EUMETSAT POLAR SYSTEM

MetOpizer Users Guide

DOCUMENT SIGNATURE TABLE

	NAME	FUNCTION	SIGNATURE	DATE
Prepared by	**Mike Elson**	Software Engineer		
	Sukhdav Sagoo	Software Engineer		
	Theo Steenbergen	Software Engineer		
Issued by	**Simon Elliott**	Product Implementation Manager		

DOCUMENT CHANGE LOG

Issue	Revision	Date	DCR no.	Pages Affected	Reason for change
1	0	x	-	-	Initial issue
1	1	13/03/03			Update for tVCDU generation.
2	0	20/07/04	-	-	Updates for Metopizer v3.0 - ccsds_displayer updated to handle IASI, ASCAT, GRAS and GOME CCSDS packets. - ccsds_iasi_flags_fix developed to correct IASI CCSDS packet flag settings. - ccsds_strip_filter developed to remove CCSDS packets that do not satisfy specified UTC start and end times. - ccsds_time_and_seq_fix developed to modify/offset the sequence counter in the CCSDS primary header and the OBT time field of the CCSDS ancillary data packet. - ccsds_utc_fix developed to set CCSDS packet UTC times. - Document renamed from Metopizer Specifications and Design to Metopizer Users Guide. Previous Users Guide document removed.

3	6	12/05/2005		-	Updates for Metopizer v3.0 - ccsds_utc_fix renamed to ccsds_time_fix - ccsds_time_and_seq_fix renamed to ccsds_seq_fix - Appendix C removed - Note that Reduced MHS packets are handled. New executables: - ccsds_to_l0 - ccsds_corrupt_filter - mhs_to_ccsds New filters: - CCSDSToL0 - EPSProductExtractor - CCSDSCorruptFilter New interfaces: - EPSProductProducer New executables: - tvcdu_corrupt_filter - cadu_corrupt_filter - buffer_corrupt_filter New filters: - tVCDUCorruptFilter - CADUCorruptFilter BufferCorruptFilter

3	19	19/05/2006			New Build Configuration: - autoconf support - Python extensions support New Scripts: - metopizer_viewer - Python extensions - tvcdu_summarizer New filters: - CADUWriter - CADUSplitter New applications: - cadu_displayer - l0_to_reduced_ccsds - cadu_orbit_splitter - sband_tm_displayer New features: - ccsds_displayer: Check of OBT/UTBC correlation, IASI IIS display, display shifted GRAS OBT times

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Purpose and Scope

This document describes the MetOpizer collection of tools. The following sections describe the concepts used, the capabilities of the software and how to use it.

1.2 Document Structure

Section 1 – Introduction

Section 2 - Installation

Section 3 - Overview of the design

Section 4 - Description of the programs that make up MetOpizer

Section 5 - Filters

Section 6 – Interfaces

Section 7 – Python Interface

Appendix A - Details of how each instruments CCSDS packets are assembled

Appendix B - Instrument Configuration files

Appendix C – (Removed)

Appendix D - List of IASI flags that require fixing.

