Tawhiri Documentation

Release 0.2.0

Cambridge University Spaceflight

November 27, 2016

Contents

Tawhiri is the name given to the next version of the Cambridge University Spacefligt balloon path and landing prediction software. The name comes from a Mori god of weather, which rather aptly "drove Tangaroa and his progeny into the sea" (WP).

Contents:

CHAPTER 1

Introduction

The project is separated into three parts:

- the predictor: provides an API over which requests for a prediction can be made. This API is public, and can be used by main predictor web UI, as a live predictor by mapping software, and potential future uses we haven't thought of.
- the web UI
- the dataset downloader: runs as a single standalone separate process, watches for new datasets published by the NOAA and retrieves them.

1.1 Setup & Installation

1.1.1 Predictor

…is written for Python 3 (though is compatible with Python 2) and needs Cython:

```
$ virtualenv venv
$ source venv/bin/activate
$ pip install cython
$ python setup.py build_ext --inplace
```

The last line (re-)builds the Cython extensions, and needs to be run again after modifying any *.pyx* files.

1.1.2 Downloader

The downloader uses gevent, so we are (disappointingly) restricted to running it under Python 2 for now.

At the time of writing, pygrib head did not work (in contrast to an earlier version), and both have a broken *setup.py*. Therefore, we need to install numpy first, and pyproj separately:

```
$ sudo aptitude install libevent-dev libgrib-api-dev
$ virtualenv -p python2 venv
$ source venv/bin/activate
$ pip install numpy
$ pip install pygrib==1.9.6 pyproj 'gevent<1.0'
```

1.1.3 Web API

The web API may be run in a development web-server using the `tawhiri-webapp` script. If necessary, you can use the `TAWHIRI_SETTINGS` environment variable to load configuration from a file:

```
$ cat > devel-settings.txt <<EOL
ELEVATION_DATASET = '/path/to/ruaumoko-dataset'
WIND_DATASET_DIR = '/path/to/tawhiri-datasets'
EOL
$ tawhiri-webapp runserver -rd
```

See the output of `tawhiri-webapp -?` and `tawhiri-webapp runserver -?` for more information.

Wind Data

Forecasts are published (for free!) by the NOAA, in the form of several hundred GRIB files.

The axes of the dataset are time, pressure level, variable, latitude and longitude. That is, the "vertical" axis is not altitude; there is a forecast for various variables at certain fixed air pressure levels. The variables we are interested in are "wind u", "wind v" and "altitude"; the first two being the speed in meters of the wind due east and north respectively.

We store wind datasets as a large array of floats (32bit). This amounts to a 9GB file on disk, which is memory mapped into the predictor and downloader processes as needed. The operating system manages caching, which means that data for popular areas can be loaded very quickly, even after a cold start of the predictor process itself.

`tawhiri.download` is responsible for acquiring the wind dataset. It downloads all the relevant GRIB files (~6GB), decompresses them, and stores the wind data in a new file on disk.

`tawhiri.interpolate`, given a dataset, estimates "wind u" and "wind v" at some time, latitude, longitude and altitude, by searching for two pressure levels between which the altiutde is contained, and interpolating along the 4 axes. More details on the implementation of this are available here

Design of the predictor

3.1 Overview

The basic idea is to do something along the lines of:

```
while not k(time, lat, lon, alt):
    lat_dot, lon_dot, alt_dot = f(time, lat, lon, alt):
    lat += lat_dot * dt
    lon += lon_dot * dt
    alt += alt_dot * dt
```

where

- f is a **model** (or a combination of, see below),
- k is a **termination function**.

3.1.1 Purity

Models, altitude profiles and termination functions must all be pure.

Besides being cleaner, it allows us to use more interesting integration methods without worrying about side effects evaluating the functions.

3.1.2 Coordinates

We principally deal with position represented as latitude, longitude and metres above sea level. While we do have to consider horizontal velocities in metres per second (e.g., when consulting wind data), we convert to latitude & longitude (or rather, "change in latitude per unit time") as soon as possible since they will (probably) be simpler to work with. ("ASAP" is so that we—as much as possible—are only working in one coordinate system throughout the code.)

Time is represented as an absolute UNIX timestamp.

3.2 Models

A model is a callable that looks something like this:

```
def f(time, lat, lon, alt):
    # < calculation goes here >
    return lat_dot, lon_dot, alt_dot
```

f(time, lat, lon, alt):
Return velocities predicted by this model (example function)

The latitude and longitude "velocities" (*lat_dot* & *lon_dot*) are "change in decimal degrees per unit time"; vertical velocity (*alt_dot*) is just metres per second.

