c699util User's Guide

Similar to Introduction to the 699 Python Library, this article is a high-level description of c699util, intended for those who are only using it to write Python scripts. I do not describe low level c699util details here. For that, see the c699util developer's guide.

c699util is a Python module which provides housekeeping and science data extraction routines for MOMA telemetry files. Its interface is very similar to 699util's tmread module:

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

  #================================================================
=============
  # Packet data extraction through 699util's tmread module:
  #================================================================
=============
  import tmread

  tmfile = tmread.get_tmfile('/Users/mpallone/momadata/fm/2016-01-15-14.06.47-30410-
MSFUNC/tm.mom.m3.s3')

  wrp_current_datapoints = []
  wrp_current_hkid = 800
  pkt_17_count = 0

  for pkt in tmfile(17):
     pkt_17_count += 1
     wrp_current_datapoints.extend(pkt[wrp_current_hkid])

  # Print out the data:
  print("pkt_17_count:", pkt_17_count)
  print()
  print("WRP current sample timestamp, WRP current sample science value:")
  for timestamp, value in wrp_current_datapoints:
     print(timestamp, 't', value)



  #================================================================
=============
```

```
# Packet data extraction through c699util:
#=================================================================
=============
import c699util

c_tmfile = c699util.construct_tmfile('/Users/mpallone/momadata/fm/2016-01-15-14.06.47-30410-
MSFUNC/tm.mom.m3.s3')

wrp_current_datapoints = []
wrp_current_hkid = 800
pkt_17_count = 0

for pkt in c_tmfile(17):
    pkt_17_count += 1
    wrp_current_datapoints.extend(pkt[wrp_current_hkid])

# Print out the data:
print("pkt_17_count:", pkt_17_count)
print()
print("WRP current sample timestamp, WRP current sample science value:")

#for timestamp, value in wrp_current_datapoints:
#    print(timestamp, 't', value)

# Note the change here!
for datapoint in wrp_current_datapoints:
    print(datapoint.relative_timestamp, 't', datapoint.value)
```

As this example illustrates, some minor interface differences exist between c699util and tmread, which are documented at the end of this article.

You are highly encouraged to follow the following convention: if you use c699util in a Python script, please prefix it with "c_". For example, when I ported ioncount.py over to c699util, I made a copy of ioncount.py, called it c_ioncount.py, and then replaced tmread code with similar c699util code. This way, both scripts are available to run (a nice option to have if we encounter bugs), and we always know which underlying library we're using.

If we already have tmread, why bother with c699util?

Well, for one, c699util is about an order of magnitude faster than tmread (and sometimes more -- c_ioncount.py was 39x faster than ioncount.py when extracting selected ion counts).

More importantly, much of tmread is deprecated for MOMA. Any feature that tmread shares with c699util is no longer maintained. Instead of maintaining telemetry extraction libraries in C++, and then duplicating the exact same functionality in Python, we've decided to simply add a Python interface to the C++ code. This dramatically simplifies feature implementation, eliminates consistency issues between various tools, and speeds up software development and program run time, but it also means that tmread won't output data as accurately as c699util, and also won't have as many features.

c699util Flaw -- only one TMFile at a time

Because the C++ code around which c699util wraps frequently uses singletons, c699util can only operate on one TMFile at a time. Calling c699util.construct_tmfile() will invalidate any previously existing c699util.TMFile objects.

We rarely ever (if at all) need this capability, but when the C++ and c699util code is ported over to future missions, this design should be reconsidered.

Obtaining c699util

If you do not have the 699 Python tools installed on your computer, then the developer instructions in Python_Tool_Installation should work, although it's probably best to just ask a MOMA software developer to do it for you. Be sure to install Python 3.4.3 from the Python website.

If you already have 699 Python tools installed, you will probably also have to ask a MOMA software developer to setup c699util. Most likely, Python 3.4.3 will need to be installed directly from the website (sorry, Homebrew aficionados), and your virtual environment will need to be rebuilt so that it uses Python 3.4.3. However, if you happen to already be using Python 3.4.3, then SSH tunneling into repos699 and then updating ~/config699 and ~/labcode/699util should be all that you need to do to start using c699util.

TMFile

c699util provides a TMFile class which is very similar to tmread's TMFile class. c699util.TMFile objects can be obtained as follows:

```
import c699util
c_tmfile = c699util.get_tmfile('/Users/mpallone/momadata/fm/2016-01-15-14.06.47-30410-MSFUNC/tm.mom.m3.s3')
```

c699util.get_tmfile() is a thin wrapper around tmread.get_tmfile(), so it does all of the telemetry-file-finding-magic that tmread.get_tmfile() does:

```
import c699util
c_tmfile = c699util.get_tmfile() # grab the tm.mom file in the cwd
```

```
c_tmfile = c699util.get_tmfile(12345) # Grab TID 12345
c_tmfile = c699util.get_tmfile("/Users/mpallone/momadata/fm/2016-01-15-14.06.47-30410-MSFUNC") # Grab the tm.mom file in the specified directory
```

It would be preferable to not use the deprecated tmread module, but it seems unlikely to me that I'll ever get around to implementing get_tmfile()'s magic inside of c699util, at least on MOMA. Also, this tactic allows us to keep using the extremely useful TMArgumentParser class.

If tmread is not available, c699util.construct_tmfile() can be used:

```
import c699util
c_tmfile = c699util.construct_tmfile('/Users/mpallone/momadata/fm/2016-01-15-14.06.47-30410-MSFUNC/tm.mom.m3.s3')
```

Also, note the convention of using c_tmfile. I prefer to use 'tmfile' for tmread.TMFile objects, and 'c_tmfile' for c699util.TMFile objects, so that I can tell at a glance what kind of object I'm using. I strongly recommend that other script writers do the same.

Other useful methods:

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

import tmread, c699util
tmfile = tmread.get_tmfile(30410) # Give it a TID and let it find the file for us
c_tmfile = c699util.construct_tmfile(tmfile.filename)

# (alternatively, we could just do c_tmfile = c699util.get_tmfile(30410)

print("Absolute filename:", c_tmfile.filename())
print("Absolute filename:", str(c_tmfile))
# Absolute filename: /Users/mpallone/momadata/fm/2016-01-15-14.06.47-30410-MSFUNC/tm.mom.m3.s3

print("Absolute directory:", c_tmfile.absolute_directory())
# Absolute directory: /Users/mpallone/momadata/fm/2016-01-15-14.06.47-30410-MSFUNC

# Parsed from the first GSE packet in the tm.mom file:
print("TID Directory name:", c_tmfile.directory_string())
# TID Directory name: 2016-01-15-14.06.47-30410-MSFUNC

print("Mass calibration file used:", c_tmfile.get_mcal_filename())
# Mass calibration file used: 2016-01-17-23.48.39-30410.mcal

print("File size in bytes:", c_tmfile.file_size())
```

```
# File size in bytes: 48384623

print("TID:", c_tmfile.tid())
# TID: 30410

print("t0 in Unix time:", c_tmfile.t0_in_unix_time())
# t0 in Unix time: 1452884878.3959727

print("Number of packets:", c_tmfile.length())
print("Number of packets:", len(c_tmfile))
# Number of packets: 70232

print("TMFile actually exists on the filesystem:", c_tmfile.exists())
print("TMFile actually exists on the filesystem:", bool(c_tmfile))
# TMFile actually exists on the filesystem: True

pkt_at_index_10 = c_tmfile.get_pkt(10)
pkt_at_index_10 = c_tmfile[10]

# We can iterate through c_tmfile and get packet objects:
pkt_count = 0
for pkt in c_tmfile:
    pkt_count += 1

# The old tm_filter routines also work for c699util:
pkt_17_count = 0
for pkt in c_tmfile(17):
    pkt_17_count += 1

pkt_count = 0
for pkt in c_tmfile(type_filter=17,      # Only yield packets of type 17
                    marker_filter=2500,  # Only yield packets at marker 2500
                    start_time=1000,     # Relative time = 1000 seconds
                    stop_time=2000):     # Relative time = 2000 seconds
    pkt_count += 1

# If the directory name is
#     '2016-01-15-14.06.47-30410-MSFUNC'
# then the test name is
#     'MSFUNC'
```

```
print("Test name:", c_tmfile.test_name())

# tmread.MessageLog is not deprecated, and is fully compatible with c699util:
msg_log = c_tmfile.message_log()

meta_markers = c_tmfile.get_meta_markers()
for meta_marker in meta_markers:
    print(meta_marker.startRelativeTimeInSecs)
    print(meta_marker.endRelativeTimeInSecs)
    print(meta_marker.metaMarkerId)
    print(meta_marker.index)
    print(meta_marker.msg)

print("model:", c_tmfile.model())
# model: fm

print("FSW timestamp at of t0 packet:", c_tmfile.t0_in_sclk())
# FSW timestamp at of t0 packet: 19351744
```

tmread.extract_tmfields() is compatible with c699util.TMFile objects:

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

import tmread, c699util
tmfile = tmread.get_tmfile(30410) # Give it a TID and let it find the file for us
c_tmfile = c699util.construct_tmfile(tmfile.filename)

AUX_AMP_DAC_SET_HKID = 61
RF_AMP_MON_HKID      = 62

fields = tmread.extract_tmfields(c_tmfile, [AUX_AMP_DAC_SET_HKID, RF_AMP_MON_HKID])

# Note that we still get c699util.TMDatapoint objects here, which are different
# from 699util's tmread datapoints!
sum_ = 0
count = 0
for datapoint in fields[RF_AMP_MON_HKID]:
    relative_timestamp = datapoint.relative_timestamp
    unix_timestamp = datapoint.unix_timestamp
    raw_timestamp = datapoint.fsw_timestamp
```

```
        sum_ += datapoint.value
        count += 1

    if count != 0:
        print("RF Amp mon avg:", sum_ / count)
```

TMPacket

Just like TMFile, c699util provides TMPacket objects which are very
similar to tmread.TMPacket objects:

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

    import tmread, c699util
    tmfile = tmread.get_tmfile(30410) # Give it a TID and let it find the file for us
    c_tmfile = c699util.construct_tmfile(tmfile.filename)

    gse_pkt = c_tmfile[0]       # Pkt type 8, GSE generated
    pkt_22 = c_tmfile[65]       # Pkt type 22
    msg_log_pkt = c_tmfile[85]  # Pkt type 8

    print("Unmasked type of gse_pkt:", gse_pkt.full_type())
    # Unmasked type of gse_pkt: 136

    print("Unmasked type of pkt_22:", pkt_22.full_type())
    # Unmasked type of pkt_22: 22

    # This is analagous to tmread.TMPacket.type
    print("gse_pkt with gse-generated bit masked out:", gse_pkt.get_type())
    # gse_pkt with gse-generated bit masked out: 8

    print("pkt_22 length, including header and checksum:", pkt_22.total_length())
    # pkt_22 length, including header and checksum: 224
    print("pkt_22 length, just the data:", pkt_22.length())
    # pkt_22 length, just the data: 212

    print("pkt_22 checksum:", pkt_22.checksum())
    print("pkt_22 checksum:", hash(pkt_22))
    # pkt_22 checksum: 1379474455
```

```
print("Is pkt_22 gse created?", pkt_22.gse_created())
# Is pkt_22 gse created? False
print("Is gse_pkt gse created?", gse_pkt.gse_created())
# Is gse_pkt gse created? True

print("pkt_22 sequence number:", pkt_22.sequence())
# pkt_22 sequence number: 146

# Yes, I'm aware that 'sclk' is a misnomer
print("pkt_22 FSW timestamp:", pkt_22.sclk())
# pkt_22 FSW timestamp: 19540225
print("pkt_22 FSW timestamp, in seconds:", pkt_22.sclk_in_seconds())
# pkt_22 FSW timestamp, in seconds: 1954.0225

print("pkt_22 index:", pkt_22.index())
# pkt_22 index: 65

print("pkt_22 marker ID:", pkt_22.marker_id())
# pkt_22 marker ID: 0

# Packets before the first marker packet have a marker index of 0
print("pkt_22 marker index:", pkt_22.xina_marker_index())
# pkt_22 marker index: 0

print("pkt_22 unix timestamp, in seconds:", pkt_22.unix_timestamp())
# pkt_22 unix timestamp, in seconds: 1452884897.2440727

print("pkt_22 relative timestamp, in seconds:", pkt_22.relative_timestamp())
# pkt_22 relative timestamp, in seconds: 18.848099946975708

print("pkt_22 marker text:", pkt_22.marker_text())
# pkt_22 marker text: <no marker text, but you get the idea>

############################################################
#
# (this doesn't actually work for TID 30410, but I think you get the picture)
#
# HKID 100 = phase cycle readback HKID
```

```
phase_cycle_readback_raw_data = pkt_22.all_raw_data(100)
phase_cycle_readback_eng_data = pkt_22.all_eng_data(100)

phase_cycle_readback_sci_data = pkt_22.all_data(100)
phase_cycle_readback_sci_data = pkt[100]

datapoint = phase_cycle_readback_sci_data[0]
print("Datapoint unix timestamp, in seconds:", datapoint.unix_timestamp)
print("Datapoint relative timestamp, in seconds:", datapoint.relative_timestamp)
print("Datapoint value:", datapoint.value)

first_phase_cycle_readback_science_sample = pkt_22.get_first_value(100)
#
#
############################################################

print("Does gse_pkt have an earlier timestamp than pkt_22?", gse_pkt < pkt_22)
# Does gse_pkt have an earlier timestamp than pkt_22? True

print("Raw message log:", msg_log_pkt.raw_message())
print("Sanitized message log:", msg_log_pkt.message())
# (output omitted for brevity)

print("raw bytes:", pkt_22.raw_packet())
# b'x16x92x00xd4x01*)x01xfaxf3 x02x00...(etc).

meta_markers = pkt_22.get_meta_markers()
for meta_marker in meta_markers:
    print(meta_marker.startRelativeTimeInSecs)
    print(meta_marker.endRelativeTimeInSecs)
    print(meta_marker.metaMarkerId)
    print(meta_marker.index)
    print(meta_marker.msg)
```

