```
#Utilities for Pyephem for Autowow                                                   Oct2007
#------------------------------------------------------------------------------------------------------------------------------------------------------------------
#Here are a collection of utility functions augmenting the astronomy package PYEPHEM by Jason Rhodes. They are used to retrieve
#satellite path data from the internet, perform conversions between data representations, compare the telescope's actual pointing position
#with its desired position (and correct if necessary), and check the end points of the levy walk for validity with regards to the horizon at
#the telescope's location.
#All functions are in python and platform independent
#required: pyephem , urllib
#------------------------------------------------------------------------------------------------------------------------------------------------------------------
#!/usr/bin/env python
# ephem helper to get NORAD 2 line satellite descriptors
# and to read the files into ephem
# june 2007
#-------------------------------------------------
import os, sys, time
import urllib
import ephem
from ephem_mathematics import *

#for LTE: 2 lines of data, one header
numlines = 2
numdata = 3
limit = 8192
#--------------------------------------------------

def get_satellite_data(url, sat_name, filename):
		site = url + sat_name
		fp = urllib.urlopen(site)
		print site
		op = open(filename, "wb")
		#print "opening file for write.."
		n = 0

		while 1:
				s = fp.read(limit)
				if not s:
				break
				op.write(s)
				n = n + len(s)

		fp.close()
		op.close()
#-------------------------------------------------

def read_tle_file(location):
	satellites = {}
	lines = open(location).readlines()
	for i in range(len(lines) - numlines):
		if lines[i+1].startswith('1 ') and lines[i+numlines].startswith('2 '):
			satellite = ephem.readtle(*lines[i:i+numdata])
			satellites[satellite.name] = satellite
	return satellites

#---------------------------------------------------
```

```
def get_transits(observer, satellite, date):
        #get unique transit times
        transit_times = []
        for minutes in range(0, 24*60, 15):
                observer.date = date
                observer.date += minutes * ephem.minute
                satellite.compute(observer)

        if satellite.transit_time is None:
                continue
        close_transit_times = [
                t for t in transit_times
                if abs(t - satellite.transit_time) < 30 * ephem.second
                ]
        if not close_transit_times:
                transit_times.append(satellite.transit_time)

        return(transit_times)
#----------------------------------------------------

def close_enough(currentposition, finalposition, threshold):
        #assumes positions are given as pairs [ra,dec]
        #if close enough return a bool true, else a bool false

        #express the position as a pair of floats of the ra and dec values
        currentpos = float(ephem.degrees(currentposition[0])), float(ephem.degrees(currentposition[1]))
        finalpos = float(ephem.degrees(finalposition[0])), float(ephem.degrees(finalposition[1]))
        difference = ephem.separation(finalpos, currentpos)
        #print "the difference in deg is: ", abs(ephem.degrees(difference))
        #print "comparing with:", threshold
        if((ephem.degrees(difference)) < threshold):
                close = True
        else:
                close = False

        return (close)
#------------------------------------------------------

def ra2angle(ra):
   #Converts an lx200 ra response into an angle in degrees
   angle = float(ephem.hours(ra))
   return angle * 360 / (2*pi)
#-------------------------------------------------------

def ra2rad(ra):
   #Converts ra to radians
    angle = float(ephem.hours(ra))
    return(angle)
#--------------------------------------------------------
def rad2angle(rad):
   #converts rad to angle in degrees( 0 - 360)
   angle = float(rad*360 / (2*pi))
   return  angle
#-------------------------------------------------------

def angle2rad(angle):
   #convert angle to radians
   rads = float(angle)
   return(rads)
#-------------------------------------------------------

def make_norm_deg(deg):
   #normalize an angle to 0 to 360 degrees
   norm_deg = deg%360
   return(norm_deg)
```

```
#--------------------------------------------------------

def make_norm_rad(rad, factor):
    #normalize an angle to 0 to factor degrees
    norm_rad = rad%(factor)
    if(norm_rad < 0.09):    #5degrees
        print "special case.."
        norm_rad = factor - norm_rad

    return(norm_rad)
#--------------------------------------------------------

def get_horizon(latitude, ra, safety_margin):
    #returns the min DEC to 'see' above the horizon, given your latitude on the earth and the direction you are pointing (ra)
    #assuming you are on the northern hemisphere

    #convert latitude to degrees:
    latitude_d = float(latitude) * 360 / (2*pi)
    #the min acceptable horizon pointing due north (northern hemisphere) is:
    min_horizon_n = (90 - latitude_d)
    #the min horizon pointing due south (northern hemisphere) is:
    min_horizon_s = (0 - latitude_d)
    #convert the ra
    angle = ra2angle(ra)
    if(angle > 180):
        angle = 360 - angle

    #get the min angle DEC to 'see' above the horizon
    min_dec = (min_horizon_n - ((angle / 180)* (min_horizon_n - min_horizon_s)) + safety_margin)
    return(min_dec)
#------------------------------------------------------------

def get_valid_levywalk_destination(sfactor, latitude, start):
    RA = start[0]; DEC = start[1]
    #make a valid new levywalk point pair (ra, dec)
    mean = 0; std = 0.1; lowerlimit = 0.0035; upperlimit = 10; location = 0.0; numpoints = 1;
    safety_margin = 5;
    #create the levy walk data point based on the new sfactor
    levy_x,levy_y = LevyWalk(mean, std, lowerlimit, upperlimit, location, sfactor, numpoints, doshow=0)
    new_ra = (ra2rad(RA) + levy_x[0])
    new_ra = make_norm_rad(new_ra, (2*pi))
    new_dec = (angle2rad(DEC) + levy_y[0])
    new_dec = make_norm_rad(new_dec, (pi/2))

    new_ra = ephem.hours(new_ra)
    new_dec = ephem.degrees(new_dec)

    min_dec = get_horizon(latitude, new_ra, safety_margin)
    if(rad2angle(float(new_dec)) > min_dec):
        print "above horizion"
        valid_ra = new_ra
        valid_dec = new_dec

    else:
        print "BELOW horizion"
        valid_ra = start[0]
        valid_dec = start[1]

    return valid_ra, valid_dec
#-------------------------------------------------------------
```