1.3 Referenced documents and pages

RD.1 Telemetry Channel Coding, CCSDS-101.0-B-3

RD.2 Simulated MHS Characteristics, Issue 1, EUM.EPS.SYS.TEN.02.005

RD.3 NOAA KLM Users Guide, www2.ncdc.noaa.gov/docs/klm

RD.4 MHS Housekeeping Telemetry to Engineering Units, Issue 3, MHS-TN-JA137-MMP

RD.6 AVHRR ICD, Issue 5 Rev 0, MO-IC-MMT-AH-001

RD.7 AMSU-A1 ICD, Issue 5 Rev 0, MO-IC-MMT-A1-001

RD.8 MHS ICD, Issue 5 Rev 0, MO-IC-MMT-MH-001

RD.9 HIRS ICD, Issue 5 Rev 0, MO-IC-MMT-HI-001

RD.10 Metop Space to Ground ICD, Issue 5, MO-IF-MMT-SYS001

RD.11 AMSU-A2 ICD, Issue 5 Rev 0, MO-IC-MMT-A2-001

RD.12 EPS Core Ground Segment Type Definitions EPS-ASPI-IR-0060

RD.13 IASI Measurement and Verification Data, Issue 2 Rev 5, IA-ID-1000-6477-AER

RD.14 GOME2 Science Data Packet Definition, Issue 7, MO-DS-LAB-GO-0006

RD.15 ASCAT Measurement Data Interface Specification, Issue 3, MO-TN-DOR-SC-0015

RD.16 GRAS Data Interface Control Document, Issue 7, P-GRM-ICD-0008-SE

RD.17 IASI Data Processing System (DPS) Specification, Issue5, IA-SP-1520-227-AER

RD.18 Python homepage, http://www.python.org

RD.19 MatPlotLib homepage, http://matplotlib.sourceforge.net.

2 INSTALLATION

For a normal install of the Metopizer use the standard autoconf commands, ie.:

- ./configure
- make
- make install

Run ‘./configure –help’ for additional options. Some common options are:

—enable-python : Specify that the optional Python module is to be built.

—enable-debug : Compile all object files with debugging flags on.

--prefix <dir> : Set the target install directory to <dir>.

The default install directory is /usr/local .

The Metopizer is written in C++. It has been developed and tested with gcc 3.3 and above, but may work with other compilers. Libxml2, libpng and libjpeg will all be used if present.

The Metopizer runs on Linux, Solaris and AIX, and on Windows using the Cygwin environment.

3 OVERVIEW OF DESIGN

MetOpizer is a collection of programs, each manipulating input files in a specific way. The components of MetOpizer are each put together using the same basic design with the intention being to eliminate duplication of code.

The main objective of the design is to produce a modular system where each piece of code performs a specific simple task. An executable in the MetOpizer is built up from a number of standard modules. Since these are reused, once a module has been tested in one executable it can be used with confidence anywhere else that functionality is needed.

3.1 executables

In the context of the MetOpizer an executable is a list of filter objects with each one processing data and feeding it on to one or more filters further down the chain.

The code specific to each executable is used to parse any command line arguments, build the chain of filters and then execute them.

In general the actual programs will be very small, with all the work carried out by the filter modules.

3.2 filters

A filter is a module of code that carries out a specific low level function, accepting input from other filters and passing data on to further filters. Filters are implemented as C++ classes.

Where possible a filter is generic enough to be used for more than one purpose.

Each filter needs to be constructed with a specific data type, so a filter can only be constructed when it has the correct input data.

3.2.1 readers

A reader is a type of filter that is designed to read in a particular data type and convert it to several different output data types. The Reader type filters are generally where most of the MetOpizer functionality lies.

3.2.2 extractors

The extractor objects are used to select one particular type of output from a Reader object and convert it to a generic binary stream (using the FileProducer interface). The data can then be sent to a filter that expects plain binary inputs such as a file writer.

For example the CCSDSExtractor object pulls out CCSDS packets from any object that supports the correct interface (CCSDSProducer). The Extractors are needed where a reader can produce multiple data types that can be written to disk (ie. the AVHRRReader can generate source packets, pictures and interpolated products).

3.3 interfaces

The links between filters are made using interfaces. An interface, as used by MetOpizer, is a list of functions to be implemented by a filter. For example the interface 'CCSDSProducer' describes a piece of code that outputs CCSDS packets. Since the interface is a fixed list of functions, any filter that generates CCSDS packets implements this interface, and can then be connected to any filter that needs to carry out processing on CCSDS packets.

In all cases the mechanism to actually pass data from one filter to another is the STL 'ostream' class. This enables either files or blocks of memory to be passed around very efficiently (disk accesses can be cached and take advantage of the systems file block size) and without the need to distinguish between memory copies and disk read operations.