Parameters

- **time** (*float*) – current absolute time, unix timestamp
- **lat** (*float*) – current latitude, decimal degrees
- **lon** (*float*) – current longitude, decimal degrees
- **alt** (*float*) – current altitude, metres above sea level

Return type 3-tuple of floats: `(lat_dot, lon_dot, alt_dot)`

3.2.1 . . . configuration

. . . is specified via closures, i.e. we have a function that takes some configuration and returns the actual model function.

3.2.2 . . . linear combinations thereof

We want to be able to specify several models, and then "swap bits" out, or pick from a selection when setting up a flight. E.g., we might want to choose a combination of

- wind velocity
- constant ascent
- something more exotic, say, parachute glide

For the majority of cases, a linear combination of the models we are interested in will suffice. Note that a function that linearly combines models is itself a model; see `tawhiri.models.make_linear_model()`.

3.3 Termination functions

A termination condition decides if the prediction (stage) should stop. They are functions that look something like:

```
def k(time, lat, lon, alt):
    return alt >= 30000
```

Note that a function returns `True` to indicate that the prediction should stop.

k(time, lat, lon, alt):
Decides if the prediction should stop (an example function)

Returns `True` if the prediction should terminate.

Parameters

- **time** (*float*) – current absolute time, unix timestamp
- **lat** (*float*) – current latitude, decimal degrees

- **lon** (*float*) – current longitude, decimal degrees
- **alt** (*float*) – current altitude, metres above sea level

Return type bool

3.3.1 ...combinations thereof

Similarly to the ability to linearly combine models, we can "OR" termination functions together with `tawhiri.models.make_any_terminator()`.

3.4 Chaining

We want to chain stages of a prediction together: this essentially amounts to running several predictions, with the initial conditions of the next prediction being the final position of the last, and concatenating the results (see `tawhiri.solver.solve()`).

`tawhiri.models` contains a few "pre-defined profiles", that is, functions that take some configuration and produce a chain of stages for a common scenario.

As an example, `tawhiri.models.standard_profile()` produces the chain containing two stages:

- stage 1
  - model: a linear combination (`tawhiri.models.make_linear_model()`) of constant ascent (`tawhiri.models.make_constant_ascent()`) and wind velocity `tawhiri.models.make_wind_velocity()`)
  - termination condition: above-a-certain-altitude (`tawhiri.models.make_burst_termination()`)
- stage 2
  - model: a linear combination of "drag descent" (`tawhiri.models.make_drag_descent()`) and wind velocity
  - termination condition: positive altitude (`tawhiri.models.ground_termination()`)

Specific implementation details

4.1 Interpolation

4.1.1 Introduction

Consider 2D linear interpolation: you know the values of some quantity at the four corners:

$$f(0,0) = a \qquad f(0,1) = b \qquad f(1,0) = c \qquad f(1,1) = d,$$

and you want an estimate for the value at (x, y).

You could first interpolate along the x axis, estimating $f(x,0)$ to be $(1-x)a + xc$ and $f(x,1)$ to be $(1-x)b + xd$.

(As an aside, you might think of $(1-x)$ being a 'weight': how much of a we should include in our estimate.)

Then, you could interpolate along the y axis, to get

$$\begin{aligned} f(x,y) &\approx (1-y)((1-x)a + xc) + y((1-x)b + xd) \\ &= (1-x)(1-y)a + (1-x)yb + x(1-y)c + xyd. \end{aligned}$$

Note, either from the symmetry or just doing it by hand, that you'd get exactly the same thing if you interpolated along the y axis first. You might interpret the quantity $(1-x)(1-y)$ as a weight for the top left corner, how much of it we should include in the answer.

4.1.2 Functions

The function *pick3* selects the indices left and right of a given point in time, latitude and longitude (but *not* altitude: see below), and then returns an eight element array (via a C 'out' pointer): each element represents a corner, and contains its indices and its weight (the product of the three numbers between 0 and 1 which represent how close the point we want is to this corner along each axis). Note that the 8 weights will sum to 1. In the implementation, weights are stored in a variable called *lerp*.

interp3, given the choices made by *pick3*, interpolates along the time, latitude and longitude axes, giving the value of a variable at any point on one of the pressure levels.

search finds the two pressure levels between which the desired altitude lies. It calls *interp3* to get the altitude at a certain point on each pressure level. It uses binary search.

interp4, given the choices made by *pick3* and a weight / lerp to use for the altitude interpolation, interpolates along all four axes.

