Package 'astronomyengine'

February 27, 2026

Title R Bindings to the 'Astronomy Engine' C Library

Version 0.1.0

Description Provides access to the 'Astronomy Engine' C library (<https://github.com/cosinekitty/astronomy>) by Don Cross. The library calculates positions of the Sun, Moon, and planets, and predicts astronomical events such as rise/set times, lunar phases, equinoxes, solstices, eclipses, and transits. It is based on the 'VSOP87' planetary model and is accurate to within approximately one arcminute. This package bundles the single-file C source so that other R packages can link against it via 'LinkingTo' without shipping their own copy.

License MIT + file LICENSE

URL https://github.com/mitchelloharawild/astronomyengine

BugReports https://github.com/mitchelloharawild/astronomyengine/issues

LinkingTo cpp11

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/cpp11/version 0.5.0

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author Mitchell O'Hara-Wild [aut, cre] (ORCID: <https://orcid.org/0000-0001-6729-7695>), Don Cross [aut, cph] (Author of the bundled Astronomy Engine C library)

Maintainer Mitchell O'Hara-Wild <mail@mitchelloharawild.com>

Repository CRAN

Date/Publication 2026-02-27 20:32:11 UTC

Contents

`astro_angle_from_sun` *Angle from the Sun*

Description

Returns the angle between a celestial body and the Sun, as seen from the center of the Earth. This angle helps determine how easy it is to see the body away from the glare of the Sun.

Usage

```
astro_angle_from_sun(body, time)
```

Arguments

`body`	Integer code identifying the celestial body. Must not be Earth.
`time`	A POSIXct date-time value indicating the observation time.

Value

A list with the following elements:

angle The angle in degrees between the Sun and the body as seen from the center of the Earth.

status Status code from the underlying C function.

Examples

```
time <- as.POSIXct("2025-06-15 12:00:00", tz = "UTC")
astro_angle_from_sun(body = astro_body["MERCURY"], time = time)
```

`astro_bary_state` *Barycentric position and velocity vectors*

Description

Calculates the barycentric (solar system barycenter) position and velocity vectors for a given celestial body at a specified time.

Usage

```
astro_bary_state(body, time)
```

Arguments

`body`	Identifier of celestial body (e.g., `astro_body["MERCURY"]`).
`time`	A POSIXct time value.

Details

The vectors are expressed in J2000 mean equator coordinates (the mean equator of the Earth at noon UTC on 1 January 2000).

Value

A list with elements:

x X position in AU.

y Y position in AU.

z Z position in AU.

vx X velocity in AU/day.

vy Y velocity in AU/day.

vz Z velocity in AU/day.

time Observation time as POSIXct.

Examples

```
time <- as.POSIXct("2025-02-19 22:10:12", tz = "UTC")
astro_bary_state(astro_body["MARS"], time)
```

`astro_body` *Celestial body codes*

Description

Integer codes used by Astronomy Engine to identify celestial bodies. Pass these constants (or their integer values) to functions that accept a body argument.

Usage

```
astro_body
```

Format

An integer vector with named elements:

MERCURY 0

VENUS 1

EARTH 2

MARS 3

JUPITER 4

SATURN 5

URANUS 6

NEPTUNE 7

PLUTO 8

SUN 9

MOON 10

EMB 11 — Earth/Moon Barycenter

SSB 12 — Solar System Barycenter

Examples

```
astro_body["SUN"]
astro_body["MARS"]
```

astro_body_code *Get the integer code for a celestial body by name*

Description

Returns the integer code corresponding to the given English name.

Usage

```
astro_body_code(name)
```

Arguments

name	One of the following strings: Sun, Moon, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto, EMB, SSB.

Value

If name is one of the listed strings (case-sensitive), the returned value is the corresponding integer code (see astro_body), otherwise it is -1.

Examples

```
astro_body_code("Sun")
astro_body_code("Neptune")
```

astro_body_name *Get the name of a celestial body*

Description

Finds the name of a celestial body.

Usage

```
astro_body_name(body)
```

Arguments

body	The celestial body whose name is to be found.

Value

The English-language name of the celestial body, or "" if the body is not valid.

Examples

```
astro_body_name(astro_body["SUN"])
astro_body_name(9L)
```

astro_combine_rotation *Combine two rotation matrices*

Description

Given two rotation matrices, returns a combined rotation matrix that is equivalent to rotating based on the first matrix, followed by the second.

Usage

```
astro_combine_rotation(a, b)
```

Arguments

a	The first rotation to apply
b	The second rotation to apply

Value

The combined rotation matrix

Examples

```
# Combine two identity matrices (result is also identity)
rot_a <- astro_identity_matrix()
rot_b <- astro_identity_matrix()
combined <- astro_combine_rotation(rot_a, rot_b)
```

astro_current_time *Current UTC time according to Astronomy Engine*

Description

Uses the computer's system clock to find the current UTC date and time. Converts that date and time to a POSIXct value and returns the result. Callers can pass this value to other Astronomy Engine functions to calculate current observational conditions.

Usage

```
astro_current_time()
```

Details

On supported platforms (Linux/Unix, Mac, Windows), the time is measured with microsecond resolution.

Value

A `POSIXct` value representing the current UTC date and time.

Examples

```
astro_current_time()
```

`astro_ecliptic` *Convert J2000 equatorial coordinates to ecliptic coordinates*

Description

Converts equatorial coordinates in the J2000 frame (mean equator of Earth at noon UTC on 1 January 2000) to true ecliptic coordinates of date (relative to the plane of Earth's orbit on the given date).

Usage

```
astro_ecliptic(x, y, z, time)
```

Arguments

`x`	X coordinate in AU.
`y`	Y coordinate in AU.
`z`	Z coordinate in AU.
`time`	A POSIXct time value.

Value

A list with elements:

x X coordinate (Cartesian) in ecliptic frame in AU.

y Y coordinate (Cartesian) in ecliptic frame in AU.

z Z coordinate (Cartesian) in ecliptic frame in AU.

lon Ecliptic longitude in degrees.

lat Ecliptic latitude in degrees.

dist Distance in AU.

time Observation time as POSIXct.

Examples

```
time <- as.POSIXct("2025-02-19 22:10:12", tz = "UTC")
astro_ecliptic(1.0, 0.5, 0.2, time)
```

`astro_ecliptic_longitude` *Heliocentric ecliptic longitude of a body*

Description

Calculates the angle around the ecliptic plane of a celestial body as seen from the center of the Sun. The angle is measured prograde (in the direction of Earth's orbit) in degrees from the true equinox of date, with values in the range [0, 360).

Usage

```
astro_ecliptic_longitude(body, time)
```

Arguments

`body`	Identifier of celestial body (e.g., `astro_body[["MARS"]]`). Must not be the Sun.
`time`	A POSIXct time value.

Value

A numeric value with the ecliptic longitude in degrees [0, 360).

Examples

```
time <- as.POSIXct("2025-02-19 22:10:12", tz = "UTC")
astro_ecliptic_longitude(astro_body[["MARS"]], time)
```

`astro_elongation` *Elongation of a celestial body*

Description

Determines visibility of a celestial body relative to the Sun, as seen from the Earth. Returns information about the elongation angle and whether the body is best observed in the morning or evening.

Usage

```
astro_elongation(body, time)
```

Arguments

`body`	Integer code identifying the celestial body.
`time`	A POSIXct date-time value indicating the observation time.

Value

A list with the following elements:

visibility Integer flag indicating morning (0) or evening (1) visibility.

elongation The angle in degrees between the Earth-Sun and Earth-body vectors. Range: [0, 180].

ecliptic_separation The absolute difference in ecliptic longitude between the body and the Sun. Range: [0, 180].

time A POSIXct value representing the observation time.

status Status code from the underlying C function.

Examples

```
time <- as.POSIXct("2025-06-15 12:00:00", tz = "UTC")
astro_elongation(body = astro_body["MERCURY"], time = time)
```

`astro_equator` *Topocentric equatorial coordinates of a celestial body*

Description

Calculates equatorial coordinates of a celestial body as seen by an observer on Earth's surface.

Usage

```
astro_equator(
  body,
  time,
  latitude,
  longitude,
  height = 0,
  equdate = FALSE,
  aberration = TRUE
)
```

Arguments

`body`	Identifier of celestial body (e.g., `astro_body[["SUN"]]`, `astro_body[["MARS"]]`). Must not be the Earth.
`time`	A POSIXct time value.
`latitude`	Observer's geographic latitude in degrees (positive north).
`longitude`	Observer's geographic longitude in degrees (positive east).
`height`	Observer's height in meters above sea level.
`equdate`	One of `TRUE` (true-equator-of-date) or `FALSE` (J2000). Default is `FALSE`.
`aberration`	One of `TRUE` (correct for aberration) or `FALSE`. Default is `TRUE`.

Details

This function corrects for light travel time and topocentric parallax (the angular shift depending on the observer's location on Earth). Parallax correction is most significant for the Moon but has a small effect on other bodies.

Value

A list with elements:

ra Right ascension in sidereal hours.

dec Declination in degrees.

dist Distance in AU.

Examples

```
time <- as.POSIXct("2025-02-19 22:10:12", tz = "UTC")
astro_equator(astro_body[["MARS"]], time, latitude = -33.87, longitude = 151.21)
```

---

astro_equator_from_vector

Convert Cartesian Vector to Equatorial Coordinates

Description

Given an equatorial vector, calculates equatorial angular coordinates (right ascension and declination).

