Introduction to the 699 Python Library

Introduction

NOTE: For MOMA, the core tmread functionality described in this article is deprecated, because it is replaced by c699util. Please use the routines described here instead.

This guide aims to provide a high-level description of the 699 Python library and is intended for those who are writing Python scripts that rely on this library (and who might have reason to "peek under the hood"). It does not include the details of low level operations such as parsing and buffering.

Note that all code examples have been tested, so they should be able to run on a workstation that has the Python tools set up (although you'll have to change the hardcoded directory names).

The instructions for installing the python tools are here:
http://699wiki.com/wiki/index.php/Python_Tool_Installation

Common Objects and Functions

tmread and TMFile

tmread is the Python package for interacting with telemetry files created by 699 instruments. It contains several useful modules, classes, and routines, including the TMFile class and the get_tmfile() function.

A TMFile object represents a telemetry file. (I will refer to telemetry files as tm.mom files, but this also applies to tm.sam or tm.mav files.) TMFile provides many useful methods, such as:

```
#!/usr/bin/env python3


from tmread import get_tmfile


tmfile = get_tmfile("/Users/mpallone/momadata/etu/tm.mom.m2")


directory = tmfile.absolute_directory()
print("directory:", directory)
# output: directory: /Users/mpallone/momadata/etu
```

```
start_time = tmfile.start_time()
print("start_time:", start_time) # returns the timestamp of the earliest packet
# output: start_time: 0.0

file_size = tmfile.file_size() # returns the number of bytes of the tm.mom file
print("file_size:", file_size)
# output: file_size: 610

tid = tmfile.tid()
print("tid:", tid)
# output: tid: 7515
```

The most useful feature of TMFile is that it can be iterated through, generating packets:

```
#!/usr/bin/env python3

from tmread import get_tmfile

tmfile = get_tmfile("/Users/mpallone/momadata/etu/tm.mom.m2")

num_pkts = 0
for pkt in tmfile:
    num_pkts += 1

print("num_pkts:", num_pkts)
# output: num_pkts: 544
```

When writing scripts (or when processing any large amount of data at any time), it is generally much faster to make a single pass through the file. This is particularly true for the MOMA Python tools, where packets are not cached in memory.

get_tmfile() can be passed a TID (as a string or integer), or a TID directory name, or (as we just saw) a tm.mom file, or nothing if you're currently in a TID directory.

```
(py699)gs699-mpallone:~ mpallone$
(py699)gs699-mpallone:~ mpallone$ # Starting in the home directory:
(py699)gs699-mpallone:~ mpallone$
(py699)gs699-mpallone:~ mpallone$ pwd
/Users/mpallone
(py699)gs699-mpallone:~ mpallone$
(py699)gs699-mpallone:~ mpallone$
```

```
(py699)gs699-mpallone:~ mpallone$ # We can use the 'tid' command to jump to a
(py699)gs699-mpallone:~ mpallone$ # TID directory:
(py699)gs699-mpallone:~ mpallone$
(py699)gs699-mpallone:~ mpallone$ tid 7421
(py699)gs699-mpallone:~ mpallone$
(py699)gs699-mpallone:~ mpallone$
(py699)gs699-mpallone:2014-12-12-15.33.44-07421-steves_test mpallone$ pwd
/Users/mpallone/momadata/etu/2014-12-12-15.33.44-07421-steves_test
(py699)gs699-mpallone:~ mpallone$
(py699)gs699-mpallone:~ mpallone$
(py699)gs699-mpallone:2014-12-12-15.33.44-07421-steves_test mpallone$ python
Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> # We can use the tmread module from the Python interpreter, from any
>>> # directory:
>>>
>>> from tmread import get_tmfile
>>>
>>> tmfile = get_tmfile()
>>> tmfile.tid()
7421
>>>
>>>
>>> tmfile = get_tmfile(30100)
>>> tmfile.tid()
30100
>>>
>>>
>>> tmfile = get_tmfile("30101")
>>> tmfile.tid()
30101
>>>
>>>
>>>
>>> tmfile = get_tmfile("/Users/mpallone/momadata/qsm/2014-12-12-09.48.18-20000-woa-204-wrp-load-test")
>>> tmfile.tid()
20000
>>>
```

```
>>> count = 0
>>> for pkt in tmfile:
...     count += 1
...
>>> count
702
```

