

# ASTRO-F FIS/Survey Data Structure

H. Baba, I. Yamamura, A. Kawamura, S. Makiuti, H. Kaneda, T. Nakagawa and C. Yamauchi

December 4, 2006

## Revised Record

2004 Aug 11:	Ver. 3.0	Completely revised for TSD3.
2004 Sep 1:	Ver. 3.0.1	Minor correction.
2004 Oct 6:	Ver. 3.0.2	Add app.2 and correction in AOCU/ADS.
2004 Oct 15:	Ver. 3.1	Added PR(Pointing Reconstucution) module to clarify position-related information.
2004 Oct 20:	Ver. 3.1.1	Renamed ADS to GADS
2005 Dec 27:	Ver. 3.99.901	Alpha release of 4.0.0. <ul style="list-style-type: none"><li>- Changed ‘<code>mposcnt</code>’ definition and added ‘<code>mposcntorg</code>’ in the FIS_OBS extension.</li><li>- Renamed/moved/added a number of columns and revised the status bits in the FIS_HK,IRC_HK,HK_2,AOCU and GADS extensions, to follow to the latest output SIB.</li><li>- Reversed/clarified the bit order of status column in each extension.</li></ul>
2006 Jan 6:	Ver. 3.99.902	Alpha2 release of 4.0.0. <ul style="list-style-type: none"><li>- Removed ‘<code>mposcntorg</code>’ in the FIS_OBS extension.</li></ul>
2006 Jan 10:	Ver. 3.99.903	Beta release of 4.0.0.
2006 Jan 23:	Ver. 3.99.904	RC1 of 4.0.0. <ul style="list-style-type: none"><li>- Revised the status of HK_2 and AOCU extensions.</li></ul>
2006 Feb 8:	Ver. 3.99.905	RC2 of 4.0.0. <ul style="list-style-type: none"><li>- Added ‘<code>untrusted_frame</code>’ flag.</li></ul>
2006 Mar 1:	Ver. 3.99.906	RC3 of 4.0.0. <ul style="list-style-type: none"><li>- Changed the <code>tform</code> of ‘<code>pimti</code>’ and ‘<code>fisti</code>’ to 64bits integer.</li></ul>
2006 Mar 18:	Ver. 3.99.907	RC4 of 4.0.0. <ul style="list-style-type: none"><li>- Moved IRC temperature data from IRC_HK to HK_2 extension.</li><li>- Revised the status of HK_2 and FIS_HK extensions.</li></ul>
2006 Mar 23:	Ver. 4.0	Added notes of I/O methods.
2006 Dec 4:	Ver. 4.0.1	Minor TSD structure changes. <ul style="list-style-type: none"><li>- Added the status of STTs to the AOCU HDU</li><li>- Changed column name from ‘<code>RANG</code>’ to ‘<code>ROLL</code>’ in AOCU, GADS and PR.</li><li>- Changed column name from ‘<code>SAT_x</code>’ to ‘<code>SAT_POSx</code>’ in SE.</li></ul>

## Revised Record before Ver. 3.0

2002 Jun 4:	Ver. $\alpha$	Initial draft based on Japanese text ver.2.2 (02/05/30).
2002 Jun 6:	Ver. $\beta$	Final draft before Kent meeting.
2002 Aug 8:	Ver. 1.0	Draft before meeting held at Univ. of Tokyo.
2002 Nov 1:	Ver. 1.1	Renamed time-line data to time-series data.
2002 Nov 22:	Ver. 1.2	Updated some status and flag names.
2003 Sep 1:	Ver. 1.3	Revised definitions of functions.
2003 Nov 26:	Ver. 2.0 $\alpha$	Re-built structure.
2003 Dec 3:	Ver. 2.0 $\beta$	For a meeting in 1st IT (not completed yet).
2003 Dec 18:	Ver. 2.0	Completely revised for TSD2.
2004 Jan 8:	Ver. 2.0.1	Minor correction.
2004 Jan 27:	Ver. 2.0.2	Minor correction in FIS_HK.

## 1 Purpose of this document

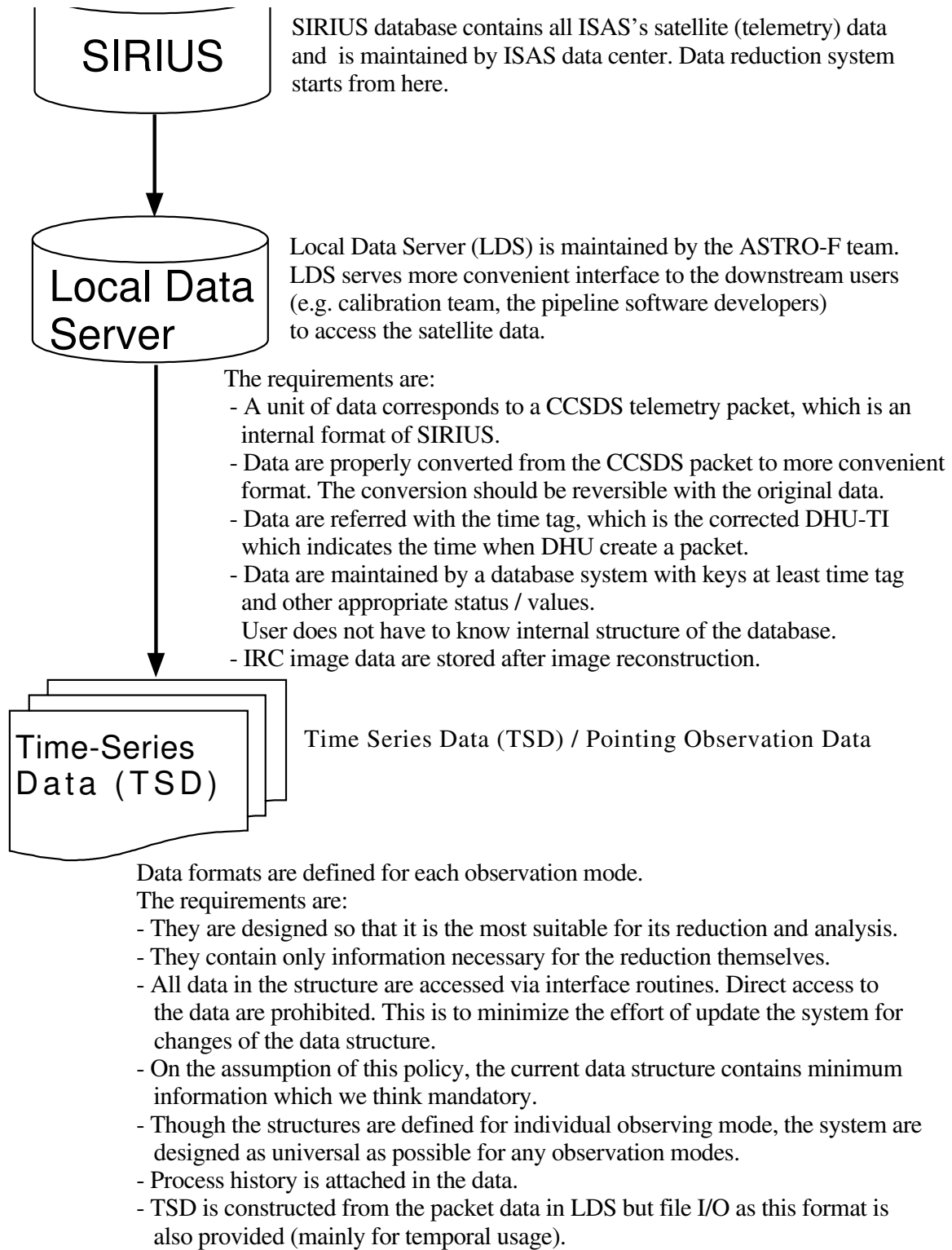
A proper data structure must be designed for efficient processing of the ASTRO-F/FIS survey data. This document describes the data reduction platform, the data structure and interface functions for the ASTRO-F/FIS survey pipeline. Policy of the data handling and overview of data-flow are also described.

## 2 Requirement

We set following requirements for the data reduction platform.

- The data structure must be flexible for possible change of elements.
- All elements are accessed via interface routines. Direct access to the member elements of data structure is not allowed.
- Pointing observation data and IRC data are to be treated basically in the same system, with a minimum changes of structure and interface routines.
- The system are easy to be maintained.
- Most of the system are implemented in IDL (Interactive Data Language).
- The system are to be as exportable as possible. Trivial incompatibility between the system such as byte order between Intel CPU and others must be taken into account.

### 3 Outline of data-flow



## 4 Time-series Data (TSD) : Overview and structure

### 4.1 Overview

An overview of the TSD structure is presented. The data consists of a header part and a data array. A record (corresponds to a sampling of detector) consists of the instrument data (mainly FIS or IRC) and necessary information from other house keeping (HK) instruments. Usually the sampling rate of HK-data is much slower than that of instrument data. These information are regridded by proper method (e.g. interpolation) to synchronize with the clock of the target instrument.

The data of individual instrument (or source of information) are attached to the main structure as “branches”. The main structure only provide the “mount point” for each branch.

The “Key time” is copy of that of the main instrument (e.g. for the FIS survey data, the FIS-PIM-TI will be the key time). This will be the reference time through the data reduction.

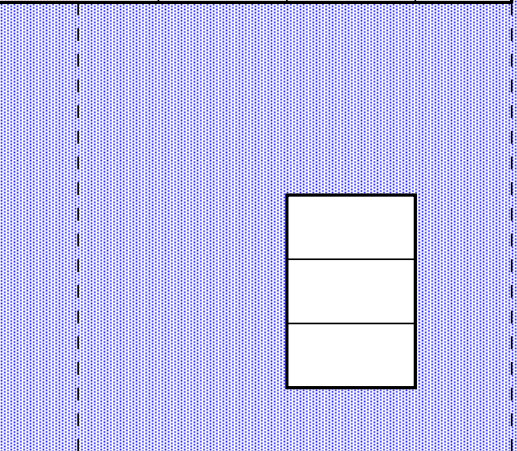
TSD Branch	Time	Status (boolean)	Telemetry (analog)	Detector Data (analog)		Flags (boolean)	Quality	Counter
	Non-Editable				Editable			
FIS_OBS				det	flux ferr	frame flags pixel flags		
FIS_HK								
IRC_HK								
HK_2								
AOCU								
GADS								
PR								
SE								

Figure 1: An overview of Time-Series Data (TSD) structure.