3.4 Multiple interfaces

Some filters can output several different types of data. For example the HIRSReader filter accepts Level 1b files as input, but can generate four types of output:

-CCSDS packets (via CCSDSProducer)

-Graphics image files (via ImageProducer)

-Text version of the product header (via TextHeaderProducer)

-Text version of a single record (via TextRecordProducer)

This system works because the HIRSProducer implements the functions needed for all four interfaces. A filter attached to the HIRSReader (the CCSDSExtractor for example) pulls out a particular data type by calling the functions of the CCSDSProducer interface.

3.5 Example interface - CCSDSProducer

The full list of interfaces is presented in Section 6, but in this section we examine one particular interface in detail as an example of how the principle works.

Interfaces are implemented as C++ classes. The definition of CCSDSProducer is given below:

```
class CCSDSProducer
{
```

```
   virtual CCSDS* get_next_ccsds(void) =0;
};
```

Any object which generates a stream of CCSDS packets will implement this interface, and any object that works with CCSDS packets as its input will call functions on a CCSDSProducer interface. In general the constructor function of a filter would include a reference to a Producer interface.

3.6 Code layout

All MetOpizer C++ code is placed in the 'src' subdirectory, with some additional scripts in the top directory. The installation and setup procedure is performed by using the ***autoconf*** install files which consist of the following:

File with relative location	Purpose	Scope
Metopizer/configure.ac	Setup & installation configuration scripting file used to automate code compilation for specific platforms	Changes will alter the initial build process
Metopizer/Makefile.am	Main make file that adds additional install steps and files	Affects which additional files are installed on local machine
Metopizer/autogen.sh	Main setup and install script which is the first file that is run, note that parameters can be passed to this script. This script first calls ***autoclean.sh***	Affects which system level programs are used to setup the *configure* and *make* files used with autoconf
Metopizer/autoclean.sh	Standalone script used to cleanup all temporary files created during the configuration, make and install processes	Changes will decide which files are deleted when the script is run
Metopizer/src/Makefile.am	Defines each binary and library file built with indication of source files that apply to individual binaries. The extended compiler options are also specified here	Changes will affect the Makefile generation behaviour and any custom compilation steps

4 METOPIZER EXECUTABLES

The following executables give the main MetOpizer functionality.

In each case only the overall program is described, the details of each step in the processing chains are covered in Section 6.

Also note in most cases the programs use TextSummaryDisplayer objects to give the user information of the processing carried out, unless this is suppressed with a command line switch. These have been omitted from diagrams for clarity.

4.1 amsua_to_ccsds

4.1.1 Purpose

To convert a NOAA L1b format AMSU-A product into a stream of CCSDS packets, or a series of images.

4.1.2 Inputs

NOAA AMSU-A Level 1b product.

4.1.3 Outputs

Depending on parameters used:

-A file of concatenated CCSDS packets

-A series of images, one per radiometric channel.

4.1.4 Usage

amsua_to_ccsds [--in <filename>] [--out <filename>] [--config-a1 <filename>] [--config-a2 <filename>] [--show-config] [--check-config] [--image <filename prefix>] [--help] [--version] [--jpeg]

where:

--in <filename> : Specify name of input file. If omitted stdin is used.

--out <filename> : Specify name of output file. If omitted (and the –image tag is not found) a stream of source packets will be written to stdout.

--config-a1 <filename> : Specify the configuration file used for generating AMSU-A1 source packets as detailed in Section 9.1. If the switch is absent the filename “$METOPIZER_HOME/amsua1.config” will be searched, otherwise

'$HOME/.metopizer/amsua1.config'. If no configuration file can be found a set of default values will be used.

--config-a2 <filename> : As above, but containing options for AMSU-A2 packets and using 'amsua2.config' as the default filename.

--image <filename prefix> : Generate a sequence of images in TIFF format based on the input file. The output consists of 15 monochrome images. No CCSDS packets are generated and the configuration files are not used.