TMPacket

A TMPacket object represents a telemetry packet. MOMA packets start with an 8 byte header and end with a 4 byte checksum. TMPacket provides several useful methods and properties, including:

```
#!/usr/bin/env python3

from tmread import get_tmfile

tmfile = get_tmfile("/Users/mpallone/momadata/etu/tm.mom.m2")

for pkt in tmfile:
    print(pkt.raw_time)    # raw_time is the timestamp in the packet header
    print(pkt.type)        # 8 for message logs, 7 for digital status packets, etc.
    print(pkt.gse_created) # always False for genuine MOMA packets
    print(pkt.sequence)    # sequence number of the packet
    print(pkt.length)      # length of the packet (not including the 8 byte header or 4 byte checksum)
    print(pkt.mkid)        # Marker ID of the packet
    print(pkt.marker_text) # text of the marker packet that preceded the current packet (if such a packet exists)
```

Use the all_data() method to extract telemetry from the packet:

```
#!/usr/bin/env python3

from tmread import get_tmfile

tmfile = get_tmfile("/Users/mpallone/momadata/fm/2015-02-10-12.26.48-30005-woa-245-wrp-test/tm.mom.m2")

for pkt in tmfile:

    # This isn't the best way to extract data. This is just a TMPacket example.
    # See extract_tmfields() for a better way.
    if pkt.type == 17: # MAIF packet
```

```
    ps_temp = pkt.all_data(806) # HKID 806 is PS_TEMP

    # ps_temp is now a list of (timestamp, value) tuples
    first_datapoint = ps_temp[0]
    timestamp = first_datapoint.ts
    value = first_datapoint.val

    print("timestamp:", timestamp)
    print("value:", value)
    # output:
    # timestamp: -1.895219
    # value: 29.495738

    break
```

SebScienceDataPkt

SebScienceDataPkt is an example of how TMPacket can be subclassed. When the Python tools encounter a packet of type 26, the tools return a packet object specific to packet 26: SebScienceDataPkt. Similar packets exist for other packet types (see tmpacket.py and moma.py for other examples).

SebScienceDataPkt has methods specific to packet 26:

```
#!/usr/bin/env python3

from tmread import get_tmfile

tmfile = get_tmfile("/Users/mpallone/momadata/fm/2015-02-25-20.51.11-30076-msfunc_gc_ldi_ec_increasing")

for pkt in tmfile:

    # This isn't the best way to extract data. This is just a TMPacket example.
    # See extract_tmfields() for a better way.       if pkt.type == 26: # SEB Science Data Packet

        print(type(pkt)) # <class 'tmread.moma.SebScienceDataPkt'>         print(pkt.starting_bin_number) # 1

        # SebScienceDataPkt has a special iterator that yields bin numbers
        # and ion counts. This 'for' loop wouldn't work on an ordinary          # TMPacket object.          ion_count = 0
        for bin_number, count in pkt:               ion_count += count

        print(ion_count) # 0 (no ions in the first pkt 26 of this TID!)

        break
```

Filtering by marker and/or packet type

When iterating through a TMFile object, packets can be filtered by marker type, or packet type, or both.

For marker filters, the format of the filter parameter can be pretty much anything you want:

```
(py699)gs699-mpallone:2015-02-25-18.49.20-30072-msfunc_rf_ramp_time mpallone$ tmmarker.py
  75.321    1000   Check RF Freq
  75.542    1010   Pressure Check
  96.347    1020   Filament on
 102.327    1040   Initialize SEB
 117.294    2500   SCN 13 START rf ramp time
 120.530    2501   SCN 13 END rf ramp time
(py699)gs699-mpallone:2015-02-25-18.49.20-30072-msfunc_rf_ramp_time mpallone$
(py699)gs699-mpallone:2015-02-25-18.49.20-30072-msfunc_rf_ramp_time mpallone$
(py699)gs699-mpallone:2015-02-25-18.49.20-30072-msfunc_rf_ramp_time mpallone$ python
Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from tmread import *
>>> tmfile = get_tmfile(30072)
>>> markers = set()
>>>
>>>
>>> # A string can be used as the marker filter. A dash indicates a range
>>> # of desired values.
>>>
>>> for pkt in tmfile(marker_filter="1010-1040"):
...     markers.add(pkt.mkid)
...
>>>
>>> markers
{1040, 1010, 1020}
>>>
>>>
>>>
>>> # An integer can be used as a marker filter.
>>>
>>> markers = set()
>>> for pkt in tmfile(marker_filter=1010):
...     markers.add(pkt.mkid)
...
>>> markers
```

```
{1010}
>>>
>>>
>>>
>>> # A comma separated string can be used to specify a list of marker IDs
>>> # to filter by. Note that marker ID 1020 isn't present, unlike when
>>> # we filtered using "1010-1040".
>>>
>>> markers = set()
>>> for pkt in tmfile(marker_filter="1010,1040"):
...     markers.add(pkt.mkid)
...
>>> markers
{1040, 1010}
>>>
>>>
>>>
>>>
>>> # A list can be used to filter markers.
>>>
>>> markers = set()
>>> for pkt in tmfile(marker_filter=[2500, 2501]):
...     markers.add(pkt.mkid)
...
>>> markers
{2500, 2501}
>>>
>>>
>>>
>>> # A generator can be used to filter markers.
>>>
>>> markers = set()
>>> for pkt in tmfile(marker_filter=range(1000,2000)):
...     markers.add(pkt.mkid)
...
>>>
>>> markers
{1000, 1040, 1010, 1020}
```