- In order to reduce the data size, each branch is a subset of full packet data and contains only necessary information.
- We define “extension” for every instrument / information source such that the source of information is clearly indicated.
- In order to reduce the data size, extension other than the target instrument are subsets of full telemetry packet data and contains only necessary information.

The practical procedures to create TSD are the followings:  
(For example: FIS survey data)

1. FIS packet data at certain time range are retrieved from LDS.

2. Detector array channels (either SW or LW) are extracted from the data.
3. Corrected TI is put for each sampling. Data array is prepared and the data are stored in the time-series structure. If there is gap between two data, blank records are inserted to fill.
4. FIS instrument status, which may have different sampling timing are regridded to the FIS detector timing then attached to the FIS data to complete the branch.
5. FIS branch is plugged in to the TSD structure (see Fig. 1). FIS TI are copied to the main time field.
6. Other component data are retrieved from LDS.
7. Data of other instruments are retrieved from LDS. Necessary information for the current data is extracted, and then the data are regridded using the main TI.
8. The branch is plugged into the TSD structure.

## 4.2 Physical file format of TSD

The physical file format of TSD to exchange the data is based on *FITS Binary Table Extension*. In the table, each row corresponds to a sampling which is identified by PIM-TI. In order to manipulate the TSD files, we use “The IDL Astronomy User’s Library” <sup>1</sup>, so that users need to install the library files before starting the analysis. You should use interface functions defined in section 6. Please do not use the astrolib functions or procedures directly.

For the detailed information about FITS Binary Table Extension, see “section 8.3 Binary Table Extension” in “Definition of Current FITS Standard - NOST 100-2.0” <sup>2</sup>.

Size of one TSD file is T.B.D. A rough estimate shows that one orbit data will be about 100–200 MB and this should be used in the pipeline system (internally).

---

<sup>1</sup><http://idlastro.gsfc.nasa.gov/>

<sup>2</sup><http://fits.gsfc.nasa.gov/>

## 5 Time-Series Data: detailed description

### 5.1 Nomenclature of Data Type

In this document, data type are expressed in C-like notation in the following descriptions.

bit	: 1 bit	0 or 1
byte	: 1 byte	unsigned byte ( $\geq 0$ )
short	: 2 bytes	integer
long	: 4 bytes	integer
longlong	: 8 bytes	integer
float	: 4 bytes	single precision floating point
double	: 8 bytes	double precision floating point

We express the elements of status column ‘n:’ where n is the element index defined in the FITS.

### 5.2 Time stamp

The basic policy of the time management in the ASTRO-F survey data are;

- The main purpose of time stamp in the data reduction system is to synchronize the information taken with different instruments and other information sources (e.g. for pointing reconstruction).
- For this purpose, time information maintained in TSD is that already corrected various delays. It is the time when the data is sampled by the instruments.
- For the FIS survey data, we use the calibrated time in the FIS data packet.
- Time in the onboard clock is 36 bits with LSB of 1/512 sec. Practically 32 bits data either with 1/32 sec or 1/512 sec are stored in the packet depending on the instruments.
- Internally we use a 64 bits integer (long64 in IDL) for time field.
- The original point of the time in the data reduction system is 2000 January 1st, 00:00:00 UTC.

### 5.3 Status

The ‘Status’, ‘Flags’ and ‘Quality’ of TSD columns are composed of multiple elements of the TBIT type (i.e., the TFORM definitions of FITS standard are ‘8X’, ‘16X’, etc. ).

Figure 2 shows an example of the status section of this document. The left hand number shows the element index. The ‘fv’ displays this index when we click the ‘expand’ button. The first element (index=1) corresponds to the MSB (bit31) in the binary. The next is the element name, which cannot be defined in the FITS standard. However, we define the element name and write them to each header of the HDU. For example, the element names of status in FIS\_OBS extension are written in the header of its HDU as follows:

```
TELEM6 = 'CREON,SHTOP,FWPOS0N,FWPOS_B1,FWPOS_B0,MPOS0N,MPOS_B1,MPOS_B0, ...
```

The user should use the element name to access the status bits, rather than element index.

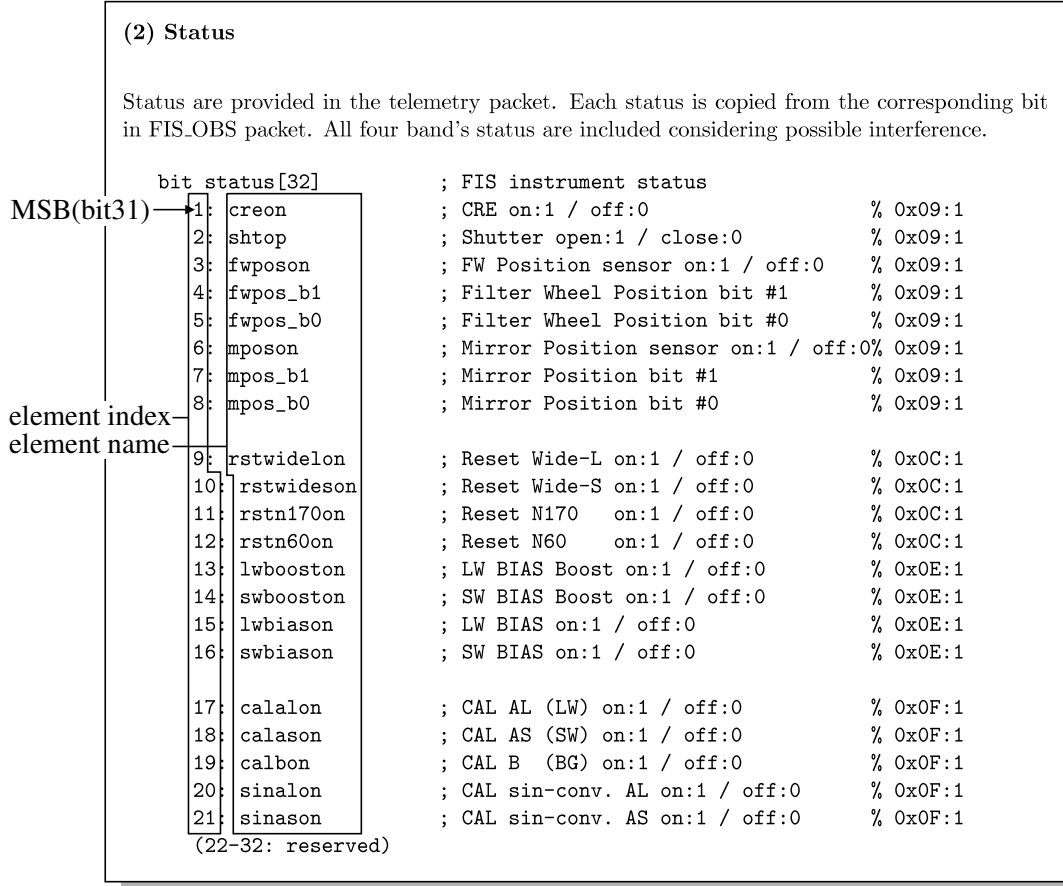


Figure 2: An example of status section.

## 5.4 Primary Header

```

string  FMTTYPE      ; Type of Format in FITS file
integer FTYPEVER     ; Version of FMTTYPE
string  CNTTYPE      ; Type of data content
string  DATE         ; File Creation Date
string  CREATOR      ; Data generator program name
string  CRTRVER      ; Version of CREATOR
string  PIPELINE     ; Data Processing Pipeline name
string  DATASTAT    ; Data status
string  ORIGIN       ; Organization creating FITS file
string  TELESCOP     ; AKARI mission
string  INSTRUME     ; Identifier of the instrument
string  DETECTOR     ; Detector name
string  OBSERVER     ; PI Name (Observer's ID)
string  PROPOSAL     ; Proposal ID
string  OBS-CAT      ; Observation Category
integer PNTNG-ID     ; Pointing ID
integer TARGETID     ; Target ID
integer SUBID        ; Sub ID
string  OBJECT       ; Object name

```



```

double OBJ-RA           ; [degree] Target position
double OBJ-DEC          ; [degree] Target position
string AOT              ; Observation AOT
string AOTPARAM         ; AOT Parameter
string INSTMODE         ; Instrument operation mode
string TIMESYS          ; Explicit time scale specification
string DATE-OBS         ; Observation start date+time
string DATE-END         ; Observation end date+time
string DATE-REF         ; Reference time in the Observation
double AFTM-OBS         ; DATE-OBS in ASTRO-F Time
double AFTM-END         ; DATE-END in ASTRO-F Time
double AFTM-REF         ; DATE-REF in ASTRO-F Time
string PIMTIOBS         ; DATE-OBS in PIM-TI (36bits DHUTI)
string PIMTIEND         ; DATE-END in PIM-TI (36bits DHUTI)
string PIMTIREF         ; DATE-REF in PIM-TI (36bits DHUTI)
double EQUINOX          ; Epoch of Coordinate
double RA              ; [degree] Target position at DATE-REF
double DEC              ; [degree] Target position at DATE-REF
double ROLL             ; [degree] Roll Angle at DATE-REF
double AA-SOL           ; [degree] Solar avoidance angle at DATE-REF
double AA-EAR           ; [degree] Earth avoidance angle at DATE-REF
double AA-LUN           ; [degree] Lunar avoidance angle at DATE-REF
double TM-SAA           ; [sec] Duration since last SAA passage at DATE-REF
double SAT-POSX         ; [km] Satellite position at DATE-REF
double SAT-POSY         ; [km] Satellite position at DATE-REF
double SAT-POSZ         ; [km] Satellite position at DATE-REF
string DAYNIGHT         ; day/night status at DATE-REF
integer STTA-NUM        ; number of stars in STT-A at DATE-REF
integer STTB-NUM        ; number of stars in STT-B at DATE-REF
string STTA-MOD         ; STT-A Mode status at DATE-REF
string STTB-MOD         ; STT-B Mode status at DATE-REF
string HISTORY          ; Processing History. Any length text.