--show-config : Display the contents of the configuration files used after the main processing.

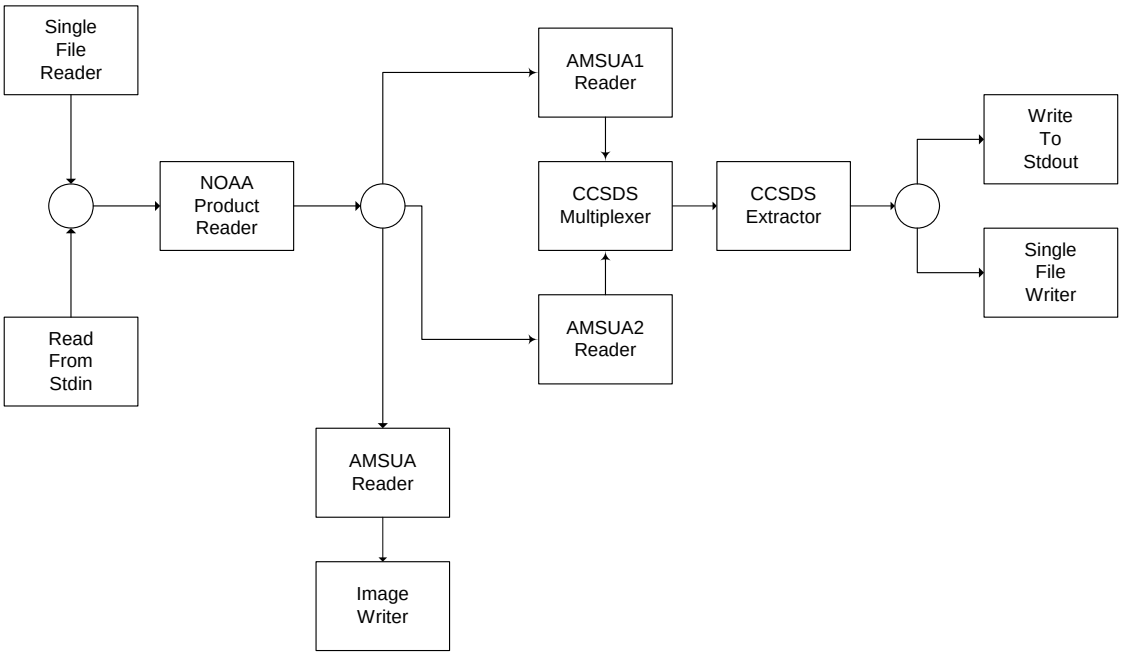
--check-config : The configuration files are read in and their contents displayed. No other processing is performed.

--help : show usage information

--version : show program version

--jpeg : Output JPEG format images instead of TIFF.

4.1.5 Functional diagram



4.1.6 Description

When source packet output is selected both AMSU-A1 and AMSU-A2 packets are generated. The two types are written alternately to file, and the ccsds_apid_filter program can be used to separate them if needed.

When image output is used each channel is normalised to cover the full range of colour values.

4.2 amsub_to_mhs_ccsds

4.2.1 Purpose

To convert a NOAA L1b format AMSU-B product into a stream of MHS format CCSDS packets, or to a series of images.

Additional telemetry information required for the MHS but not present in the AMSU-B data is read from a configuration file. This is intended to simulate a fully operational MHS unit in nominal measurement mode.

4.2.2 Inputs

NOAA AMSU-B Level 1b product.

4.2.3 Outputs

Depending on parameters used:

-A file of concatenated CCSDS packets

-A series of images, one per radiometric channel.

4.2.4 Usage

amsub_to_mhs_ccsds [--in <filename>] [--out <filename>] [--config <filename>] [--show-config] [--check-config] [--image <filename prefix>] [--help] [--version] [--jpeg]

where:

--in <filename> : Specify name of input file. If omitted stdin is used.

--out <filename> : Specify name of output file. If omitted and the –image option is not found then stdout is used.