We can also filter by packet type. Continuing with the previous example:

```
>>>
>>>
>>> # The type_filter can be an integer:
>>>
>>> types = set()
>>> for pkt in tmfile(type_filter=25):
...     types.add(pkt.type)
...
>>> types
{25}
>>>
>>>
>>> # Or the type filter can be a list (really, any iterable container):
>>>
>>> types = set()
>>> for pkt in tmfile(type_filter=[7, 8, 9]):
...     types.add(pkt.type)
...
>>>
>>> types
{8, 9, 7}
>>>
>>>
>>>
>>> # Or the type filter can be a generator:
>>>
>>> types = set()
>>> for pkt in tmfile(type_filter=range(7,10)):
...     types.add(pkt.type)
...

>>>
>>> types
{8, 9, 7}
>>>
```

Finally, type_filter and marker_filter can be used at the same time:

```
>>> # marker_filter and type_filter can be combined:
>>>
```

```
>>> for pkt in tmfile(marker_filter=2501, type_filter=10):
...     print(pkt.index, pkt.type, pkt.mkid)
...
5646 10 2501
5649 10 2501
5652 10 2501
# etc.
```

TMArgumentParser

The TMArgumentParser class is a child class of the built-in ArgumentParser: https://docs.python.org/3/howto/argparse.html . It is extremely useful when building 699 Python scripts.

It adds a few additional 699-telemetry-specific bells and whistles (see libs-py/tmread/scripts.py for the full implementation), as the example below shows. TMArgumentParser also finds and returns relevant TMFile objects:

```
#!/usr/bin/env python3

from tmread import TMArgumentParser
import tmread

import os

def main():

    # python scripts are usually run inside of a TID directory, so pretend to do
    # that here.
    os.chdir("/Users/mpallone/momadata/fm/2015-02-25-20.51.11-30076-msfunc_gc_ldi_ec_increasing/")

    parser = TMArgumentParser(description="This title-string shows up when the -h flag is used.")

    # This allows the user to specify --unix to see Unix time, --relative to see
    # relative time, etc. When "tmpacketObject.timestamp" is accessed, it will
    # default to whatever flag the user specified.
    parser.add_time_modes_group()



    # This allows the user to specify --raw to see raw telemetry values, --eng to
    # see telemetry values converted to engineering units, etc. Housekeeping
    # objects will take on whatever format this flag specifies.
```

```
    parser.add_data_modes_group()


    # This allows the user to specify "-m 50" or "-m 50-100" to see marker 50
    # packets or to see marker packets between marker 50 and marker 100
    parser.add_marker_argument()


    # After setting up the parser, let actually parse the arguments:
    args = parser.parse_args()


    #
    # *This* is the preferred way to get a TMFile object:
    #
    tmfile = args.tmfile
    if not tmfile:
        return tmread.ReturnCodes.missing_tmfile


    # Do something with the tmfile object
    print(tmfile.absolute_directory())
    # output: /Users/mpallone/momadata/fm/2015-02-25-20.51.11-30076-msfunc_gc_ldi_ec_increasing


main()
```

moma_packet_types

moma_packet_types is an enumeration that allows us to refer to packet types without hardcoded magic numbers.

```
#!/usr/bin/env python3

from tmread import MomaPktIds

print(MomaPktIds.fsw_time_pkt_id == 2)   # True
print(MomaPktIds.digital_hk_pkt_id == 7) # True
print(MomaPktIds.msg_log_pkt_id == 8)    # True
```