Given an equatorial vector, calculates equatorial angular coordinates (right ascension and declination).

Usage

```
astro_equator_from_vector(vector)

astro_equator_from_vector(vector)
```

Arguments

`vector`	A list with components: **x** The Cartesian x-coordinate in AU **y** The Cartesian y-coordinate in AU **z** The Cartesian z-coordinate in AU **t** The date and time (POSIXct) at which this vector is valid

Value

A list representing equatorial coordinates with elements:

- `ra`: Right ascension in sidereal hours (0-24)
- `dec`: Declination in degrees (-90 to +90)
- `dist`: Distance in AU
- `vec`: The original vector
- `status`: Status code (0 = success)

A list with components:

ra Right ascension in sidereal hours

dec Declination in degrees

dist Distance to the celestial body in AU

vec Equatorial coordinates in Cartesian vector form

status Status code (0 = success)

Examples

```
vec <- list(x = 1, y = 0.5, z = 0.25, t = as.POSIXct("2024-01-01", tz = "UTC"))
equ <- astro_equator_from_vector(vec)

# Convert vector to equatorial coordinates
vec <- list(x = 1, y = 0, z = 0, t = as.POSIXct("2024-01-01", tz = "UTC"))
equ <- astro_equator_from_vector(vec)
```

---

astro_geo_vector *Geocentric position vector of a celestial body*

Description

Calculates the position of a celestial body as a vector using the center of the Earth as the origin. The result is expressed as a Cartesian vector in the J2000 equatorial system (the mean equator of the Earth at noon UTC on 1 January 2000).

Usage

```
astro_geo_vector(body, time, aberration = "ABERRATION")
```

Arguments

`body`	Identifier of celestial body (e.g., `astro_body["MERCURY"]`).
`time`	A POSIXct time value.
`aberration`	One of `"ABERRATION"` or `"NO_ABERRATION"`. Default is `"ABERRATION"`.

Details

This function corrects for light travel time. The position of the body is back-dated by the amount of time it takes light to travel from that body to an observer on Earth.

The position can optionally be corrected for aberration, an effect causing the apparent direction of the body to be shifted due to transverse movement of the Earth.

Value

A list with elements:

x X coordinate in AU.

y Y coordinate in AU.

z Z coordinate in AU.

time Observation time as POSIXct.

Examples

```
time <- as.POSIXct("2025-02-19 22:10:12", tz = "UTC")
astro_geo_vector(astro_body["MARS"], time)
```

---

astro_helio_vector *Heliocentric position vector of a celestial body*

Description

Calculates the position of a celestial body as a vector using the center of the Sun as the origin. The result is expressed as a Cartesian vector in the J2000 equatorial system (the mean equator of the Earth at noon UTC on 1 January 2000).

Usage

```
astro_helio_vector(body, time)
```

Arguments

body	Identifier of celestial body (e.g., astro_body[["SUN"]], astro_body[["MARS"]]).
time	A POSIXct time value.

Details

The position is not corrected for light travel time or aberration.

Value

A list with elements:

x X coordinate in AU.

y Y coordinate in AU.

z Z coordinate in AU.

time Observation time as POSIXct.

Examples

```
time <- as.POSIXct("2025-02-19 22:10:12", tz = "UTC")
astro_helio_vector(astro_body[["MARS"]], time)
```

---

astro_horizon *Horizontal coordinates of a celestial body*

Description

Calculates the apparent location of a body relative to the local horizon of an observer on Earth.

Usage

```
astro_horizon(
  time,
  latitude,
  longitude,
  ra,
  dec,
  refraction = "REFRACTION_NORMAL"
)
```

Arguments

time	A POSIXct time value.
latitude	Observer's geographic latitude in degrees (positive north).
longitude	Observer's geographic longitude in degrees (positive east).
ra	Right ascension of the body in sidereal hours.
dec	Declination of the body in degrees.
refraction	One of "REFRACTION_NORMAL", "REFRACTION_JPLHOR", or "REFRACTION_NONE".

Details

Given a date, time, geographic location, and equatorial coordinates of a celestial body, this function returns horizontal coordinates (azimuth and altitude) relative to the horizon.

The `ra` and `dec` must be equator-of-date coordinates. Equator-of-date coordinates can be obtained by calling `astro_equator()` with `equdate = "EQUATOR_OF_DATE"` and `aberration = "ABERRATION"`.

Atmospheric refraction correction is recommended. Pass `refraction = "REFRACTION_NORMAL"` to correct for optical lensing of the Earth's atmosphere that causes objects to appear higher above the horizon than they actually are.

Value

A list with elements:

azimuth Azimuth angle in degrees (eastward from north).

altitude Altitude angle in degrees (positive above horizon).

ra Right ascension of the body in sidereal hours.

dec Declination of the body in degrees.

Examples

```
time <- as.POSIXct("2025-02-19 22:10:12", tz = "UTC")
astro_horizon(time, latitude = -33.87, longitude = 151.21, ra = 10.5, dec = -20.0)
```

`astro_horizon_from_vector` *Convert Cartesian Vector to Horizontal Coordinates*

Description

Given a horizontal Cartesian vector, returns horizontal azimuth and altitude.

Given a horizontal Cartesian vector, returns horizontal azimuth and altitude.

Usage

```
astro_horizon_from_vector(vector, refraction = 1L)

astro_horizon_from_vector(vector, refraction = 1L)
```

Arguments

`vector`	A list with components: **x** North component in AU **y** West component in AU **z** Zenith (up) component in AU **t** The date and time (POSIXct)

`refraction` Refraction option (0 = REFRACTION_NONE, 1 = REFRACTION_NORMAL, 2 = REFRACTION_JPLHOR). REFRACTION_NORMAL: correct altitude for atmospheric refraction (recommended). REFRACTION_NONE: no atmospheric refraction correction is performed. REFRACTION_JPLHOR: for JPL Horizons compatibility testing only; not recommended for normal use.

Details

IMPORTANT: This function differs from `astro_sphere_from_vector` in two ways:

1. The longitude value represents azimuth defined clockwise from north (e.g., east = +90), preserving traditional navigational conventions.
2. This function optionally corrects for atmospheric refraction.

The returned structure contains the azimuth in the `lon` field, measured in degrees clockwise from north: east = +90 degrees, west = +270 degrees.

IMPORTANT: This function differs from `astro_sphere_from_vector()` in two ways:

- `astro_sphere_from_vector()` returns a `lon` value that represents azimuth defined counter-clockwise from north (e.g., west = +90), but this function represents a clockwise rotation (e.g., east = +90). The difference is because `astro_sphere_from_vector()` is intended to preserve the vector "right-hand rule", while this function defines azimuth in a more traditional way as used in navigation and cartography.
- This function optionally corrects for atmospheric refraction, while `astro_sphere_from_vector()` does not.

The returned structure contains the azimuth in `lon`. It is measured in degrees clockwise from north: east = +90 degrees, west = +270 degrees.

The altitude is stored in `lat`.

The distance to the observed object is stored in `dist`, and is expressed in astronomical units (AU).

Value

A list representing horizontal coordinates with elements:

- `lat`: Altitude in degrees (-90 to +90)
- `lon`: Azimuth in degrees (0-360, clockwise from north)
- `dist`: Distance in AU
- `status`: Status code (0 = success)

A list with components:

lat Altitude angle in degrees (corrected for refraction if requested)

lon Azimuth in degrees clockwise from north

dist Distance in AU

status Status code (0 = success)

Examples

```
# Vector pointing east at 45° altitude
vec <- list(x = 0, y = -0.707, z = 0.707,
            t = as.POSIXct("2024-01-01", tz = "UTC"))
hor <- astro_horizon_from_vector(vec, refraction = 1)

# Convert horizontal vector to angular coordinates
vec <- list(x = 0, y = -1, z = 0, t = as.POSIXct("2024-01-01", tz = "UTC"))
hor <- astro_horizon_from_vector(vec, refraction = 1)
```

---

astro_hour_angle *Calculate the hour angle of a body*

Description

Finds the hour angle of a body for a given observer and time. The hour angle indicates the body's position in the sky with respect to Earth's rotation.

Usage

```
astro_hour_angle(body, time, latitude, longitude, height = 0)
```

Arguments

body	Integer body code (see astro_body).
time	A POSIXct date/time in UTC.
latitude	Observer's geographic latitude in degrees.
longitude	Observer's geographic longitude in degrees.
height	Observer's height above sea level in metres. Default 0.

Details

The hour angle is 0 when the body culminates (reaches its highest point), and increases by 1 unit for every sidereal hour that passes. A value of 12 means the body is at its lowest point. A value of 24 is equivalent to 0.

Value

A numeric value representing the hour angle in the range [0, 24), where each unit is one sidereal hour.

Examples

```
t <- as.POSIXct("2025-06-21 12:00:00", tz = "UTC")
astro_hour_angle(astro_body[["SUN"]], t,
                 latitude = -33.8688, longitude = 151.2093)
```

astro_identity_matrix *Create an identity rotation matrix*

Description

Returns a rotation matrix that has no effect on orientation. This matrix can be the starting point for other operations, such as using a series of calls to `astro_pivot()` to create a custom rotation matrix.

