IDF Cookbook

Bernard GODARD

Version 1.0, 4 November 2004

Contents

1 Introduction

CalFUSE, the FUSE calibration pipeline, was originally designed before the launch of the satellite. As unexpected instrumental effects were discovered during the telescope operations, changes were made to the pipeline to correct for these, leading to a cumbersome design.

In 2002, a new pipeline design was conceived. The goal was not only to take into account all the newly discovered instrumental effects, but to make it easier to add further instrumental corrections. The new design is more flexible, thus maintenance is easier. Moreover it makes the pipeline work considerably faster.

Up to version 2.4, CalFUSE converts time-tag data to a 2-dimensional image in an early step, making it hard to add a module to correct for time-dependent effects such as mirror motions. CalFUSE 3 keeps the data in the format of a time tagged events list until the extraction. Histogram files are also converted to a list of events, but in that case all events are tagged with the same time value.

This events list is stored in the Intermediate Data File (IDF). This FITS file is an unified input/output format for most of the pipeline modules. The pipeline subroutines read or write the keywords, rows and columns in this file. The unified input/output format for most of the pipeline routines means that the order of operation of the pipeline modules can be changed (to a certain extent) or new modules can easily be added.

The Intermediate Data File (IDF) is at the center of the new design. It is a FITS file with 4 Header Data Units (HDU).

The first HDU consists of the header originally copied from the raw file which gets modified as the IDF goes through the different pipeline steps. It contains information about the proposal, the exposure, the observation and the calibration as well as engineering and housekeeping data.

The second HDU is a time-tagged list of events with their raw XY positions, weights, pulse height, as well as other parameters set to dummy values at the creation of the file (like their XY corrected positions and assigned wavelengths) and computed when the IDF goes through the appropriate pipeline module.

The third HDU is the list of good time intervals.

The fourth and last HDU lists, for each second of the exposure, housekeeping parameters like limb angle, orbital velocity, latitude and longitude, as well as the LiF and SiC Y centroids. If present, a housekeeping file, gen-

erated from the engineering telemetry, is used to fill some columns of this time-line. Otherwise these columns are filled with the best guess from the engineering keywords of the main header. This is the first time the pipeline makes use of the housekeeping files. It allows for a time-dependent correction of such effects as dead-time and count-rate Y distortion as well as screening for detector high voltage variations.

Since in the IDF file events are flagged as good or bad, but never discarded, users can change their selection criteria without rerunning the pipeline.

This document deals with the analysis tools that work on IDF files. The tools described are from CalFUSE 3.0.8. If you are using a different version of the pipeline, the examples in this document might not work. Note also that a lot of these tools are not available in pipeline 3.0 before 3.0.8. The goal of this document is to help you:

- modify IDF files to get better output spectra.
- extract interesting information from IDF files.

2 Combining IDF files

IDF files for different exposures can be combined into a single one. This results in a higher Signal to Noise ratio, allowing a better computation of the background and the Y centroid. Thus it is recommanded to combine your exposures before extracting, especially if your target is faint.

The exposures to be combined should generally come from the same observation (or two observations of the same target in the same aperture that are close in time) , the same detector, and the same segment.

2.1 Command line method

You can combine your IDF files from the command line with **idf_combine**.

```
Usage:
  idf_combine [-ahbc]  [-v level] output_idf_file input_idf_files

Options:
  -h:  this help message
  -v:  verbosity level (=1; 0 is silent)
```

```
-a:  ignore EXP_STAT keyword
-c:  recalculates Y centroids (use target events)
-b:  store ORBITAL_VEL in a float
```

Here is how to combine all exposures of observation M1010101 for segment 2A.

```
idf_combine -c  M1010101all2attagfidf.fit M1010101*2attagfidf.fit
```

Without the **c** switch, the Y centroid of the LiF and SiC target apertures wouldn't be re-calculated. The EXP_ID keyword in the main header of the output file is set to '999'. Thus after extraction, you will get the two files:

```
M10101019992alifttagfcal.fit
M10101019992asicttagfcal.fit
```

The **a** switch makes the program ignore the EXP_STAT keyword. This means that exposures that have EXP_STAT strictly positive will not be rejected as would be the case without the **a** switch. The EXP_STAT keyword is set to a strictly postive value by the pipeline if the exposure is bad or for histogram (HIST) files that contain bursts, SAA crossing or low limb angle: these exposures contain many bad events that cannot be screened because HIST files are not time-tagged.

