Het Python Interface

Release 1.0.0

James R. Fowler

April 23, 2018

CONTENTS

CHAPTER

ONE

INTRODUCTION

The Telescope Control System (Tcs) at the Hobby-Eberly Telescope (Het) consists of a approximately ten control system programs and associated subsystems. There is a built-in messaging interface to allow communications between systems. Control systems act as servers and can also be clients of the other control systems. About half the control systems within Tcs manage hardware associated with pointing and operating the telescope. The Tcs software is written in C++. In order to allow easy program development and scripting capabilities a set of python interfaces have been developed. This documentation describes the Python interface and the application programming interface (API) for the Tcs.

The Python interface to the Tcs Control System at Het consists of the **Tcs Python** modules and the **Hetlib** package. **Hetlib** provides wrappers around some of the routines in the **Tcs Python** modules as well as scripts to assist in the operation of the telescope system. These packages and modules are available to other software developers and provide an easy interface to the Tcs Control Systems.

Note that the **Tcs Python** modules will be migated to the **pytcs** package at some future date which will be soon I hope.

1.1 Definition of Terms

1.1.1 Control System

A Control System is a standalone program. It may receive *Handler* requests (basically commands) from clients and it may emit events that other programs can receive. The control systems with in the Tcs are the tracker_server, tcs_server, pasServer, pfipserver, legacyServer, tcsGui, apcServer, tcs_monitor, tcs_log_relay, and tcsnamed.

1.1.2 Subsystem

A subsystem is the name for a part of a *Control System*. For example, the legacyServer controls the dome and shutter as well as collecting the weather information. The dome, shutter, and weather are all subsystems of the legacy *Control System*.

1.1.3 Route

A route is a URL (Uniform Resource Locator) that is used by clients to talk to the *Control System*. Requests to a *Control System* are sent to the route URL. The URL is of the form `aaa://xxx.xxx.xxx.xxx:nnnnn` where aaa is the protocol (e.g. tcp or udp), xxx.xxx.xxx.xxx is an IP address, and nnnnn is a port number. For example, the route to the tcs_server is tcp://192.168.66.31:30300. All *Control System* within Tcs use the TCP protocol. Routes are available from the tcsnamed server.

1.1.4 Event-route

An event route is a URL that is used by a *Control System* to emit *Events*. Clients and other *Control System* may listen to these *Events* to obtain state or activity information about the *Control System*. The Event-route's port number is normally one higher than the *Control System Route* port. For example, the tcs_server route is on port 30300 and its event-route is on port 30301. Event-routes are available from the tcsnamed server.

1.1.5 Handler

A handler is a function within a *Control System* that may be called by a client. A handler is called through the *Route* to the *Control System*. When the handler completes a return value is sent to the client. Some handlers return immediately even though the action may take longer and the client will have to look at *Events* from the *Control System* in order to see when the action might be complete.

1.1.6 Events

Events are emitting by a *Control System* via a broadcast on the *Event-route* URL. Clients may subscribe to events from some or all *Control System*. Clients are not able to pick which events from a particular *Control System* they subscribe to. Client must filter events on their end and provide any call back required to handle the event.

CHAPTER

TWO

INSTALLATION

2.1 Location of the source code

The source code for the **Tcs** system may be obtained from the SVN repository located at the University of Texas at Austin, McDonald Observatory. The machine does not have an anonymous download mechanism and you will need an account on the machine in order to obtain the software. Assumming that you do have an account the command to get the software is

```
svn checkout svn+ssh://<user>@hetdex-pr.as.utexas.edu/repos/het/trunk
```

At the HET the main directory for the source code and the operating environment is `/opt/het/hetdex`. The working copy of the source for the **Tcs** python scripting package is usually in `/opt/het/hetdex/src/tcs/trunk/scripting`. The **Hetlib** library can be found in `/opt/het/hetdex/src/tcs/trunk/scripting/hetlib` and the scripts can be found in `/opt/het/hetdex/src/tcs/trunk/scripting/scripts`. Third-party packages are in various sub-directories under `/opt/het/hetdex/src`. Normally these directories are mounted via NFS from gringotts and should be accessible on all our machines.

The binaries are typically installed in `/opt/het/hetdex/bin`; the python packages are typically installed in `/opt/het/hetdex/lib/python2.7/site-packages`; configuration files are typically found in `/opt/het/hetdex/etc/conf`.

2.2 Required packages

The **Tcs Python** scripting package requires the **numpy**, **scipy**, **sqlite3**, and **zmq** python packages to support operations. The **Tcs** C++ code will require the **zeromq** and **czmq** libraries which can be found at zeromq.org.

The **Hetlib** python module only requires the **tcssubsystem** package and standard python modules.

The scripts that have been written using **Hetlib** require a number of additional packages that may not come with the standard python installation. The packages currently required are **mysql**, **numpy**, **pyephem**, **matplotlib**, and **six**. Additional packages may be required for future scripts.

2.3 Setting up your environment

Because the executable and libraries are not in one of the standard system locations you will need to modify your environment in order to let the operating system know where to find executable, libraries, and configuration files.

Note that we have a local install of Python 2.7. This was done because the Python 2.6 package available by default from Redhat is woefully behind the times.

If you use the bash shell, then you should include the following lines in your .bashrc file.:

```
#
# .bashrc setup for HetDex
#

#
# location of third-party and hetdex software executables
#
export HETDEX_DIR=/opt/het/hetdex

#
# location of src tree for Hetdex software source
#  (this is only required to build the software)
#
export HET_SRC_ROOT=$HETDEX_DIR/src/tcs/trunk

#
# modify environment variables to gain access to the system
#  (this is required to operate the software)
#
export LIBRARY_PATH=$HETDEX_DIR/lib:$HETDEX_DIR/epics/lib/linux-x86_64

export LD_LIBRARY_PATH=$HETDEX_DIR/lib:$HETDEX_DIR/epics/lib/linux-x86_64:$HETDEX_DIR/plugins/designe

export PATH=$HETDEX_DIR/Python-2.7/bin:$HETDEX_DIR/bin:$PATH

export PYTHONPATH=$HETDEX_DIR/lib/python2.7/site-packages:$HETDEX_DIR/lib64/python2.7/site-packages:$
```

If you use csh or tcsh, then include the following in your .cshrc file.:

```
#
# .cshrc set up for HetDex
#

#
# location of third-party and hetdex software executables
#
setenv HETDEX_DIR /opt/het/hetdex

#
# location of src tree for Hetdex software source
#  (this is only required to build the software)
#
setenv HET_SRC_ROOT $HETDEX_DIR/src/tcs/trunk

#
# for third-party software locations
#  (this is required to operate the software)
#
setenv LIBRARY_PATH $HETDEX_DIR/lib:$HETDEX_DIR/epics/lib/linux-x86_64

setenv LD_LIBRARY_PATH $HETDEX_DIR/lib:$HETDEX_DIR/epics/lib/linux-x86_64:$HETDEX_DIR/plugins/designe

setenv PATH $HETDEX_DIR/Python-2.7/bin:$HETDEX_DIR/bin:$PATH

setenv PYTHONPATH $HETDEX_DIR/lib/python2.7/site-packages:$HETDEX_DIR/lib64/python2.7/site-packages
```

2.4 Installing the scripting packages

Both the **Tcs Scripting** package and the **Hetlib** package may be installed with the following commands

```
cd $HET_SRC_ROOT/scripting
```

```
make install
```

This will install the package files into $prefix/lib/python2.7/site-packages and the scripts in $prefix/bin. This is normally done at Het as the user hetdex.

2.4.1 Installing the Tcs Scripting package only

The **Tcs Scripting** package is mix of python and C++ libraries. The package can be installed with the following commands,

```
cd $HET_SRC_ROOT/scripting
```

```
python setup.py install --prefix=$HETDEX_DIR
```

This will install the package files into $prefix/lib/python2.7/site-packages and the scripts in $prefix/bin. This is normally done at Het as the user hetdex.

2.4.2 Installing the Hetlib module and scripts only

Hetlib is a pure python package and can be installed with the command

```
cd $HET_SRC_ROOT/scripting
```

```
python hetlib_setup.py install --prefix=$HETDEX_DIR
```

This will install the module in $prefix/lib/python2.7/site-packages and the scripts in $prefix/bin. This is normally done at Het as the user hetdex.

