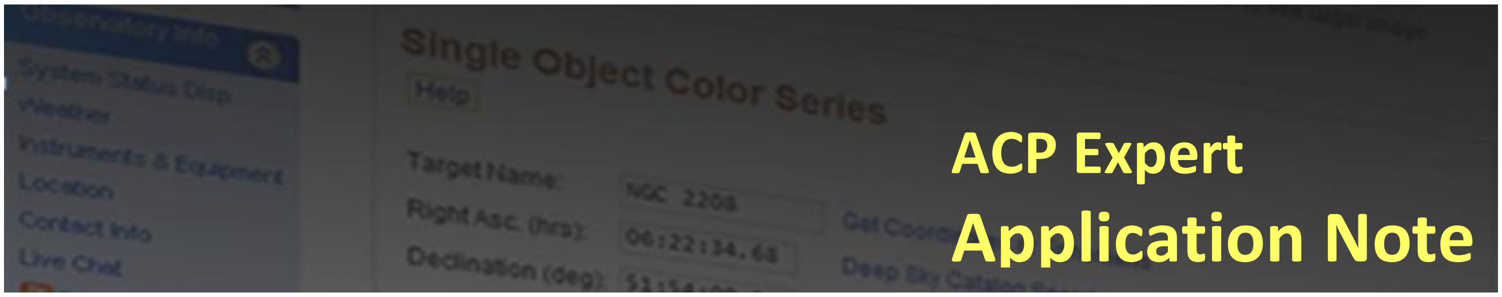

ACP Expert
Application Note

Imaging Geosynchronous Satellites with ACP

Robert B. Denny [1]
DC-3 Dreams SP, Mesa, Arizona

Abstract: This AppNote describes how to use ACP Expert's UserActions hooks, in combination with external Python code, to allow imaging geosynchronous satellites (geosats) without human intervention or stringent timing. Orbital elements (TLE) are automatically fetched from Space-Track as needed and cached locally. The application adds two new target types to ACP live observing plans: (1) stationary image(s) of a geosat field, (2) sidereally tracked image(s) of a geosat field (for subsequent photometry). These target types may be freely intermixed with conventional deep sky and solar system (orbital) targets in a single ACP live observing plan. All ACP timing, filter, binning, and repetition directives are supported for these two new target types. Also covered is useful info on providing and developing Python scripts on Windows including visual editing and debugging with Visual Studio 2015.

Prerequisites:

- Familiarity with ACP observing plan structure and directives (ACP Help)
- Familiarity with ACP UserAction extension architecture and usage (ACP Help)
- Fluency in Javascript and Windows Objects (Windows Script 5.6 Reference[1])
- Fluency in Python (Python.org)
- (recommended) Familiarity with Visual Studio Community Edition 2015

Original: ***April, 2017***
Revised: ***January 2018***

Objective and Approach

The objective of this application is to support imaging of geosynchronous satellites in multiple color bands, along with sidereally tracked images containing nearby stars. The star field images may be used for photometric analysis of the satellite images. A secondary objective is to allow the use of all of ACP's live plan directives for timing, repetition, and color band control to be used with either new type

[1] included with ACP documentation

of image. Another secondary objective is to allow standard ACP target types (equatorial coordinate, deep sky catalog, asteroids, comets) to be intermixed with the two new target types.

The approach, as is common to many ACP special applications, is to use the UserActions extension feature to augment the data acquisition process at key points. By providing UserActions, ACP allows modification of the process by code that is immune from being overwritten by ACP updates (as would a patch to provided standard code). UserActions also allows isolation of customer provided logic from the internals of ACP's logic.

Adding New Capabilities to ACP Plans

The ACP Plan language includes a #tag directive with which one can add meta-data to a standard target. In this application, the #tag directive will be used to define two new target types, and the associated standard target will be used only for its name. The #tag directive includes "name=value" string and we'll use this to pass info on the satellite. Please review the ACP Help info on the #tag directive.

```
#tag Satellite=nnnnnn
unused<tab>0<tab>0
```

Acquire tracking-off images of the satellite with NORAD Catalog number *nnnnn*. The ACP target line is a place-holder only. The name and the RA/Dec values of zero are ignored. This application patches the name to be the name of the satellite (e.g. "GALAXY 15" for catalog number 28868).

```
#tag SatField
unused<tab>0<tab>0
```

Acquire sidereally tracked images centered on the position of the previous satellite. For simplicity, this must immediately follow a Satellite target. It uses the position that was calculated from the TLE for the satellite, avoiding recalculations. We did this to more or less "force" acquisition of the star field at very nearly the same time as the satellite itself, so that the photometric measurements would be most accurate.