The **b** switch will be discussed later.

2.2 GUI method

You can also use the IDL program CF_EDIT to combine your exposures. Open all the IDF files you want to combine. Don't forget to tell CF_EDIT to recalculate the Y centroids (see the CF_EDIT user manual) and save the result. With CF_EDIT, you can also align your exposures on a spectral feature as well as modify the screening parameters.

2.3 Combining the Bad Pixels Maps

After combining your IDF files and before extracting, you should generate a Bad Pixels Map (BPM) file for your combined file. This is done with **bpm_combine**. **cf_bad_pixels**, the pipeline BPM generator generates a BPM file for each exposure from its IDF file (and if present the associated

housekeeping and jitter files), but cannot generate a BPM for an IDF file containing several exposures. **bpm_combine** will combine the BPM files in the same way idf_combine or CF_EDIT combine IDF files. It takes into account the possible alignment performed on the exposures with CF_EDIT.

The syntax for **bpm_combine** is:

```
bpm_combine  M1010101all2attagfbpm.fit M1010101all2attagfidf.fit
```

The program uses the information in the main header of the combined IDF file to find which IDF files were used to make it. It then opens these IDF files to find the associated BPM files filenames. Last it opens the BPM files for each exposures and combines them. For this program to work, all the IDF files that were used to generate the combined IDF file as well as their associated BPM files must be in the current directory.

3 Screening IDF files

Selecting the night-time events only for extraction to make the geocoronnal lines disappear or keeping the events that happen during a burst are two examples of what can be done with **idf_screen** or CF_EDIT. Screening concerns only time-tagged (TTAG) IDF files.

3.1 Command line method

idf_screen allows the user to alter the screening performed by the pipeline.

```
Usage:
  idf_screen [-h]  [-v level] input_idf_file output_idf_file
                              timeflag value

Arguments :
  timeflag :  USER/JITR/OPUS/BRST/HV/SAA/LIMB/DAY
  value    :  GOOD/BAD/EITHER

Options:
  -h:  this help message
  -v:  verbosity level (=1; 0 is silent)
```

The following example selects only the night-time events.

```
idf_screen M10101010011attagfidf.fit M10101010011a_night_ttagfidf.fit \
           DAY GOOD
```

This just modifies the header keyword DAYNIGHT and recalculates some other header keywords like EXPTIME and EXPNIGHT. For timeflags other than DAY, the behaviour is different. It modifies the events list and timeline.

```
idf_screen M10101010011attagfidf.fit M10101010011a_ignorebursts_ttagfidf.fit \
           BRST EITHER
```

In the above example, the timeline and events BURSTS timeflags will all be set to zero.

```
idf_screen M10101010011attagfidf.fit M10101010011a_jitter_ttagfidf.fit \
           JITR BAD
```

In the above example, the timeline and events list JITR timeflags will all be inverted. Thus the extraction routine will think that the events where the jitter is good are the events where in fact the jitter is bad. It is then tricked into extracting only the events for which the jitter is actually bad (and the other conditions are satisfied).

Note that using one of the timeflags other than DAY with the value GOOD doesn't change the screening parameters. In that case, **idf_screen** doesn't generate the output file.

idf_screen does not update the GTIs table.

3.2 GUI method

With CF_EDIT, you can perform screening in the same way as **idf_screen** does, using the EVENTS SELECTION button, with the added advantage of seeing in real-time the effects of your change. CF_EDIT also allows PHA screening and selection of Good Time Intervals (GTIs). For more information, consults the CF_EDIT user manual.