2.5 Building the Documentation

The documentation is created through the Sphinx documentation system. The source files are located in `$HET_SRC_ROOT/scripting/docs`.

You can run `make` to get a list of available target formats. Normally the documentation is built by running `make html latexpdf` in the `./doc` directory. This will build the html pages and the PDF version of the document. The PDF file will be in `./_build/latex/HetPython.pdf` while the HTML files can be found in `./_build/html`. These files may be installed in the directories of your choice.

CHAPTER

THREE

TCS PYTHON INTERFACE

The **Tcs Python** Interface modules provide low-level access to the Tcs Control Systems. These are not a pure python modules since there are C++ modules from the Tcs system that are linked into the Python code.

Note that the modules in the **Tcs Python** Interface will be migrated to the **pytcs** package some time soon.

3.1 Tcs Python Modules

3.1.1 tcssubsystem.py

class `tcssubsystem.`**`TCSSubSystem`**(*name*, *system_route*)

The workhorse class `TCSSubSystem` is the only class defined in this file. `name` should be a unique string that will not conflict with any other named connection on the computer. `system_route` is the *Route* URL for the *Control System* you want to connect to. The URL's for a specific *Control System* can be found through the TCSNamed class defined in *TCSNamed.py*.

```
import tcssubsystem
import os, random
basename = os.path.basename(__file__) + str(random.random()) # a unique name

tracker = tcssubsystem.TCSSubSystem( basename, 'tcp://192.168.66.31:30400')
```

classmethod `tcssubsystem.`**`generic_handler_call`**()

The client object returned by will contact the *Control System* and generate class methods for all the *Handler* available in the *Control System*. For example, the tracker *Control System* has a *Handler* named `moveRTF` which is called with a set of RTF coordinates. Accessing this *Handler* through `tcssubsystem.py` is as simple as

```
tracker.moveRTF(x=0.0, y=0.0, z=0.0, theta=0.0, phi=0.0,
                rho=0.0, type='abs', vel='TRK_FAST')
```

The *Handler* description for a *Control System* can be found in the documentation for that specific *Control System*. [DEFINE LOCATION]

classmethod `tcssubsystem.`**`help`**(*method_name*)

This client object has a help function which will list the documentation for a single method if the optional parameter `handler_name` is given or for all commands if no command is given in the function call.

```
tracker.help(command='moveRTF')
  tracker::moveRTF                                                    (8182)

  Sends a move command in RTF coordinates. Sends with tmcs_mova_rtf or
  tmcs_movr_rtf command depending on the type parameter.
```

```
Required parameters are:

  phi<double> -
  rho<double> -
  theta<double> -
  type<string> -
  vel<string> -
  x<double> -
  y<double> -
  z<double> -

or tracker.help()
.
.
.
```

classmethod `tcssubsystem.`**`wait`**(*msgid*)

The wait() function will wait on a pending command given by `msgid` and blah, blah, blah. [Find out what this does!]

```
tracker.wait(msgid)
(this does something)
```

3.1.2 TCSNamed.py

class `TCSNamed.`**`TCSNamed`**(*name*, *named_route*)

The Tcs interface uses a number of network protocols, addresses, and ports numbers. Rather than making the users remember the URL's for *Route*, *Event-route*, and other configurable variables, the Tcs interface provides a standalone process knowns as `tcsnamed` that will respond to queries and provide the answers. See the `tcsnamed` documentation [DEFINE LOCATION] for further details

The `TCSNamed.py` interface provides a connection to the tcsnamed server and that will do query look ups. Of course, you must know the route to `tcsnamed` prior to calling it. The commonly used named_route is defined in the HET Interface package `tcsutils.py`, see *tcsutils.py*

```
import TCSNamed

named_route = 'tcp://192.168.66.99:30000'

n = TCSNamed.TCSNamed(named_route)
```

classmethod `TCSNamed.`**`lookup`**(*key*)

To use the named service the method `lookup` is provided. The key names that can be queried are defined in [DEFINE LOCATION]

n.lookup(key='tcs-route') 'tcp://192.168.66.31:30300'

3.1.3 TCSLog.py

class `TCSLog.`**`TCSLog`**(*name*, *logRelayRoute*, *namedRoute*)

The **Tcs Python** interface does not provide the full capability for python scripts to issue *Events* to other system. However, it does provide a mechanism to issue log *Events* of the type `pytcs.<sysname>.log_[debug|info|warn|error|fatal|alarm]` (where <sysname> is your unique name for the connection to the *logrelay-label*:. These *Events* can be monitored by other programs and will

be placed in the nightly database log. See the documentation for *Control System* and related *Events* [DEFINE LOCATION] for further details.

The class TCSLog requires a unique name and can take either the `logRelayRoute` or the `namedRoute`. It is not necessary to pass in both the `logRelayRoute` and the `namedRoute`. If the `logRelayRoute` is not give but the `namedRoute` is, then TCSLog will do a look up of `logRelayRoute` through *TCSNamed.py*:. If `logRelayRoute` is given, then TCSLog uses it directly and does not use the `namedRoute` at all.

classmethod `TCSLog.`**`log_`** (*fmt*, **args*)

The available methods are:

`TCSLog.`**`log_debug`** (*fmt*, **args*)

`TCSLog.`**`log_info`** (*fmt*, **args*)

`TCSLog.`**`log_warn`** (*fmt*, **args*)

`TCSLog.`**`log_error`** (*fmt*, **args*)

`TCSLog.`**`log_fatal`** (*fmt*, **args*)

`TCSLog.`**`log_alarm`** (*fmt*, **args*)

Where `fmt` and `args` are treated as a python format string and a list of arguments.

As an illustration of the use of this module this code sequence will issue a log_debug message

```
import TCSLog

named_route = 'tcp://192.168.66.99:30000'

l = TCSLog.TCSLog( 'my_random_name', namedRoute=named_route )

l.log_debug('This is test number {} of {}', 1, 2)
```

The result as shown by watching the log relay server output with the the *monitor* program is

```
> monitor -v -r

{ "time": "2017-08-08T16:42:41.108",
  "pytcs.my_random_name.log_debug": {
      "file": "<stdin>",
      "line": 1
      "function":     "<module>",
      "message":      "This is test number 1 of 2",
      "__system":     "pytcs",
      "__source":     "my_random_name",
      "__key":        "log_debug",
      "__data_time":  "1502210561.108609033",
      "__wire_time":  "1502210561.108689667",
      "__data":       "false"
} }
```

3.1.4 tcsdb.py

class `tcsdb.`**`tcsdb`** (*dbname*)

Events that are generated by the *Control System* and by scripts are recorded in an sqlite3 database. The tcsdb module provides an interface to the daily sqlite3 database or databases generated by the `monitor` script.

```
import tcsdb

d = tcsdb.tcsdb('/opt/het/hetdex/logs/Mon/2017/08/20170802T180005.db')
```