Calculating GeoSat Position (Alt/Az)

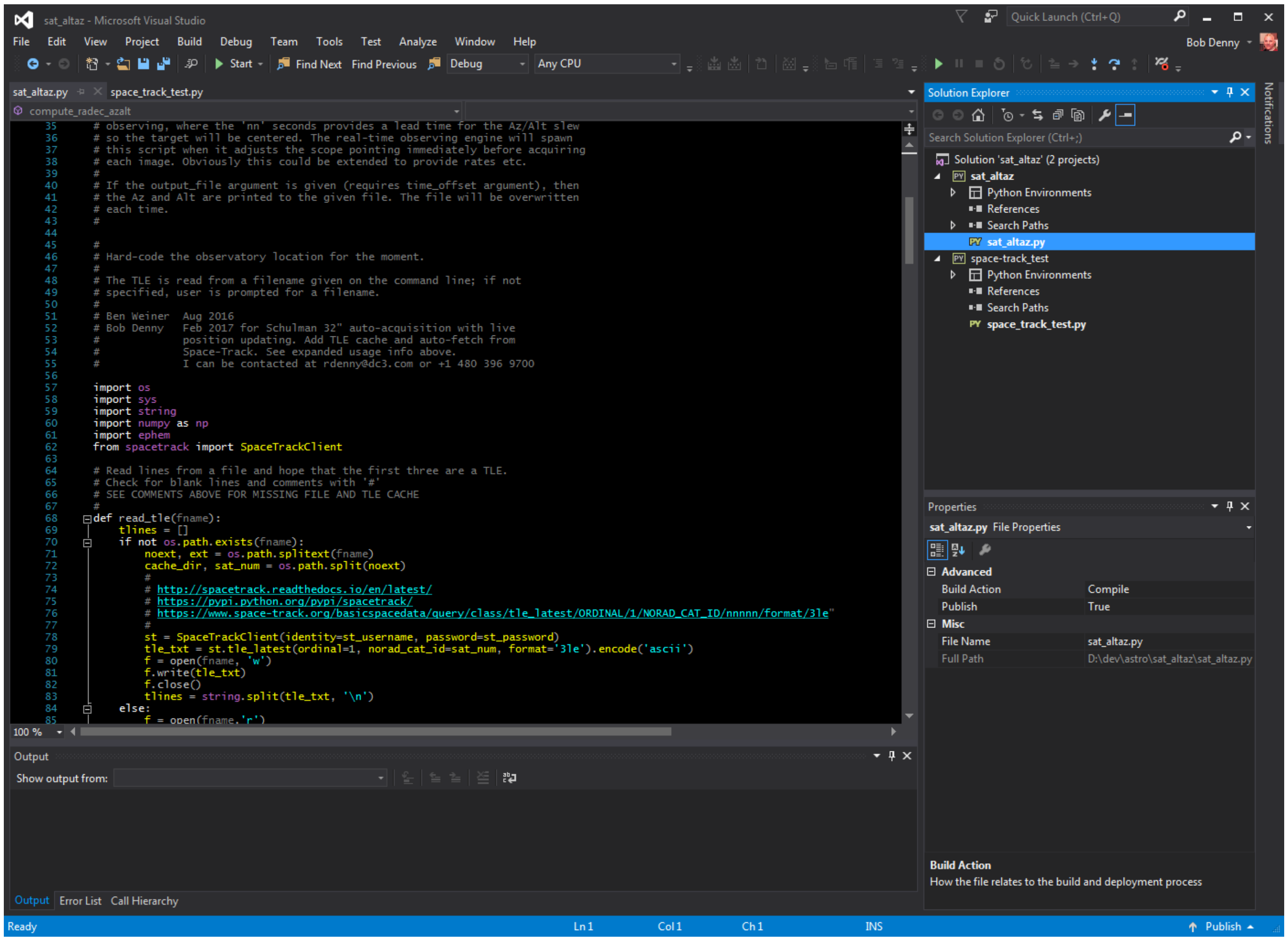
Calculation of current satellite position (alt/az) is done by a separate Python script **sat_altaz.py** originally developed at National Optical Astronomy Observatories by Ben Weiner to print out tables for manual telescope aiming and for multiple NOAO observatories. With his permission we enhanced the script to optionally return a short-form result for a single satellite, and to automatically fetch and cache NORAD Two Line Elements (TLEs) from the SpaceTrack website as needed (using the SpaceTrack REST API). The enhanced scroipt uses the **PyEphem** and **SpaceTrack** modules to greatly simplify our work. We used Python 2, but of course you can try this under Python 3.

Interfacing to the Python Script

Within the UserAction Javascript, it is a simple matter to use The ACP API along with the native Windows **FileSystemObject** to establish the file path to ACP user files. We created a new directory **My Documents\ACP Astronomy\ SatObs** to hold the **sat_altaz.py** script, as well as a subdirectory **TLECache** below that to hold the TLE cache. The ACP API has a very simple set of properties and methods to shell out to run the Python script with command line parameters. For simplicity the results (the alt/az coordinates) are written to a file and read back by the Javascript. This is all accomplished, including error detection and handling, in 37 lines of Javascript.

