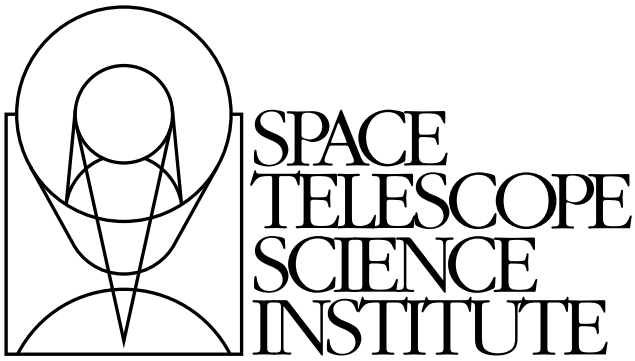

costools Documentation

Release 1.0.6 (14-Aug-2012)

Warren Hack, Nadia Dencheva, Chris Sontag, Megan Sosey, Micha

April 18, 2016

CONTENTS

This package provides data processing tools for working with COS data.

Contents:

CHAPTER

ONE

SAAMODEL

costools.saamodel.**saaModel**(*model*)

Get vertices for SAA model number *model*.

This was copied from UVMpdbelem.cgi, downloaded from: http://www.sesd.stsci.edu/prd/files/UVMpdbelem.cgi?ELEM=pdb/sv

See http://www.sesd.stsci.edu/prd/icd/icd26_pt3_10_svdf.html for a description of the SAA vertex description file.

Parameters

model: int

The SAA model number (0 - 32, inclusive).

Returns

list

List of (latitude, longitude) tuples, one for each vertex.

CHAPTER

TWO

TIMEFILTER

timefilter - filter a corrtag table based on the TIMELINE extension

1. To run this task from within Python:

```
>>> from costools import timefilter
>>> timefilter.TimelineFilter("xyz_corrtag.fits", "temp_corrtag.fits",
                              "sun_alt > 0.")
```

Note: make sure the costools package is on your Python path

2. To run this task using the TEAL GUI to set the parameters under PyRAF:

```
>>> import costools
>>> epar costools.timefilter   # or "teal timefilter"
```

3. To run this task from the operating system command line:

```
# just print info:
% timefilter.py xyz_corrtag.fits

# flag events with sun_alt > 0 as bad, writing output to a new file
# temp_corrtag.fits:
% timefilter.py xyz_corrtag.fits temp_corrtag.fits 'sun_alt > 0'

# clear the bad time interval flag (2048) from the DQ column
% timefilter.py temp_corrtag.fits '' reset
% timefilter.py temp_corrtag.fits '' clear
```

Note: make sure the file "timefilter.py" is on your executable path

class costools.timefilter.**TimelineFilter**(*input*, *output=None*, *filter=None*, *verbose=False*)
Filter a TIME-TAG table by setting a flag in the DQ column.

There are no user-callable methods. Instantiating the class does all the work.

Attributes

input: str	Name of input corrtag file.
output: str	Name of output file (may be None).
filter_str: str	The filter, either as specified by the user, or possibly with further modification.
filter: str	Info about column name, relation, and cutoff, or "info" or "clear".
verbose: bool	True if messages should be printed.
fd: file	File handle for input file.
events_list: list of two integers	[extver, hdunum] for each EVENTS extension.
gti_list: list of two integers	[extver, hdunum] for each GTI extension.
tl_list: list of two integers	[extver, hdunum] for each TIMELINE extension.
events_hdunum: int	HDU number for EVENTS extension.
gti_hdunum: int	HDU number for last GTI extension.
tl_hdunum: int	HDU number for TIMELINE extension.
events_time: array_like	Array of times from the TIME column in the EVENTS table, but copied to a local array in native format. This will initially be set to None, then assigned later if we need the times.
dq: array_like	The DQ column from the EVENTS table.
first_gti_hdunum: int	HDU number for the first GTI extension.
first_gti: list of two-element lists	The contents of the first GTI table. Each element of first_gti is a list giving the start and stop times (in seconds since EXPSTART) for a "good time interval," before any change to the GTI table resulting from filtering by this module.