The full definition (as of 05/26/15):

```
class MomaPktIds(enum.IntEnum):
    """Enumeration of packet IDs used in MOMA."""
    memory_dump_pkt_id           = 1
    fsw_time_pkt_id              = 2
    digital_hk_pkt_id            = 7
    msg_log_pkt_id               = 8
    marker_pkt_id                = 9

    rsim_nominal_pkt_id          = 10

    fill_pkt_pkt_id              = 11
    gc_science_and_hk_pkt_id     = 14
    laser_pulse_tlm_pkt_id       = 15
    laser_periodic_status_pkt_id  = 16
    maif_1_sec_pkt_id            = 17
    maif_1_min_pkt_id            = 18
    micro_pirani_pressure_pkt_id  = 19
    seb_ack_pkt_id               = 21
    seb_hk_pkt_id                = 22
    seb_sequence_mem_dump_pkt_id  = 23
    seb_dds_mem_dump_pkt_id      = 24
    seb_scan_status_pkt_id       = 25
    seb_science_data_pkt_id      = 26
    seb_summed_histogram_pkt_id   = 27
    seb_combined_histogram_pkt_id = 28

    rsim_fake_timestamp_pkt_id    = 74
```

extract_tmfields

extract_tmfields() takes an iterable of packets and a list of HKIDs to extract, and returns a dictionary containing the data. The keys of the returned dictionary are the HKIDs, and the values of the dictionary are the list of datapoints that match the HKID.

```
#!/usr/bin/env python3

from tmread import get_tmfile, extract_tmfields
```

```
tmfile = get_tmfile("/Users/mpallone/momadata/fm/2015-02-25-20.51.11-30076-
msfunc_gc_ldi_ec_increasing/tm.mom.m2")

list_of_hkids = [802] # Arbitrarily extracting HKID 802

dictionary_of_datapoints = extract_tmfields(tmfile, list_of_hkids)

first_datapoint = dictionary_of_datapoints[802][0]

timestamp = first_datapoint.ts
value = first_datapoint.val

print(timestamp, value) # Output: 1.012687 1.498488
```

TID Directories

tmread also provides two variables to facilitate working with TID directories on a user's computer. Remember that these directories can be passed to get_tmfile().

```
>>> from tmread import moma_tmdirs
>>> from tmread import moma_tids
>>>
>>>
>>> # moma_tids is a dictionary where the key is the TID (which can be an int
>>> # or a string), and the value is the directory of that TID on the user's
>>> # computer.
>>>
>>> moma_tids[7421]
'/Users/mpallone/momadata/etu/2014-12-12-15.33.44-07421-steves_test'
>>>
>>> moma_tids["7421"]
'/Users/mpallone/momadata/etu/2014-12-12-15.33.44-07421-steves_test'
>>>
>>>
>>>
>>> # moma_tmdirs is simply a list of TID directories on the user's computer.
>>>
>>> moma_tmdirs[0]
'/Users/mpallone/momadata/etu/2014-07-24-14.16.31-00268-'
```

```
>>>
>>>
>>> moma_tmdirs[5]
'/Users/mpallone/momadata/etu/2014-07-29-17.00.38-00273-MEB-GC-Integration-ETU'
>>> moma_tmdirs[100]
'/Users/mpallone/momadata/etu/2014-09-07-15.35.24-07078-firing_laser'
```

Examples

t0.py

"Cookbooking" off of t0.py is a good way to start writing a script:

```
#!/usr/bin/env python3
"""A script display start time from a telemetry file.

Many telemetry file processing programs use time since start of file.
This program displays that time.

Author: Eric Raaen
Date:   Feb 1, 2012

"""
from time import asctime, ctime, gmtime
import tmread

def main():
    """Main function for t0.py."""
    parser = tmread.TMArgumentParser(description='Display start time of a '
                                     'telemetry file')
    args = parser.parse_args()

    t0 = args.tmfile.start_time()
    gmt = gmtime(t0.unix)
    print("SCLK (raw):", t0.sclk)
    print("     ctime:", t0.unix)
    print("       UTC:", asctime(gmt))
    print("Local time:", ctime(t0.unix))
```

```
    return 0



if __name__ == '__main__':
    exit(main())
```

Revision #3
Created 22 March 2023 19:43:08 by Nick Dobson
Updated 24 March 2023 13:46:57 by Nick Dobson