```

## 5.5 FIS\_OBS extension

Here we show the data structure when the main data is the FIS survey data.

### (1) Time

```
double aftime          ; ASTRO-F Satellite Time (calibrated)
                        ; which is the seconds since 2000/January/1 00:00 (UTC)
longlong pimtiorg      ; PIMTI counter (from original packet)
longlong fistiorg      ; FISTI counter (from original packet)
longlong pimti         ; PIMTI counter (interpolated)
longlong fisti         ; FISTI counter (interpolated)
```

### (2) Status

Status are provided in the telemetry packet. Each status is copied from the corresponding bit in FIS\_OBS packet. All four band's status are included considering possible interference.

```
bit status[32]         ; FIS instrument status
1: creon               ; CRE on:1 / off:0                % 0x09:1
2: shtop               ; Shutter open:1 / close:0        % 0x09:1
3: fwposon             ; FW Position sensor on:1 / off:0  % 0x09:1
4: fwpos_b1            ; Filter Wheel Position bit #1     % 0x09:1
5: fwpos_b0            ; Filter Wheel Position bit #0     % 0x09:1
6: mposon              ; Mirror Position sensor on:1 / off:0 % 0x09:1
7: mpos_b1             ; Mirror Position bit #1         % 0x09:1
8: mpos_b0             ; Mirror Position bit #0         % 0x09:1

9: rstwidelon          ; Reset Wide-L on:1 / off:0                % 0x0C:1
10: rstwideson         ; Reset Wide-S on:1 / off:0                % 0x0C:1
11: rstn170on          ; Reset N160 on:1 / off:0                 % 0x0C:1
12: rstn60on           ; Reset N60 on:1 / off:0                  % 0x0C:1
13: lwbooston          ; LW BIAS Boost on:1 / off:0              % 0x0E:1
14: swbooston          ; SW BIAS Boost on:1 / off:0              % 0x0E:1
15: lwbiason           ; LW BIAS on:1 / off:0                   % 0x0E:1
16: swbiason           ; SW BIAS on:1 / off:0                   % 0x0E:1

17: calalon            ; CAL AL (LW) on:1 / off:0                % 0x0F:1
18: calason            ; CAL AS (SW) on:1 / off:0                % 0x0F:1
19: calbon             ; CAL B (BG) on:1 / off:0                % 0x0F:1
20: sinalon            ; CAL sin-conv. AL on:1 / off:0           % 0x0F:1
21: sinason            ; CAL sin-conv. AS on:1 / off:0           % 0x0F:1
(22-32: reserved)
```

The 'rstn170on' should be renamed to 'rstn160on'. However, we do not change it for historical reasons.

### (3) Analog Telemetry

```
byte packetid          ; PacketID (FIS obsmode)          % 0x00:8
```

```

byte rstcntwidel      ; Reset counter Wide-L          % 0x0A:4
byte rstcntwides      ; Reset counter Wide-S          % 0x0A:4
byte rstcntn170       ; Reset counter N160            % 0x0B:4
byte rstcntn60        ; Reset counter N60             % 0x0B:4

byte calplscntal      ; CAL pulse counter AL         % 0x0C:4
byte calplscntas      ; CAL pulse counter AS         % 0x0D:4
byte calplscntb       ; CAL pulse counter B          % 0x0D:4
byte aderrcntsw       ; SW A/D error counter         % 0x19:4
byte aderrcntlw       ; LW A/D error counter         % 0x19:4
long mposcnt          ; Assigned mirror position      % 0x1A-0x2F
                    ; Assigned/interpolated frame
                    ; counter in the survey mode

byte dacbiaswides     ; DAC BIAS Wide-S level        % 0x10:8
byte dacbiasn60       ; DAC BIAS N60 level           % 0x11:8
byte dacbiaswidel     ; DAC BIAS Wide-L level        % 0x12:8
byte dacbiasn170      ; DAC BIAS N160 level          % 0x13:8

byte daccalas         ; DAC CAL AS level             % 0x14:8
byte daccalal         ; DAC CAL AL level             % 0x15:8
byte daccalb          ; DAC CAL B level              % 0x16:8
byte dacsinas         ; DAC CAL sin-conv. AS level    % 0x17:8
byte dacsinal         ; DAC CAL sin-conv. AL level    % 0x18:8

short tpboddy         ; FIS body temperature         % 0x3E-3F
short tpsw            ; SW Detector temperature      % 0x40-41
short tplw            ; LW Detector temperature      % 0x42-43
short tpcalft         ; CAL FT temperature           % 0x44-45

```

The 'rstcntn170' and 'dacbiasn170' should be renamed to 'rstcntn160' and 'dacbiasn160' respectively. However, we do not change them for historical reasons.

#### (4) Detector data

Either SW(NDET=100) or LW (NDET=75) data are included in one data set.

```

long det[NDET]        ; Detector signal [ADU]
float flux[NDET]       ; Detector data in physical unit
float ferr[NDET]       ; Flux uncertainty

```

#### (5) Flags

Flags are those set in the reduction software. These flags are set for each frame (sampling).

```

bit flag[8]           ; Flag for detector condition
    1: bad_frame      ; set at all the unrecoverable frames
                        ; described by the other flags:1
    2: undef_anom_frame ; unusable data but the anomaly cannot

```

```

; be described by the existing flags.
3: blank          ; blank:1
4: in_saa         ; in SAA:1
5: near_moon      ; near moon:1
                  ; "aa_lun" is used for lunar avoidance
                  ; angle in GADS extension.
6: untrusted_frame ; incomplete restitution of higher-order bits of DET.
(7-8: reserved)

```

These flags are set for each frame (sampling) and each pixel.

```

bit pix_flag[32*NDET] ; Flag for each pixel condition
1: bad                ; bad pixel:1
2: undef_anom         ; unusable data but the anomaly is not
                      ; described by any other flags.
3: arith_err          ; special values for undefined results;
                      ; NaN and INFINITY
4: dead               ; dead pixel:1
5: saturate           ; saturated pixel:1
6: reset              ; data taken just after reset:1
7: rstanom            ; anomaly seen in a few certain
                      ; samplings just after reset.
8: no_diff            ; No differentiation enable:1
9: no_rp_corr         ; impossible to carry out ramp curve correction
10: no_dk_corr         ; dark has not been subtracted successfully.
11: no_dccal_corr     ; DC responsivity correction has not been
                      ; corrected successfully.
12: no_tr_corr        ; transient effect has not been corrected successfully.
13: no_flat_field     ; flat fielding has not been carried out successfully.
14: no_gpgl           ; unable to search/correct for glitch
15: no_mtgl           ; unable to search/correct for glitch
16: no_abscale        ; abs. flux calibration has not been done successfully.

; For glitch detection using Gaussian Processing method(Kester)
17: gpgl_type1        ; Glitch Type 1
18: gpgl_type2        ; Glitch Type 2
19: gpgl_type3        ; Glitch Type 3
20: gpgl_type4        ; Glitch Type 4
21: gpgl_tail         ; Glitch Tail

; For glitch detection using Median Transformation method (Serjeant)
22: mtgl_type1        ; Glitch Type 1
23: mtgl_type2        ; Glitch Type 2
24: mtgl_type3        ; Glitch Type 3
25: mtgl_type4        ; Glitch Type 4
26: mtgl_tail         ; Glitch Bad

27: no_peri_corr      ; No periodic noise correction available

;; For SUSSEXtractor
28: sx_peak           ; threshold detection

```

```

29: sx_source          ; source detected:1
30: sx_obj             ; anomolous detected
(31-32: reserved)

```

## (6) Quality

```

bit quality[40*NDET]    ; Quality Flag for each pixel condition

;; gb_conv_to_volt ;;
1-2: qual_cv_param      ; conversion quality from ADU to Volts.

;; gb_ramp_curve_corr ;;
3-4: qual_rc_param      ; accuracy of correction parameters
5-6: qual_rc_cf         ; correction factor (difference between
                        ; corrected and the original data)

;; gb_differentiation ;;
7-8: qual_df_eq         ; order of polynomials/number of points used.

;; gb_dc_responsibility_corr / make_DC_cal_lamp_table ;;
9-10: qual_rp_data      ; accuracy of the derived cal. lamp intensity
                        ; from data in a "cal. lamp on."
11-12: qual_rp_param    ; availability of "cal. lamp on" to derive
                        ; a correction factor
13-14: qual_rp_table    ; quality of the correction table (fluctuation etc.)

;; "correction for flat fielding" ;;
15-16: qual_ff_param    ; quality of the parameters for flat fielding
17-18: qual_ff_cf       ; correction factor (difference between
                        ; corrected and the original data)

;; gb_set_glitch_status_gp & gb_set_glitch_status_gp ;;
;; -> correction after glitch detection
19-20: qual_gppl_corr; correction of glitch tails for Gaussian
                        ; processing method
21-22: qual_mtgl_corr; correction of glitch tails for Median
                        ; transformation method

;; gb_transient_corr ;;
23-24: quql_tr_hist     ; history information availability
25-26: qual_tr_param    ; correction parameter availability

;; gb_dark_subtraction/ make_dark_table ;;
27-28: qual_dk_data     ; quality of the dark data in one "shutter-close".
29-30: qual_dk_param    ; availability of the data of a "shutter-close"
                        ; to derive a dark value.
31-32: qual_dk_table    ; quality of the dark values of the correction
                        ; table (fluctuation etc.)

```

```
;; gb_flux_calib ;;  
33-34: qual_fx_corr ; availability of calibration parameters  
35-36: qual_fx_param ; error in calibration factors  
(37-40: reserved)
```

## (7) Counter

```
short cnt_saa[NDET] ; time in seconds since the last SAA passage  
; # "tm_saa" is used in SE extension  
short cnt_glitch_gp[NDET] ; a duration since the last corresponding glitch(GP).  
short cnt_glitch_mt[NDET] ; a duration since the last corresponding glitch(MMT).
```

## 5.6 FIS HK extension

Outputs of APID='FIS\_HK' (which contains some minor status of FIS) should be stored in this extension.