3.3 Screening, Combining and the Bad Pixels Map

Once you have modified the screening parameters (except for pulse-height), you need to regenerate the BPM file with **cf_bad_pixels**. Since **cf_bad_pixels** doesn't operate on a combined file, if you use CF_EDIT to simultaneously

combine and screen a file, you won't be able to generate a BPM file for the output.

If you really care about the bad-pixels (for example if there is a pothole in a spectral feature you are interested in) , here is the way to do it. Let us assume, we have a stack of exposures M1010101*1attagfidf.fit that we want to combine, selecting only the night-time events. Run **idf_screen** or CF_EDIT on each exposures selecting only the night-time events. The result is a stack of IDF files M1010101*1a_night_ttagfidf.fit. On each of these files, run **cf_bad_pixels**. Combine the screened IDF files with **idf_combine** or CF_EDIT. Run **bpm_combine** on the combined IDF file.

Here is a shell script that does all of this:

```
#!/bin/sh
#
# Takes one argument : observation rootname (example: M1010101)
# Selects the night-time events only
# Combines all the IDF files
# Creates the associated BPM file
# This for each segment: 1a, 1b, 2a and 2b
#

rm *ttagfbpm.fit
rm *_night_ttagfidf.fit


for seg in "1a" "1b" "2a" "2b"
do
    str=${1}[0-8]??${seg}ttagfidf.fit
    exposures=`ls $str`
    for expo in $exposures
    do
        exposcreen=`echo $expo | sed -e 's/ttag/_night_ttag/g`
        idf_screen $expo $exposcreen DAY GOOD
        cf_bad_pixels $exposcreen
    done
    str2=${1}[0-8]??${seg}_night_ttagfidf.fit
    idf_combine -c ${1}all${seg}_night_ttagfidf.fit $str2
    bpm_combine ${1}all${seg}_night_ttagfbpm.fit \
```

```
                ${1}all${seg}_night_ttagfidf.fit
done

exit 0
```

4 Removing the orbital motion of a source in a close binary system

If you have a TTAG observation of a source in a close binary system, you might be interested in removing the orbital motion that smears out the photospheric lines. For a circular orbit, you can use **remove_target_orbital_motion**.

```
Usage:
  remove_target_orbital_motion [-hb] [-v level]
    input_idf_file output_idf_file
    mjd0 period vrmax
    [ra_h ra_m ra_s dec_d dec_m dec_s]

Arguments:
    mjd0      :  (Geocentric or Heliocentric) Modified Julian Day
                 of closest approch on orbit.
    period    :  Period in seconds.
    vrmax     :  Maximum radial velocity in km/s
    ra_h ra_m ra_s dec_d dec_m dec_s  :  Optional RA and DEC of target.
                                         If present mjd0 is interpreted
                                         as Heliocentric.

Options:
  -h:  this help message
  -v:  verbosity level (=1; 0 is silent)
  -b:  update timeline doppler information (required for BPM)
```

Note that you should have:

$$\frac{(obstime - mjd0)}{T} * \delta T \ll T$$

where T is the orbital period, dT the uncertainty on the period, *obstime* the time of the observation and *mjd0* a modified julian day for which the target is at its closest approch on an orbit. If this is not the case or if you don't know *mjd0*, you should try with different values of *mjd0* going from *obstime* to *obstime*+T.