The start/stop times of the database in Unix seconds can be obtained with

classmethod `tcsdb.`**`start`**`()`

classmethod `tcsdb.`**`stop`**`()`

for example

```
d.start()
1501624142.5000026
d.stop()
1501804799.9950001
```

classmethod `tcsdb.`**`events`**`()`

The events method returns a list of all event names seen in the database. Events have the naming convention `system.source.key`. Note that they are not listed in alphabetical order. However, this is a list so you may invoke the sort() method if you want alphabetical ordering.

```
from pprint import pprint

pprint( d.events())
['tcs.tracker.position',
 'tracker.tmcs.status',
 'tcs.root.ra_dec',
 'pfip.VirusMonitor2.status',
 'apc.VEncl2Misc.VEncl2MiscPDU_status',
 'apc.ControlRoom.ControlRoomPDU_status',
 'log-relay.receiver.heartbeat',
 'legacy.receiver.heartbeat',
 'apc.PFIP.PFIPPDU_status',
 'tcs.root.ra_dec',
 ...
]
```

classmethod `tcsdb.`**`keys`**`(`*'event_name'*`)`

This method will provide a list of the keywords for a specific event given in the argument.

```
pprint(d.keys('tcs.root.ra_dec'))
['__data',
 '__data_time',
 '__key',
 '__pid',
 '__source',
 '__system',
 '__wire_time',
 'az',
 'correction.emp.x',
 'correction.emp.y',
 'correction.sky.x',
 'correction.sky.y',
 'dec',
 'dec_2000',
 'dec_offset',
 'el',
 'hour_angle',
```

```
 'itf.phi',
 'itf.rho',
 'itf.t',
 'itf.theta',
 'itf.x',
 'itf.y',
 'itf.z',
 'lst',
 'offset.focus.w',
 'offset.rho',
 'offset.sky.x',
 'offset.sky.y',
 'offset.tiptilt.phi',
 'offset.tiptilt.theta',
 'parallactic_angle',
 'ra',
 'ra_2000',
 'ra_offset',
 'setup',
 'zenith_distance']
```

classmethod `system.source.key.`**`keword`**`()`

The actual values of a particular keyword can be found by using the constructed method name based on the event/keyword name. These functions are of the form `system.source.key.keword()` and they return two lists; the first is a list of the time stamps of the events and the second being the values of the event/keyword for the time stamp. The following example illustrates this. Note that the daily database is quite large and it may take a few minutes for the database to be read.

```
# Note that this can take a few minutes on a slow machine
ra_t, ra = d.tcs.root.ra_dec.ra()
len(ra_t)
418498
len(ra)
418498

pprint(ra_t)
[1501696806.765
 1501696806.968
 1501696807.170
 1501696807.373
 1501696807.575
 ...
]

pprint(ra)
[5.709843
 5.709899
 5.709956
 5.710012
 5.710069
 ...
]
```

It is also possible to return a full event and do the parsing yourself.

```
radec_t, radec = d.tcs.root.ra_dec()
len(radec)
426640
```

```
# Note that there are more ra_dec events then there are ra keys
# shown above. Events may have missing keys and if you do your own
# parsing you need to be aware of this

pprint(radec)
['__data': 'false',
 '__data_time': 1501696806.765,
 '__key': 'ra_dec',
 '__pid': 57480,
 '__source': 'root',
 '__system': 'tcs',
 '__wire_time': 1501696806.782876,
 'az': 285.551159,
 'correction': {'emp': {'x': 0, 'y': 0}, 'sky': {'x': 0, 'y': 0}},
 'dec': 34.051544,
 'dec_2000': 34.042592,
 'dec_offset': 0,
 'el': 55.079064,
 'hour_angle': 2.125675,
 'itf': {'phi': -0.001449,
         'rho': -0.040128,
         't': 1501696806.765,
         'theta': 0.220831,
         'x': 11.001893,
         'y': -1836.970907,
         'z': -4.965364},
 'lst': 7.835518,
 'offset': {'focus': {'w': -1.95},
            'rho': 0,
            'sky': {'x': 0, 'y': 0},
            'tiptilt': {'phi': 0, 'theta': 0}},
 'parallactic_angle': 262.910534,
 'ra': 5.709843,
 'ra_2000': 5.690575,
 'ra_offset': 0,
 'setup': 'false',
 'zenith_distance': 34.920936},
{'__data': 'false',
 '__data_time': 1501696806.968,
 '__key': 'ra_dec',
 '__pid': 57480,
 '__source': 'root',
 '__system': 'tcs',
 '__wire_time': 1501696806.9851422,
 'az': 285.551159,
 'correction': {'emp': {'x': 0, 'y': 0}, 'sky': {'x': 0, 'y': 0}},
 'dec': 34.051544,
 'dec_2000': 34.042593,
 'dec_offset': 0,
 'el': 55.079064,
 'hour_angle': 2.125675,
 'itf': {'phi': -0.001449,
         'rho': -0.040128,
         't': 1501696806.968,
        'theta': 0.220831,
         'x': 11.001792,
         'y': -1836.970806,
         'z': -4.965364},
```

```
 'lst': 7.835574,
 'offset': {'focus': {'w': -1.95},
            'rho': 0,
            'sky': {'x': 0, 'y': 0},
            'tiptilt': {'phi': 0, 'theta': 0}},
 'parallactic_angle': 262.910534,
 'ra': 5.709899,
 'ra_2000': 5.690631,
 'ra_offset': 0,
 'setup': 'false',
 'zenith_distance': 34.920936},
 ...]

ra = []
for r in radec:
  try:
    r.append(r['ra'])
  except:
    pass
```

Keys and events are more fully described in [DEFINE LOCATION]

CHAPTER

FOUR

HETLIB PYTHON INTERFACE

The module, **hetlib**, defines useful functions and the basic configuration information. The **hetlib** package imports the low level packages *tcssubsystem.py* and provides wrappers for functionality so it is not necceasary to explicitly import the th low level package. You only need to have the line

```
import hetlib
```

in your file to get the full functionality of these packages.

Contants defined in hetlib:

`hetlib.`**named_route**
The default URL for the *Control System* **tcsnamed**, defined as 'tcp://192.168.66.99:30000'

A number of site specific constants are defined:

`hetlib.`**Latitude_deg**
The latitude of the Het is 30.681436 degrees

`hetlib.`**Latitude_rad**
Defined as `radians(Latitude_deg)`

`hetlib.`**LongitudeWest_deg**
The longitude of the het is 104.014742 degrees west

`hetlib.`**LongitudeWest_rad**
Defined as `radians(LongitudeWest_deg)`

`hetlib.`**LongitudeEast_deg**
Defined as `360.0 - LongitudeWest_deg`

`hetlib.`**LongitudeEast_rad**
Defined as `radians(LongitudeEast_deg)`

`hetlib.`**Altitude_deg**
The elevation alitude of the Het primary mirror is 55.055223 degrees

`hetlib.`**Altitude_rad**
Defined as `radians(Altitude_deg)`

`hetlib.`**Height_meters**
The height of the Het telescope center of the primary is 2003.0 meters

`hetlib.`**Fs_meters**
The Focal length of the Het primary mirror is 13.37884180 meters

`hetlib.`**Fs_millimeters**
Defined as `Fs_meters * 1000.0`

hetlib.**het_P**
Defined as `cos(Altitude_rad) * cos(Latitude_rad)` radians

hetlib.**het_Q**
Defined as `sin(Altitude_rad) * sin(Latitude_rad)` radians

hetlib.**BetaLimit_deg**
The angular limit for trajectories is 8.5 degrees

hetlib.**BetaLimit_rad**
Defined as `radians(BetaLimit_deg)`

4.1 Hetlib Python Modules

4.1.1 tcsutils.py

`tcsutils` contains the workhorse functions for your python scripts. The principal one for client programs is `start_clients()`.

The `tcsutils` module provides the following constant.

tcsutils.**named_route**
The default URI for the *tcs_named-label*, defined as 'tcp://192.168.66.99:30000'

The following functions are defined:

tcsutils.**start_clients**(*tcs*, *tracker*, *legacy*, *pas*, *pfip*, *apc*, *lrs2*, *vdas*, *named*, *verbose*)

Start one or a number of clients links. The default values for the system names in ''start_client()' are False. Set them to be true to create a handle (as described in *tcssubsystem.py*. The function returns a dictionary of handles to Control Systems.

```
client_dict = start_clients(tcs=True, virus=True)
tcsClient = client_dict['tcs']
virusClient = client_dict['virus']
```

4.1.2 hetutils.py

The hetutils.py package provides functions to do calculations specific to the Hobby Eberly Telescope at McDonald Observatory. These functions provide information about telescope pointing as well as tracker information.

hetutils.**Att**(*az_deg*, *dec_deg*)
Calculate the total possible track time for a give azimuth and declination. Input arguments are in degrees. The function returns the track time in minutes or 0 if no trajectory is available.

hetutils.**BestAzimuth**(*dec_deg*)
This function calculates the best azimuth for a given declination. The function returns an azimuth only on the east side. If you want the west azimuth, simply add 180 degrees to the return value. The input declination is in degrees, The return value is the azimuth in degrees or None is the declinination is not visible at HET.

hetutils.**Ho**(*azimuth_deg*)
Calculate the telecentric hour angle for a given azimuth. Negative values are east hour angles. The input azimuth is in degrees. Returns the hour angle in degrees.

hetutils.**Pa**(*azimuth_deg*)
Calculate the zero-crossing paralactic angle as a function of azimuth. The input azimuth is in degrees. Returns the paralactic angle in degrees.

hetutils.**Tde**(*azimuth_deg*)
Calculate the telecentric declination as a function of azimuth. The input azimuth is in degrees. Returns the declination in degrees or None if the input is not between 0 <= az < 360.0

hetutils.**clipToCircle_deg**(*x*)
Return the value (dX mod 360). This has the result of clipping the value between 0 <= dR < 360.0.