### (1) Time

```
double aftime          ; ASTRO-F Satellite Time (calibrated)
                        ; which is the time since 2000/January/1 00:00 (UTC)
longlong pimtior      ; PIMTI counter (from original packet)
```

### (2) Status

```
bit status[8]          ; Satellite attitude status
    1: blank           ; blank:1
    2: blank_subcom_data ; blank bit for subcom data (for sw_heater_tmp)
    (3-8:reserved)
```

Note that some FIS status are stored in the HK\_2 extension.

### (3) Analog data

```
byte fis_fw_f_pul      ; Filter wheel pulse counter (Forward)
byte fis_fw_r_pul      ; Filter wheel pulse counter (Rewind)
byte fis_lw_wire_light ; LW wire light DAC set value
byte fis_run_mode       ; RUN mode
byte fis_obs_mode       ; Observation mode
byte fis_asq_pattern    ; ASQ pattern
long fis_asq_step       ; ASQ step
byte sw_heater_tmp      ; SW heater temperature
```

## 5.7 IRC\_HK extension

Outputs of APID='IRC\_HK' (which contains status of temperature of the focal plane instruments) should be stored in this extension.

### (1) Time

```
double aftime          ; ASTRO-F Satellite Time (calibrated)
                        ; which is the time since 2000/January/1 00:00 (UTC)
longlong pimtiorg      ; PIMTI counter (from original packet)
```

### (2) IRC status

```
bit status[8]          ; Status of IRC functions which may cause
                        ; interference
    1: blank            ; Data blank:
    (2-8:reserved)     ; Details TBD with IRC team
```

### (3) Analog data

TBD



## 5.8 HK\_2 extension

Outputs of APID='HK\_2' (which contains basic data of Cryostat, FIS and IRC) should be stored in this extension.

### (1) Time

```
double aftime          ; ASTRO-F Satellite Time (calibrated)
                        ; which is the time since 2000/January/1 00:00 (UTC)
longlong pimtiorg      ; PIMTI counter (from original packet)
```

### (2) Cryostat/FIS/IRC status

```
bit status[16]         ; CRYO/FIS status
  1: blank              ; blank:1
  2: blank_cryo_temp_data; blank bit for Cryo temperature data
  (3-4:reserved)
  5: cryo_temp_data_en_ds; Cryo temperature data enabled/disabled
  (6:reserved)
  7: fis_cpu_run_rst    ; FIS CPU run/reset
  8: fis_fis_on_off     ; FIS on/off
  9: fis_tmp_pol_pos_neg; Thermometer polarity
 10: fis_obs_seq        ; Observation sequence start/stop
 11: fis_rsw_auto_cmnd; Auto (periodic) reset mode on/off
 12: fis_fcal_frq_b1    ; CAL overlay sine wave frequency B1 bit #1
 13: fis_fcal_frq_b0    ; CAL overlay sine wave frequency B1 bit #0
  (14:reserved)
 15: irc_cpu_ru_rs      ; IRC CPU run/reset
 16: irc_irc_on_of      ; IRC on/off
```

### (3) Cryostat temperature

```
float cryo_he_tank_1    ; He tank 1 temperature[K] (one close to heat strap)
float cryo_baffle_1     ; Baffle 1 temperature[K] (one close to the tel.?)
float cryo_baffle_2     ; Baffle 2 temperature[K]
```

### (4) Cryostat current

```
float cryo_cold_hd_a_dr_i; Cold head A driving current[A]
float cryo_cold_hd_b_dr_i; Cold head B driving current[A]
float cryo_comp_a_dr_i   ; Compressor A driving current[A]
float cryo_comp_b_dr_i   ; Compressor B driving current[A]
```

### (5) IRC Analog data

```
float irc_mainref_tmp1   ; Primary mirror temperature[K] (IRC)
float irc_subref_tmp     ; Secondary mirror temperature[K] (IRC)
float irc_fpip_tmp1      ; FPI plate[K] (IRC)
```

## 5.9 Attitude and Orbit Control Unit (AOCU) extension

Outputs of APID='AOCU' (Attitude and Orbit Control Unit) should be stored in this extension.

### (1) Time

```
double aftime          ; ASTRO-F Satellite Time (calibrated)
                        ; which is the time since 2000/January/1 00:00 (UTC)
longlong pimtiorg      ; PIMTI counter (from original packet)
```

### (2) AOCU Status

```
bit status[8]          ; AOCU operation status
  1: blank              ; blank:1
  2: ao_rom_ram         ; ROM mode:1 / RAM mode:0
                        ; i.e., the blank bit for the ads_mode_b1, ads_mode_b0
                        ; and aodu_ads_q.
  3: ads_mode_b1        ; Attitude determination mode (AOCU_ADS_MODE) bit #1
  4: ads_mode_b0        ; Attitude determination mode (AOCU_ADS_MODE) bit #0
  (5-8:reserved)
```

### (3) Analog telemetry

```
double aocu_ads_q[4]    ; Quaternion at boresight
double aocu_body_rate[3]; Body rate (Angular velocity[deg/s]) at boresight

byte aocu_saab_mode     ; STT-A operation mode
byte aocu_sbab_mode     ; STT-B operation mode
byte aocu_saab_win_num  ; Number of stars in STT-A
byte aocu_sbab_win_num  ; Number of stars in STT-B

double ra               ; J2000 R.A. (deg) at boresight
double dec              ; J2000 Dec. (deg) at boresight
double roll             ; Roll angle at boresight (deg)
double d_ra             ; Scan rate in J2000 R.A.
double d_dec            ; Scan rate in J2000 Dec.
double d_roll           ; Scan rate in roll angle

double crs_off          ; Offset in cross-scan direction from
                        ; Nominal scan path
```

### (4) Quality

```
bit quality[8]          ; Quality Flag for condition
  1: interpolated       ; interpolated:1
; Details will be discussed with ISAS/ESA.
```

## 5.10 Ground-based Attitude Determination System (GADS) extension

Results of ground-based attitude determination should be stored in this extension.

### (1) Time

```
double aftime          ; ASTRO-F Satellite Time (calibrated)
                        ; which is the time since 2000/January/1 00:00 (UTC)
longlong pmtiorg       ; PIMTI counter (from original packet)
```

### (2) Status

```
bit status[8]          ; Satellite attitude status
    1: blank           ; blank: 1
    (2-8:reserved)
```

### (3) Analog telemetry

```
double attitude[4]     ; Quaternion at boresight
double attitude_er[3]  ; Estimated error of position
double angl_vel[3]     ; Body rate (Angular velocity) at boresight

double ra              ; J2000 R.A. (deg) at boresight
double dec             ; J2000 Dec. (deg) at boresight
double roll           ; Roll angle at boresight (deg)
double d_ra           ; Scan rate in J2000 R.A.
double d_dec          ; Scan rate in J2000 Dec.
double d_roll         ; Scan rate in roll angle

double el_sol          ; Solar elongation
double aa_sol          ; Solar Avoidance Angle
double aa_ear         ; Earth Avoidance Angle
double aa_lun          ; Lunar Avoidance Angle

double crs_off         ; Offset in cross-scan direction from
                        ; Nominal scan path
```

### (4) Quality

```
bit quality[8]         ; Quality Flag for condition
    1: interpolated    ; interpolated:1
; Details will be discussed with ISAS/ESA.
```

### 5.11 Pointing Reconstruction (PR) extension

Results of pointing reconstruction produced by ESA should be stored in this extension.

#### (1) Time

```
double aftime          ; ASTRO-F Satellite Time (calibrated)
                        ; which is the time since 2000/January/1 00:00 (UTC)
longlong pmtiorg       ; PIMTI counter (from original packet)
```

#### (2) Status

```
bit status[8]          ; Satellite attitude status
    1: blank           ; blank: 1
    (2-8:reserved)     ; Details will be discussed with ESA.
```

#### (3) Analog telemetry

```
double quaternion[4]   ; Quaternion at boresight
double pos_err[3]      ; Estimated error of position
double body_rate[3]    ; Body rate (Angular velocity) at boresight

double ra              ; J2000 R.A. (deg) at boresight
double dec             ; J2000 Dec. (deg) at boresight
double roll           ; Roll angle at boresight (deg)
double d_ra           ; Scan rate in J2000 R.A.
double d_dec          ; Scan rate in J2000 Dec.
double d_roll         ; Scan rate in roll angle

double el_sol          ; Solar elongation
double aa_sol          ; Solar Avoidance Angle
double aa_ear         ; Earth Avoidance Angle
double aa_lun          ; Lunar Avoidance Angle

double crs_off         ; Offset in cross-scan direction from
                        ; Nominal scan path
```

#### (4) Quality

```
bit quality[8]         ; Quality Flag for condition
    1: interpolated    ; interpolated:1
; Details will be discussed with ISAS/ESA.
```

## 5.12 Satellite Ephemeris (SE) extension

The following information is provided from the satellite orbit prediction. The delivery route is TBD. Data may be replaced when “actual” ephemeris are provided by the ground attitude determination system after data are downlinked.

### (1) Time

```
double aftime          ; ASTRO-F Satellite Time (calibrated)
                        ; which is the time since 2000/January/1 00:00 (UTC)
longlong pimtiorgr     ; PIMTI counter (from original packet)
```

### (2) Status

```
bit status[8]          ; Satellite attitude status
  1: blank              ; blank:1
  2: type               ; Data type (Prediction/Determined)
  3: daynight           ; Day / Night
  4: in_saa             ; in SAA region
  5: in_polar           ; in Polar region
  (6-8:reserved)
```

### (3) Analog value

```
double tm_saa          ; seconds since last SAA passage.
                        ; < 0 during the passage
                        ; Clock start at end of Bias Boost ?

double sat_posx         ; Satellite position in Earth reference frame
double sat_posy         ; Satellite position in Earth reference frame
double sat_posz         ; Satellite position in Earth reference frame
```

## 6 The TSD Interface Functions

As was described in the previous section, the basic policy of data interface is that user programs (pipeline routines) are NOT allowed inquired / set the data fields directly. The data I/F routines are written in the manner of object-oriented, so that all access should be done through the data I/F routines, which are exclusively maintained by the data I/F team. Note that it is not necessary to make users program an object-oriented model, but usual procedure model.