If *mjd0* and *obstime* are not close in time, you should take into account the heliocentric light-time delay (Note that the light-time delay due to the systemic velocity is generally already accounted for in the period as measured from Earth on a long time-base). Thus you should probably use the heliocentric version of **remove_target_orbital_motion**, which means that you need to input the RA and DEC of the target. If your value of mjd0 is geocentric, you will need to convert it to heliocentric. You can use **mjd2hjd** to do so.

```
Usage:
  mjd2hjd [-h] [-v level] mjd ra_h ra_m ra_s dec_d dec_m dec_s

Arguments:
    mjd                                 :  Modified Julian Day.
    ra_h ra_m ra_s dec_d dec_m dec_s    :  RA and DEC of target.

Options:
  -h:  this help message
  -v:  verbosity level (=1; 0 is silent)
```

The **b** switch in **remove_target_orbital_motion** tells the program to update the timeline doppler information in the output IDF file. This is required if you want to correct for potholes. In that case you should run **cf_bad_pixels** on the output of **remove_target_orbital_motion**. This also means that if you want to correct for bad pixels, you should not have combined your exposures into a single IDF file before removing the orbital motion.

The IDF file with updated doppler information uses a bigger container for doppler information than a normal IDF file because close binary systems can have very large orbital velocity. **idf_combine** can read these files, but you may get an overflow when it tries to write the output file. To prevent this, use the **b** switch with **idf_combine** to combine such files.

The following script remove the target orbital motion, combines the exposures into a single IDF file and generates an associated BPM file.

```
#!/bin/sh
#
# Takes 10 arguments :
# 1 : observation rootname (example: M1010101)
# 2 : Heliocentric Modified Julian Day of closest approch on orbit.
# 3 : period in seconds
# 4 : Maximum radial velocity in km/s
# 5 6 7 : RA of target  - h  , min, sec
# 8 9 10: DEC of target - deg, min, sec
#
# Removes the orbital motion
# Combines all the IDF files
# Creates the associated BPM file
# This for each segment: 1a, 1b, 2a and 2b
#

rootname=$1
hmjd=$2
period=$3
vmax=$4
rah=$5
ram=$6
ras=$7
decd=$8
decm=$9
shift 1
decs=$9

rm *ttagfbpm.fit
rm *_corr_ttagfidf.fit


for seg in "1a" "1b" "2a" "2b"
do
    str=${rootname}[0-8]??${seg}ttagfidf.fit
    exposures=`ls $str`
    for expo in $exposures
    do
```

```
       expocorr=‘echo $expo | sed -e ’s/ttag/_corr_ttag/g‘
       remove_target_orbital_motion -b $expo $expocorr \
                                       $hmjd $period $vmax \
                                       $rah $ram $ras \
                                       $decd $decm $decs
       cf_bad_pixels $expocorr
    done
    str2=${rootname}[0-8]??${seg}_corr_ttagfidf.fit
    idf_combine -c -b ${rootname}all${seg}_corr_ttagfidf.fit $str2
    bpm_combine ${rootname}all${seg}_corr_ttagfbpm.fit \
                ${rootname}all${seg}_corr_ttagfidf.fit
done

exit 0
```

5 Time-resolved Spectroscopy

The **idf_cut** program allows you to cut an IDF file into several smaller IDF files into which the records of the input IDF files are sorted according to their time-phase.

```
 Usage:
  idf_cut [-hm]  [-v level] idf_file RefTime Period Nout

Arguments:
  RefTime :  Reference Time in seconds since EXPSTART
  Period  :  Period in seconds
  Nout    :  Number of output files.

Options:
  -m:  interprets RefTime as MJD
  -h:  this help message
  -v:  verbosity level (=1; 0 is silent)
```

The name of the *Nout* output IDF files are

```
{input_IDF_filename}.p{X}.fit where X=0..Nout-1
```

An event from the input file gets sorted into output file X if it happened at time t such that there exists an integer k that satisfies :

$$\frac{X}{Nout}T \leq t - RefTime - kT < \frac{X+1}{Nout}T$$

where T is the period.

The **m** switch is used to specify that *RefTime* is a Geocentric Modified Julian Day instead of a number of seconds since start of exposure.

If you just want to cut your files in time bins without folding on a period, use start of exposure for the reference time, end of exposure minus start of exposure for the period and choose the number of bins you want with the last parameter.

If you are running this program on a combined IDF files, note that some of the output files may be empty, because of the gaps between the exposures.