Python on Windows

The Python Software Foundation provides a first-class Python package for Windows. It is not necessary to install any Linux emulation such as CygWin, nor Anaconda, nor IronPython, etc. With the addition of Visual Studio Community Edition[2] and the Python Tools for Visual Studio (PTVS), you have perhaps the most advanced timesaving Python development and debugging environment on any platform. Installation of modules uses the usual Python PIP tool.



```
# observing, where the 'nn' seconds provides a lead time for the Az/Alt slew
# so the target will be centered. The real-time observing engine will spawn
# this script when it adjusts the scope pointing immediately before acquiring
# each image. Obviously this could be extended to provide rates etc.
#
# If the output_file argument is given (requires time_offset argument), then
# the Az and Alt are printed to the given file. The file will be overwritten
# each time.
#

#
# Hard-code the observatory location for the moment.
#
# The TLE is read from a filename given on the command line; if not
# specified, user is prompted for a filename.
#
# Ben Weiner   Aug 2016
# Bob Denny    Feb 2017 for Schulman 32" auto-acquisition with live
#              position updating. Add TLE cache and auto-fetch from
#              Space-Track. See expanded usage info above.
#              I can be contacted at rdenny@dc3.com or +1 480 396 9700

import os
import sys
import string
import numpy as np
import ephem
from spacetrack import SpaceTrackClient

# Read lines from a file and hope that the first three are a TLE.
# Check for blank lines and comments with '#'
# SEE COMMENTS ABOVE FOR MISSING FILE AND TLE CACHE
#
def read_tle(fname):
    tlines = []
    if not os.path.exists(fname):
        noext, ext = os.path.splitext(fname)
        cache_dir, sat_num = os.path.split(noext)
        #
        # http://spacetrack.readthedocs.io/en/latest/
        # https://pypi.python.org/pypi/spacetrack/
        # https://www.space-track.org/basicspacedata/query/class/tle_latest/ORDINAL/1/NORAD_CAT_ID/nnnnn/format/3le"
        #
        st = SpaceTrackClient(identity=st_username, password=st_password)
        tle_txt = st.tle_latest(ordinal=1, norad_cat_id=sat_num, format='3le').encode('ascii')
        f = open(fname, 'w')
        f.write(tle_txt)
        f.close()
        tlines = string.split(tle_txt, '\n')
    else:
        f = open(fname,'r')
```

Properties

sat_altaz.py File Properties

Advanced	
Build Action	Compile
Publish	True
Misc	
File Name	sat_altaz.py
Full Path	D:\dev\astro\sat_altaz\sat_altaz.py

Build Action
How the file relates to the build and deployment process

[2] We use Visual Studio 2015. The newly released VS 2017 may work fine with the Python Tools

Installing and Configuring the Package

1. Install Python for Windows on your system. Recommend Python 2.7.14 or later, but if you have Python 3.x you can try that before adding the confusing dual-version Python installation[3]. You don't need Visual Studio, just the Python for Windows vanilla installation.
2. Add the Python27 directory to your PATH.
3. Use Python's **pip** utility to install the following libraries (`pip install xxxxx`)
   ```
   numpy
   ephem
   spacetrack
   ```
4. Create a new directory My Documents\ACP Astronomy\SatObs and then a sub-directory My Documents\ACP Astronomy\SatObs\TLECache
5. Unzip UserActionsGeosyncSatellites.zip and put UserActionsGeosyncSatellites.wsc into C:\Program Files (x86)\ACP Obs Control
6. If you have a currently active UserActions with other special code, you will have to merge your existing code into this new UserActions.
7. Using the instructions in ACP Help, Customizing ACP, Adding to ACP's Logic, scroll down on that page to Activating UserActions, register the UserActionsGeosyncSatellites.wsc. Note that the instructions say to register UserActions.wsc, but you need to use the actual name UserActionsGeosyncSatellites.wsc.
8. Unzip sat_altaz.zip and put sat_altaz.py into the My Documents\ACP Astronomy\SatObs\ directory
9. Edit sat_altaz.py and search for "Observatory Location". Put your observatory latitude (positive north), longitude (positive east), and elevation (meters) into the obvious places. The sexagesimals must be strings in ''.
10. Edit sat_altaz.py and search for "Spacetrack Credentials". Put your Space-Track username and password into the obvious places.
11. Test the Python script by opening a CMD shell and setting the current directory to My Documents\ACP Astronomy\SatObs\ then typing
    ```
    ..> Python sat_altaz.py 28884 +30
    ```
    The test may take 30 seconds to a minute the first time for that satellite. You should see a response azimuth,altitude.