4.2 Overview

`tawhiri.interpolate.make_interpolator()` casts the dataset to a pointer (see `tawhiri.interpolate.DatasetProxy`) and wraps the Cython function *get_wind* in a closure, which does the main work.

get_wind:

- calls *pick3*,
- calls *search*,
- uses *interp3* to get the altitude on the pressure levels above and below the desired point,
- calculates the weight / lerp value for interpolating along the altitude axis,
- calls *interp4* to get the final "wind u" and "wind v" values.

4.3 Extrapolation

If the altitude is below the lowest level (quite common) or above the highest (rarer), we can switch to extrapolation by allowing the weight for altitude interpolation to go out of the range $[0, 1]$.

CHAPTER 5

API

Tawhiri provides a simple API for requesting predictions. The current API version is *Version 1*.

5.1 Version 1

5.1.1 API Endpoint

There is a single endpoint, http://predict.cusf.co.uk/api/v1/, to which `GET` requests must be made with request parameters in the query string.

5.1.2 Profiles

Tawhiri supports multiple flight profiles which contain a description of the model chain to be used when predicting a specific flight type.

Tawhiri currently supports the following profiles:

- Standard Profile - `standard_profile`
- Float Profile - `float_profile`

Standard Profile

A profile for the standard high altitude balloon situation of ascent at a constant rate followed by burst and subsequent descent at terminal velocity under parachute with a predetermined sea level descent rate.

The API refers to this profile as `standard_profile`.

Float Profile

A profile for the typical floating balloon situation of ascent at constant altitude to a float altitude which persists for some amount of time before stopping. Descent is not predicted when using this profile.

The API refers to this profile as `float_profile`.

5.1.3 Requests

The following parameters are accepted for all requests to the predictor API. In addition, each profile accepts various additional parameters.

Parameter	Required	Default Value	Description
`profile`	optional	`standard_profile`	The profile to use for this prediction.
`dataset`	optional	The latest dataset.	The dataset to use for this prediction formatted as a RFC3339 timestamp.
`launch_latitude`	required		Launch latitude in decimal degrees. Must be between `-90.0` and `90.0`.
`launch_longitude`	required		Launch longitude in decimal degrees. Must be between `0.0` and `360.0`.
`launch_datetime`	required		Time and date of launch formatted as a RFC3339 timestamp.
`launch_altitude`	optional	Defaults to elevation at launch location looked up using Ruaumoko.	Elevation of launch location in metres above sea level.

Standard Profile

The standard profile accepts the following parameters in addition to the general parameters above.

Parameter	Required	Default Value	Description
`ascent_rate`	required		The ascent rate of the balloon in metres per second. Must be greater than `0.0`.
`burst_altitude`	required		The burst altitude of the balloon in metres above sea level. Must be greater than the launch altitude.
`descent_rate`	required		The descent rate of the balloon in metres per second. Must be greater than `0.0`.

Float Profile

The float profile accepts the following parameters in addition to the general parameters above.

Parameter	Required	Default Value	Description
`ascent_rate`	required		The ascent rate of the balloon in metres per second. Must be greater than `0.0`.
`float_altitude`	required		The float altitude of the balloon in metres above sea level. Must be greater than the launch altitude.
`stop_datetime`	required		Time and date to stop the float prediction formatted as a RFC3339 timestamp. Must be after the launch datetime.

5.1.4 Responses

Responses are returned in JSON and consist of various fragments. Successful responses contain `request`, `prediction` and `metadata` fragments. Error responses contain `error` and `metadata` fragments only.

The predictor API returns HTTP Status Code `200 OK` for all successful predictions.

Request Fragment

The request fragment contains a copy of the request with any optional parameters filled in. If the latest dataset is being used, its timestamp is included. The API version is also included.

Example:

```
"request": {
  "ascent_rate": 5.0,
  "burst_altitude": 30000.0,
  "dataset": "2014-08-19T12:00:00Z",
  "descent_rate": 10.0,
  "launch_altitude": 0,
  "launch_datetime": "2014-08-19T23:00:00Z",
  "launch_latitude": 50.0,
  "launch_longitude": 0.01,
  "profile": "standard_profile",
  "version": 1
}
```

Prediction Fragment

The prediction fragment consists of a list of stages according to the profile in use. Each stage has a name and a trajectory. The trajectory is a list of points. A point consists of a `latitude` (decimal degrees), a `longitude` (decimal degrees), an `altitude` (metres above sea level) and a `datetime` (RFC3339 timestamp).