Extracting Science Data

c699util has a get_scans() function, just like tmread.momascience:

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

import c699util
c_tmfile = c699util.get_tmfile(30410)

scan_count = 0
for scan in c699util.get_scans(c_tmfile):
    scan_count += 1

print("Number of scans:", scan_count)
```

The usual packet type, marker filtering, start time, etc. filtering options are available.

Unlike tmread, c699util provides a wrapper around MOMA Data View's ScienceDataCache class:

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

import c699util

# Not needed
# c_tmfile = c699util.construct_tmfile(tmfile.filename)

# Notice that this factory function takes the absolute filename, just like
# construct_tmfile
# science_cache = c699util.construct_science_data_cache('/Users/mpallone/momadata/fm/2016-01-15-14.06.47-30410-MSFUNC/tm.mom.m3.s3')

science_cache = c699util.get_science_data_cache(30410)
```

c699util.get_science_data_cache() is analagous to c699util.get_tmfile(), so it can be passed no arguments if the programmer wants to search the cwd, or just the TID to search for, or the absolute path of the tm.mom file, etc.

The science cache can be iterated through to get scan objects (just c699util.get_scans()), or it can be indexed, as demonstrated below.

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

import c699util

science_cache = c699util.get_science_data_cache(30410)
```

```
last_scan = science_cache[-1]

print("Last scan data:")

print("Unix timestamp, in seconds:", last_scan.unix_timestamp())
# Unix timestamp, in seconds: 1452890221.7956727

print("Relative timestamp, in seconds:", last_scan.relative_timestamp())
# Relative timestamp, in seconds: 5343.399699926376

print("marker ID:", last_scan.marker_id())
# marker ID: 2050

print("marker Index:", last_scan.xina_marker_index())
# marker Index: 136

print("marker text:", last_scan.marker_text())
# marker text: SCN 602 START Emission Current Source B

list_of_ion_counts = last_scan.counts()
print("total ion count:", sum(list_of_ion_counts))
print("total ion count:", last_scan.total_ion_count())
# total ion count: 40463

print("Highest count in the last_scan:", last_scan.highest_count())
# Highest count in the last_scan: 708

print("Bin number containing the highest count:", last_scan.highest_count_bin_num())
# Bin number containing the highest count: 1710

print("Lowest count:", last_scan.lowest_count())
# Lowest count: 0

# There should be one entry for every 100 bins
rf_status_entry_list = last_scan.rf_status_entries()
rf_entry = rf_status_entry_list[0]

print("First RF Entry start bin:", rf_entry.start_bin)
# First RF Entry start bin: 0
```