--config <filename> : Specify name of CCSDS options file as detailed in Section 9.5. If the parameter is omitted the filename “$METOPIZER_CONFIG_DIR/mhs.config” will be tried, otherwise “$HOME/.metopizer/mhs.config” is looked for. If no configuration file is found set of default values are used.

--show-config : After processing display a list of the configuration parameters used.

--check-config : Only show the configuration parameters. No other processing is performed.

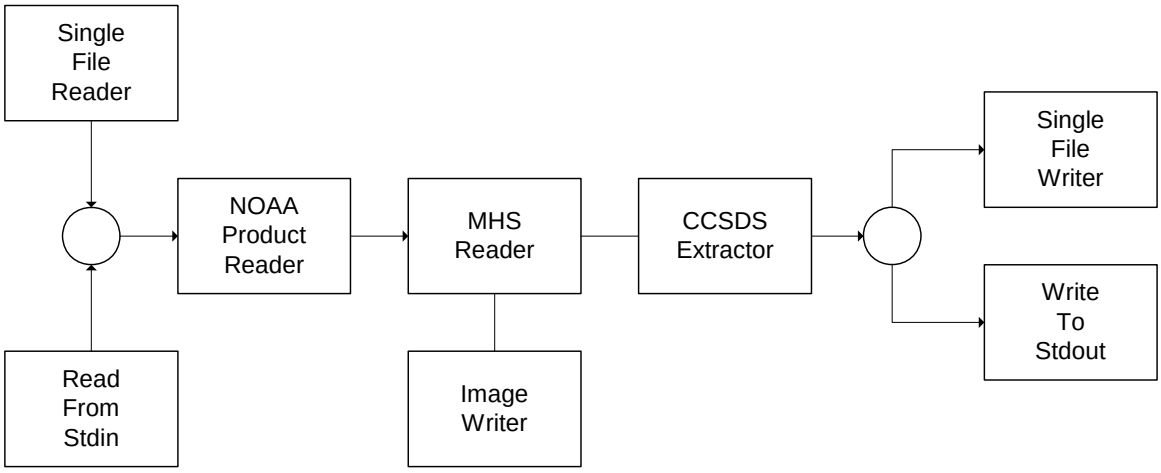
--image <filename prefix> : Generate a sequence of images in TIFF format based on the input file. The output consists of 15 monochrome images. No CCSDS packets are generated.

--help : show usage information

--version : show program version

--jpeg : Output JPEG format images instead of TIFF.

4.2.5 Functional diagram



4.2.6 Description

When image output is used, each channel is normalised to cover the full range of colour values.

4.3 avhrr_to_ccsds

4.3.1 Purpose

To read in a NOAA format AVHRR Level 1b file in either LAC or GAC format and convert the data to a stream of MetOp source packets, or graphics images.

If the input is in GAC format an interpolation step can be used to simulate high resolution data.

4.3.2 Inputs

NOAA AVHRR Level 1b file.

4.3.3 Outputs

A stream of CCSDS packets either one per file or concatenated together, or a set of images.

4.3.4 Usage

avhrr_to_ccsds [--in <filename>] [--out <filename (prefix)>] [--config <filename>] [--show-config] [check-config] [--image <filename prefix>] [--interpolate] [--bps x] [--channels xxxxx] [--write-product] [--help] [--version] [--jpeg]

where:

--in <filename> : Specify name of input file. If omitted stdin is used.

--out <filename> : Specify name of output file. If omitted stdout is used.

--config <filename> : Specify name of CCSDS options file as detailed in Section 9.3. If the parameter is omitted the file “avhrr.config” is looked for first in the directory $METOPIZER_CONFIG_DIR, then “$HOME/.metopizer”. If no configuration file is found set of default values are used.

--show-config : After processing display a list of the configuration parameters used.

--check-config : Only show the configuration parameters. No other processing is performed.