Profile	Stages
`standard_profile`	`ascent`, `descent`
`float_profile`	`ascent`, `float`

Example (truncated for brevity):

```
"prediction": [
  {
    "stage": "ascent",
    "trajectory": [
      {
        "altitude": 0.0,
        "datetime": "2014-08-19T23:00:00Z",
        "latitude": 50.0,
        "longitude": 0.01
      },
      {
        "altitude": 29997.65625,
        "datetime": "2014-08-20T00:39:59.53125Z",
        "latitude": 50.016585320900354,
        "longitude": 1.0037172612852707
      }
    ]
  },
  {
    "stage": "descent",
    "trajectory": [
      {
        "altitude": 29997.65625,
        "datetime": "2014-08-20T00:39:59.53125Z",
        "latitude": 50.016585320900354,
        "longitude": 1.0037172612852707
```

```
      },
      {
        "altitude": 69.78466142247058,
        "datetime": "2014-08-20T01:02:50.625Z",
        "latitude": 50.01827279347765,
        "longitude": 1.2934223933861644
      }
    ]
  }
]
```

Metadata Fragment

The `metadata` fragment contains `start_datetime` and `complete_datetime` which are RFC3339 formatted timestamps representing the time and date when the prediction was started and completed.

Example:

```
"metadata": {
  "complete_datetime": "2014-08-19T21:32:52.036925Z",
  "start_datetime": "2014-08-19T21:32:51.929028Z"
}
```

Error Fragment

The API currently outputs the following types of errors in the error fragment:

Type	HTTP Status Code	Description
`RequestException`	`400 Bad Request`	Returned if the request is invalid.
`InvalidDatasetException`	`404 Not Found`	Returned if the requested dataset is invalid.
`PredictionException`	`500 Internal Server Error`	Returned if the predictor's solver raises an exception.
`InternalException`	`500 Internal Server Error`	Returned when an internal error occurs.
`NotYetImplementedException`	`501 Not Implemented`	Returned when the functionality requested has not yet been implemented.

Example:

```
"error": {
  "description": "Parameter 'launch_datetime' not provided in request.",
  "type": "RequestException"
}
```

5.1.5 Full Examples

Successful Standard Prediction

Request:

```
$ curl "http://predict.cusf.co.uk/api/v1/?launch_latitude=50.0&launch_longitude=0.01&launch_datetime=
```

Response (prediction truncated for brevity):

```
{
  "metadata": {
    "complete_datetime": "2014-08-19T21:32:52.036925Z",
    "start_datetime": "2014-08-19T21:32:51.929028Z"
  },
  "prediction": [
    {
      "stage": "ascent",
      "trajectory": [
        {
          "altitude": 0.0,
          "datetime": "2014-08-19T23:00:00Z",
          "latitude": 50.0,
          "longitude": 0.01
        },
        {
          "altitude": 29997.65625,
          "datetime": "2014-08-20T00:39:59.53125Z",
          "latitude": 50.016585320900354,
          "longitude": 1.0037172612852707
        }
      ]
    },
    {
      "stage": "descent",
      "trajectory": [
        {
          "altitude": 29997.65625,
          "datetime": "2014-08-20T00:39:59.53125Z",
          "latitude": 50.016585320900354,
          "longitude": 1.0037172612852707
        },
        {
          "altitude": 69.78466142247058,
          "datetime": "2014-08-20T01:02:50.625Z",
          "latitude": 50.01827279347765,
          "longitude": 1.2934223933861644
        }
      ]
    }
  ],
  "request": {
    "ascent_rate": 5.0,
    "burst_altitude": 30000.0,
    "dataset": "2014-08-19T12:00:00Z",
    "descent_rate": 10.0,
    "launch_altitude": 0,
    "launch_datetime": "2014-08-19T23:00:00Z",
    "launch_latitude": 50.0,
    "launch_longitude": 0.01,
    "profile": "standard_profile",
    "version": 1
  }
}
```

Missing Parameters

Request:

```
$ curl "http://predict.cusf.co.uk/api/v1/?launch_latitude=50.0&launch_longitude=0.01"
```

Response:

```
{
  "error": {
    "description": "Parameter 'launch_datetime' not provided in request.",
    "type": "RequestException"
  },
  "metadata": {
    "complete_datetime": "2014-08-19T21:40:08.697297Z",
    "start_datetime": "2014-08-19T21:40:08.697059Z"
  }
}
```

CHAPTER 6

Web Interface

Details of the web interface.

tawhiri

7.1 tawhiri package

7.1.1 Submodules

7.1.2 tawhiri.dataset module

Open a wind dataset from file by memory-mapping

Datasets downloaded from the NOAA are stored as large binary files that are memmapped into the predictor process and thereby treated like a huge array.