Set DQ flag to mark bad time intervals.

Parameters

input: str

Name of input corrtag file.

output: str or None

Optional name of output file. If an output file was specified, the input file will be copied to output and possibly modified.

filter: str or None

This string specifies which time intervals should be flagged as bad, based on columns in the TIMELINE extension. filter = "info" means that information about the input file should be printed. filter = "clear" or "reset" means that flag 2048 (bad time interval) should be cleared from the DQ column.

verbose: bool

If True, information will be printed.

clearDqFlag()

Clear (reset) the bad-time-interval flag in the DQ column.

The bit corresponding to the bad-time-interval flag value 2048 will be set to 0 for every row of the DQ column in the EVENTS table.

If there is more than one GTI table, the next to last one will be copied to overwrite the last one (based on keyword EXTVER). This is not foolproof; the last one may record intervals rejected due to FUV bursts,

and this information could be lost. A safer way to clear the bad-time-interval flag would be to go back to a previous version of the file.

findExtensions()
Find EVENTS, GTI and TIMELINE extensions.

This checks each extension in the input file to find all extensions with keyword EXTNAME equal to (case insensitive) "EVENTS" (there should be exactly one), "GTI" (we expect one or two), or "TIMELINE" (we expect one, but a raw file or an old corrtag file might have none).

These three attributes will be assigned by this method:

```
self.events_list        EVENTS tables
self.gti_list           GTI tables
self.tl_list            TIMELINE tables
```

Each element is a two-element list (extver and hdunum) that identifies one extension in the input file. extver is the value of keyword EXTVER, the extension version number. hdunum is the header/data unit number of the extension (primary header is 0).

findHduNum()
Select a header/data unit from each list.

A RuntimeError exception will be raised if there is more than one EVENTS table or more than one TIMELINE table.

These three attributes will be assigned by this method:

```
self.events_hdunum      HDU number of EVENTS table
self.first_gti_hdunum   HDU number of first GTI table
self.gti_hdunum         HDU number of last GTI table
self.tl_hdunum          HDU number of TIMELINE table
```

These are the header/data unit numbers of the EVENTS table, the last (highest EXTVER) GTI table, and the TIMELINE table respectively. The value will be None if there are no elements in the corresponding self.events_list, self.gti_list, or self.tl_list.

getFirstGTI()
Get the contents of the first GTI table.

Attribute first_gti will be assigned by this method.

interpretFilter(*filter*)
Split filter into its parts.

Parameters
filter: str

Specification of how to filter, e.g. column name in TIMELINE table, cutoff value, and whether values to be flagged as bad are greater than the cutoff, less than, etc. filter may alternatively be "info" or "clear" or "reset".

mergeGTI(*first_gti*, *second_gti*, *precision=None*)
Merge two good time intervals tables.

Parameters
first_gti: list of two-element lists

A list of [start, stop] good time intervals. This is the list from the first GTI table.

second_gti: list of two-element lists

A second list of [start, stop] good time intervals. This is the list based on the DQ column.

Returns:

A new gti list, consisting of intervals that overlap both first_gti and second_gti.

printInfo()
Print information about the input file.

The information printed includes:

- The names of the input and output files.
- The good-time intervals table (the one with largest EXTVER).
- **The following values at the beginning, middle, and end of the**

  range of times in the TIMELINE TIME column:
  -sun altitude, target altitude, longitude, latitude, shift1.
- The minimum, maximum, median, of shift1, ly_alpha, darkrate.

recomputeExptime()
Compute the exposure time and update the EXPTIME keyword.

Returns
gti: list of two-element lists

Each element of gti is a two-element list, the start and stop times (in seconds since EXPSTART) for a "good time interval."

roundGTI(*input_gti*, *precision=3*)
Round the start and stop times to precision decimals.

Parameters
input_gti: list of two-element lists

A list of [start, stop] good time intervals.

precision: int

The number of decimal places for rounding.