```
print("First RF Entry RF Amp V:", rf_entry.rf_amp_v)
# First RF Entry RF Amp V: 66.67613871395588

print("First RF Entry RF Amp DAC V:", rf_entry.rf_amp_dac_v)
# First RF Entry RF Amp DAC V: 70.00512770935893

print("First RF Entry Aux Amp DAC V:", rf_entry.aux_amp_dac_v)
# First RF Entry Aux Amp DAC V: 1.0998702980577946

print("RF Amp Start:", last_scan.rf_amp_start())
# RF Amp Start: 66.67613871395588

print("RF Amp End:", last_scan.rf_amp_end())
# RF Amp End: 993.6579284607433

print("RF Amp DAC Start:", last_scan.rf_amp_dac_start())
# RF Amp DAC Start: 70.00512770935893

print("RF Amp DAC End:", last_scan.rf_amp_dac_end())
# RF Amp DAC End: 996.3826513544906

print("Aux Amp DAC Start:", last_scan.aux_amp_dac_start())
# Aux Amp DAC Start: 1.0998702980577946

print("Total scan time, in seconds:", last_scan.total_scan_time())
# Total scan time, in seconds: 0.05000000074505805

print("Peak counts per second:", last_scan.peak_counts_per_sec())
# Peak counts per second: 70799998.9449978

print("Pkt source type (26 or 27):", last_scan.src_pkt_type())
# Pkt source type (26 or 27): 27

print("Number of scans that went into this scan (typically 10 for sum packets):")
print(last_scan.num_scans())
# Number of scans that went into this scan (typically 10 for sum packets):
# 10
```

```
print("Does this scan have a scan status packet?", last_scan.has_scan_status())
# Does this scan have a scan status packet? True

print("Number of scan status packets:", last_scan.num_scan_status_pkts())
# Number of scan status packets: 10

print("Does this scan have a preceding SEB HK packet?", last_scan.has_seb_hk_pkt())
# Does this scan have a preceding SEB HK packet? True

print("Is this scan a dark count scan?", last_scan.is_dark_count_scan())
# Is this scan a dark count scan? False

print("Sum of the ion counts for bins 500-600:")
print(last_scan.selected_ion_count_sum_by_bin(500,600))
# Sum of the ion counts for bins 500-600:
# 3981

print("Sum of the ion counts between masses 138-140:")
print(last_scan.selected_ion_count_sum_by_mass(138,140))
# Sum of the ion counts between masses 138-140:
# 78

print("Maximum ion count between bins 500-600:")
print(last_scan.selected_ion_count_max_by_bin(500,600))
# Maximum ion count between bins 500-600:
# 537

print("Maximum ion count between masses 138-140:")
print(last_scan.selected_ion_count_max_by_mass(138,140))
# Maximum ion count between masses 138-140:
# 21

print("seb_bin_time_readback:", last_scan.seb_bin_time_readback())
# seb_bin_time_readback: 10.000000149011612

print("seb_bin_time_readback_in_seconds:", last_scan.seb_bin_time_readback_in_seconds())
# seb_bin_time_readback_in_seconds: 1.0000000149011612e-05

print("seb_special_scan_id:", last_scan.seb_special_scan_id())
```

```
# seb_special_scan_id: 602

print("seb_raw_scan_value:", last_scan.seb_raw_scan_value())
# seb_raw_scan_value: 127

print("seb_scan_value:", last_scan.seb_scan_value())
# seb_scan_value: 49.80392238497734

print("seb_rf_freq_hz:", last_scan.seb_rf_freq_hz())
# seb_rf_freq_hz: 988400.0

print("seb_last_rf_amp_mon:", last_scan.seb_last_rf_amp_mon())
# seb_last_rf_amp_mon: 993.6579284607433

print("seb_rf_amp_dac_set:", last_scan.seb_rf_amp_dac_set())
# seb_rf_amp_dac_set: 996.3826513544906

print("seb_aux_amp_dac_set:", last_scan.seb_aux_amp_dac_set())
# seb_aux_amp_dac_set: 7.297474631876097

print("seb_counter_sel:", last_scan.seb_counter_sel())
# seb_counter_sel: 1

print("seb_bin_count_set:", last_scan.seb_bin_count_set())
# seb_bin_count_set: 5000

