costools Documentation

Release 1.2.4

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

Dec 21, 2021

CONTENTS

1 costoolsutil 3

2 saamodel 5

3 splittag 7

4 timefilter 9

5 x1dcorr 15

6 Indices and tables 17

Python Module Index 19

Index 21

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

Contents:

CHAPTER

ONE

COSTOOLSUTIL

costools.costoolsutil.**createOutputDirectory**(*outdir*)
Check whether outdir exists, and create it if necessary.

If outdir was specified but doesn't exist, create it.

Parameters

outdir: str or None Name of output directory.

costools.costoolsutil.**expandFilename**(*filename*)
Get the actual file name.

Parameters

filename: str A file or directory name.

Returns

full_file_name: str The real directory name.

costools.costoolsutil.**splitInputString**(*input*)
Split on comma and/or space.

Parameters

input: str One or more values (e.g. file names), separated by a comma and/or a space.

Returns

words: list of strings

costools.costoolsutil.**uniqueInput**(*infiles*, *unique=False*)
Remove effective duplicates from the list of files.

This function also expands environment variables and wildcards. Aside from that, the order of the input file names will be preserved.

Parameters

infiles: list of strings List of input file names.

unique: bool If unique is True, drop duplicates and effective duplicates from the list of file names. "Effective duplicate" means that the root names are the same, but the segment might differ.

Returns

unique_files: list of strings The list of input files but with duplicates removed.

CHAPTER

TWO

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/svdf.dat

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

THREE

SPLITTAG

splittag - Split TIME-TAG files into multiple files.

1. To run this task from within Python:

```
>>> import costools
>>> from stsci.tools import teal
>>> teal.teal("splittag")

or:

>>> import costools
>>> costools.splittag.splittag("rootname_corrtag_a.fits", "split3s",
        starttime=0., increment=3., endtime=1000.,
        time_list="")

or:

>>> from costools import splittag
>>> splittag.splittag("rootname_corrtag_a.fits", "split250s",
                      starttime=None, increment=None, endtime=None,
                      time_list="0, 250, 500, 750, 1000")
```

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
>>> teal splittag                    # or 'epar splittag'
```

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

costools.splittag.**help**()

costools.splittag.**main**()
Split corrtag files by time.

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

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

costools.splittag.**splittag**(*infiles*, *outroot*, *starttime=None*, *increment=None*, *endtime=None*, *time_list=None*, *verbosity=1*)

CHAPTER

FOUR

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.

Methods

clearDqFlag()	Clear (reset) the bad-time-interval flag in the DQ column.
findExtensions()	Find EVENTS, GTI and TIMELINE extensions.
findHduNum()	Select a header/data unit from each list.
getFirstGTI()	Get the contents of the first GTI table.
interpretFilter(filter)	Split filter into its parts.
mergeGTI(first_gti, second_gti[, precision])	Merge two good time intervals tables.
printInfo()	Print information about the input file.
recomputeExptime()	Compute the exposure time and update the EXPTIME keyword.
roundGTI(input_gti[, precision])	Round the start and stop times to precision decimals.
saaFilter(filter_col, model)	Flag within the specified SAA contour as bad.
saveNewGTI(gti)	Append new GTI information as a BINTABLE extension.
setDqFlag()	Set the bad-time-interval flag in the DQ column.
shift1Info(shift1, cutoff)	Compute information about the SHIFT1 column in TIMELINE.
writeNewOutputFile()	Write the current HDUList to a new output file.

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

FIVE

X1DCORR

x1dcorr - Extract 1-D spectra.

1. To run this task from within Python:

```
>>> from costools import x1dcorr
>>> x1dcorr.x1dcorr(["rootname_corrtag_a.fits",
                     "rootname_corrtag_b.fits"], outdir="out/",
                    update_input=True,
                    find=True, cutoff=5., verbosity=2)

>>> x1dcorr.x1dcorr(["rootname_corrtag_a.fits"], outdir="out/",
                    update_input=True,
                    find=False,
                    location=527.88, extrsize=15,
                    verbosity=2)
>>> x1dcorr.x1dcorr(["rootname_corrtag_b.fits"], outdir="out/",
                    update_input=True,
                    find=False,
                    location=586.68, extrsize=15,
                    verbosity=2)