Dataset contains some utility methods to find/list datasets in a directory, and can open (& create) dataset files.

Note: once opened, the dataset is mmaped as *Dataset.array*, which by itself is not particularly useful. *tawhiri.interpolate* casts it (via a memory view) to a pointer in Cython.

class `tawhiri.dataset.`**`Dataset`**

A wind dataset

`__init__`(*ds_time*, *directory='/srv/tawhiri-datasets'*, *new=False*)

Open the dataset file for *ds_time*, in *directory*

Parameters

- **`directory`** (*string*) – directory containing the dataset
- **`ds_time`** (`datetime.datetime`) – forecast time
- **`new`** (*bool*) – should a new (blank) dataset be created (overwriting any file that happened to already be there), or should an existing dataset be opened?

See also:

open_latest()

After initalisation, the following attributes are available:

`array`

A `mmap.mmap` object; the entire dataset mapped into memory.

`ds_time`

The forecast time of this dataset (`datetime.datetime`).

. . . and this method:

close()
Close the dataset

This deletes *array*, thereby releasing (a) reference to it. Note that other objects may very well hold a reference to the array, keeping it open.

(The file descriptor is closed as soon as the dataset is mapped.)

The following attributes are class attributes:

shape = (65, 47, 3, 361, 720)
The dimensions of the dataset

Note `len(axes[i]) == shape[i]`.

axes
The values of the points on each axis: a 5-(named)tuple `(hour, pressure variable, latitude, longitude)`.

For example, `axes.pressure[4]` is `900`—points in cells `dataset.array[a][4][b][c][d]` correspond to data at 900mb.

element_type = 'float32'
The data type of dataset elements

element_size = 4
The size in bytes of *element_type*

size = 9528667200
The size in bytes of the entire dataset

SUFFIX_GRIBMIRROR = '.gribmirror'
The filename suffix for "grib mirror" files

DEFAULT_DIRECTORY = '/srv/tawhiri-datasets'
The default location of wind data

These "utility" class methods are available:

classmethod filename(*ds_time*, *directory='/srv/tawhiri-datasets'*, *suffix=''*)
Returns the filename under which we expect to find a dataset

... for forecast time *ds_time*, in *directory* with an optional *suffix*

Parameters

- **directory** (*string*) – directory in which dataset resides/will reside
- **ds_time** (`datetime.datetime`) – forecast time

Return type string

classmethod listdir(*directory='/srv/tawhiri-datasets'*, *only_suffices=None*)
Scan for datasets in *directory*

... with filenames matching those generated by *filename()* and (optionally) filter by only looking for certian suffices.

Parameters

- **directory** (*string*) – directory to search in
- **only_suffices** (*set*) – if not `None`, only return results with a suffix contained in this set

Return type (named) tuples `(dataset time, suffix, filename, full path)`

classmethod **`open_latest`**(*directory='/srv/tawhiri-datasets'*, *persistent=False*)
Find the most recent datset in *directory*, and open it

Parameters

- **`directory`** (*`string`*) – directory to search
- **`persistent`** (*`bool`*) – should the latest dataset be cached, and re-used?

Return type *`Dataset`*

7.1.3 tawhiri.download module

7.1.4 tawhiri.interpolate module

`tawhiri.interpolate.`**`make_interpolator`**(*dataset*)
Produce a function that can get wind data from *dataset* (a *`tawhiri.dataset.Dataset`*).

This function returns a closure:

`closure.`**`f`**(*hour*, *alt*, *lat*, *lng*)

Returns delta lat, lon and alt

See also:

implementation

See also:

wind_data

The interpolation code is not documented here. Please see the source on GitHub.

7.1.5 tawhiri.models module

Provide all the balloon models, termination conditions and functions to combine models and termination conditions.

`tawhiri.models.`**`float_profile`**(*ascent_rate*, *float_altitude*, *stop_time*, *dataset*, *warningcounts*)
Make a model chain for the typical floating balloon situation of ascent at constant altitude to a float altitude which persists for some amount of time before stopping. Descent is in general not modelled.

`tawhiri.models.`**`make_any_terminator`**(*terminators*)
Return a terminator that terminates when any of *terminators* would terminate.