Usage

```
astro_identity_matrix()
```

Value

A list with components:

rot A 3x3 rotation matrix

status Status code (0 = success)

Examples

```
# Create an identity matrix
id <- astro_identity_matrix()
id
```

astro_illumination *Illumination data for a celestial body*

Description

Calculates visual magnitude, phase angle, and related illumination information for a celestial body as seen from Earth.

Usage

```
astro_illumination(body, time)
```

Arguments

`body`	An integer representing a celestial body (see `[astro_body]`). Cannot be Earth.
`time`	A `POSIXct` datetime object.

Details

Visual magnitude is a measure of brightness, where smaller (or negative) values indicate brighter objects and larger values indicate dimmer objects.

Phase angle is the angle in degrees between the Sun and Earth as seen from the body's center. It indicates what fraction of the body appears illuminated from Earth:

- Phase angle near 0° means the body appears "full"
- Phase angle near 90° means the body appears "half full"
- Phase angle near 180° means the body appears as a thin crescent

For Saturn, the returned list includes `ring_tilt`, which is the tilt angle in degrees of Saturn's rings as seen from Earth (0° means edge-on and nearly invisible).

Value

A list containing:

time The input time as a `POSIXct` object.

mag Visual magnitude (numeric).

phase_angle Phase angle in degrees (numeric).

phase_fraction Fraction of the body illuminated from 0 to 1 (numeric).

helio_dist Distance from Sun in AU (numeric).

ring_tilt Saturn's ring tilt in degrees, 0 for other bodies (numeric).

Examples

```
# Get illumination data for Mars on 2025-06-21
time <- as.POSIXct("2025-06-21", tz = "UTC")
astro_illumination(astro_body["MARS"], time)
```

`astro_inverse_rotation` *Calculate the inverse of a rotation matrix*

Description

Given a rotation matrix that performs some coordinate transform, this function returns the matrix that reverses that transform.

Usage

```
astro_inverse_rotation(rotation)
```

Arguments

`rotation`	A rotation matrix (list with `rot` component) as returned by `astro_identity_matrix()` or other rotation functions

Value

A rotation matrix that performs the opposite transformation

Examples

```
# Create a rotation and invert it
rot <- astro_identity_matrix()
inv <- astro_inverse_rotation(rot)
```

`astro_make_time` *Create an astronomical time value*

Description

Given a UTC calendar date and time, calculates a POSIXct value that can be passed to other Astronomy Engine functions for performing various calculations relating to that date and time.

Usage

```
astro_make_time(year, month, day, hour = 0L, minute = 0L, second = 0)
```

Arguments

`year`	Integer UTC calendar year (e.g. 2025).
`month`	Integer UTC calendar month in the range 1–12.
`day`	Integer UTC calendar day in the range 1–31.
`hour`	Integer UTC hour of the day in the range 0–23. Default 0.
`minute`	Integer UTC minute in the range 0–59. Default 0.
`second`	Numeric UTC floating-point second in the range [0, 60). Default 0.

Details

It is the caller's responsibility to ensure that the parameter values are correct. The parameters are not checked for validity, and this function never returns any indication of an error. Invalid values, for example passing in February 31, may cause unexpected return values.

Value

A `POSIXct` value in UTC that represents the given calendar date and time.

Examples

```
astro_make_time(2025, 6, 21)
astro_make_time(2025, 6, 21, 12, 30, 0)
```

astro_moon_phase *Moon Phase Angle*

Description

Returns the Moon's phase as an angle from 0 to 360 degrees, where 0 is new moon, 90 is first quarter, 180 is full moon, and 270 is third quarter.

Usage

```
astro_moon_phase(time)
```

Arguments

time	A POSIXct datetime representing the observation time.

Value

A numeric value representing the Moon's phase angle in degrees (0-360).

Examples

```
astro_moon_phase(as.POSIXct("2025-02-19 12:00:00", tz = "UTC"))
```

astro_next_lunar_eclipse *Find the next lunar eclipse in a series*

Description

After using astro_search_lunar_eclipse() to find the first lunar eclipse, call this function to find the next consecutive lunar eclipse. Pass in the peak value from the previous call.

Usage

```
astro_next_lunar_eclipse(prev_eclipse_time)
```

Arguments

prev_eclipse_time	A POSIXct time near a lunar eclipse peak (typically from a previous call).

Value

A list with the following elements:

kind Integer code for eclipse type (0=penumbral, 1=partial, 2=total).

obscuration Fraction of Moon's disc covered by Earth's umbra (0-1).

peak POSIXct time of eclipse peak.

sd_total Semi-duration of total phase in minutes.

sd_partial Semi-duration of partial phase in minutes.

sd_penum Semi-duration of penumbral phase in minutes.

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
first_eclipse <- astro_search_lunar_eclipse(start)
next_eclipse <- astro_next_lunar_eclipse(first_eclipse$peak)
```

---

astro_next_moon_quarter

Find Next Lunar Quarter

Description

Continues searching for lunar quarters from a previous search result. Call this function repeatedly after `astro_search_moon_quarter()` to find consecutive lunar quarters.

Usage

```
astro_next_moon_quarter(mq)
```

Arguments

`mq`	A list returned by `astro_search_moon_quarter()` or a previous call to `astro_next_moon_quarter()`.

Value

A list with:

quarter Integer 0-3: 0 = new moon, 1 = first quarter, 2 = full moon, 3 = third quarter.

time POSIXct datetime of the next lunar quarter.

Examples

```
start <- as.POSIXct("2025-02-19", tz = "UTC")
q1 <- astro_search_moon_quarter(start)
q2 <- astro_next_moon_quarter(q1)
q3 <- astro_next_moon_quarter(q2)
```

astro_next_transit *Search for the next transit of Mercury or Venus*

Description

Finds the next transit of Mercury or Venus after a previous transit. Call this repeatedly to find successive transits.

Usage

```
astro_next_transit(body, prev_transit_time)
```

Arguments

body	Integer code for the planet. Use 1 for Mercury or 2 for Venus.
prev_transit_time	A POSIXct datetime from a previous transit result.

Value

A list with elements:

start Start time of the transit (POSIXct).

peak Time of closest approach (POSIXct).

finish End time of the transit (POSIXct).

separation Angular separation at peak in arcminutes.

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
transit1 <- astro_search_transit(astro_body["MERCURY"], start)
transit2 <- astro_next_transit(astro_body["MERCURY"], transit1$peak)
```

astro_observer_gravity *Observer gravitational acceleration*

Description

Calculates the effective gravitational acceleration experienced by an observer on the Earth's surface. This combines inward gravitational acceleration with outward centrifugal acceleration due to Earth's rotation.

Usage

```
astro_observer_gravity(latitude, height)
```

Arguments

latitude	The geographic latitude of the observer in degrees north of the equator (range: -90 to +90). By formula symmetry, only the absolute value of latitude matters.
height	The elevation above sea level in meters. Accuracy is best in the range 0 to 100,000 meters.

Details

This function implements the WGS 84 Ellipsoidal Gravity Formula, which accounts for the Earth's oblate spheroid shape and rotation.

Value

The effective gravitational acceleration in meters per second squared (m/s²).

The returned value increases toward the Earth's poles and decreases toward the equator, consistent with the weight measured by a spring scale of a fixed mass moved to different latitudes and heights on Earth.

Examples

```
# Calculate gravity at sea level in different locations
gravity_equator <- astro_observer_gravity(latitude = 0, height = 0)
gravity_pole <- astro_observer_gravity(latitude = 90, height = 0)
gravity_sydney <- astro_observer_gravity(latitude = -33.8688, height = 0)

# Gravity is stronger at the poles
cat(sprintf("Equator: %.6f m/s²\n", gravity_equator))
cat(sprintf("Pole: %.6f m/s²\n", gravity_pole))
cat(sprintf("Sydney: %.6f m/s²\n", gravity_sydney))
```

---

astro_observer_state *Observer position and velocity vector from geographic coordinates*

Description

Calculates the geocentric equatorial position and velocity vectors of an observer on the surface of the Earth, taking into account the Earth's rotation.

Usage

```
astro_observer_state(time, latitude, longitude, height, of_date = FALSE)
```

Arguments

time	A POSIXct date and time for which to calculate the observer's state vector.
latitude	The geographic latitude of the observer in degrees north of the equator (range: -90 to +90).
longitude	The geographic longitude of the observer in degrees east of the prime meridian (range: 0 to 360 or -180 to +180).
height	The elevation of the observer above sea level in meters.
of_date	Logical. If TRUE, uses equator-of-date coordinates (Earth's equator at the given time). If FALSE (default), uses J2000 coordinates (Earth's equator on 2000-01-01).

Value

A list with components:

x Equatorial x-coordinate in AU

y Equatorial y-coordinate in AU

z Equatorial z-coordinate in AU

vx Equatorial x-velocity in AU/day

vy Equatorial y-velocity in AU/day

vz Equatorial z-velocity in AU/day

t The time (POSIXct) at which the state vector is valid

status Status code (0 = success)

The position vector components are expressed in Astronomical Units (AU). Multiply by #KM_PER_AU to convert to kilometers. The velocity components are in AU per day.

If only position is needed without velocity, astro_observer_vector() is slightly more efficient.