The following routines assume that the data are read from disk files into memory first, then accessed with the functions (the routines are not designed to access disk files directly).

Conversion from data in LDS to TSD are described in the separate document.

### A Simple example

Most of the functions are implemented as methods of TSD class. In order to use these methods, you should make an “object instance” of TSD class using IDL’s `OBJ_NEW()` function at first, then call methods with “->” operator.

---

```

; Create an object instance
IDL> mytsd = OBJ_NEW('tsd')
; Open a file
IDL> ret = mytsd->Read_TSD('abc.fits')
; Get status of Shutter Open
IDL> ret = mytsd->Get_Status_TSD('FIS_OBS', 'shtop')
; Get detector signal
IDL> value = mytsd->Get_Value_TSD('FIS_OBS', 'det')

      (apply your procedures to the value...)

; Set detector values
IDL> ret = mytsd->Set_Value_TSD('FIS_OBS', 'det', value)
; Set flags of Glitch
IDL> ret = mytsd->Set_Pix_Flag_TSD('FIS_OBS', 'spike', flags)
; Write a file
IDL> ret = mytsd->Write_TSD('xyz.fits')

```

---

## 6.1 Object Initialization/Destruction

### SYNOPSIS

```

; CALLING SEQUENCE:
;   tsdobj = OBJ_NEW('tsd' [, DATATYPE=datatype] [, INFILE='infile.fits'])
;
; INPUTS:
;   'tsd'          Name of object class
;
; OPTIONAL KEYWORD:
;   DATATYPE      Datatype (STRING)
;                  (now available: 'FIS_LW' and 'FIS_SW')
;   INFILE        Input filename of TSD (which is a FITS format) (STRING)
;
; OUTPUTS:
;   tsdobj        Name of object instance
;
; EXAMPLE:
;   IDL> tsdobj=OBJ_NEW('tsd')
;   IDL> help, tsdobj
;   TSDOBJ          OBJREF      = <ObjHeapVar1(TSD)>

```

---

### SYNOPSIS

```

; CALLING SEQUENCE:
;   OBJ_DESTROY, tsdobj
;
; INPUTS:
;   tsdobj        Name of object instance
;
; EXAMPLE:
;   IDL> tsdobj=OBJ_NEW('tsd')
;           :
;           :
;   IDL> OBJ_DESTROY, tsdobj

```

---

## 6.2 TSD I/O methods

### NAME

Read\_TSD

### SYNOPSIS

```
; CALLING SEQUENCE:
;   status = tsd->Read_TSD('infile.fits' [, RANGE=range])
;
; INPUTS:
;   infile      Input filename of TSD (which is a FITS format) (STRING)
;
; OPTIONAL KEYWORD:
;   range       A scalar or two element vector giving the start
;               and end rows to be retrieved. The first row is row 0.
;               If only a single value, x, is given in the range,
;               the range is assumed to be [0,x-1].
;               See comment of MRDFITS of AstroLib for more detail.
;
; OUTPUTS:
;   status      Status of function (!SUCCEED or !FAIL)
;
; EXAMPLE:
;   By default, all of infile is red into TSD internal structure on
;   IDL memory.
;
;   IDL> tsd = OBJ_NEW('tsd')
;   IDL> dummy = tsd->Read_TSD('infile.fits')
;
;   If the input file is so large and you don't have enough memory, you
;   also can get a subset of the file by specifying range you want to read.
;   Please note that range overflow might be occurred.
;
;   IDL> tsd = OBJ_NEW('tsd')
;   IDL> range = [100,200] ; Note that nrow=101
;   IDL> dummy = tsd->Read_TSD('so_laaaaaarge.fits', RANGE=range)
;   IDL> aftime = tsd->Get_Value_TSD('FIS_OBS', 'aftime')
;   IDL> help, aftime
;   AFTIME      INT      = Array[101]
```

### NAME

Write\_TSD

### SYNOPSIS

```
; CALLING SEQUENCE:
;   status = tsd->Write_TSD('outfile.fits')
;
; INPUTS:
;   outfile     Output filename of TSD (which is a FITS format) (STRING)
;
; OPTIONAL KEYWORDS:
```



```
;   None
;
; OUTPUTS:
;   status      Status of function (!SUCCEED or !FAIL)
```

---

**NAME**

Create\_TSD

**SYNOPSIS ; CALLING SEQUENCE:**

```
;   status = tsd->Create_TSD(datatype, naxis2 [, INSTRUME=instrume])
;
; INPUTS:
;   datatype    Datatype (STRING)
;               (now available: 'FIS_LW' and 'FIS_SW')
;   naxis2      Number of rows of binary table (INT)
;   instrume    String to set to 'INSTRUME' in the primary header
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   status      Status of function (!SUCCEED or !FAIL)
```

---

### 6.3 TSD I/O method with the Local Data Server (LDS)

See also the LDS client functions (section 7).

#### NAME

Read\_TSD\_From\_LDS

#### SYNOPSIS

```
; CALLING SEQUENCE:
;   filename = tsd->Read_TSD_From_LDS( MODE=mode, GB_PROCESSED=gb_processed, $
;                                     FMTTYPE(DATATYPE)=fmttype, $
;                                     DATE(TIME)=date, AFTIME=aftime, $
;                                     SPAN=span, GZIP(COMPRESS)=gzip )
;
; INPUTS:
;   NONE
;
; OPTIONAL KEYWORDS:
;   MODE           Data query mode: ('Fli' or 'Rel': default: 'Fli')
;   GB_PROCESSED   Flag whether data is processed by GreenBox or not
;                 (0: not_processed, 1: processed. Default: 0)
;   FMTTYPE        'FIS_SW' or 'FIS_LW'.
;   DATE           Start time of requested information.
;                 DATE should be set in UTC.
;                 (STRING FORMAT: '2006-07-10T23:40:00.000Z')
;   AFTIME         Start time of requested information.
;                 DATE is ignored when AFTIME is set.
;   SPAN           Span of request information (seconds)
;   GZIP           Set 1 when you want get a gzipped file.
;                 (default: 0)
;
; OUTPUTS:
;   filename       Name of filename (STRING)
;                 !FAIL will be returned if failed.
;
; EXAMPLE:
;   This method is for non-interactive use.
;
;   IDL> tsd = OBJ_NEW('tsd')
;   IDL> filename = tsd->Read_TSD_From_LDS(MODE='Fli', GB_PROCESSED=0, $
;                                     FMTTYPE='FIS_LW', $
;                                     DATE='2007-02-01T00:30:00.000Z', $
;                                     SPAN=2010, GZIP=1)
;
;   IDL> print , filename
;   FIS_LW_20070201003000_2010.fits.gz
;   IDL> aftime = tsd->Get_Value_TSD('FIS_OBS', 'AFTIME')
;   IDL> help, aftime
;   AFTIME          DOUBLE      = Array[33882]
;
```

---

## 6.4 Header I/O methods

### NAME

Get\_Header\_TSD

### SYNOPSIS

```
; CALLING SEQUENCE:
;   header = tsd->Get_Header_TSD(extname [, /POINTER])
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'Primary', 'FIS_OBS', 'FIS_HK', 'IRC_HK',
;                 'HK_2', 'AOCU', 'GADS', 'PR', and 'SE')
;
; OPTIONAL KEYWORDS:
;   POINTER      Keyword whether return value is a pointer or not.
;
; OUTPUTS:
;   header       FITS Header (STRARR)
```

---

### NAME

Set\_Header\_TSD

### SYNOPSIS

```
; CALLING SEQUENCE:
;   status = tsd->Set_Header_TSD(extname, header)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'Primary', 'FIS_OBS', 'FIS_HK', 'IRC_HK',
;                 'HK_2', 'AOCU', 'GADS', 'PR', and 'SE')
;   header       FITS Header (STRARR)
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   status       Status of function (!SUCCEED or !FAIL)
```

---

### NAME

Get\_Header\_Keyword\_TSD

### SYNOPSIS

```
; CALLING SEQUENCE:
;   value = tsd->Get_Header_Keyword_TSD(extname, keyword)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'Primary', 'FIS_OBS', 'FIS_HK', 'IRC_HK',
;                 'HK_2', 'AOCU', 'GADS', 'PR', and 'SE')
;   keyword      Header Keyword (STRING)
```

```

;           The first eight bytes of a header card image.
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   value      Value of header keyword

```

---

**NAME**

Set\_Header\_Keyword\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   status = tsd->Set_Header_Keyword_TSD(extname, keyword, value)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                (now available: 'Primary', 'FIS_OBS', 'FIS_HK', 'IRC_HK',
;                'HK_2', 'AOCU', 'GADS', 'PR', and 'SE')
;   keyword      Header Keyword (STRING)
;                The first eight bytes of a header card image.
;   value        Value of Keyword
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   status       Status of function (!SUCCEED or !FAIL)

```

---

## 6.5 Access to TSD structure elements

### NAME

Get\_Value\_TSD

### SYNOPSIS

```
; CALLING SEQUENCE:
;   valarray = tsd->Get_Value_TSD(extname, fname [, /POINTER])
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                 'AOCU', 'GADS', 'PR', and 'SE')
;   fname        Field name of FITS Binary table (STRING)
;
; OPTIONAL KEYWORDS:
;   POINTER      Keyword whether return value is a pointer or not.
;
; OUTPUTS:
;   valarray      Array of value (or pointer) in question
;
; DESCRIPTION:
;   Copy any value in data array specified by keywords into a simple
;   time sequence array.
;
; COMMENT:
;   There may be some functions which are frequently used at
;   certain occasion but practically equivalent with Get_Value_TSD().
;   See also Get_Status_TSD(), Get_Flag_TSD(), and Get_Pix_Flag_TSD().
```

---

### NAME

Set\_Value\_TSD

### SYNOPSIS