Inputs dX - the value you want clipped

Outputs none

Return (dX mod 360)

hetutils.**clipToCircle_rad**(*x*)
Return the value (dX mod 2PI). This has the result of clipping the value between 0 <= dR < 2PI.

Inputs dX - the value you want clipped

Outputs none

Return (dX mod 2PI)

hetutils.**degrees_to_hours**(*deg*)
Convert decimal degrees to decimal hours.

hetutils.**hours_to_degrees**(*hr*)
Convert decimal hours to decimal degrees.

hetutils.**hours_to_radians**(*hr*)
Convert decimal hours to decimal radians.

hetutils.**modulo**(*x*, *y*)
A general purpose modulo (x mod y) function.

Definition of 'mod' x mod y = x - y * floor(x/y) for y != 0 x mod 0 = x

if 0 < y then 0 <= (x mod y) < y if 0 > y then 0 >= (x mod y) > y

hetutils.**radians_to_hours**(*rad*)
Convert decimal radians to decimal hours.

A number of constants are defined in hetutils and are duplicated in hetlib. Their use from hetutils is deprecated and they will be removed from hetutils at a future date.

Latitude_deg = 30.681436

Latitude_rad = radians(Latitude_deg)

LongitudeWest_deg = 104.014742

LongitudeWest_rad = radians(LongitudeWest_deg)

LongitudeEast_deg = 360.0 - LongitudeWest_deg

LongitudeEast_rad = radians(LongitudeEast_deg)

Altitude_deg = 55.055223

Altitude_rad = radians(Altitude_deg)

Height_meters = 2003.0

Fs_meters = 13.37884180

Fs_millimeters = Fs_meters * 1000.0

het_P = cos(Altitude_rad) * cos(Latitude_rad) # radians

het_Q = sin(Altitude_rad) * sin(Latitude_rad) # radians

limit on the total angular travel of the tracker

BetaLimit_deg = 8.5

BetaLimit_rad = radians(BetaLimit_deg)

4.1.3 hetTime.py

Useful (or not) tools for dealing with Het time in its different formations.

Some definitions:

unix time – the number of seconds since the epoch. The epoch is defined as 00:00:00 UTC, January 1, 1970. This number may be a floating point value with factional seconds.

index time – The tracker's definition of the time of day. Index time

is reset to 0.0 at 1800 UT hours or 12 noon CST (1300 CDT). There is no information contained here that might indicate which day this index time occurred on. This number may be a floating point value.

0000 hours UT == 00000 secs UT == 0600 hours index == 21600 seconds index

0600 hours UT == 21600 secs UT == 1200 hours index == 43200 seconds index

1200 hours UT == 43200 secs UT == 1800 hours index == 64800 seconds index

1800 hours UT == 64800 secs UT == 2400 hours index == 86400 seconds index (0000)

0. 2400 hours UT == 86400 secs UT == 0600 hours index == 21600 seconds index

ISO time – an ISO-8601 string of the format YYYY-MM-DDThh:mm:ss.mmmmmm, where YYYY is the four digit year, MM is the two digit month (starts at 01), DD is the two digit day (starts at 01), hh is the two digit hour, mm is the two digit minute, ss is the two digit second, and mmmmmm is the fractional part of the second.

datetime – a python datetime.datetime() object. This is the intermediate object through which the time conversion routines work. It is also a value which Python can work with. See the documentation for the datetime package within Python.

Note that the timezone information in the datetime.datetime() object is explicitly set to None and the assumption is that all times are UT. This may change in the future if we need to compare local time with HET times. In this case the timezone will be explicitly set to UTC.

Future Plans.

Convert to object hetTime(), subclass of datetime, which takes an index time, unix time, datetime object or ISO string at instantion. added functions hetTime.indexTime(), hetTime.unixTime(), hetTime.ISOtime() to return time in different format. ISOtime and unixTime contain year/mon/day information but index time does not. Use the current year/mon/day unless there is a specified year/mon/day at instantation.

e.g. hetTime(index|unix|iso, year=now.year, mon=year.mon, day=now.day)

The following functions are defined:

hetTime.**DT_To_ISO**(*dt*)

Convert a datatime.datetime() object to a ISO time string.

input: datetime object

output: ISO-8601 format string

hetTime.**DT_To_Index**(*dt*)
Convert datetime.datetime() to an index time.

input: datetime.datetime()

output: float

Note that we lose the date information from the datetime() object.

hetTime.**DT_To_Unix**(*dt*)
Convert a datetime.datetime() object to a Unix time value.

input: datetime.datetime()

output: Unix time value

hetTime.**ISO_To_DT**(*isoStr*)
Convert an ISO-8601 time string to a datetime.datetime() object.

input: ISO-8601 time string

output: datetime object

hetTime.**ISO_To_Index**(*isoStr*)
Convert a ISO string to an Index time.

input: ISO-8601 format string

output: Index time

Note that we lose the date information from the ISO string.

hetTime.**ISO_To_Unix**(*isoStr*)
Convert an ISO time string to a Unix time value.

input: ISO-8601 format string

output: Unix time value

hetTime.**Index_To_DT**(*ind*)
Convert an Index time into a datetime.datetime() object.

input: Index time

output: datetime.datetime()

Note that Index time has no date information

hetTime.**Index_To_ISO**(*idxtime*)
Convert an Index time to a ISO time string.

input: Index time

output: ISO-8601 format string

Note that Index time has no date information

hetTime.**Index_To_Unix**(*idxtime*)
Convert an Index time to a Unix time value

input: Index time

output: Unix time value

Note that Index time has no date information

hetTime.**Unix_To_DT**(*utime*)
Convert a Unix time value to a datatime.datetime) object. This function duplicates datetime.fromtimestamp() and is included only for consistence with the other time functions.

input: Unix time value

output: datetime.datetime()

hetTime.**Unix_To_ISO**(*utime*)
Convert a Unix time value to an ISO time string.

input: Unix time value

output: ISO-8601 format string

hetTime.**Unix_To_Index**(*utime*)
Convert a Unix time value to an Index time.

input: datetime object

output: ISO-8601 format string

Note that we lose the date information from the Unix time value.

The commands available are shown in the following matrix

From/To	DT	ISO	Index	Unix
DT		DT_To_ISO()	DT_To_Index()	DT_To_Unix()
ISO	ISO_To_DT()		ISO_To_Index()	ISO_To_Unix()
Index	Index_To_DT()	Index_To_ISO()		Index_To_Unix()
Unix	Unix_To_DT()	Unix_To_ISO()	Unix_To_Index()	

4.1.4 Points.py

This file contains the list `Points` which consists of of 97 dictionary defining points in ITF coordinates that lie on the focal sphere. The dictionarys are defined as