Examples

```
# Get observer state at Sydney Observatory
time <- as.POSIXct("2024-01-01 12:00:00", tz = "UTC")
obs_state <- astro_observer_state(time, latitude = -33.8688, longitude = 151.2093, height = 0)
obs_state
```

---

astro_observer_vector *Observer position vector from geographic coordinates*

Description

Calculates the geocentric equatorial position vector of an observer on the surface of the Earth, taking into account the Earth's rotation. This is the inverse function of astro_vector_observer().

Usage

```
astro_observer_vector(time, latitude, longitude, height, of_date = FALSE)
```

Arguments

time	A POSIXct date and time for which to calculate the observer's position vector.
latitude	The geographic latitude of the observer in degrees north of the equator (range: -90 to +90).
longitude	The geographic longitude of the observer in degrees east of the prime meridian (range: 0 to 360 or -180 to +180).
height	The elevation of the observer above sea level in meters.
of_date	Logical. If TRUE, uses equator-of-date coordinates (Earth's equator at the given time). If FALSE (default), uses J2000 coordinates (Earth's equator on 2000-01-01).

Value

A list with components:

x Equatorial x-coordinate in AU

y Equatorial y-coordinate in AU

z Equatorial z-coordinate in AU

t The time (POSIXct) at which the vector is valid

status Status code (0 = success)

The vector represents the displacement from Earth's center to the observer, expressed in Astronomical Units (AU). Multiply by #KM_PER_AU to convert to kilometers.

Examples

```
# Get observer position at Sydney Observatory
time <- as.POSIXct("2024-01-01 12:00:00", tz = "UTC")
obs_vec <- astro_observer_vector(time, latitude = -33.8688, longitude = 151.2093, height = 0)
obs_vec
```

---

astro_pair_longitude *Ecliptic longitude of one body relative to another*

Description

Determines where one body appears around the ecliptic plane as seen from Earth, relative to another body's apparent position.

Usage

```
astro_pair_longitude(body1, body2, time)
```

Arguments

body1	First body (e.g., astro_body["SUN"]).
body2	Second body (e.g., astro_body["MOON"])
time	A POSIXct time value.

Details

The returned angle is in the range [0, 360) degrees. The angle is 0 when the two bodies are at the same ecliptic longitude. The angle increases in the prograde direction (the direction planets orbit the Sun). When the angle is 180 degrees, the bodies appear on opposite sides of the sky.

Neither body1 nor body2 may be the Earth.

Value

A list with element:

angle Ecliptic longitude difference in degrees [0, 360).

Examples

```
time <- as.POSIXct("2025-02-19 22:10:12", tz = "UTC")
astro_pair_longitude(astro_body["SUN"], astro_body["MOON"], time)
```

---

astro_pivot *Re-orient a rotation matrix by pivoting around an axis*

Description

Given a rotation matrix, a selected coordinate axis, and an angle in degrees, this function pivots the rotation matrix by that angle around that coordinate axis.

Usage

```
astro_pivot(rotation, axis, angle)
```

Arguments

rotation	The input rotation matrix
axis	An integer that selects which coordinate axis to rotate around: 0 = x, 1 = y, 2 = z. Any other value will fail with an error.
angle	An angle in degrees indicating the amount of rotation around the specified axis. Positive angles indicate rotation counterclockwise as seen from the positive direction along that axis, looking towards the origin point of the orientation system. Any finite number of degrees is allowed, but best precision will result from keeping angle in the range [-360, +360].

Details

For example, if you have rotation matrix that converts ecliptic coordinates (ECL) to horizontal coordinates (HOR), but you really want to convert ECL to the orientation of a telescope camera pointed at a given body, you can use `astro_pivot()` twice: (1) pivot around the zenith axis by the body's azimuth, then (2) pivot around the western axis by the body's altitude angle. The resulting rotation matrix will then reorient ECL coordinates to the orientation of your telescope camera.

Value

A pivoted rotation matrix

Examples

```
# Create an identity matrix and pivot it
rot <- astro_identity_matrix()
pivoted <- astro_pivot(rot, axis = 2, angle = 45)  # Rotate 45° around z-axis
```

`astro_rotate_vector` *Apply a rotation to a vector*

Description

This function transforms a vector in one orientation to a vector in another orientation.

Usage

```
astro_rotate_vector(rotation, vector)
```

Arguments

`rotation`	A rotation matrix that specifies how the orientation of the vector is to be changed
`vector`	The vector whose orientation is to be changed. A list with components: **x** The Cartesian x-coordinate in AU **y** The Cartesian y-coordinate in AU **z** The Cartesian z-coordinate in AU **t** The date and time (POSIXct) at which this vector is valid

Value

A vector in the orientation specified by `rotation`

Examples

```
# Create a vector and rotate it
vec <- list(x = 1, y = 0, z = 0, t = as.POSIXct("2024-01-01", tz = "UTC"))
rot <- astro_identity_matrix()
rotated <- astro_rotate_vector(rot, vec)
```

astro_rotation_ECL_EQD

Rotation Matrix from J2000 Ecliptic to Equatorial of-Date

Description

Calculates a rotation matrix from J2000 mean ecliptic (ECL) to equatorial of-date (EQD).

Usage

```
astro_rotation_ECL_EQD(time)
```

Arguments

time	A POSIXct object representing the date and time.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: ECL = ecliptic system, using equator at J2000 epoch.

Target: EQD = equatorial system, using equator of date.

Value

A rotation matrix that converts ECL to EQD.

Examples

```
astro_rotation_ECL_EQD(Sys.time())
```

astro_rotation_ECL_EQJ

Rotation Matrix from ECL to EQJ

Description

Calculates a rotation matrix from J2000 mean ecliptic (ECL) to J2000 mean equator (EQJ).

Usage

```
astro_rotation_ECL_EQJ()
```

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: ECL = ecliptic system, using equator at J2000 epoch.

Target: EQJ = equatorial system, using equator at J2000 epoch.

Value

A rotation matrix that converts ECL to EQJ.

Examples

```
astro_rotation_ECL_EQJ()
```

---

astro_rotation_ECL_HOR

Rotation Matrix from ECL to HOR

Description

Calculates a rotation matrix from J2000 mean ecliptic (ECL) to horizontal (HOR).

Usage

```
astro_rotation_ECL_HOR(time, latitude, longitude, height)
```

Arguments

`time`	A POSIXct object representing the date and time of observation.
`latitude`	The observer's geographic latitude in degrees.
`longitude`	The observer's geographic longitude in degrees.
`height`	The observer's elevation above sea level in meters.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: ECL = ecliptic system, using equator at J2000 epoch.

Target: HOR = horizontal system (x = north, y = west, z = zenith).

Value

A rotation matrix that converts ECL to HOR. The components of the horizontal vector are: x = north, y = west, z = zenith (straight up from the observer).

Examples

```
astro_rotation_ECL_HOR(Sys.time(), latitude = -35.28, longitude = 149.12, height = 0)
```

`astro_rotation_ECT_EQD`

Rotation Matrix from ECT to EQD

Description

Calculates a rotation matrix from true ecliptic of date (ECT) to equator of date (EQD).

Usage

```
astro_rotation_ECT_EQD(time)
```

Arguments

`time`	A POSIXct object representing the date and time of the ecliptic/equator conversion.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: ECT = true ecliptic of date.

Target: EQD = equator of date.

Value

A rotation matrix that converts ECT to EQD.

Examples

```
astro_rotation_ECT_EQD(Sys.time())
```

astro_rotation_ECT_EQJ

Rotation Matrix from ECT to EQJ

Description

Calculates a rotation matrix from true ecliptic of date (ECT) to J2000 mean equator (EQJ).

Usage

```
astro_rotation_ECT_EQJ(time)
```

Arguments

time	A POSIXct object representing the date and time at which the Earth's ecliptic defines the source orientation.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: ECT = ecliptic system, using true equinox of the specified date/time.

Target: EQJ = equatorial system, using mean equator at J2000 epoch.

Value

A rotation matrix that converts ECT to EQJ.

Examples

```
astro_rotation_ECT_EQJ(Sys.time())
```

astro_rotation_EQD_ECL

Rotation Matrix from Equatorial of-Date to J2000 Ecliptic

Description

Calculates a rotation matrix from equatorial of-date (EQD) to J2000 mean ecliptic (ECL).

Usage

```
astro_rotation_EQD_ECL(time)
```

Arguments

time	A POSIXct object representing the date and time.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: EQD = equatorial system, using equator of date.

Target: ECL = ecliptic system, using equator at J2000 epoch.

Value

A rotation matrix that converts EQD to ECL.

Examples

```
astro_rotation_EQD_ECL(Sys.time())
```

---

astro_rotation_EQD_ECT

Rotation Matrix from EQD to ECT

Description

Calculates a rotation matrix from equator of date (EQD) to true ecliptic of date (ECT).

Usage

```
astro_rotation_EQD_ECT(time)
```

Arguments

time	A POSIXct object representing the date and time of the equator/ecliptic conversion.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: EQD = equator of date.

Target: ECT = true ecliptic of date.

Value

A rotation matrix that converts EQD to ECT.

Examples

```
astro_rotation_EQD_ECT(Sys.time())
```

`astro_rotation_EQD_EQJ` *Rotation Matrix from EQD to EQJ*

Description

Calculates a rotation matrix from equatorial of-date (EQD) to J2000 mean equator (EQJ).