```
; CALLING SEQUENCE:
;   status = tsd->Set_Value_TSD(extname, fname, val)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                 'AOCU', 'GADS', 'PR', and 'SE')
;   fname        Field name of FITS Binary table (STRING)
;   val          Value
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   status        Status of function (!SUCCEED or !FAIL)
;
; DESCRIPTION:
```

```
; Write time-sequence data array into the field in TSD specified by
; keyword. Keyword and corresponding data type, and data field in
; TSD must be all consistent, otherwise the process is failure.
```

---

**NAME**

Get\_Status\_TSD

**SYNOPSIS**

```
; CALLING SEQUENCE:
;   valarray = tsd->Get_Status_TSD(extname, key)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                 'AOCU', 'GADS', 'PR', and 'SE')
;   key          Name of status of the extension (STRING)
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   valarray     Array of value (or pointer) in question
;
; DESCRIPTION:
;   Inquire status in TSD specified by keywords.
;
; COMMENT
;   This function is equivalent as far as a status is corresponding
;   to a bit in TSD. In addition, this function will treat a status
;   expressed by combination of two or more status bits.
```

---

**NAME**

Set\_Status\_TSD

**SYNOPSIS**

```
; CALLING SEQUENCE:
;   status = tsd->Set_Status_TSD(extname, key, val)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                 'AOCU', 'GADS', 'PR', and 'SE')
;   key          Name of flag of the extension (STRING)
;   val          Value
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   status       Status of function (!SUCCEED or !FAIL)
```

```

;
; DESCRIPTION:
;   Set status bits in TSD.
;
; COMMENT:
;   As our definition is that status are those contained in the telemetry
;   data or other initial data and are not allowed to change by the
;   reduction process, this function is only used to create a new TSD.

```

---

**NAME**

Get\_Flag\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   valarray = tsd->Get_Flag_TSD(extname, key)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                'AOCU', 'GADS', 'PR', and 'SE')
;   key          Name of flag of the extension (STRING)
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   valarray     Array of value (or pointer) in question
;
; DESCRIPTION:
;   Inquire flag. Flag keyword name are defined separately. This
;   function is practically equivalent with Get_Value_TSD().

```

---

**NAME**

Set\_Flag\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   status = tsd->Set_Flag_TSD(extname, key, val)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                'AOCU', 'GADS', 'PR', and 'SE')
;   key          Name of flag of the extension (STRING)
;   val          Value
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   status       Status of function (!SUCCEED or !FAIL)

```

---

**NAME**

Get\_Pix\_Flag\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   valarray = tsd->Get_Pix_Flag_TSD(extname, key [, PIXNO=pixno])
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                 'AOCU', 'GADS', 'PR', and 'SE')
;   key          Name of pixel-flag of the extension (STRING)
;
; OPTIONAL KEYWORDS:
;   PIXNO        FIS Pixel number (0-99 for SW, 0-74 for LW)
;                 An integer or array expression is allowed.
;
; OUTPUTS:
;   valarray     Array of value (or pointer) in question
;
; DESCRIPTION:
;   Inquire flag for each pixel. Flag keyword name are defined
;   separately. This function is practically equivalent with
;   Get_Value_TSD().

```

---

**NAME**

Set\_Pix\_Flag\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   status = tsd->Set_Pix_Flag_TSD(extname, key, val [, PIXNO=pixno])
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                 'AOCU', 'GADS', 'PR', and 'SE')
;   key          Name of pixel-flag of the extension (STRING)
;   val          Value
;
; OPTIONAL KEYWORDS:
;   PIXNO        FIS Pixel number (0-99 for SW, 0-74 for LW)
;                 An integer or array expression is allowed.
;
; OUTPUTS:
;   status       Status of function (!SUCCEED or !FAIL)

```

---

**NAME**

Get\_Quality\_TSD



**SYNOPSIS**

```

; CALLING SEQUENCE:
;   valarray = tsd->Get_Quality_TSD(extname, key [, PIXNO=pixno])
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'AOCU', 'GADS' and 'PR')
;   key          Name of FIS Quality flag (STRING)
;
; OPTIONAL KEYWORDS:
;   PIXNO        FIS Pixel number (0-99 for SW, 0-74 for LW)
;                 An integer or array expression is allowed.
;
; OUTPUTS:
;   valarray      Array of value (or pointer) in question
;
; DESCRIPTION:
;   Inquire quality for each pixel. Flag keyword name are defined
;   separately. This function is practically equivalent with
;   Get_Value_TSD().

```

---

**NAME**

Set\_Quality\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   status = tsd->Set_Quality_TSD(extname, key, val [, PIXNO=pixno])
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'AOCU', 'GADS' and 'PR')
;   key          Name of FIS Quality flag (STRING)
;   val          Value
;
; OPTIONAL KEYWORDS:
;   PIXNO        FIS Pixel number (0-99 for SW, 0-74 for LW)
;                 An integer or array expression is allowed.
;
; OUTPUTS:
;   status        Status of function (!SUCCEED or !FAIL)

```

---

## 6.6 Utility methods

### NAME

Get\_Filename

### SYNOPSIS

```
; CALLING SEQUENCE:
;   filename = tsd->Get_Filename()
;
; INPUTS:
;   None
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   filename      Name of filename (STRING)
```

---

### NAME

Set\_Filename

### SYNOPSIS

```
; CALLING SEQUENCE:
;   status = tsd->Set_Filename(filename)
;
; INPUTS:
;   filename      Name of filename (STRING)
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   status        Status of function (!SUCCEED or !FAIL)
```

---

### NAME

Get\_TSDVER

### SYNOPSIS

```
; CALLING SEQUENCE:
;   version = tsd->Get_TSDVER()
;
; INPUTS:
;   None
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   version      Version number (INT)
```

---

**NAME**

Get\_Datatype

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   datatype = tsd->Get_Datatype()
;
; INPUTS:
;   None
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   datatype      Datatype (STRING)
;                  (now available: 'FIS_LW' and 'FIS_SW')

```

---

**NAME**

Get\_Value\_Members\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   members = tsd->Get_Value_Members_TSD(extname)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                  (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                  'AOCU', 'GADS', 'PR', and 'SE')
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   members      Names of members of extname (STRARR)

```

---

**NAME**

Get\_Status\_Members\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   members = tsd->Get_Status_Members_TSD(extname)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                  (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                  'AOCU', 'GADS', 'PR', and 'SE')
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   members      Names of status members of extname (STRARR)

```

---

**NAME**

Get\_Flag\_Members\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   members = tsd->Get_Flag_Members_TSD(extname)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                 'AOCU', 'GADS', 'PR', and 'SE')
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   members      Names of flag members of extname (STRARR)

```

---

**NAME**

Get\_Pix\_Flag\_Members\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   members = tsd->Get_Pix_Flag_Members_TSD(extname)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'FIS_HK', 'IRC_HK', 'HK_2',
;                 'AOCU', 'GADS', 'PR', and 'SE')
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   members      Names of pix_flag members of extname (STRARR)

```

---

**NAME**

Get\_Quality\_Members\_TSD

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   members = tsd->Get_Quality_Members_TSD(extname)
;
; INPUTS:
;   extname      Name of FITS extension (STRING)
;                 (now available: 'FIS_OBS', 'AOCU', 'GADS' and 'PR')
;
; OPTIONAL KEYWORDS:
;   None

```

```
;
; OUTPUTS:
;   members      Names of quality members of extname (STRARR)
```

---

## 6.7 Data handling methods

### NAME

Subset\_TSD

### SYNOPSIS

```
; CALLING SEQUENCE:
;   subset = tsd->Subset_TSD(index)
;
; INPUTS:
;   index      Index array which specifies row numbers of original TSD
;
; OPTIONAL KEYWORDS:
;   None
;
; OUTPUTS:
;   subset      A TSD object instance which is newly created. Specified
;               row index of original TSD (current instance) is copied.
;               Size of subset is equivalent to N_ELEMENTS(index).
;
; EXAMPLE:
;   You can get subset of current TSD between row# = 100-109 by entering:
;
;   IDL> offset = 100
;   IDL> index = INDGEN(10) + offset
;   IDL> subset = tsd->Subset_TSD(index)
;   IDL> dummy = subset->Write_TSD('output.fits')
;
;   You can check the subset TSD file (output.fits) as follows:
;
;   IDL> newtsd = OBJ_NEW('tsd')
;   IDL> dummy = newtsd->Read_TSD('output.fits')
;   IDL> aftime = newtsd->Get_Value_TSD('FIS_OBS', 'aftime')
;   IDL> help, aftime
;   AFTIME      INT      = Array[10]
```

---

## 7 The LDS Client Functions

The LDS version 2 and its client communicate each other using XMLRPC-like unique protocol. The users can easily access to the LDS using functions described in this section. The results are returned as a nested structure. A column element can be obtained like this:

```
print, info.(row_index).date_obs
```

and `N.TAGS(info)` shows the number of rows. See the code of `print.results()` method to know how to use returned information.