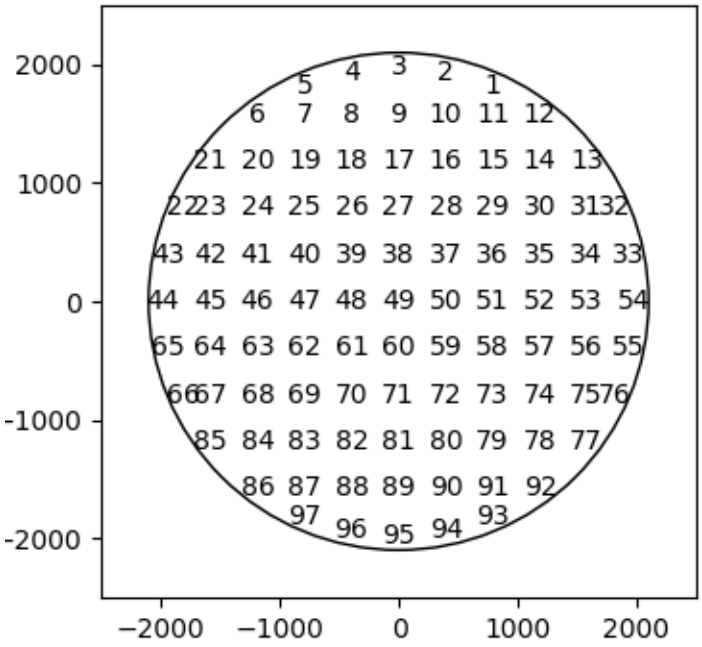
```
Points = [
{'Index':  1,
 'X':  790.877147,
 'Y': 1812.127196,
 'Z':  146.930811,
 'Theta': 7.799501970,
 'Phi':   3.421093209,
 'Rho':   0.000000000,
 'Beta':  8.500000000},
 ...
 ...
 ...
 {'Index': 97,
  'X': -790.877147,
  'Y':-1812.127196,
  'Z':  146.930811,
  'Theta': -7.799501970,
  'Phi':   -3.421093209,
  'Rho':    0.000000000, '
  'Beta':   8.500000000},
 ]
```

The parameters `X`, `Y`, and `Z` are given in millimeter while `Theta`, `Phi`, `Rho`, and `Beta` are given in units of degrees

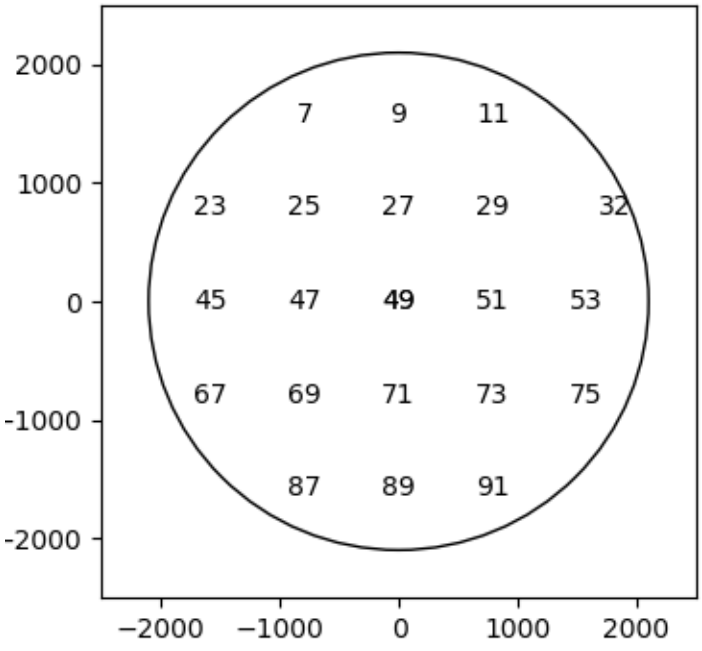
These points are primarily used in the program mountTest (DEFINE LINK HERE) but can be used anywhere. The points were calculated using a tracker sphere of ????? mm.

There are a number of pre-defined lists of points given in the dictionary `knownLists`. The dictionary contains keys of the form `'List_97'` and the value of such a key is a list of the form [1,2,3,4,5,..95,96,97]. Note that the number given in the list is one more than the entry value in Points.

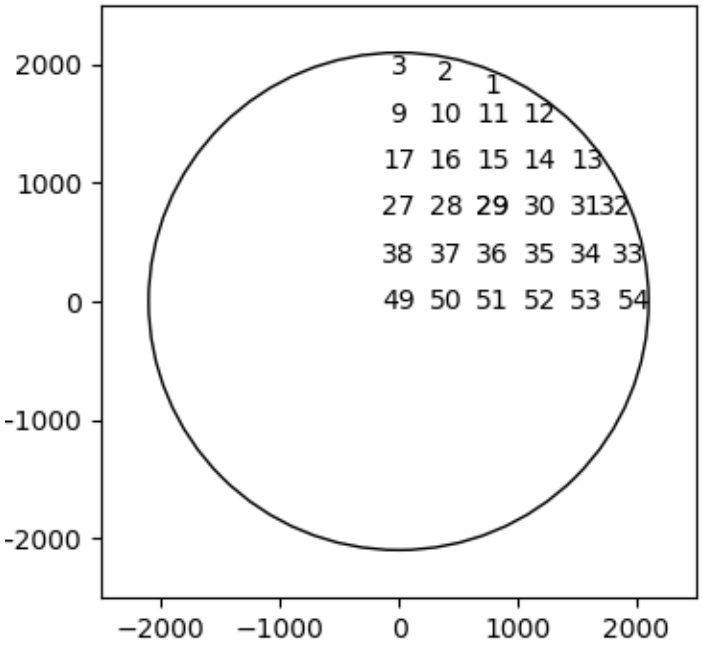
'List_97' is the array of all points,



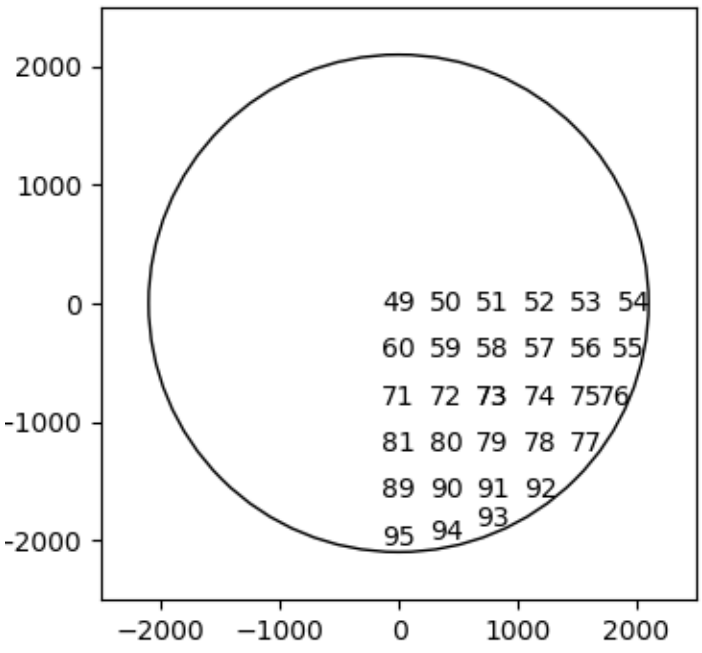
'List_21' is the sparse 21 point array,



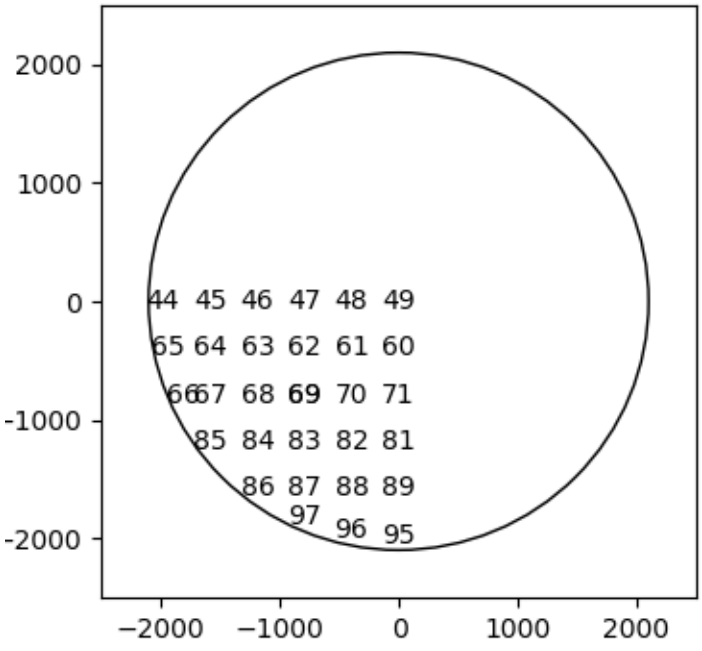
'List_Q1' is the array of all points in the upper right quadrant,