print("seb_bin_count_reg:", last_scan.seb_bin_count_reg())
# seb_bin_count_reg: 20000

print("seb_phase_cycle_rdbk:", last_scan.seb_phase_cycle_rdbk())
# seb_phase_cycle_rdbk: 3

print("seb_rf_step_set:", last_scan.seb_rf_step_set())
# seb_rf_step_set: 0.1870379802200142

print("seb_rf_step_reg:", last_scan.seb_rf_step_reg())
# seb_rf_step_reg: -0.038366765173336245

print("seb_aux_step_set:", last_scan.seb_aux_step_set())
```

```
# seb_aux_step_set: 0.0012397955291065842

print("seb_aux_step_reg:", last_scan.seb_aux_step_reg())
# seb_aux_step_reg: 0.0006103608758678569

print("seb_ionization_time_in_milliseconds:", last_scan.seb_ionization_time_in_milliseconds())
# seb_ionization_time_in_milliseconds: 4.999999888241291

print("seb_scan_em1:", last_scan.seb_scan_em1())
# seb_scan_em1: -2069.6365661583154

print("seb_scan_em1_raw:", last_scan.seb_scan_em1_raw())
# seb_scan_em1_raw: 197

print("seb_scan_em2:", last_scan.seb_scan_em2())
# seb_scan_em2: 994.4167251074475

print("seb_scan_em2_raw:", last_scan.seb_scan_em2_raw())
# seb_scan_em2_raw: 0

print("seb_src_a_foc_a_dac_set:", last_scan.seb_src_a_foc_a_dac_set())
# seb_src_a_foc_a_dac_set: -39.6078431372549

print("seb_src_b_foc_a_dac_set:", last_scan.seb_src_b_foc_a_dac_set())
# seb_src_b_foc_a_dac_set: -39.6078431372549

print("seb_src_a_foc_a_dac_reg:", last_scan.seb_src_a_foc_a_dac_reg())
# seb_src_a_foc_a_dac_reg: -67.45098039215686

print("seb_src_b_foc_a_dac_reg:", last_scan.seb_src_b_foc_a_dac_reg())
# seb_src_b_foc_a_dac_reg: -67.45098039215686

print("meta markers:", last_scan.get_meta_markers())
# output omitted due to laziness, it's the same interface that's documented in the TMFile and TMPacket sectio
ns

print("m/z values:", last_scan.m_over_z_values())
# (long tuple of doubles)
```

```
print("list of TICs:", last_scan.total_ion_counts())
# list of TICs: (3680, 3882, 4092, 4100, 4147, 4119, 4103, 4021, 4214, 4105)

print("cTIC:", last_scan.corrected_total_ion_count())
# cTIC: 1715842

print("cTIC Factor:", last_scan.corrected_total_ion_count_factor())
# cTIC Factor: 42.40522135001076

print("cTIC Factors:", last_scan.corrected_total_ion_count_factors())
# cTIC Factors: ()

print("Packet version:", last_scan.science_data_pkt_version())
# Packet version: 1

print("seb_is_fil_a_on:", last_scan.seb_is_fil_a_on())
# seb_is_fil_a_on: False

print("seb_is_fil_b_on:", last_scan.seb_is_fil_b_on())
# seb_is_fil_b_on: True

print("Scan mode:", last_scan.scan_mode_as_string())
# Scan mode: EI

print("SEB sequence packet indexes:", last_scan.seb_sequence_pkt_indexes())
# SEB sequence packet indexes: (68463, 68464, 68465)