`tawhiri.models.`**`make_burst_termination`**(*burst_altitude*)
Return a burst-termination criteria, which terminates integration when the altitude reaches *burst_altitude*.

`tawhiri.models.`**`make_constant_ascent`**(*ascent_rate*)
Return a constant-ascent model at *ascent_rate* (m/s)

`tawhiri.models.`**`make_drag_descent`**(*sea_level_descent_rate*)
Return a descent-under-parachute model with sea level descent *sea_level_descent_rate* (m/s). Descent rate at altitude is determined using an altitude model courtesy of NASA: http://www.grc.nasa.gov/WWW/K-12/airplane/atmosmet.html

For a given altitude the air density is computed, a drag coefficient is estimated from the sea level descent rate, and the resulting terminal velocity is computed by the returned model function.

`tawhiri.models.`**`make_elevation_data_termination`**(*dataset=None*)
A termination criteria which terminates integration when the altitude goes below ground level, using the elevation data in *dataset* (which should be a ruaumoko.Dataset).

`tawhiri.models.`**`make_linear_model`**(*models*)
Return a model that returns the sum of all the models in *models*.

`tawhiri.models.`**`make_time_termination`**(*max_time*)
A time based termination criteria, which terminates integration when the current time is greater than *max_time* (a UNIX timestamp).

`tawhiri.models.`**`make_wind_velocity`**(*dataset*, *warningcounts*)
Return a wind-velocity model, which gives lateral movement at the wind velocity for the current time, latitude, longitude and altitude. The *dataset* argument is the wind dataset in use.

`tawhiri.models.`**`sea_level_termination`**(*t*, *lat*, *lng*, *alt*)
A termination criteria which terminates integration when the altitude is less than (or equal to) zero.

Note that this is not a model factory.

`tawhiri.models.`**`standard_profile`**(*ascent_rate*, *burst_altitude*, *descent_rate*, *wind_dataset*, *elevation_dataset*, *warningcounts*)
Make a model chain for the standard high altitude balloon situation of ascent at a constant rate followed by burst and subsequent descent at terminal velocity under parachute with a predetermined sea level descent rate.

Requires the balloon *ascent_rate*, *burst_altitude* and *descent_rate*, and additionally requires the dataset to use for wind velocities.

Returns a tuple of (model, terminator) pairs.

7.1.6 tawhiri.solver module

`tawhiri.solver.`**`solve`**(*t*, *lat*, *lng*, *alt*, *chain*)
Solve from initial conditions *t*, *lat*, *lng* and *alt*, using models and termination criteria from *chain*, an iterable of (model, terminator) pairs which make up each stage of the flight.

7.1.7 tawhiri.api module

Provide the HTTP API for Tawhiri.

exception `tawhiri.api.`**`APIException`**
Bases: `exceptions.Exception`

Base API exception.

`status_code` **= 500**

exception `tawhiri.api.`**`InternalException`**
Bases: *`tawhiri.api.APIException`*

Raised when an internal error occurs.

`status_code` **= 500**

exception `tawhiri.api.`**`InvalidDatasetException`**
Bases: *`tawhiri.api.APIException`*

Raised if the dataset specified in the request is invalid.

`status_code` **= 404**

exception `tawhiri.api.`**`NotYetImplementedException`**
Bases: *`tawhiri.api.APIException`*

Raised when the functionality has not yet been implemented.

status_code = 501

exception tawhiri.api.**PredictionException**
Bases: *tawhiri.api.APIException*

Raised if the solver raises an exception.

status_code = 500

exception tawhiri.api.**RequestException**
Bases: *tawhiri.api.APIException*

Raised if request is invalid.

status_code = 400

tawhiri.api.**handle_exception**(*error*)
Return correct error message and HTTP status code for API exceptions.

tawhiri.api.**main**()
Single API endpoint which accepts GET requests.

tawhiri.api.**parse_request**(*data*)
Parse the request.

tawhiri.api.**ruaumoko_ds**()

tawhiri.api.**run_prediction**(*req*)
Run the prediction.

7.1.8 Module contents

Tawhiri is trajectory prediction software for high altitude balloons.

See http://www.cusf.co.uk/wiki/tawhiri:start for further details.

License & Authors

Tawhiri is Copyright 2014 (see AUTHORS & individual files) and licensed under the GNU GPL 3.

CHAPTER 9

See also

- The CUSF wiki contains pages on Tawhiri and prediction in general.
- The source is on GitHub.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

t

Index

Symbols

A

C

D

E

F

H

I

L

M

N

O

P

R

S

T