### 7.1 Object Initialization/Destruction

#### SYNOPSIS

```
; CALLING SEQUENCE:
;   ldc = OBJ_NEW('ldsclient')
;
; INPUTS:
;   'ldsclient' Name of object class
;
; OUTPUTS:
;   ldc          Object instance
;
; EXAMPLE:
;   IDL> ldc=OBJ_NEW('ldsclient')
;   IDL> help, ldc
;   LDC          OBJREF      = <ObjHeapVar1(LDSCLIENT)>
```

---

#### SYNOPSIS

```
; CALLING SEQUENCE:
;   OBJ_DESTROY, ldc
;
; INPUTS:
;   ldc          Name of object instance
;
; EXAMPLE:
;   IDL> ldc=OBJ_NEW('ldsclient')
;           :
;           :
;   IDL> OBJ_DESTROY, ldc
```

---

## 7.2 Methods

### NAME

Get\_TermInfo

### SYNOPSIS

```
; CALLING SEQUENCE:
;   info = ldc->get_terminfo( MODE=mode, GB_PROCESSED=gb_processed, $
;                               FMTTYPE=fmttype, $
;                               DATE=date, AFTIME=aftime, SPAN=span )
;
; PURPOSE:
;   This method sends a request to get the registration information of
;   the LDS files and receives results from the LDS.
;
; INPUTS:
;   NONE
;
; OPTIONAL KEYWORDS:
;   MODE           Data query mode: ("Fli" or "Rel": default: "Fli")
;   GB_PROCESSED   Flag whether data is processed by GreenBox or not
;                   (0: not_processed, 1: processed. Default: 0)
;   FMTTYPE        'FIS_SW' or 'FIS_LW'.
;   DATE           Start time of requested information.
;                   DATE should be set in UTC.
;                   (STRING FORMAT: '2006-07-10T23:40:00.000Z')
;   AFTIME         Start time of requested information.
;                   DATE is ignored when AFTIME is set.
;   SPAN           Span of request information (seconds)
;
; OUTPUTS:
;   info           Array of information structure
;                   0 is returned if failed.
;
; EXAMPLE:
;
;   IDL> ldc = obj_new('ldsclient')
;   IDL> info = ldc->get_terminfo( MODE='Fli', FMTTYPE='FIS_LW', $
;                                   DATE='20070201003000', SPAN=86400 )
;   IDL> n = ldc->print_results()
;   IDL> OBJ_DESTROY, ldc
;
```

---

### NAME

ClipOut\_TSD

### SYNOPSIS

```
; CALLING SEQUENCE:
;   info = ldc->clipout_tsd( MODE=mode, GB_PROCESSED=gb_processed, $
;                               FMTTYPE=fmttype, $
;                               DATE=date, AFTIME=aftime, SPAN=span, $
```



```

;                                     GZIP=gzip, FILENAME=filename, DIRNAME=dirname )
;
;
; PURPOSE:
;   This method sends a request for clipping out a TSD file, downloads
;   it and receives results from the LDS.
;
;
; INPUTS:
;   NONE
;
;
; OPTIONAL KEYWORDS:
;   MODE           Data query mode: ('Fli' or 'Rel': default: 'Fli')
;   GB_PROCESSED   Flag whether data is processed by GreenBox or not
;                  (0: not_processed, 1: processed. Default: 0)
;   FMTTYPE        'FIS_SW' or 'FIS_LW'.
;   DATE           Start time of requested information.
;                  DATE should be set in UTC.
;                  (STRING FORMAT: '2006-07-10T23:40:00.000Z')
;   AFTIME         Start time of requested information.
;                  DATE is ignored when AFTIME is set.
;   SPAN           Span of request information (seconds)
;   GZIP           Set 1 when you want get a compressed (gzipped) file.
;                  (default: 0)
;   FILENAME       Name of downloaded TSD file. The name of this file
;                  depends on the server's response, when FILENAME is
;                  not set.
;   DIRNAME        Name of the destination directory to save the downloaded
;                  file.
;
;
; OUTPUTS:
;   info           Array of information structure
;                  0 is returned if failed.
;
;
; EXAMPLE:
;
;   IDL> ldc = obj_new('ldsclient')
;   IDL> info = ldc->clipout_tsd( MODE='Fli', FMTTYPE='FIS_LW', $
;                               DATE='20070201003000', SPAN=2010, $
;                               GZIP=1, DIRNAME='dest_dir' )
;
;   IDL> n = ldc->print_results()
;   IDL> OBJ_DESTROY, ldc
;
;

```

## NAME

Get\_PosInfo

## SYNOPSIS

[illegible]

```

;                                     DATE=date, AFTIME=aftime, SPAN=span, $
;                                     LON=lon, LAT=lat, RADIUS=radius, $
;                                     SCANSPEED_MIN=scanspeed_min, $
;                                     SCANSPEED_MAX=scanspeed_max, $
;                                     [/J2000], [/B1950], [/EQUATORIAL], [/GALACTIC] )
;
; INPUTS:
;   NONE
;
; PURPOSE:
;   This method sends a request to get the positional information of
;   the TSD files and receives results from the LDS.
;
; OPTIONAL KEYWORDS:
;   MODE          Data query mode: ("Fli" or "Rel": default: "Fli")
;   GB_PROCESSED  Flag whether data is processed by GreenBox or not
;                 (0: not_processed, 1: processed. Default: 0)
;   FMTTYPE       'FIS_SW' or 'FIS_LW'
;   PACKETID      Packetid in the FIS_OBS BTE.
;   ORIGIN        Origin of positional information.
;                 'PR','GADS' or 'AOCU' can be set.
;   DATE          Start time of requested information.
;                 DATE should be set in UTC.
;                 (STRING FORMAT: '2006-07-10T23:40:00.000Z')
;   AFTIME        Start time of requested information.
;                 DATE is ignored when AFTIME is set.
;   SPAN          Span of request information (seconds)
;   LON           R.A. or longitude (degree)
;   LAT           Dec. or latitude (degree)
;   RADIUS        Radius of FOV (degree)
;   /J2000        Original equinox of coordinates (default)
;   /B1950        Another equinox of coordinates
;   /EQUATORIAL   Input is equatorial coordinate system(RA,DEC) (default)
;   /GALACTIC     Input is galactic coordinate system
;
; OUTPUTS:
;   info          Array of information structure
;                 0 is returned if failed.
;
; OUTPUT KEYWORDS:
;   NONE
;
; RESTRICTIONS:
;   Return value becomes 0 if both time and coordinates are specified.
;
; EXAMPLE:
;   IDL> ldc = obj_new('ldsclient')
;   IDL> info = ldc->get_posinfo( MODE='Fli', FMTTYPE='FIS_SW', $
;                                LON=268.0, LAT=66.0, RADIUS=0.5, $
;                                SCANSPEED_MIN=0.05, SCANSPEED_MAX=0.07 )

```

```

;      IDL> n = ldc->print_results()
;      IDL> OBJ_DESTROY, ldc
;

```

---

**NAME**

Print\_Results

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   n = ldc->print_results( RESULTS=info, MAX=max )
;
; INPUTS:
;   NONE
;
; PURPOSE:
;   This method prints the results received from the LDS.
;
; OPTIONAL KEYWORDS:
;   RESULTS      Return value of get_terminfo(), clipout_tsd() or
;                 get_posinfo()
;   MAX          Number of rows to print results
;
; OUTPUTS:
;   n            Number of rows.
;
; OUTPUT KEYWORDS:
;   NONE
;

```

---

**NAME**

Clipout\_TSDs\_From\_Results

**SYNOPSIS**

```

; CALLING SEQUENCE:
;   n = ldc->clipout_tsds_from_results( RESULTS=info, MAX=max, $
;                                     MODE=mode, GB_PROCESSED=gb_processed, $
;                                     FMTTYPER=fmtype, GZIP=gzip, $
;                                     DIRNAME=dirname )
;
; INPUTS:
;   NONE
;
; PURPOSE:
;   This method downloads the TSDs using info from the LDS.
;
; OPTIONAL KEYWORDS:
;   RESULTS      Return value of get_terminfo() or get_posinfo()
;   MAX          Number of downloaded TSD files.
;   MODE         Data query mode: ("Fli" or "Rel": default: "Fli")
;   GB_PROCESSED Flag whether data is processed by GreenBox or not

```

```
;          (0: not_processed, 1: processed.  Default: 0)
;  FMTTYPE  'FIS_SW' or 'FIS_LW'
;  GZIP     Set 1 when you want get a compressed (gzipped) file.
;          (default: 0)
;  DIRNAME  Name of the destination directory to save the downloaded
;          file.
;
; OUTPUTS:
;  filenames  Name of downloaded files (STRARR).
;
; OUTPUT KEYWORDS:
;  NONE
;
; EXAMPLE:
;  IDL> ldc = obj_new('ldsclient')
;  IDL> info = ldc->get_posinfo( LON=268.0, LAT=66.0, RADIUS=0.5, $
;                               SCANSPEED_MIN=0.05, SCANSPEED_MAX=0.07 )
;  IDL> filenames = ldc->clipout_tsds_from_results(RESULTS=info,MAX=4,GZIP=1)
;  IDL> OBJ_DESTROY, ldc
;
```

---

## Appendix: Brief summary of differences between TSD definition version 3 and 4

### Reasons of re-structuring

- \* Renamed/moved/added a number of columns and revised the status bits in the FIS\_HK, IRC\_HK, HK\_2, AOCU and GADS extensions, to follow to the latest output SIB.
- \* Reversed/clarified the bit order of status column in each extension.
- \* And else.

### Added fields:

- \* FIS\_OBS extension
  - flag
  - untrusted\_frame
  - pix\_flag
  - no\_peri\_corr
- \* FIS\_HK extension
  - status
  - blank\_subcom\_data
  - fis\_lw\_wire\_light(1B), fis\_run\_mode(1B), fis\_obs\_mode(1B),
  - fis\_asq\_pattern(1B), fis\_asq\_step(1J), sw\_heater\_tmp(1B)
- \* HK\_2 extension
  - status
  - blank\_cryo\_temp\_data, fis\_cpu\_run\_rst, fis\_fis\_on\_off,
  - fis\_tmp\_pol\_pos\_neg, fis\_obs\_seq, fis\_rsw\_auto\_cmnd,
  - fis\_fcsl\_frq\_b1, fis\_fcsl\_frq\_b0, irc\_cpu\_ru\_rs, irc\_irc\_on\_of
  - cryo\_baffle\_2(1E), cryo\_cold\_hd\_a\_dr\_i(1E), cryo\_cold\_hd\_b\_dr\_i(1E),
  - cryo\_comp\_a\_dr\_i(1E), cryo\_comp\_b\_dr\_i(1E),
  - irc\_mainref\_tmp1(1E), irc\_subref\_tmp(1E), irc\_fpip\_tmp1(1E)
- \* AOCU extension
  - status
  - ao\_rom\_ram
  - aocu\_saab\_mode(1B), aocu\_sbab\_mode(1B),
  - aocu\_saab\_win\_num(1B), aocu\_sbab\_win\_num(1B)
  - quality
  - interpolated
- \* GADS extension
  - quality
  - interpolated
- \* PR extension
  - quality
  - interpolated

### Removed fields:

- \* FIS\_HK extension
  - tmppol
- \* IRC\_HK extension
  - tp\_pmrr, tp\_smrr, tp\_fpip