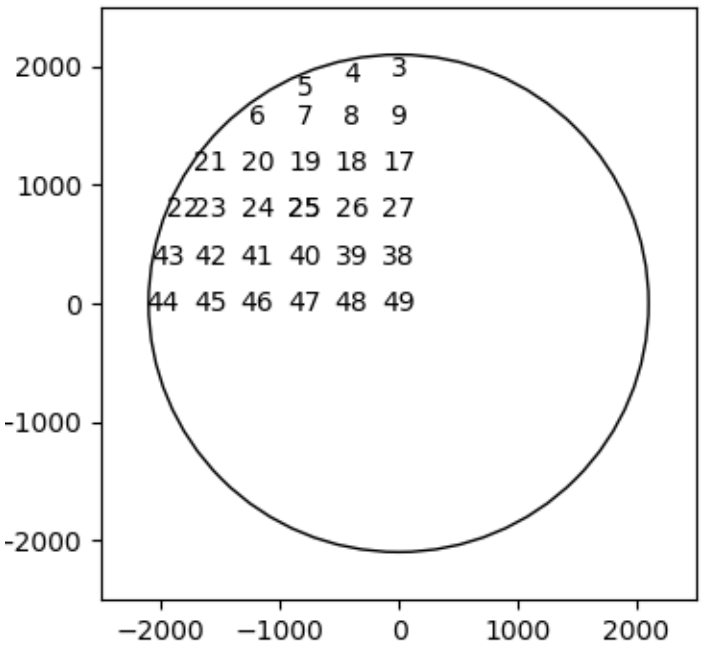
'List_Q2' is the array of all points in the lower right quadrant,



'List_Q3' is the array of all points in the lower left quadrant,



'List_Q4' is the array of all points in the upper left quadrant,


2000
1000
0
-1000
-2000
−2000
−1000
0
1000
2000
5
4
3
6
7
8
9
21
20
19
18
17
22
23
24
25
26
27
43
42
41
40
39
38
44
45
46
47
48
49

CHAPTER

FIVE

USING THE SCRIPTS

The scripts that I have written are in pure python. They are used to do various things with the *Control System* and clients. Since they read the `tcsutils.py` files you don't need to know the URL's for the *Routes_def-label* and *Event_Routes_def-label*. All scripts should be located in `/opt/het/hetdex/bin`. These scripts utilize the python interface described in Chapter 3 *Hetlib Python Interface*.

5.1 User Scripts

5.1.1 chksys

This bash shell script searches the system process table and prints the lines associated with the control systems to the standard output. At this time it is a shell script with no command line flags. There is no general way to search for control systems so this script simply searchs by name. If you create a new control system, then you will have to modifiy this script.

5.1.2 getposition

Usage: getposition [options]

Get the position of the tracker and print all the coordinates, ITF, RTF, and TSF

Options:

```
-h, --help  show this help message and exit
```

Example:

```
$ getposition
{u'__ack': u'false',
u'__async': u'false',
u'__data': u'false',
u'__data_time': u'1502812057.453914950',
u'__done': u'true',
u'__effective_id': 262987,
u'__error': u'false',
u'__handler': u'tracker_position',
u'__origin': u'tcsutils.pyc0.530180389939_tcs107451.htcs.107451.0327b23c6->tcp://192.168.66.31:30300
u'__original_id': 2,
u'__pid': 121862,
u'__wire_time': u'1502812057.453915299',
u'itf.phi': 0.047562,
u'itf.rho': -0.039519,
```

```
u'itf.theta': 0.235037,
u'itf.x': 1824.1775,
u'itf.y': -1837.147486,
u'itf.z': -2.73196,
u'rtf.phi': 0,
u'rtf.rho': -0.039519,
u'rtf.theta': 3e-06,
u'rtf.x': 1799.1873,
u'rtf.y': -1799.4097,
u'rtf.z': -3e-05,
u'time': u'1502812057.265000000',
u'trajectory': u'false',
u'tsf.h1': -0.0001,
u'tsf.h2': 0,
u'tsf.h3': 0,
u'tsf.h4': 0,
u'tsf.h5': 0.0001,
u'tsf.h6': -0.0001,
u'tsf.lx': 1799.1878,
u'tsf.rho': -0.0395,
u'tsf.ux': 1799.1868,
u'tsf.y': -1799.4097}
```

5.1.3 getSpectTemperatures

Usage: getSpectTemperatures [options]

Get the spectrograph hardware status.

Options:

```
-h, --help            show this help message and exit
-v, --version         show program's version number and exit
-o FILE, --output FILE
                      append output to FILE
-l, --lrs2, --ldas    get LRS2 temperatures
-V, --virus, --vdas   get Virus temperatures
-s sort_flag, --sort sort_flag
                      sort_flag must be one of ['IFU', 'SSA', 'SID', 'CID',
                      'MUX', 'IFU', 'HIGHCCD', 'LOWCCD', 'HIGHCRYO',
                      'LOWCRYO'] (default IFUSLOT)
-p print_options, --print print_options
                      print_options, must one or more of ['ALL', 'TEMPS',
                      'CCDTEMPS', 'CRYOTEMPS', 'CRYOPRESS', 'HEATER',
                      'TROUBLE'] (default TEMPS)
--no-update           do not ask for a hardware update, just use the last
                      value
--no-header           Don't print the header string
--print-commands      print the set_ccd_temp() commands
```

Example:

```
 $ getSpectTemperatures -Vl -s SID
 2017-08-15T19:07:13       Cryo    Left  Right    Left  Right   Left Right
    IFU SSA SID CID MUX    Temp   SetPt  SetPt    Temp   Temp  Volts Volts
 1  093 157 008 040 007  -180.0  -105.0 -105.0  -105.0 -105.0  0.848 0.644
 2  106 136 012 103 005  -173.6  -105.0 -105.0  -105.0 -105.0  0.941 0.749
 3  103 152 013 061 007  -179.9  -110.0 -105.0  -110.0 -105.0  0.822 0.754
```

```
 4  104 153 016 055 007  -178.7  -110.0 -110.0  -110.0 -110.0  0.801 0.782
 5  086 135 017 026 005  -174.3  -110.0 -110.0  -110.0 -110.0  0.677 1.138
 6  095 158 020 065 007  -177.2  -100.0 -105.0  -100.0 -105.0  0.613 0.653
 7  083 137 024 030 005  -179.2  -110.0 -110.0  -110.0 -110.0  0.983 0.901
 8  076 154 025 056 007  -175.7  -110.0 -110.0  -110.0 -110.0  0.785 0.814
 9  085 122 027 090 004  -170.9  -100.0 -105.0  -100.0 -105.0  0.956 0.569
10  094 155 032 098 007  -175.9   -90.0 -100.0   -77.3 -100.0  0.000 0.603
11  073 151 037 059 006  -179.8  -110.0 -105.0  -110.0 -105.0  0.606 0.791
12  096 134 038 062 005  -184.6  -100.0 -110.0  -100.0 -110.0  0.722 0.760
13  084 133 041 041 003  -179.1  -105.0 -105.0  -105.0 -105.0  0.984 1.154
14  075 132 047 052 005  -181.2  -100.0 -105.0  -100.0 -105.0  0.724 0.766
15  105 138 051 080 005  -172.1   -95.0 -100.0   -95.0 -100.0  0.842 0.522
16  044 238 202 038 002  -177.2  -105.0 -105.0  -105.0 -105.0  1.320 1.675
17  045 248 305 078 001  -173.2  -105.0 -105.0  -105.0 -105.0  1.251 1.174
18  074 131 307 105 005  -174.2  -100.0 -100.0  -100.0 -100.0  0.874 0.622
19  066 121 502 100 000  -200.6  -110.0 -110.0  -110.0 -110.0  1.050 1.190
20  056 111 503 069 000  -182.7  -110.0 -110.0  -110.0 -110.0  0.536 1.183
```

5.1.4 getUptime

Usage: getUptime [options]

Show the server uptime in decimal days, decimal hours, decimal minutes, decimal seconds, or days, hours, minutes, seconds as ddd:hh:mm:ss.

