SPACEPY - QUICKSTART

JOSEF KOLLER AND OTHERS ...

1. Installation

Option 1) to install it in the standard loction (depending on your system)

```
>>> python setup.py install
```

Option 2) to install in custom location, e.g.

```
>>> python setup.py install --home=/n/packages/lib/python
```

2. Toolbox - a box full of tools

Contains tools that don't fit anywhere else but are, in general, quite useful. The following function are implemented....

3. Time and Coordinate Transformations

Import the module as `import spacepy.spacetime as st`

3.1. **TickTock class.** The TickTock class provides as number of time conversion routines. The following time coordinates are provided

UTC: Coordinated Universal Time implemented as a `datetime.datetime` class

ISO: standard ISO 8601 format like `2002-10-25T14:33:59`

TAI: International Atomic Time in units of seconds since Jan 1, 1958 (midnight) and includes leap seconds, i.e. every second has the same length

JD: Julian Day

MJD: Modified Julian Day

UNX: UNIX time in seconds since Jan 1, 1970

RDT: Rata Die Time (lat. fixed date) in days since Jan 1, 1 AD midnight

CDF: CDF Epoch time in milliseconds since Jan 1, 0

DOY: Day of Year including fractions

leaps: Leap seconds according to ftp://maia.usno.navy.mil/ser7/tai-utc.dat

To access these time coordinates, you'll create an instance of a TickTock class, e.g.

```
>>> t = st.TickTock('2002-10-25T12:30:00', 'ISO').
```

Instead of ISO you may use any of the formats listed above. You can also use numpy arrays or lists of time points. `t` has now the class attributes `t.dtype = 'ISO', t.data = '2002-10-25T12:30:00', t.UTC = datetime.datetime(2002, 10, 25, 12, 30)`. UTC is added automatically.

If you want to convert/add a class attribute from the list above, simply type e.g. `t.RTD` or `t.getRTD()`. You can replace RTD with any from the list above.

You can find out how many leap seconds were used by issuing the command

```
>>> t.getleapsecs()
```

3.2. **TickDelta class.** You can add/substract time from a TickTock class instance by creating a TickDelta instance first.

```
>>> dt = st.TickDelta(days=2.3)
```

Then you can add by e.g. `t+dt`

3.3. **SpaCo class.** The spatial coordinate class includes the following coordinate systems in cartesian and sphericals.

GDZ: (altitude, latitude, longitude in km, deg, deg

GEO: cartesian, Re

GSM: cartesian, Re

GSE: cartesian, Re

SM: cartesian, Re

GEI: cartesian, Re

MAG: cartesian, Re

SPH: same as GEO but in spherical

RLL: radial distance, latitude, longitude, Re, deg, deg.

Create a SpaCo instance with spherical='sph' or cartesian='car' coordinates:

```
>>> coord = st.SpaCo([[1,2,4],[1,2,2]], 'GEO', 'car')
```

This will let you request for example all y-coordinates by `coord.y` or if given in spherical coordinates by `coord.lati`. One can transform the coordinates by `newcoord =`

`coord.convert('GSM', 'sph')`. This will return GSM coordinates in a spherical system. Since GSM coordinates depend on time, you'll have to add first a TickTock vector like `coord.ticktock = st.TickTock(['2002-02-02T12:00:00', '2002-02-02T12:00:00'], 'ISO')`

4. RadBelt Module

The radiation belt module currently include a simple radial diffusion code as a class. Import the module and create a class

```
>>> import spacepy.radbelt as sprb
```

```
>>> rb = sprb.RBmodel()
```

Add a time grid for a particular period that you are interested in:

```
>>> rb.setup_ticks('2002-02-01T00:00:00', '2002-02-10T00:00:00', 0.25)
```

This will automatically lookup required geomagnetic/solar wind conditions for that period. Run the diffusion solver for that setup and plot the results.

```
>>> rb.evolve()
```

```
>>> rb.plot()
```

5. OMNI Module

bla bla

6. ONERA-DESP Module

bla bla

7. The testing.py module

Is supposed to test the implementation of spacepy modules.