Usage

```
astro_rotation_EQD_EQJ(time)
```

Arguments

`time`	A POSIXct object representing the date and time at which the Earth's equator defines the source orientation.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: EQD = equatorial system, using equator of the specified date/time.

Target: EQJ = equatorial system, using mean equator at J2000 epoch.

Value

A rotation matrix that converts EQD to EQJ.

Examples

```
astro_rotation_EQD_EQJ(Sys.time())
```

`astro_rotation_EQD_HOR` *Rotation Matrix from EQD to HOR*

Description

Calculates a rotation matrix from equatorial of-date (EQD) to horizontal (HOR).

Usage

```
astro_rotation_EQD_HOR(time, latitude, longitude, height)
```

Arguments

`time`	A POSIXct object representing the date and time at which the Earth's equator applies.
`latitude`	The observer's geographic latitude in degrees.
`longitude`	The observer's geographic longitude in degrees.
`height`	The observer's elevation above sea level in meters.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: EQD = equatorial system, using equator of the specified date/time.

Target: HOR = horizontal system (x = north, y = west, z = zenith).

Value

A rotation matrix that converts EQD to HOR. The components of the horizontal vector are: x = north, y = west, z = zenith (straight up from the observer).

Examples

```
astro_rotation_EQD_HOR(Sys.time(), latitude = -35.28, longitude = 149.12, height = 0)
```

astro_rotation_EQJ_ECL

Rotation Matrix from EQJ to ECL

Description

Calculates a rotation matrix from J2000 mean equator (EQJ) to J2000 mean ecliptic (ECL).

Usage

```
astro_rotation_EQJ_ECL(time_posix)
```

Arguments

time_posix	A POSIXct object specifying the date and time at which the Earth's equator defines the target orientation.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: EQJ = equatorial system, using equator at J2000 epoch.

Target: ECL = ecliptic system, using equator at J2000 epoch.

Value

A rotation matrix that converts EQJ to ECL at the specified time.

Examples

```
time <- as.POSIXct("2024-01-01", tz = "UTC")
astro_rotation_EQJ_ECL(time)
```

astro_rotation_EQJ_ECT

Rotation Matrix from EQJ to ECT

Description

Calculates a rotation matrix from J2000 mean equator (EQJ) to true ecliptic of date (ECT).

Usage

```
astro_rotation_EQJ_ECT(time_posix)
```

Arguments

time_posix	A POSIXct object specifying the date and time at which the Earth's equator defines the target orientation.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: EQJ = equatorial system, using mean equator at J2000 epoch.

Target: ECT = ecliptic system, using true equinox of the specified date/time.

Value

A rotation matrix that converts EQJ to ECT at the specified time.

Examples

```
time <- as.POSIXct("2024-01-01", tz = "UTC")
astro_rotation_EQJ_ECT(time)
```

---

astro_rotation_EQJ_EQD

Rotation Matrix from EQJ to EQD

Description

Calculates a rotation matrix from J2000 mean equator (EQJ) to equatorial of-date (EQD).

Usage

```
astro_rotation_EQJ_EQD(time_posix)
```

Arguments

time_posix	A POSIXct object specifying the date and time at which the Earth's equator defines the target orientation.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: EQJ = equatorial system, using equator at J2000 epoch.

Target: EQD = equatorial system, using equator of the specified date/time.

Value

A rotation matrix that converts EQJ to EQD at the specified time.

Examples

```
time <- as.POSIXct("2024-01-01", tz = "UTC")
astro_rotation_EQJ_EQD(time)
```

`astro_rotation_EQJ_GAL` *Rotation Matrix from EQJ to GAL*

Description

Calculates a rotation matrix from J2000 mean equator (EQJ) to galactic (GAL).

Usage

```
astro_rotation_EQJ_GAL()
```

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: EQJ = equatorial system, using the equator at the J2000 epoch.

Target: GAL = galactic system (IAU 1958 definition).

Value

A rotation matrix that converts EQJ to GAL.

Examples

```
astro_rotation_EQJ_GAL()
```

astro_rotation_EQJ_HOR *Rotation Matrix from J2000 Equatorial to Horizontal*

Description

Calculates a rotation matrix from J2000 mean equator (EQJ) to horizontal (HOR).

Usage

```
astro_rotation_EQJ_HOR(time, latitude, longitude, height)
```

Arguments

time	A POSIXct object representing the date and time of observation.
latitude	The observer's geographic latitude in degrees.
longitude	The observer's geographic longitude in degrees.
height	The observer's elevation above sea level in meters.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: EQJ = equatorial system, using the equator at J2000 epoch.

Target: HOR = horizontal system (x=North, y=West, z=Zenith).

Value

A rotation matrix that converts EQJ to HOR. The components represent: x = north, y = west, z = zenith (straight up from observer).

Examples

```
astro_rotation_EQJ_HOR(Sys.time(), latitude = -35.28, longitude = 149.12, height = 0)
```

astro_rotation_GAL_EQJ

Rotation Matrix from GAL to EQJ

Description

Calculates a rotation matrix from galactic (GAL) to J2000 mean equator (EQJ).

Usage

```
astro_rotation_GAL_EQJ()
```

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: GAL = galactic system (IAU 1958 definition).

Target: EQJ = equatorial system, using the equator at the J2000 epoch.

Value

A rotation matrix that converts GAL to EQJ.

Examples

```
astro_rotation_GAL_EQJ()
```

astro_rotation_HOR_ECL

Rotation Matrix from HOR to ECL

Description

Calculates a rotation matrix from horizontal (HOR) to J2000 mean ecliptic (ECL).

Usage

```
astro_rotation_HOR_ECL(time, latitude, longitude, height)
```

Arguments

time	A POSIXct object representing the date and time of observation.
latitude	The observer's geographic latitude in degrees.
longitude	The observer's geographic longitude in degrees.
height	The observer's elevation above sea level in meters.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: HOR = horizontal system (x = north, y = west, z = zenith).

Target: ECL = ecliptic system, using equator at J2000 epoch.

Value

A rotation matrix that converts HOR to ECL.

Examples

```
astro_rotation_HOR_ECL(Sys.time(), latitude = -35.28, longitude = 149.12, height = 0)
```

---

astro_rotation_HOR_EQD

Rotation Matrix from HOR to EQD

Description

Calculates a rotation matrix from horizontal (HOR) to equatorial of-date (EQD).

Usage

```
astro_rotation_HOR_EQD(time, latitude, longitude, height)
```

Arguments

time	A POSIXct object representing the date and time at which the Earth's equator applies.
latitude	The observer's geographic latitude in degrees.
longitude	The observer's geographic longitude in degrees.
height	The observer's elevation above sea level in meters.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: HOR = horizontal system (x = north, y = west, z = zenith).

Target: EQD = equatorial system, using equator of the specified date/time.

Value

A rotation matrix that converts HOR to EQD.

Examples

```
astro_rotation_HOR_EQD(Sys.time(), latitude = -35.28, longitude = 149.12, height = 0)
```

`astro_rotation_HOR_EQJ`

Rotation Matrix from Horizontal to J2000 Equatorial

Description

Calculates a rotation matrix from horizontal (HOR) to J2000 mean equator (EQJ).

Usage

```
astro_rotation_HOR_EQJ(time, latitude, longitude, height)
```

Arguments

`time`	A POSIXct object representing the date and time of observation.
`latitude`	The observer's geographic latitude in degrees.
`longitude`	The observer's geographic longitude in degrees.
`height`	The observer's elevation above sea level in meters.

Details

This is one of the family of functions that returns a rotation matrix for converting from one orientation to another.

Source: HOR = horizontal system (x=North, y=West, z=Zenith).

Target: EQJ = equatorial system, using equator at J2000 epoch.

Value

A rotation matrix that converts HOR to EQJ at `time` and for the observer.

Examples

```
astro_rotation_HOR_EQJ(Sys.time(), latitude = -35.28, longitude = 149.12, height = 0)
```

`astro_search_altitude` *Search for when a body reaches a specified altitude*

Description

Finds when the center of a given body ascends or descends through a given altitude angle, as seen by an observer at the specified location on the Earth.

Usage

```
astro_search_altitude(
  body,
  time,
  latitude,
  longitude,
  height = 0,
  direction = 1L,
  limit_days = 1,
  altitude = 0
)
```

Arguments

`body`	Integer body code (see astro_body).
`time`	A `POSIXct` date/time in UTC to start the search from.
`latitude`	Observer's geographic latitude in degrees.
`longitude`	Observer's geographic longitude in degrees.
`height`	Observer's height above sea level in metres. Default `0`.
`direction`	`1L` to find the body ascending, `-1L` to find descending. Default `1L` (ascending).
`limit_days`	Maximum number of days to search. When positive, searches forward in time; when negative, searches backward. Default `1`.
`altitude`	The desired altitude angle above (positive) or below (negative) the observer's local horizon, in degrees. Must be in the range [-90, +90].

Details

This function is useful for finding twilight times. For example:

- Civil dawn: `direction = 1, altitude = -6`
- Civil dusk: `direction = -1, altitude = -6`
- Nautical twilight: `altitude = -12`
- Astronomical twilight: `altitude = -18`

Value

A `POSIXct` value in UTC, or `NA` if no event is found within `limit_days`.