print("EM on times (ms):", last_scan.em_on_times_in_ms())
# EM on times (ms): (600.1000052131712, 0.0)
```

Interface Differences Between tmread and c699util

This section is intended for those who have used tmread on SAM, LADEE, or MAVEN. My hope is that such users can use c699util just as they use tmread, using this section as a cheatsheet as needed.

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

import tmread, c699util
tmfile = tmread.get_tmfile(30410) # Give it a TID and let it find the file for us
c_tmfile = c699util.construct_tmfile(tmfile.filename)

AUX_AMP_DAC_SET_HKID = 61
RF_AMP_MON_HKID      = 62

#-------------------------------------------------------------------------------
# TMFile interface differences
#-------------------------------------------------------------------------------

# Getting a TMFile:
tmfile = tmread.get_tmfile(30410) # Can be passed TID or absolute filename
c_tmfile = c699util.construct_tmfile(tmfile.filename) # requires absolute filename

filename =   tmfile.filename
filename = c_tmfile.filename()

dir_string =   tmfile.directory_string
dir_string = c_tmfile.directory_string()

t0_in_unix_time =   tmfile.start_time().unix   # Returns TMTimestamp object
t0_in_unix_time = c_tmfile.t0_in_unix_time() # Returns a float

msg_log =   tmfile.message_log
msg_log = c_tmfile.message_log()

#-------------------------------------------------------------------------------
# TMPacket interface differences
#-------------------------------------------------------------------------------

pkt   = tmfile[-1]
c_pkt = c_tmfile[-1]

# Packet type with GSE bit masked out:
pkt_type =   pkt.type
pkt_type = c_pkt.get_type()
```

```
# Full packet type:
full_pkt_type =   pkt.full_type
full_pkt_type = c_pkt.full_type()

total_length =   pkt.total_length
total_length = c_pkt.total_length()

# Doesn't included the 8 byte header and 4 byte checksum, unlike total_length
data_length =   pkt.length
data_length = c_pkt.length()

checksum =   pkt.checksum
checksum = c_pkt.checksum()

gse_created =   pkt.gse_created
gse_created = c_pkt.gse_created()

sequence_number =   pkt.sequence
sequence_number = c_pkt.sequence()

fsw_timestamp =   pkt.raw_time
fsw_timestamp = c_pkt.sclk()

unix_timestamp =   pkt.timestamp.unix     # TMTimestamp object
unix_timestamp = c_pkt.unix_timestamp()   # float

relative_timestamp =   pkt.timestamp.relative   # TMTimestamp object
relative_timestamp = c_pkt.relative_timestamp() # float

marker_text =   pkt.marker_text
marker_text = c_pkt.marker_text()

marker_id =   pkt.mkid
marker_id = c_pkt.marker_id()

msg =   pkt.message
msg = c_pkt.message()
```

```
#
# Housekeeping data interface:
#
data_dict = tmread.extract_tmfields(tmfile, [800])
hkid_800_datapoints = data_dict[800]
first_datapoint = hkid_800_datapoints[0]

timestamp = first_datapoint.ts
value = first_datapoint.val

data_dict = tmread.extract_tmfields(c_tmfile, [800])
hkid_800_datapoints = data_dict[800]
first_datapoint = hkid_800_datapoints[0]

unix_timestamp = first_datapoint.unix_timestamp
relative_timestamp = first_datapoint.relative_timestamp
value = first_datapoint.value

#-------------------------------------------------------------------------------
# Scan interface differences
#-------------------------------------------------------------------------------

scan =   list(tmread.get_scans(tmfile))[-1]
c_scan = list(c699util.get_scans(c_tmfile))[-1]

result =   scan.timestamp.unix        # Returns a TMTimestamp object
result = c_scan.unix_timestamp();     # Returns a float

result =   scan.timestamp.relative       # Returns a TMTimestamp object
result = c_scan.relative_timestamp();    # Returns a float

result =   scan.marker_text
result = c_scan.marker_text();

result =   scan.scanpoints   # list of MomaScanPoint objects
result = c_scan.counts();    # list of integers

result =   scan.total_ion_count
result = c_scan.total_ion_count();
```

```
result =   scan.highest_count
result = c_scan.highest_count();

result =   scan.highest_count_bin_num