Integrating with ACP's Observing Logic

This application will use three of ACP's UserAction functions.

`TargetStart()` - is used to detect the special target types and set things up for the special telescope pointing needed. Using the sat_altaz.py script TLE is fetched and cached if needed, then the alt/az coordinates are calculated from the TLE for the current time and location and read into the

[3] If you do have a dual version Python installation (V2/3) you will need to elarn how to launch the appropriate version via the "py" command taht gets put into C:\Windows. If you have Python in your PATH you'll have to resolve which one gets run etc. It's beyond the scope of this document to explain all of this.

UserAction's global variables. Due to ACP's pre-slewing feature, this requires some extra logic and care, see the next section.

`ImageStart()` - on the first image, tracking is stopped (for the alt/az slew) then the scope is slewed to the alt/az coordinates for the satellite. Then for a `Satellite` target the tracking is left off. For a `SatField`, the tracking is turned back on. For subsequent images nothing is done in ImageStart().

`TargetEnd()` - If the tracking is off, it is turned back on at this point, as expected by ACP's multi-target acquisition logic.

ACP's Pre-Slew Logic - Looking Ahead

One of ACP's professional features is its ability to detect the last (or only) image for a given target, and start the slew to the next target before starting the download of the image data, thus providing overlapped slewing and image downloading. This creates a problem for the satellite application.

Suppose the current target in the ACP plan is a normal deep sky object, and it is followed in the plan by a Satellite target. The UserAction functions are always called in the context of the *current* target. So on the last image of the (current) deep sky target ACP will start a slew to the coordinates of the (next) Satellite target. It will, of course, use the RA and Dec of the next target, which for a Satellite, will be zero! How can we prevent this from hapening?

Fortunately ACP's TargetStart() function is passed handles to both the *current* target object (from the Plan Complier) as well as the *upcoming* target object (if any). To make the pre-slew work to our advantage, if the upcoming target is a Satellite or SatField, we patch its RA/Dec to equatorial coordinates of the satellite. Since the ACP Plan has been compiled into proper Windows Objects, we just overwrite the zero values that the compiler put there for NextTarget.RA and NextTarget.Dec with the RA and Dec of the current satellite position (again using the Python script etc.). Of course this will only be approximate, so later, after the final image of the current target has been downloaded, and the next (Satellite or SatField) target is processed, the TargetStart() is called again and the Alt/Az is calculated for the precision slew in the first call to ImageStart(). The name of the satellite overwrites the upcoming target name as well.

Sample ACP Observing Plan

Here is a sample ACP observing plan with intermixed deep sky and satellite targets.

```
#binning 1,1
#count 2,2
#filter B,V
#posang 0
#tag Satellite=27715
www   0     0
;
;
#interval 20,16
#binning 1,1
#filter B,V
#count 1,1
#posang 0
#tag SatField=yes
xxx   0     0
;
;
#interval 45
#binning 1
#filter Blue
#count 1
#posang 45
Block Nebula      2.5   22
```

Real-World Issues

The number one source of "issues" for this application is behavioral quirks of the mount.

Some mounts (properly) insist on sidereal tracking being off for alt/az slews, others need sidereal tracking on even for an alt/az slew, with the result that the mount tracks away from the destination Alt/Az immediately upon completion of the slew. Worse, a mount might calculate the destination Alt/Az at the start of the slew and miss it by the sidereally tracked distance over the duration of the slew. Expect some wrestling with this.

Another issue is a spontaneous change in sidereal tracking as a side effect of an Alt/Az or Equatorial slew. You may need to force tracking to be on or off as needed at places in the logic that you might not have thought of.

Ideas for Enhancement

Depending on the capabilities of the mount, this could be extended to image fast moving satellites. The coordinate rates are available, but the ASCOM Telescope interface doesn't provide a generic alt/az tracking rate capability. If it were possible to access this in the telescope controller, then tracked images of fast moving satellites would be possible. Timing is everything! It would be important to run MaxIm DL's shutter latency measurement so that MaxIm DL can accurately time-stamp images.

The TLE cache should hold TLEs for a limited amount of time. At a minimum, all TLE files in the cache could simply be deleted at the beginning of each night. Other more "intelligent" ageing schemes are obviously possible.

Acknowledgements:

Adam Block - University of Arizona Mt. Lemmon Sky Center, for whom this application was originally developed. It is being run on the Schulman 32" RC at MLSC.

Ben Weiner - National Optical Astronomy Observatories, for the original sat_altaz.py script

[1]**Contact Information**

Robert B. Denny
DC-3 Dreams, SP
6665 E. Vanguard St.
Mesa, AZ 85215-7737 USA
+1 480 396 9700
rdenny@dc3.com