>>> x1dcorr.x1dcorr(["rootname_corrtag.fits"], outdir="out/",
                    find=False,
                    location=[196.84, 290.87, 424.62],
                    extrsize=[15, 15, 15])
```

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
>>> teal x1dcorr                        # or 'epar x1dcorr'
```

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

```
# Extract a 1-D spectrum xxx
```

costools.x1dcorr.**getHelpAsString**(*fulldoc=True*)
Return help info from <module>.help in the script directory

costools.x1dcorr.**help**()

costools.x1dcorr.**main**()
Run the CalCOS 1-D extraction function.

costools.x1dcorr.**prtOptions**()
Print a list of command-line options and arguments.

costools.x1dcorr.**run**(*configobj=None*)
TEAL interface for running this code.

costools.x1dcorr.**x1dcorr**(*input*, *outdir=''*, *update_input=False*, *find=False*, *cutoff=None*, *location=''*, *extrsize=''*, *verbosity=1*)

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

costools.costoolsutil, 3
costools.saamodel, 5
costools.splittag, 7
costools.timefilter, 9
costools.x1dcorr, 15

INDEX

C

clearDqFlag() (*costools.timefilter.TimelineFilter method*), 10
costools.costoolsutil
    module, 3
costools.saamodel
    module, 5
costools.splittag
    module, 7
costools.timefilter
    module, 9
costools.x1dcorr
    module, 15
createOutputDirectory() (*in module costools.costoolsutil*), 3

E

expandFilename() (*in module costools.costoolsutil*), 3
expandFilename() (*in module costools.timefilter*), 13

F

findExtensions() (*costools.timefilter.TimelineFilter method*), 11
findHduNum() (*costools.timefilter.TimelineFilter method*), 11
findMedian() (*in module costools.timefilter*), 13

G

getFirstGTI() (*costools.timefilter.TimelineFilter method*), 11
getHelpAsString() (*in module costools.splittag*), 7
getHelpAsString() (*in module costools.timefilter*), 13
getHelpAsString() (*in module costools.x1dcorr*), 15

H

help() (*in module costools.splittag*), 7
help() (*in module costools.timefilter*), 13
help() (*in module costools.x1dcorr*), 15

I

interpretFilter() (*costools.timefilter.TimelineFilter method*), 11

M

main() (*in module costools.splittag*), 7
main() (*in module costools.timefilter*), 13
main() (*in module costools.x1dcorr*), 16
mergeGTI() (*costools.timefilter.TimelineFilter method*), 11
module
    costools.costoolsutil, 3
    costools.saamodel, 5
    costools.splittag, 7
    costools.timefilter, 9
    costools.x1dcorr, 15

P

printInfo() (*costools.timefilter.TimelineFilter method*), 12
prtOptions() (*in module costools.splittag*), 7
prtOptions() (*in module costools.timefilter*), 14
prtOptions() (*in module costools.x1dcorr*), 16

R

recomputeExptime() (*costools.timefilter.TimelineFilter method*), 12
roundGTI() (*costools.timefilter.TimelineFilter method*), 12
run() (*in module costools.splittag*), 7
run() (*in module costools.timefilter*), 14
run() (*in module costools.x1dcorr*), 16

S

saaFilter() (*costools.timefilter.TimelineFilter method*), 12
saaModel() (*in module costools.saamodel*), 5
saveNewGTI() (*costools.timefilter.TimelineFilter method*), 12
setDqFlag() (*costools.timefilter.TimelineFilter method*), 12
shift1Info() (*costools.timefilter.TimelineFilter method*), 13
splitInputString() (*in module costools.costoolsutil*), 3
splittag() (*in module costools.splittag*), 8

T

testWithinSAA() (*in module costools.timefilter*), 14
TimelineFilter (*class in costools.timefilter*), 9
toRect() (*in module costools.timefilter*), 14

U

uniqueInput() (*in module costools.costoolsutil*), 3

W

writeNewOutputFile() (*costools.timefilter.TimelineFilter method*), 13

X

x1dcorr() (*in module costools.x1dcorr*), 16