Options:

```
-h, --help            show this help message and exit
-v, --version         show program's version number and exit
--verbose             print useful(?) information (does nothing at this time)
--pretty-print {dd,dh,dm,ds,dhms} pretty print the output (default dhms)
-A, --all             uptime for all the systems,
-a, --apc             uptime for the apc system,
-L, --legacy          uptime for the legacy system,
-l, --lrs2            uptime for the lrs2 system,
-n, --named           uptime for the named server,
-p, --pas             uptime for the pas server,
-P, --pfip            uptime for the pfip server,
-t, --tracker         uptime for the tracker server,
-T, --tcs             uptime for the tcs server,
-V, --virus           uptime for the virus system
```

Example:

```
$ getUptime -V
virus uptime: 003:21:54:25 ddd:hh:mm:ss

$ getUptime -V -l
virus uptime: 003:21:54:25 ddd:hh:mm:ss
lrs2 uptime: 003:21:04:57 ddd:hh:mm:ss

$ getUptime -T --pretty-print dd
tcs uptime: 2.894 days
```

5.1.5 hettime

Usage: hettime [options] [time]

Convert between various notions of time at HET. 'time' may be either index time, an ISO-8601 string, or unixtime. The return values are the input time converted to all three formats. If no time is given on the command line, use the current time.

Options:

```
--version   show program's version number and exit
-h, --help  show this help message and exit
```

Example:

```
# Index time
$ hettime 77145
77145.0 55545.0 1970-01-01T15:25:45

# ISO time string
$ hettime 2017-08-15T15:25:45
77145.0 1502810745.0 2017-08-15T15:25:45

# Unix time
$ hettime 1502810745.0
77145.0 1502810745.0 2017-08-15T15:25:45
```

5.1.6 monitor

Usage: monitor [options]

Monitor one or several processes within Tcs.

Options:

```
--version             show program's version number and exit
-h, --help            show this help message and exit
-t, --tracker         catch the tracker server output.
-T, --tcs             catch the tcs server output.
-p, --pas             catch the pas server output.
-P, --pfip            catcj the pfip server output.
-l, --logger          catch the log server output
-L, --legacy          catch the legacy system output.
-g, --gui             catch the gui system output.
-v, --verbose         Send output to stdout as well as the output file.
-o FILE, --output=FILE
                      Send output to FILE (default
                      ./data/Mon/yyyymmddThhmmss_mon.log)
--system-filter=SYSTEM_FILTER
                      Specify filters on the system..
--source-filter=SOURCE_FILTER
                      Specify filters on the source
--key-filter=KEY_FILTER
                      Specify filters on the key.
```

Description:

The monitor program has the almost the same flags as startsys. I usually start it separately from the other systems. You can specify the filters us usual but you should probably quote the string values. This is a python programs so the command line parser is slightly different from the C++ version in the control systems.

Example:

A typical usage would be `monitor -t -T -L -p -o 20141119a_mon.txt` which will listen to the tracker_server, tcs_server, legacyServer, and pasServer event routes and place the output in 20141119a_mon.txt. Not all the available flags in the tcs_monitor are available in the python version. So if you want to specify alternate routes or configure files you will need to use tcs_monitor directly.

5.1.7 moveITF

Usage: moveITF [options] left|right|home|zero|x y z rho theta phi

> Move to the requested ITF position. The position may be one of the pre-defined positions or six ITF coordinates. The default velocity is track. If both track and slew speeds are set on the command line, then the script will exit since it does not know what you want. Verbose mode will print useful information about what is happening. Quiet mode will print nothing at all, not even errors. The default is to not print the useful information but to print error messages.
>
> left == (-1800, -1800, 0, 0, 0, 0)
>
> right = (+1800, -1800, 0, 0, 0, 0)
>
> home = (0, -1800, 0, 0, 0, 0)
>
> zero = (0, 0, 0, 0, 0, 0)

Options:

```
-h, --help     show this help message and exit
-t, --track    move at tracker speed
-s, --slew     move at slew speed (default)
-S, --slow     move at slow speed (default Fast)
-v, --verbose  display more useful (or not) information about progress
-q, --quiet    do not print any information
```

5.1.8 moveRTF

Usage: moveRTF [options] left|right|home|zero|x y z rho theta phi

> Move to the requested RTF position. The position may be one of the pre-defined positions or six RTF coordinates. The default velocity is track. If both track and slew speeds are set on the command line, then the script will exit since it does not know what you want. Verbose mode will print useful information about what is happening. Quiet mode will print nothing at all, not even errors. The default is to not print the useful information but to print error messages.
>
> left == (-1800, -1800, 0, 0, 0, 0)
>
> right = (+1800, -1800, 0, 0, 0, 0)
>
> home = (0, -1800, 0, 0, 0, 0)
>
> zero = (0, 0, 0, 0, 0, 0)

Options:

```
-h, --help     show this help message and exit
-t, --track    move at tracker speed
-s, --slew     move at slew speed (default)
-S, --slow     move at slow speed (default Fast)
-v, --verbose  display more useful (or not) information about progress
-q, --quiet    do not print any information
```

5.1.9 moveTSF

Usage: moveTSF [options] left|right|home|zero|ux lx y rho h1 h2 h3 h4 h5 h6

Move to the requested TSF position. The position may be one of the pre-defined positions or ten TSF coordinates. The default velocity is track. If both track and slew speeds are set on the command line, then the script will exit since it does not know what you want. Verbose mode will print useful information about what is happening. Quiet mode will print nothing at all, not even errors. The default is to not print the useful information but to print error messages.

left == (-1800, -1800, -1800, 0, 0, 0, 0, 0, 0, 0)

right = (+1800, +1800, -1800, 0, 0, 0, 0, 0, 0, 0)

home = (0, 0, -1800, 0, 0, 0, 0, 0, 0, 0)

zero = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Options:

```
-h, --help     show this help message and exit
-t, --track    move at tracker speed
-s, --slew     move at slew speed (default)
-S, --slow     move at slow speed (default Fast)
-v, --verbose  display more useful (or not) information about progress
-q, --quiet    do not print any information
```

5.1.10 startsys

This has been replaced by init scripts run by the `service` command. The command is deprecated.

The `startsys` command will start the various control systems depending on the command line flags. These controls systems are started with the standard command line flags defined in `tcsconfig.py` and the default configuration file in `/opt/het/hetdex/etc/conf`. If the `-v` flag is also given then the standard output of the control systems is sent to the screen; otherwise the output is sent to `/dev/null`. The script finishes by sleeping within a while loop. This allows the user to interupt the script and thereby kill all the control system there were started. (This works mostly.)

Usage: startsys [options]

Set up and start processes involved in the Tcs system.

Options:

```
--version             show program's version number and exit
-h, --help            show this help message and exit
-x, --no-tmcs         no TMCS hardware available, use simulator if starting
                      tracker.
-t, --tracker         start the tracker server.
-T, --tcs             start the tcs server.
-p, --pas             start the pas server.
-P, --pfip            start the pfip server.
-l, --logger          turn on logging and start the log server
-L, --legacy          start the legacy system.
-m, --monitor         start the monitor system.
-g, --gui             start the gui system.
--plot                start the plotters
-v, --verbose         Send output to stdout.
--tap                 Turn on message tapping for all systems
--disable-mount-model
```

```
                      Turn off TCS mount model.
--valgrind=FILE       Enable valgrind for a particular system.  The log file
                      is written to <system>-valgrind.log in the current
                      directory.
```

5.1.11 syscmd

Usage: syscmd option command

Send a synchronous command to a particular control system

If no command is specified, then the help() command is executed. This results in a list of available handlers for the control system. To find out specific information about a particular command, use the extended form of the help command. For example, to find out the detail of the Tcs go_next() command use

syscmd -T "help(command='go_next')"

The return value of syscmd is the return value of the command.