Examples

```
t <- as.POSIXct("2025-06-21", tz = "UTC")
# Find civil dawn (Sun at -6 degrees)
astro_search_altitude(astro_body[["SUN"]], t,
                      latitude = -33.8688, longitude = 151.2093,
                      direction = 1L, altitude = -6)
```

`astro_search_hour_angle` *Search for when a body reaches a specified hour angle*

Description

Searches for the time when the center of a body reaches a specified hour angle as seen by an observer on the Earth. The hour angle is 0 when the body reaches its highest point (culmination) above the horizon in a given day.

Usage

```
astro_search_hour_angle(
  body,
  time,
  latitude,
  longitude,
  height = 0,
  hour_angle = 0,
  direction = 1L
)
```

Arguments

`body`	Integer body code (see astro_body).
`time`	A `POSIXct` date/time in UTC to start the search from.
`latitude`	Observer's geographic latitude in degrees.
`longitude`	Observer's geographic longitude in degrees.
`height`	Observer's height above sea level in metres. Default `0`.
`hour_angle`	An hour angle value in the range [0, 24) indicating the number of sidereal hours after the body's most recent culmination. Default `0` (culmination).
`direction`	`1L` to search forward in time, `-1L` to search backward. Default `1L` (forward).

Details

To find when a body culminates (reaches maximum altitude), use `hour_angle = 0`. To find when a body reaches its minimum altitude, use `hour_angle = 12`.

Value

A list with elements:

time `POSIXct` time of the event.

azimuth Azimuth angle in degrees (0° = North, 90° = East).

altitude Altitude angle in degrees above the horizon.

Examples

```
t <- as.POSIXct("2025-06-21", tz = "UTC")
# Find when the Sun culminates (hour_angle = 0)
astro_search_hour_angle(astro_body[["SUN"]], t,
                        latitude = -33.8688, longitude = 151.2093)
```

`astro_search_lunar_eclipse` *Search for a lunar eclipse*

Description

Searches for the first lunar eclipse that occurs after the given start time. A lunar eclipse may be penumbral, partial, or total.

Usage

```
astro_search_lunar_eclipse(start_time)
```

Arguments

`start_time`	A `POSIXct` date and time for starting the search.

Value

A list with the following elements:

kind Integer code for eclipse type (0=penumbral, 1=partial, 2=total).

obscuration Fraction of Moon's disc covered by Earth's umbra (0-1).

peak POSIXct time of eclipse peak.

sd_total Semi-duration of total phase in minutes.

sd_partial Semi-duration of partial phase in minutes.

sd_penum Semi-duration of penumbral phase in minutes.

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
astro_search_lunar_eclipse(start)
```

`astro_search_max_elongation` *Search for maximum elongation*

Description

Finds the next date and time when Mercury or Venus reaches its maximum angle from the Sun as seen from the Earth. Maximum elongation events are the best opportunities for observing these inner planets.

Usage

```
astro_search_max_elongation(body, start_time)
```

Arguments

`body`	Integer code for the celestial body. Must be Mercury (3) or Venus (2). Other values will fail.
`start_time`	A POSIXct date-time value indicating the search start time. The maximum elongation event found will be the first one occurring after this time.

Value

A list with the following elements:

visibility Integer flag indicating morning (0) or evening (1) visibility.

elongation The maximum elongation angle in degrees.

ecliptic_separation The ecliptic separation at maximum elongation.

time A POSIXct value representing the time of maximum elongation.

status Status code from the underlying C function.

Examples

```
start <- as.POSIXct("2025-01-01 00:00:00", tz = "UTC")
astro_search_max_elongation(body = astro_body["MERCURY"], start_time = start)
```

`astro_search_moon_phase` *Search for a Specific Moon Phase*

Description

Searches for the time when the Moon reaches a specified phase angle.

Usage

```
astro_search_moon_phase(target_lon, start_time, limit_days)
```

Arguments

`target_lon`	A numeric value in the range [0, 360) representing the target phase angle. Common values: 0 = new moon, 90 = first quarter, 180 = full moon, 270 = third quarter.
`start_time`	A POSIXct datetime to begin the search.
`limit_days`	A numeric value specifying the search window in days. Positive values search forward, negative values search backward.

Value

A POSIXct datetime when the Moon reaches the target phase.

Examples

```
start <- as.POSIXct("2025-02-19", tz = "UTC")
astro_search_moon_phase(0, start, 30)  # Find next new moon
```

`astro_search_moon_quarter` *Find First Lunar Quarter*

Description

Finds the first lunar quarter (new moon, first quarter, full moon, or third quarter) after the specified date and time.

Usage

```
astro_search_moon_quarter(start_time)
```

Arguments

`start_time`	A POSIXct datetime to begin the search.

Value

A list with:

quarter Integer 0-3: 0 = new moon, 1 = first quarter, 2 = full moon, 3 = third quarter.

time POSIXct datetime of the lunar quarter.

Examples

```
start <- as.POSIXct("2025-02-19", tz = "UTC")
astro_search_moon_quarter(start)
```

---

astro_search_peak_magnitude

Search for peak magnitude of Venus

Description

Searches for the next date and time when Venus will appear brightest as seen from Earth. This function currently only supports Venus.

Usage

```
astro_search_peak_magnitude(body, start_time)
```

Arguments

body	An integer representing the celestial body. Currently only `astro_body["VENUS"]` is supported. Returns error for other bodies.
start_time	A `POSIXct` datetime object specifying when to begin the search.

Details

Venus reaches peak magnitude (maximum brightness) at certain times in its orbit. This is distinct from other brightness events: Mercury's peak magnitude occurs at superior conjunction when it's invisible, and outer planets reach peak magnitude at opposition.

The search may require iterating through multiple synodic periods of Venus to find an event after the specified start time. The function will search forward from the start time until it finds a valid peak magnitude event.

Value

A list containing:

time The time of peak magnitude as a `POSIXct` object.

mag Visual magnitude at peak brightness (numeric).

phase_angle Phase angle in degrees (numeric).

phase_fraction Fraction of Venus illuminated (numeric).

helio_dist Distance from Sun in AU (numeric).

ring_tilt Always 0 for Venus (numeric).

Examples

```
# Find when Venus will next reach peak magnitude after 2025-01-01
start <- as.POSIXct("2025-01-01", tz = "UTC")
astro_search_peak_magnitude(astro_body["VENUS"], start)
```

`astro_search_relative_longitude`
Search for relative longitude event between Earth and another planet

---

Description

Searches for the next time when the relative longitude (angle measured in the ecliptic plane from one planet to another as seen from the Sun) reaches a specified target angle.

Usage

```
astro_search_relative_longitude(body, target_rel_lon, start_time)
```

Arguments

`body`	An integer representing a planet other than Earth (see `[astro_body]`). Cannot be the Earth, Moon, or Sun.
`target_rel_lon`	Numeric. The desired relative longitude in degrees. Must be in the range [0, 360).
`start_time`	A `POSIXct` datetime object specifying when to begin the search.

Details

Relative longitude defines several important astronomical events:

0° Conjunction (inferior for Mercury/Venus, opposition for outer planets)

180° Superior conjunction (planet on opposite side of Sun from Earth)

For planets orbiting closer to the Sun than Earth (Mercury, Venus), a relative longitude of 0° indicates inferior conjunction. For planets orbiting farther from the Sun, 0° indicates opposition (closest approach).

Value

A `POSIXct` datetime object indicating when the target relative longitude is reached.

Examples

```
# Find next opposition of Mars after 2025-01-01
start <- as.POSIXct("2025-01-01", tz = "UTC")
astro_search_relative_longitude(astro_body["MARS"], 0, start)
```

`astro_search_rise_set` *Search for the next rise or set time of a celestial body*

Description

Searches for the next time a celestial body rises or sets as seen by an observer on the Earth. Rise time is when the body first starts to be visible above the horizon. Set time is when the body appears to vanish below the horizon. This function adjusts for the apparent angular radius of the observed body (significant only for the Sun and Moon) and corrects for atmospheric refraction.

Usage

```
astro_search_rise_set(
  body,
  time,
  latitude,
  longitude,
  height = 0,
  direction = 1L,
  limit_days = 1,
  meters_above_ground = 0
)
```

Arguments

`body`	Integer body code (see astro_body).
`time`	A `POSIXct` date/time in UTC to start the search from.
`latitude`	Observer's geographic latitude in degrees.
`longitude`	Observer's geographic longitude in degrees.
`height`	Observer's height above sea level in metres. Default `0`.
`direction`	`1L` to find the next rise, `-1L` to find the next set. Default `1L` (rise).
`limit_days`	Maximum number of days to search. When positive, searches forward in time; when negative, searches backward. Default `1`.
`meters_above_ground`	Height of observer above the ground (not sea level) in metres, for computing the dip of the horizon. Default `0`.

Details

Rise or set may not occur in every 24-hour period. For example, near the Earth's poles, there are long periods where the Sun stays below the horizon, never rising.

Value

A POSIXct value in UTC, or NA if no event is found within limit_days.

Examples

```
t <- as.POSIXct("2025-06-21", tz = "UTC")
# Find next sunrise at Sydney Observatory
astro_search_rise_set(astro_body[["SUN"]], t,
                      latitude = -33.8688, longitude = 151.2093)
```

---

astro_search_sun_longitude

Search for Sun longitude

Description

Searches for the time when the Sun reaches a specified apparent ecliptic longitude as seen from the center of the Earth.

Usage