```

* AOCU extension
  - status
    stt, tfss
  pos_err(3D)

```

### Renamed or Redefined fields:

```

* FIS_OBS extension
  pimtior(1J) -> pimtior(1K)
  fistior(1J) -> fistior(1K)
  pimti(1J) -> pimti(1K)
  fisti(1J) -> fisti(1K)
  mposcnt(11I) -> mposcnt(1J)
  - status
    fwposb1 -> fwpos_b1
    fwposb0 -> fwpos_b0
    mposb1 -> mpos_b1
    mposb0 -> mpos_b0
* FIS_HK extension
  pimtior(1J) -> pimtior(1K)
  fwfpul -> fis_fw_f_pul
  fwrpul -> fis_fw_r_pul
* IRC_HK extension
  pimtior(1J) -> pimtior(1K)
* HK_2 extension
  pimtior(1J) -> pimtior(1K)
  - status
    invalid -> cryo_temp_data_en_ds
    tp_he_tank -> cryo_he_tank_1
    tp_baffle -> cryo_baffle_1
* AOCU extension
  pimtior(1J) -> pimtior(1K)
  - status
    mode -> ads_mode_b1, ads_mode_b0
    quaternion -> aocu_ads_q
    body_rate -> aocu_body_rate
    rang -> roll
    dra -> d_ra
    ddec -> d_dec
    drang -> d_roll
* GADS extension
  pimtior(1J) -> pimtior(1K)
  quaternion -> attitude
  pos_err -> attitude_er
  body_rate -> angl_vel
  rang -> roll
  dra -> d_ra
  ddec -> d_dec
  drang -> d_roll
* PR extension

```

```
pimtiorg(1J) -> pimtiorg(1K)
rang -> roll
dra -> d_ra
ddec -> d_dec
drang -> d_roll
* SE extension
sat_x -> sat_posx
sat_y -> sat_posy
sat_z -> sat_posz
```

## Appendix 2: Names of available arguments for TSD4 methods

### Available extension names

TSD::(Get|Set)\_(Value|Status|Flag|Pix\_Flag|Quality)(\_Members)\_TSD()

Following args can be set as the first argument of each method:

'FIS\_OBS', 'FIS\_HK', 'IRC\_HK', 'HK\_2', 'AOCU', 'GADS', 'PR', 'SE'  
which correspond to each binary extension name of TSD.

### FIS\_OBS

- TSD::(Get|Set)\_Value\_TSD('FIS\_OBS', keyword)
  - aftime pimtiorg fistiorg pimti fisti status packetid
  - rstcntwidel rstcntwides rstcntn170 rstcntn60 calplscntal
  - calplscntas calplscntb aderrcntsw aderrcntlw mposcnt
  - dacbiaswides dacbiasn60 dacbiaswidel dacbiasn170 daccalas
  - daccalal daccalb dacsinas dacsinal tpbodys tpsw tplw tpcalft
  - det flux ferr flag pix\_flag quality cnt\_saa cnt\_glitch\_gp cnt\_glitch\_mt
- TSD::(Get|Set)\_Status\_TSD('FIS\_OBS', keyword)
  - creon shtop fwposon fwpos\_b1 fwpos\_b0 mposon mpos\_b1 mpos\_b0
  - rstwidelon rstwideson rstn170on rstn60on swbooston lwbooston
  - swbiason lwbiason calalon calason calbon sinalon sinason
- TSD::(Get|Set)\_Flag\_TSD('FIS\_OBS', keyword)
  - bad\_frame undef\_anom\_frame blank in\_saa near\_moon untrusted\_frame
- TSD::(Get|Set)\_Pix\_Flag\_TSD('FIS\_OBS', keyword)
  - bad undef\_anom arith\_err dead saturate reset rstanom
  - no\_diff no\_rp\_corr no\_dk\_corr no\_dccal\_corr no\_tr\_corr
  - no\_flat\_field no\_gpgl no\_mtgl no\_abscal
  - gpgl\_type1 gpgl\_type2 gpgl\_type3 gpgl\_type4 gpgl\_tail
  - mtgl\_type1 mtgl\_type2 mtgl\_type3 mtgl\_type4 mtgl\_tail
  - no\_peri\_corr sx\_peak sx\_source sx\_obj
- TSD::(Get|Set)\_Quality\_TSD('FIS\_OBS', keyword)
  - qual\_cv\_param qual\_rc\_param qual\_rc\_cf qual\_df\_eq
  - qual\_rp\_data qual\_rp\_param qual\_rp\_table qual\_ff\_param
  - qual\_ff\_cf qual\_gpgl\_corr qual\_mtgl\_corr qual\_tr\_hist
  - qual\_tr\_param qual\_dk\_data qual\_dk\_param qual\_dk\_table
  - qual\_fx\_corr qual\_fx\_param

### FIS\_HK

- TSD::(Get|Set)\_Value\_TSD('FIS\_HK', keyword)
  - aftime pimtiorg status fis\_fw\_f\_pul fis\_fw\_r\_pul
  - fis\_lw\_wire\_light fis\_run\_mode fis\_obs\_mode fis\_asq\_pattern
  - fis\_asq\_step sw\_heater\_tmp
- TSD::(Get|Set)\_Status\_TSD('FIS\_HK', keyword)
  - blank blank\_subcom\_data
- TSD::(Get|Set)\_Flag\_TSD('FIS\_HK', keyword)
  - (nothing)
- TSD::(Get|Set)\_Pix\_Flag\_TSD('FIS\_HK', keyword)
  - (nothing)
- TSD::(Get|Set)\_Quality\_TSD('FIS\_HK', keyword)



(nothing)

## IRC\_HK

- TSD::(Get|Set)\_Value\_TSD('IRC\_HK', keyword)  
    aftime pimtior status
- TSD::(Get|Set)\_Status\_TSD('IRC\_HK', keyword)  
    blank
- TSD::(Get|Set)\_Flag\_TSD('IRC\_HK', keyword)  
    (nothing)
- TSD::(Get|Set)\_Pix\_Flag\_TSD('IRC\_HK', keyword)  
    (nothing)
- TSD::(Get|Set)\_Quality\_TSD('IRC\_HK', keyword)  
    (nothing)

## HK\_2

- TSD::(Get|Set)\_Value\_TSD('HK\_2', keyword)  
    aftime pimtior status cryo\_he\_tank\_1 cryo\_baffle\_1  
    cryo\_baffle\_2 cryo\_cold\_hd\_a\_dr\_i cryo\_cold\_hd\_b\_dr\_i  
    cryo\_comp\_a\_dr\_i cryo\_comp\_b\_dr\_i irc\_mainref\_tmp1  
    irc\_subref\_tmp irc\_fpip\_tmp1
- TSD::(Get|Set)\_Status\_TSD('HK\_2', keyword)  
    blank blank\_cryo\_temp\_data cryo\_temp\_data\_en\_ds  
    fis\_cpu\_run\_rst fis\_fis\_on\_off fis\_tmp\_pol\_pos\_neg  
    fis\_obs\_seq fis\_rsw\_auto\_cmnd fis\_fcal\_frq\_b1 fis\_fcal\_frq\_b0  
    irc\_cpu\_ru\_rs irc\_irc\_on\_of
- TSD::(Get|Set)\_Flag\_TSD('HK\_2', keyword)  
    (nothing)
- TSD::(Get|Set)\_Pix\_Flag\_TSD('HK\_2', keyword)  
    (nothing)
- TSD::(Get|Set)\_Quality\_TSD('HK\_2', keyword)  
    (nothing)

## AOCU

- TSD::(Get|Set)\_Value\_TSD('AOCU', keyword)  
    aftime pimtior status aocu\_ads\_q aocu\_body\_rate  
    ra dec roll d\_ra d\_dec d\_roll crs\_off
- TSD::(Get|Set)\_Status\_TSD('AOCU', keyword)  
    blank ao\_rom\_ram ads\_mode\_b1 ads\_mode\_b0
- TSD::(Get|Set)\_Flag\_TSD('AOCU', keyword)  
    (nothing)
- TSD::(Get|Set)\_Pix\_Flag\_TSD('AOCU', keyword)  
    (nothing)
- TSD::(Get|Set)\_Quality\_TSD('AOCU', keyword)  
    interpolated

## GADS

- TSD::(Get|Set)\_Value\_TSD('GADS', keyword)

```

    aftime pmtiorg status attitude attitude_er angl_vel
    ra dec roll d_ra d_dec d_roll el_sol aa_sol aa_ear aa_lun crs_off
- TSD::(Get|Set)_Status_TSD('GADS', keyword)
    blank
- TSD::(Get|Set)_Flag_TSD('GADS', keyword)
    (nothing)
- TSD::(Get|Set)_Pix_Flag_TSD('GADS', keyword)
    (nothing)
- TSD::(Get|Set)_Quality_TSD('GADS', keyword)
    interpolated

```

## PR

```

- TSD::(Get|Set)_Value_TSD('PR', keyword)
    aftime pmtiorg status quaternion quat_err body_rate
    ra dec roll d_ra d_dec d_roll el_sol aa_sol aa_ear aa_lun crs_off
- TSD::(Get|Set)_Status_TSD('PR', keyword)
    blank
- TSD::(Get|Set)_Flag_TSD('PR', keyword)
    (nothing)
- TSD::(Get|Set)_Pix_Flag_TSD('PR', keyword)
    (nothing)
- TSD::(Get|Set)_Quality_TSD('PR', keyword)
    interpolated

```

## SE

```

- TSD::(Get|Set)_Value_TSD('SE', keyword)
    aftime pmtiorg status tm_saa sat_posx sat_posy sat_posz
- TSD::(Get|Set)_Status_TSD('SE', keyword)
    blank type daynight in_saa in_polar
- TSD::(Get|Set)_Flag_TSD('SE', keyword)
    (nothing)
- TSD::(Get|Set)_Pix_Flag_TSD('SE', keyword)
    (nothing)
- TSD::(Get|Set)_Quality_TSD('SE', keyword)
    (nothing)

```