Options:

```
--version      show program's version number and exit
-h, --help     show this help message and exit
-t, --tracker  send a command to the tracker server (default help())
-T, --tcs      send a command to the tcs server (default help()).
-p, --pas      send a command to the pas server (default help())
-P, --pfip     send a command to the pfip server (default help()).
-L, --legacy   send a command to the legacy system (default help()).
-g, --gui      send a command to the gui system (default help()).
-v, --verbose  Send output to stdout.
```

Examples:

```
$ syscmd -l 'get_heater_voltage()'
{u'__ack': u'false',
u'__async': u'false',
u'__data': u'false',
u'__data_time': u'1502813354.967412900',
u'__done': u'true',
u'__effective_id': 6574,
u'__error': u'false',
u'__handler': u'get_heater_voltage',
u'__origin': u'tcsutils.pyc0.0929679765573_lrs2108337.htcs.108337.0327b23c6->tcp://192.168.66.15:3130
u'__original_id': 2,
u'__pid': 29135,
u'__wire_time': u'1502813354.967414133',
u'lrs2.mux.000.spec.502.cntl.100.ccd.left.15609.voltage': 1.046802,
u'lrs2.mux.000.spec.502.cntl.100.ccd.right.15612.voltage': 1.185606,
u'lrs2.mux.000.spec.503.cntl.069.ccd.left.21162.voltage': 0.53953,
u'lrs2.mux.000.spec.503.cntl.069.ccd.right.21090.voltage': 1.18294}
```

5.1.12 tap

Usage: tap [options]

Send the tap command to control systems.

Options:

```
--version      show program's version number and exit
-h, --help     show this help message and exit
-t, --tracker  tap the tracker server.
-T, --tcs      tap the tcs server.
-p, --pas      tap the pas server.
-P, --pfip     tap the pfip server.
-L, --legacy   tap the legacy system.
-g, --gui      tap the gui system.
-v, --verbose  Send output to stdout.
```

5.1.13 trajectory

Usage: trajectory [options] [- - ra_hr dec_deg]

Set up and start a trajectory

Options:

```
--version             show program's version number and exit
-h, --help            show this help message and exit

Object trajectories:
  -s SATELLITE_NAME, --satellite=SATELLITE_NAME
                      generate a trajectory from a satellite..
  -S FILE, --sat-data=FILE
                      use FILE instead of /opt/het/hetdex/etc/point/geo.txt
                      for satellite information.
  -f FILE, --file=FILE
                      generate a trajectory from FILE.
  -e FILE, --ephemeris=FILE
                      generate a trajectory from the ephemeris file FILE.

Engineering trajectories:
  -c, --current       generate a trajectory for lst at the current azimuth
  -F, --Full          generate a full trajectory (must specify azimuth)
  -l, --lst           a trajectory at az=180.0 and Ra=LST (starts from
                      center of tracker)
  --lst-east          a trajectory at az=65.0 and Ra=LST (starts from center
                      of tracker)
  --lst-west          a trajectory at az=295.0 and Ra=LST (starts from
                      center of tracker)

Global options:
  -a AZIMUTH, --azimuth=AZIMUTH
                      [cur|current|best|nnn.nnn] generate a trajectory with
                      selected azimuth (default best).
  -E EQUINOX, --equinox=EQUINOX
                      use EQUINOX instead of 2000.0.
  -w, --west          generate a trajectory in the west (default east).
  --use-struct        move the structure if necessary. legacyServer must be
                      running.
  --use-dome          move the dome if necessary. legacyServer must be
                      running.
  -v, --verbose       be noisy about our actions.
```

Description:

`trajectory` will generate and run a trajectory with in the Tcs. If requested, it will move the structure as well. This is a single step process. It is not like to old Tcs where we would load a trajectory first, then tell the system when to go later. Both the load and gonext steps are performed with this command. If a trajectory is currently running, this command will return an error message.

Example:

There are a couple of ways that I use trajectory. The first way is just the usual track, where the structure moves to the calculated azimuth

`trajectory -v --use-struct -- <ra> <dec>` #where <ra> <dec> are in decimal degrees.

The other way is to use the current structure azimuth

`trajectory -v -a current -- <ra> <dec>`

or you can ignore the structure all together and specify an azimuth yourself

`trajectory -v -a <az> -- <ra> <dec>` # if you do not specify the –use-struct flag, the structure will not move.

If you get an error message that the azimuth and direction don't agree, then try using the –west flag. This message should only occur on tracks in the north or the south.

5.1.14 vnctrk

This is a simple shell script that excutes `vncviewer` with the proper geometry and connects to the tracker engineering computer. This allows the operator to remotely view the Control Desk interface.

5.1.15 rdesktrk

This is a simple shell script that excutes `rdesktop` with the proper geometry and connects to the tracker engineering computer. This allows the operator to remotely view the Control Desk interface.

5.2 Test Scripts

5.2.1 mountTest

Usage: mountTest [options]

Run part or all of a 97 point test

Options:

```
-h, --help            show this help message and exit
-l LIST, --list=LIST  run the "97", "21", "Q1" through "Q4" point list, or
                      some other defined list (default "97")
-o OUTPUT_FILE, --output-file=OUTPUT_FILE
                      use the file OUTPUT_FILE for the results (default is
                      "./data/mountTest_out.csv).
```

5.2.2 dmipoints

Usage: dmipoints [options]

Run part or all of a 97 point test collecting DMI and TTC data. See *Points.py* for more information about the available points.

Options:

```
--version             show program's version number and exit
-h, --help            show this help message and exit
-l LIST, --list=LIST  run the "97", "21", "Q1" through "Q4" point list, or
                      some other defined list (default "97")
-v, --verbose         monitor all systems on stderr
-g, --gui             start gui so operator can monitor
```

5.2.3 dmittc

Usage: dmittc.py [options] azimuth

This script runs a full trajectory at the given azimuth and collects DMI and TTC data. These data are collected in the log /opt/het/hetdex/logs/data/Mon/dmittcs/dmittc_YYYYMMDDThhmmss_az.mon

Options:

```
-h, --help       show this help message and exit
-v, --verbose    monitor all systems on stderr
-V, --version    print version infomation
-g, --gui        start gui so operator can monitor
-t, --test       use the test values in the monitor output
-n, --no-struct  do not use the structure (default - use structure)
--closeTT        close the Tip/Tilt control loop
--closeDMI       close the DMI control loop
```

5.3 Admin Scripts

5.3.1 get_geo

A shell script that retrieves the geo-sychronous satellite orbital elements from http://celestrak.com/NORAD/elements/geo.txt. The script is run daily at 23:00 hours on the machine htcs from the hetdex user's crontab file. The file resides in `/opt/het/hetdex/etc/point/data/geo.txt`.

5.3.2 get_ser7

A shell script that retrieves the Earth Orientation Bulletin from the U.S. Naval Observatory at ftp://maia.usno.navy.mil/ser7/ser7.dat. The script is run weekly on Thursdays at 23:00 hours on the machine htcs from the hexdex user's crontab file. The file resides in `/opt/het/hetdex/etc/ser7.dat`.

5.3.3 tcs_query

Query a database for a specific event type

$ ./tcs_query –help

This is a utility for generating a csv-formatted text file by querying a TCS database containing events identified by: <system>.<source>.<key>

The key to extract is specified by: –key=<system>.<source>.<key>

For example, to extract the key identified by system=pfip, source='acq_shutter', key='status', use
–key='pfip.acq_shutter.status'

The entries listed in the output file will be the attributes for the specified key(s), ordered by time. If no key parameter is provided, the output from the utility will be a list of the unique keys contained in the database file.

The file output will be sent to stdout, unless the –outfile parameter is provided with a file name. When a specific key is requested using the –key option, the output file generated will contain values for all the attributes of that event key, ordered by time.

The name of the output file is specifed using the option: –outfile=<filename>

The output file name is defaulted to stdout The name of the database file to search is specified with the option:

–dbfile=<filename>

The meta-data attributes are, by default, included in the output. If you wish to reduce data redundancy and query time, specify a list of meta attributes to include, e.g.:

–meta='__data_time_string,__pid'

By default, the data delimiter in the output file is <tab>. If you wish to specify a different delimiter, use:
–delim=','

Names of the attributes will be listed at the beginning of the output file, if the –attribs option is used.

CHAPTER

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

h

p

S

T

U

W