```
astro_search_sun_longitude(target_lon, start_time, limit_days)
```

Arguments

target_lon	Numeric. The desired ecliptic longitude in degrees, relative to the true equinox of date. Must be in the range [0, 360). Conventional values: 0 = March equinox, 90 = June solstice, 180 = September equinox, 270 = December solstice.
start_time	POSIXct. The date and time for starting the search.
limit_days	Numeric. The number of days to search forward from start_time. Recommended range: 1 to 10 days.

Details

This function can be used to determine equinoxes and solstices. However, for calculating all equinoxes and solstices for a calendar year, astro_seasons() is usually more convenient and efficient.

The search is performed within the time window from start_time to start_time + limit_days. It is recommended to keep the search window smaller than 10 days when possible.

Value

A list with element:

time POSIXct value indicating when the Sun reaches the target longitude.

Examples

```
# Find the March equinox in 2025
start <- as.POSIXct("2025-03-15", tz = "UTC")
astro_search_sun_longitude(0, start, 10)
```

`astro_search_transit` *Search for a transit of Mercury or Venus*

Description

Finds the first transit of Mercury or Venus after a specified date. A transit occurs when an inferior planet passes between the Sun and Earth, with the planet's silhouette visible against the Sun.

Usage

```
astro_search_transit(body, start_time)
```

Arguments

`body`	Integer code for the planet. Use `1` for Mercury or `2` for Venus.
`start_time`	A `POSIXct` datetime for starting the search.

Value

A list with elements:

start Start time of the transit (POSIXct).

peak Time of closest approach (POSIXct).

finish End time of the transit (POSIXct).

separation Angular separation at peak in arcminutes.

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
astro_search_transit(astro_body["MERCURY"], start)
```

`astro_seasons` *Equinoxes and Solstices for a Given Year*

Description

Calculates the dates and times of both equinoxes and both solstices for a given calendar year.

Usage

```
astro_seasons(year)
```

Arguments

`year`	Integer calendar year. While any integer is accepted, only the years 1800 through 2100 have been validated for accuracy. Unit testing against data from the United States Naval Observatory confirms that all equinoxes and solstices for this range are within 2 minutes of the correct time.

Details

The equinoxes are the moments twice each year when the plane of the Earth's equator passes through the center of the Sun. In other words, the Sun's declination is zero at both equinoxes. The March equinox defines the beginning of spring in the northern hemisphere and the beginning of autumn in the southern hemisphere. The September equinox defines the beginning of autumn in the northern hemisphere and the beginning of spring in the southern hemisphere.

The solstices are the moments twice each year when one of the Earth's poles is most tilted toward the Sun. More precisely, the Sun's declination reaches its minimum value at the December solstice, which defines the beginning of winter in the northern hemisphere and the beginning of summer in the southern hemisphere. The Sun's declination reaches its maximum value at the June solstice, which defines the beginning of summer in the northern hemisphere and the beginning of winter in the southern hemisphere.

Value

A list of `POSIXct` values (in UTC):

mar_equinox March equinox.

jun_solstice June solstice.

sep_equinox September equinox.

dec_solstice December solstice.

Examples

```
astro_seasons(2025)
```

`astro_sphere_from_vector` *Convert Cartesian Vector to Spherical Coordinates*

Description

Given a Cartesian vector, returns latitude, longitude, and distance in spherical coordinates.

Given a Cartesian vector, returns latitude, longitude, and distance.

Usage

```
astro_sphere_from_vector(vector)

astro_sphere_from_vector(vector)
```

Arguments

`vector`	A list with components: **x** The Cartesian x-coordinate in AU **y** The Cartesian y-coordinate in AU **z** The Cartesian z-coordinate in AU **t** The date and time (POSIXct) at which this vector is valid

Value

A list representing spherical coordinates with elements:

- `lat`: Latitude in degrees
- `lon`: Longitude in degrees (0-360)
- `dist`: Distance in AU
- `status`: Status code (0 = success)

A list with components:

lat Latitude angle: -90..+90 degrees

lon Longitude angle: 0..360 degrees

dist Distance in AU

status Status code (0 = success)

Examples

```
vec <- list(x = 1, y = 1, z = 1, t = as.POSIXct("2024-01-01", tz = "UTC"))
sphere <- astro_sphere_from_vector(vec)

# Convert Cartesian to spherical
vec <- list(x = 1, y = 0, z = 0, t = as.POSIXct("2024-01-01", tz = "UTC"))
sphere <- astro_sphere_from_vector(vec)
```

`astro_sun_position` *Sun's position in ecliptic coordinates*

Description

Calculates the geocentric ecliptic coordinates of the Sun. The returned coordinates are based on the true equinox of date (the instantaneous intersection of the Earth's equatorial plane and the ecliptic plane at the given time).

Usage

```
astro_sun_position(time)
```

Arguments

`time`	A `POSIXct` object representing the date and time.

Details

This function accounts for light travel time from the Sun and corrects for precession and nutation of the Earth's axis.

Value

A list containing:

elon Ecliptic longitude in degrees.

elat Ecliptic latitude in degrees.

vec A list with Cartesian coordinates:

- **x** X-coordinate in AU.
- **y** Y-coordinate in AU.
- **z** Z-coordinate in AU.
- **t** Time as `POSIXct`.

Examples

```
time <- as.POSIXct("2025-03-20 09:00:00", tz = "UTC")
astro_sun_position(time)
```

`astro_vector_from_horizon` *Convert Horizontal Coordinates to Cartesian Vector*

Description

Given apparent angular horizontal coordinates, calculates a horizontal Cartesian vector.

Given apparent angular horizontal coordinates, calculate horizontal vector. The input azimuth is measured in degrees clockwise from north (east = +90). The returned vector is in the horizontal system: x = north, y = west, z = zenith (up).

Usage

```
astro_vector_from_horizon(sphere, time, refraction = 1L)

astro_vector_from_horizon(sphere, time, refraction = 1L)
```

Arguments

`sphere`	A list with components: **lat** Refracted altitude angle in degrees **lon** Azimuth in degrees clockwise from north **dist** Distance from the observer to the object in AU
`time`	POSIXct time of the observation
`refraction`	Refraction option (0 = REFRACTION_NONE, 1 = REFRACTION_NORMAL, 2 = REFRACTION_JPLHOR). This specifies how refraction is to be removed from the altitude stored in `sphere$lat`.

Value

A list representing a horizontal Cartesian vector with elements:

- `x`: North component in AU
- `y`: West component in AU
- `z`: Zenith (up) component in AU
- `t`: Time value
- `status`: Status code (0 = success)

A list representing a vector in the horizontal system with components:

x North component in AU

y West component in AU

z Zenith (up) component in AU

t The date and time (POSIXct)

status Status code (0 = success)

Examples

```
# 30° altitude, facing south
hor <- list(lat = 30, lon = 180, dist = 1)
time <- as.POSIXct("2024-01-01 12:00:00", tz = "UTC")
vec <- astro_vector_from_horizon(hor, time, refraction = 1)

# Convert horizontal coordinates to vector
sphere <- list(lat = 45, lon = 90, dist = 1)  # 45° altitude, 90° azimuth (east)
time <- as.POSIXct("2024-01-01 12:00:00", tz = "UTC")
vec <- astro_vector_from_horizon(sphere, time, refraction = 1)
```

---

astro_vector_from_sphere

Convert Spherical Coordinates to Cartesian Vector

Description

Given spherical coordinates and a time at which they are valid, returns a vector of Cartesian coordinates. The returned value includes the time, as required by the vector structure.

Given spherical coordinates and a time at which they are valid, returns a vector of Cartesian coordinates. The returned value includes the time, as required by the vector type.

Usage

```
astro_vector_from_sphere(sphere, time)

astro_vector_from_sphere(sphere, time)
```

Arguments

`sphere`	A list with components: **lat** Latitude angle: -90..+90 degrees **lon** Longitude angle: 0..360 degrees **dist** Distance in AU
`time`	POSIXct time at which the coordinates are valid

Value

A list representing a Cartesian vector with elements:

- `x`, `y`, `z`: Cartesian coordinates in AU
- `t`: Time value
- `status`: Status code (0 = success)

A list representing a vector with components:

x The Cartesian x-coordinate in AU

y The Cartesian y-coordinate in AU

z The Cartesian z-coordinate in AU

t The date and time (POSIXct) at which this vector is valid

status Status code (0 = success)

Examples

```
sphere <- list(lat = 45, lon = 90, dist = 1.5)
time <- as.POSIXct("2024-01-01 12:00:00", tz = "UTC")
vec <- astro_vector_from_sphere(sphere, time)

# Convert spherical to Cartesian
sphere <- list(lat = 0, lon = 0, dist = 1)
time <- as.POSIXct("2024-01-01", tz = "UTC")
vec <- astro_vector_from_sphere(sphere, time)
```

---

astro_vector_observer *Geographic coordinates from observer position vector*

Description

Given a geocentric equatorial position vector, calculates the geographic latitude, longitude, and elevation of the observer on Earth's surface. This is the inverse function of `astro_observer_vector()`.

