrolib.coords.position.Position method), 8
dms() (in module astrolib.coords.position), 9

E
ecliptic() (astrolib.coords.position.Position method), 8

galactic() (astrolib.coords.position.Position method), 8
gcdist() (in module astrolib.coords.position), 9

H
hav() (in module astrolib.coords.position), 9
Hmsdms (class in astrolib.coords.position), 11
hmsdms() (astrolib.coords.position.Position method), 8

J
J2000 (in module astrolib.coords.astrodate), 17
j2000 (in module astrolib.coords.pytpm), 19
j2000() (astrolib.coords.position.Position method), 8
jd2jyear() (in module astrolib.coords.astrodate), 16
JulianDate (class in astrolib.coords.astrodate), 15
jyear() (astrolib.coords.astrodate.BesselDate method), 15
jyear2jd() (in module astrolib.coords.astrodate), 17

M
MJD_0 (in module astrolib.coords.astrodate), 17

P
Position (class in astrolib.coords.position), 7

R
rad() (astrolib.coords.position.Position method), 8
Radians (class in astrolib.coords.position), 11
radians() (astrolib.coords.angsep.AngSep method), 14

S
s01 (in module astrolib.coords.pytpm), 20
s02 (in module astrolib.coords.pytpm), 20
s03 (in module astrolib.coords.pytpm), 20
s04 (in module astrolib.coords.pytpm), 20
s05 (in module astrolib.coords.pytpm), 20
s06 (in module astrolib.coords.pytpm), 21
setunits() (astrolib.coords.angsep.AngSep method), 14

T
tpmstate() (astrolib.coords.position.Position method), 8

U
utc2jd() (in module astrolib.coords.astrodate), 17

W
within() (astrolib.coords.position.Position method), 9