Returns:

A new gti list with times rounded off.

saaFilter(*filter_col*, *model*)
Flag within the specified SAA contour as bad.

Parameters
filter_col: arbitrary

This argument is not used. It's included so that the calling sequence will be the same as functions np.greater, etc.

model: int

The SAA model number. Currently these range from 2 to 32 inclusive. (Models 0 and 1 are radio frequence interference contours.)

Returns
flag: array_like

This is a boolean array, one element for each row of the TIMELINE table. True means that HST was within the SAA contour (specified by model) at the time corresponding to the TIMELINE row.

saveNewGTI(*gti*)
Append new GTI information as a BINTABLE extension.

Create and save a GTI extension. If there is no GTI extension, or if there is only one, the new GTI will be appended as a new extension. If there are already two or more GTI extensions, the last one (highest EXTVER) will be replaced.

Parameters
gti: list of two-element lists

A list of [start, stop] good time intervals.

setDqFlag()
Set the bad-time-interval flag in the DQ column.

The bit corresponding to the bad-time-interval flag value (2048) will be set to 1 in the DQ column in the EVENTS table for each event that is within a time interval that is "bad," according to the filter specified by the user.

The time resolution (0.032 s) in the EVENTS table is finer than that of the TIMELINE table (1 s). The bad time intervals are determined by using the TIMELINE table; the flagging can therefore be off by some fraction of a second.

The GTI table will be updated. If the input file has two or more GTI table extensions, the last one (highest EXTVER) will be overwritten with the new good time intervals; otherwise, a new GTI extension will be appended to the file.

shift1Info(*shift1*, *cutoff*)
Compute information about the SHIFT1 column in TIMELINE.

Parameters
shift1: array_like

Array of SHIFT1[abc] values during the exposure.

cutoff: float

The cutoff value specified by the user; in this case this will be interpreted as the factor by which the standard deviation of SHIFT1 is to be multiplied in order to get the actual cutoff value.

Returns
tuple of two floats

The first element is the median of the shift1 values, taken after clipping outliers. The second element is the cutoff value specified by the user multiplied by the standard deviation of the sigma-clipped shift1 values.

writeNewOutputFile()
Write the current HDUList to a new output file.

costools.timefilter.**expandFilename**(*filename*)
Expand environment variables to get the real file name.

Parameters
filename: str

A file name.

Returns
str

The real file name.

costools.timefilter.**findMedian**(*x*)
Compute the median of x.

Parameters
x: array_like

An array that can be sorted.

Returns
median_x: float

If there are an odd number of elements in x, median_x is the middle element; otherwise, median_x is the average of the two middle elements.

costools.timefilter.**getHelpAsString**(*fulldoc=True*)
Return help info from <module>.help in the script directory

costools.timefilter.**help**()

costools.timefilter.**main**()
Filter a corrtag file using its timeline extension.

costools.timefilter.**prtOptions**()
Print a list of command-line options and arguments.

costools.timefilter.**run**(*configobj=None*)
TEAL interface for running this code.

costools.timefilter.**testWithinSAA**(*hst*, *vertices*, *middle_SAA*)
Test whether HST is within the polygon for an SAA contour.

Parameters
hst: array_like

Unit vector pointing from the center of the Earth toward the location of HST at a particular time.

vertices: array_like, shape (nvertices,3)

vertices[i] is a unit vector from the center of the Earth toward vertex number i of a polygon that defines one of the SAA contour.

middle_SAA: array_like

Unit vector from the center of the Earth toward a point near the middle of the SAA region. This is for making a quick check that hst is close enough to the SAA contour to be worth making a detailed check.

Returns
boolean

True if hst is within the SAA contour defined by vertices.

costools.timefilter.**toRect**(*longitude*, *latitude*)
Convert longitude and latitude to rectangular coordinates.

Parameters
longitude: float

longitude in degrees.

latitude: float

latitude in degrees.

Returns

array_like

Unit vector in rectangular coordinates.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

C