If you care about the bad pixels correction, you need to run **cf_bad_pixels** on each of the output files. Again this means that at this point you should not have combined the exposures.

6 Time-resolved Photometry

6.1 Extracting the lightcurve

You can extract the lightcurve from an IDF file with **ttag_lightcurve**.

```
Usage:
  ttag_lightcurve [-hf] [-v level] input_file output_file windows_file
                                    bins LiForSiC

Arguments:
  input_file     : ttag IDF file

  output_file    : ASCII file with 2 columns:
                         - time (MJD)
                         - countrate corrected for deadtime

  windows_file   : input ASCII file with 2 columns :
                         - start of spectral windows
```

```
                        - stop of spectral windows

  bins            : bin size in seconds
  LiForSiC        : 1=LiF, 2=SiC

Options:
  -h:  this help message
  -f:  output calibrated flux (erg/cm2/s) instead of countrate
  -v:  verbosity level (=1; 0 is silent)
```

All photons that are not in the selected channel, that are not in the spectral windows (defined in the spectral windows file), that have LOCATION_FLAGS not equal to zero or that are not in the good times are not selected. Thus the lightcurve extraction result depends on the way you screened the IDF file.

The **f** switch makes the program output the calibrated flux instead of the countrate. Note that no background substraction is performed.

The output lightcurve is not defined outside the good time intervals. The output ASCII file is tagged with the Geocentric Modified Julian Day. To convert the time-tag from Geocentric to Heliocentric in the whole file, use **ttag_lightcurve_mjd2hmjd**:

```
Usage:
  ttag_lightcurve_mjd2hmjd [-h] [-v level] output_file input_file
                                    hh mm ss dd mm ss

Arguments:
  input_file      : ASCII file with 2 columns :
                      - time (MJD)
                      - signal

  output_file     : ASCII file with 2 columns :
                      - time (HMJD)
                      - signal

  hh mm ss dd mm ss : RA and DEC of target

Options:
```

```
  -h:  this help message
  -v:  verbosity level (=1; 0 is silent)
```

6.2 Adding lightcurves

If you want to sum the lightcurves from different channels, use **ttag_lightcurve_channel_sum**

```
Usage:
  ttag_lightcurve_channel_sum [-h] [-v level] output_file
                  input_file1 input_file2

Arguments:
  input_files    : ASCII files with 2 columns :
                     - time
                     - signal

  output_file    : ASCII file with 2 columns :
                     - time
                     - signal

Options:
  -h:  this help message
  -v:  verbosity level (=1; 0 is silent)
```

You can only sum 2 files at a time.

Note that the difference *dt* between the two first times appearing in the first input file is critical as explained below.

For each time *t* appearing in the first input file, the program looks for a time *t2* appearing in the second input file that has value between *t-dt/2* and *t+dt/2*. If *t2* exists then *t* will appear in the output file.

Only the times that are common (in the way described above) to both inputs will appear in the output file.

6.3 Combining lightcurves

If you want to combine lightcurves for different exposures or observations, use **ttag_lightcurve_combine**

```
Usage:
```

```
  ttag_lightcurve_combine [-h]  [-v level] output_file input_files

Options:
  -h:  this help message
  -v:  verbosity level (=1; 0 is silent)
```

6.4 Spectral Power Density Estimation

To generate a periodogram for the lightcurve, use **ttag_lightcurve_periodogram**

```
Usage:
  ttag_lightcurve_periodogram [-h] [-v level] input_file output_file
                                                 minf, maxf, stepf
Arguments:
  input_file  : an ASCII file with 2 columns :
                  - time (JD or MJD, Helio- or Geo-centric)
                  - countrate

  output_file : an ASCII file with 2 columns :
                  - frequency (Hz)
                  - normalized estimated power spectral density

  minf        : start frequency (Hz)
  maxf        :   end frequency (Hz)
  stepf       :  frequency step (Hz)

Options:
  -h:  this help message
  -v:  verbosity level (=1; 0 is silent)
```