Usage

```
astro_vector_observer(vector, of_date = FALSE)
```

Arguments

`vector`	A list with components representing the observer's position vector: **x** Equatorial x-coordinate in AU **y** Equatorial y-coordinate in AU **z** Equatorial z-coordinate in AU **t** The time (POSIXct) at which the vector is valid The components are expressed in Astronomical Units (AU). Divide kilometers by #KM_PER_AU to convert to AU.
`of_date`	Logical. If `TRUE`, interprets `vector` as equator-of-date coordinates. If `FALSE` (default), interprets it as J2000 coordinates.

Value

A list with components:

latitude Geographic latitude in degrees north of the equator (range: -90 to +90)

longitude Geographic longitude in degrees east of the prime meridian (range: 0 to 360)

height Elevation above sea level in meters

Examples

```
# Convert a position vector back to geographic coordinates
obs_vec <- list(
  x = 0.00005, y = 0.00005, z = 0.00005,
  t = as.POSIXct("2024-01-01 12:00:00", tz = "UTC")
)
obs <- astro_vector_observer(obs_vec)
obs
```

---

next_global_solar_eclipse

Search for the next global solar eclipse

Description

Finds the next solar eclipse in a series after the given eclipse time. Typically, you pass the peak value from a previous `search_global_solar_eclipse` or `next_global_solar_eclipse` call.

Usage

```
next_global_solar_eclipse(prev_eclipse_time)
```

Arguments

`prev_eclipse_time`	A `POSIXct` time near a previous eclipse peak.

Value

A list with the same structure as `search_global_solar_eclipse`.

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
eclipse1 <- search_global_solar_eclipse(start)
eclipse2 <- next_global_solar_eclipse(eclipse1$peak)
```

next_local_solar_eclipse *Search for the next local solar eclipse*

Description

Finds the next solar eclipse in a series at a specific location. Typically, you pass the peak value from a previous `search_local_solar_eclipse` or `next_local_solar_eclipse` call.

Usage

```
next_local_solar_eclipse(prev_eclipse_time, latitude, longitude)
```

Arguments

`prev_eclipse_time`	A `POSIXct` time near a previous eclipse peak.
`latitude`	Latitude of the observer in degrees (-90 to 90).
`longitude`	Longitude of the observer in degrees (-180 to 180).

Value

A list with the same structure as `search_local_solar_eclipse`.

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
eclipse1 <- search_local_solar_eclipse(start, 37.77, -122.41)
eclipse2 <- next_local_solar_eclipse(eclipse1$peak$time, 37.77, -122.41)
```

next_lunar_apsis *Find the next lunar apsis event*

Description

Given a lunar apsis event (perigee or apogee), finds the next apsis event in the series. This function alternates between finding perigees and apogees.

Usage

```
next_lunar_apsis(apsis)
```

Arguments

`apsis`	A list returned from `search_lunar_apsis()` or a previous call to `next_lunar_apsis()`.

Value

A list with the same structure as `search_lunar_apsis()`:

time A `POSIXct` datetime of the next lunar apsis.

kind Integer code: 0 for perigee, 1 for apogee.

dist_au Distance in astronomical units.

dist_km Distance in kilometers.

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
apsis1 <- search_lunar_apsis(start)
apsis2 <- next_lunar_apsis(apsis1)
apsis3 <- next_lunar_apsis(apsis2)
```

`next_planet_apsis` *Find the next planetary apsis in a series*

Description

Given an aphelion event, this function finds the next perihelion event, and vice versa. This requires an apsis event obtained from a call to `search_planet_apsis()` or a previous call to `next_planet_apsis()`.

Usage

```
next_planet_apsis(body, apsis)
```

Arguments

`body`	Integer constant identifying the planet. Use `astro_body["PLANET_NAME"]` where `PLANET_NAME` is one of: `"MERCURY"`, `"VENUS"`, `"EARTH"`, `"MARS"`, `"JUPITER"`, `"SATURN"`, `"URANUS"`, `"NEPTUNE"`, or `"PLUTO"`. Must match the body passed into the call that produced the `apsis` parameter. Not allowed to be `"SUN"` or `"MOON"`.
`apsis`	An apsis event (a list) obtained from a call to `search_planet_apsis()` or `next_planet_apsis()`.

Value

A list with the same structure as returned by `search_planet_apsis()`:

kind An integer flag: 0 for perihelion, 1 for aphelion.

time A `POSIXct` value representing the date and time of the next planetary apsis.

dist_au The distance from the planet to the Sun in astronomical units.

dist_km The distance from the planet to the Sun in kilometers.

See Also

search_planet_apsis()

Examples

```
# Find successive apsis events for Mars
start <- as.POSIXct("2025-01-01", tz = "UTC")
apsis1 <- search_planet_apsis(astro_body["MARS"], start)
apsis2 <- next_planet_apsis(astro_body["MARS"], apsis1)
apsis3 <- next_planet_apsis(astro_body["MARS"], apsis2)
```

---

search_global_solar_eclipse

Search for a global solar eclipse

Description

Searches for the first solar eclipse visible anywhere on Earth's surface that occurs after the given start time. A solar eclipse may be partial, annular, or total. To find a series of eclipses, use `next_global_solar_eclipse` with the `peak` time from the previous result.

Usage

```
search_global_solar_eclipse(start_time)
```

Arguments

`start_time`	A `POSIXct` date/time for starting the search.

Value

A list containing:

status Status code (0 = success).

kind Type of eclipse: 0 = partial, 1 = annular, 2 = total.

peak Peak time of the eclipse as `POSIXct`.

distance Distance in kilometers from Earth's center to Moon's shadow axis.

latitude Latitude of peak eclipse.

longitude Longitude of peak eclipse.

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
search_global_solar_eclipse(start)
```

search_local_solar_eclipse *Search for a local solar eclipse*

Description

Searches for the first solar eclipse visible at a specific location on Earth's surface that occurs after the given start time. Note: an eclipse reported by this function might be partly or completely invisible due to the time of day. To find a series of eclipses, use `next_local_solar_eclipse` with the `peak` time from the previous result.

Usage

```
search_local_solar_eclipse(start_time, latitude, longitude)
```

Arguments

`start_time`	A `POSIXct` date/time for starting the search.
`latitude`	Latitude of the observer in degrees (-90 to 90).
`longitude`	Longitude of the observer in degrees (-180 to 180).

Value

A list containing:

status Status code (0 = success).

kind Type of eclipse: 0 = partial, 1 = annular, 2 = total.

partial_begin Start of partial eclipse (list with `time` and `altitude`).

total_begin Start of total/annular eclipse (list with `time` and `altitude`).

peak Peak of eclipse (list with `time` and `altitude`).

total_end End of total/annular eclipse (list with `time` and `altitude`).

partial_end End of partial eclipse (list with `time` and `altitude`).

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
eclipse <- search_local_solar_eclipse(start, latitude = 37.77, longitude = -122.41)
```

`search_lunar_apsis` *Search for lunar apsis events*

Description

Finds the date and time of the Moon's closest distance (perigee) or farthest distance (apogee) with respect to the Earth after a given start time.

Usage

```
search_lunar_apsis(start_time)
```

Arguments

`start_time`	A `POSIXct` datetime object indicating when to start the search.

Details

The closest point is called *perigee* and the farthest point is called *apogee*. The word *apsis* refers to either event.

To iterate through consecutive alternating perigee and apogee events, call `search_lunar_apsis()` once, then use the return value to call `next_lunar_apsis()`. After that, keep feeding the previous return value from `next_lunar_apsis()` into another call of `next_lunar_apsis()` as many times as desired.

Value

A list containing:

time A `POSIXct` datetime of the next lunar apsis.

kind Integer code: 0 for perigee (APSIS_PERICENTER), 1 for apogee (APSIS_APOCENTER).

dist_au Distance in astronomical units.

dist_km Distance in kilometers.

Examples

```
start <- as.POSIXct("2025-01-01", tz = "UTC")
apsis <- search_lunar_apsis(start)
apsis$time
apsis$kind
```

`search_planet_apsis` *Search for the next planetary apsis*

Description

Finds the date and time of a planet's perihelion (closest approach to the Sun) or aphelion (farthest distance from the Sun) after a given time.

Usage

```
search_planet_apsis(body, start_time)
```

Arguments

`body`	Integer constant identifying the planet. Use `astro_body["PLANET_NAME"]` where `PLANET_NAME` is one of: `"MERCURY"`, `"VENUS"`, `"EARTH"`, `"MARS"`, `"JUPITER"`, `"SATURN"`, `"URANUS"`, `"NEPTUNE"`, or `"PLUTO"`. Not allowed to be `"SUN"` or `"MOON"`.
`start_time`	A `POSIXct` value indicating the date and time at which to start searching for the next perihelion or aphelion.

Details

The closest point is called perihelion and the farthest point is called aphelion. To iterate through consecutive alternating perihelion and aphelion events, call `search_planet_apsis()` once, then use the return value to call `next_planet_apsis()`. After that, keep feeding the previous return value into another call of `next_planet_apsis()` as many times as desired.

Value

A list containing:

kind An integer flag: 0 for perihelion, 1 for aphelion.

time A `POSIXct` value representing the date and time of the next planetary apsis.

dist_au The distance from the planet to the Sun in astronomical units.

dist_km The distance from the planet to the Sun in kilometers.

See Also

`next_planet_apsis()`

Examples

```
# Find the next perihelion/aphelion of Mars after 2025-01-01
start <- as.POSIXct("2025-01-01", tz = "UTC")
apsis <- search_planet_apsis(astro_body["MARS"], start)
apsis
```

Index