result = c_scan.highest_count_bin_num();

result =   scan.rf_amp_start
result = c_scan.rf_amp_start();

result =   scan.rf_amp_end
result = c_scan.rf_amp_end();

result =   scan.rf_amp_dac_start
result = c_scan.rf_amp_dac_start();

result =   scan.rf_amp_dac_end
result = c_scan.rf_amp_dac_end();

result =   scan.aux_amp_dac_start
result = c_scan.aux_amp_dac_start();

result =   scan.aux_amp_dac_end
result = c_scan.aux_amp_dac_end();

result =   scan.total_scan_time
result = c_scan.total_scan_time();

result =   scan.peak_counts_per_sec
result = c_scan.peak_counts_per_sec();

result =   scan.src_pkt_type
result = c_scan.src_pkt_type();

result =   scan.num_scans
result = c_scan.num_scans();

result = bool(scan.seb_state.num_scan_status_pkt_updates)
result = c_scan.has_scan_status();
```

```
result =   scan.seb_state.num_scan_status_pkt_updates
result = c_scan.num_scan_status_pkts();

result =   scan.seb_state.scan_bin_time_readback
result = c_scan.seb_bin_time_readback()

result =   scan.seb_state.scan_bin_time_readback_in_seconds
result = c_scan.seb_bin_time_readback_in_seconds()

result =   scan.seb_state.special_scan_id
result = c_scan.seb_special_scan_id()

result =   scan.seb_state.raw_scan_value
result = c_scan.seb_raw_scan_value()

result =   scan.seb_state.scan_value
result = c_scan.seb_scan_value()

result =   scan.seb_state.rf_freq
result = c_scan.seb_rf_freq_hz()

result =   scan.seb_state.last_rf_amp_mon
result = c_scan.seb_last_rf_amp_mon()

result =   scan.seb_state.rf_amp_dac_set
result = c_scan.seb_rf_amp_dac_set()

result =   scan.seb_state.aux_amp_dac_set
result = c_scan.seb_aux_amp_dac_set()

result =   scan.seb_state.counter_sel
result = c_scan.seb_counter_sel()

result =   scan.seb_state.bin_count_set
result = c_scan.seb_bin_count_set()

result =   scan.seb_state.bin_count_reg
result = c_scan.seb_bin_count_reg()
```

```
result =   scan.seb_state.phase_cycle_rdbk
result = c_scan.seb_phase_cycle_rdbk()

result =   scan.seb_state.rf_step_set
result = c_scan.seb_rf_step_set()

result =   scan.seb_state.rf_step_reg
result = c_scan.seb_rf_step_reg()

result =   scan.seb_state.rf_step_reg
result = c_scan.seb_aux_step_set()

result =   scan.seb_state.aux_step_reg
result = c_scan.seb_aux_step_reg()

result =   scan.seb_state.ionization_time
result = c_scan.seb_ionization_time_in_milliseconds()

result =   scan.seb_state.scan_em1
result = c_scan.seb_scan_em1()

result =   scan.seb_state.scan_em2
result = c_scan.seb_scan_em2()

result =   scan.seb_state.srcA_foc_a_dac_set
result = c_scan.seb_src_a_foc_a_dac_set()

result =   scan.seb_state.srcB_foc_a_dac_set
result = c_scan.seb_src_b_foc_a_dac_set()

result =   scan.seb_state.srcA_foc_a_dac_reg
result = c_scan.seb_src_a_foc_a_dac_reg()

result =   scan.seb_state.srcB_foc_a_dac_reg
result = c_scan.seb_src_b_foc_a_dac_reg()

result = bool(scan.seb_state.num_hk_pkt_updates)
result = c_scan.has_seb_hk_pkt();
```

```
result =   scan.seb_state.is_emon_a
result = c_scan.seb_is_emon_a();

result =   scan.seb_state.is_emon_b
result = c_scan.seb_is_emon_b();

result =   scan.count_ions(min_bin=min_bin, max_bin=max_bin)
result = c_scan.selected_ion_count_sum_by_bin(min_bin, max_bin);

result =   scan.count_ions(min_mass=min_mass, max_mass=max_mass)
result = c_scan.selected_ion_count_sum_by_mass(min_mass, max_mass);

result =   scan.max_ion_count(min_bin=min_bin, max_bin=max_bin)
result = c_scan.selected_ion_count_max_by_bin(min_bin, max_bin);

result =   scan.max_ion_count(min_mass=min_mass, max_mass=max_mass)
result = c_scan.selected_ion_count_max_by_mass(min_mass, max_mass);
```

c699util and the 699 Gnuplot module

If you use GnuplotArgumentParser (a class defined in 699util), be sure to pass use_c699util=True to the constructor, so that it will use c699util objects internally. (This is not default behavior because I don't want to break older Python scripts.)

Revision #3
Created 22 March 2023 17:45:00 by Nick Dobson
Updated 22 March 2023 17:53:02 by Nick Dobson