--image <filename prefix> : Generate a sequence of images in TIFF format based on the input file. The output includes 5 monochrome images and a false colour image. No CCSDS packets are generated.

--interpolate : Convert input product from GAC format to LAC format.

--bps x : Specify that the product uses x bits per sample format. Not required when a NOAA archive header is present. The default value is 10 bpp.

--channels xxxxx : Each x specifies whether a particular channel 1-5 is present in the input product. Not required if a NOAA archive header is present. The default value is 'yyyyy'.

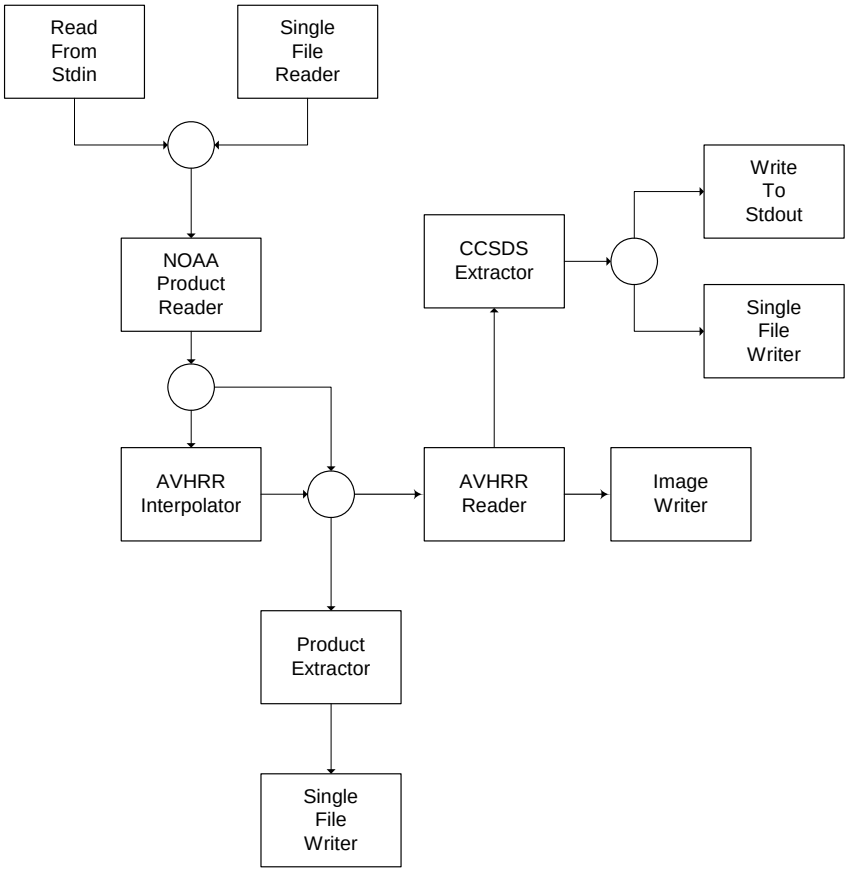
--write-product <filename> : Write the input product back to disk as <filename>. This is can be used with the --interpolate option to convert a GAC product file to a LAC format file.

--help : show usage information

--version : show program version

--jpeg : Output JPEG format images instead of TIFF.

4.3.5 Functional Diagram



4.3.6 Description

The input file must be either a LAC (1.1km resolution) or GAC (4km resolution) Level 1b AVHRR product.

CCSDS output is only available for LAC format products, so the –interpolate option must be used if the input is in the GAC format.

There are some restrictions on the format of the input products. This is due to the number of different AVHRR product formats, some of which cannot be distinguished automatically. For products with an SAA Archive Header there is no problem as this section fully describes the product, but when an archive header is not present the product must have all 5 channels present in order to be processed automatically.

If this is not the case the –bps and –channels flags must be used to tell the program what the input format is.

The --write-product option can be used to convert products on disk from GAC to LAC format in conjunction with the –interpolate flag.

4.4 buffer_corrupt_filter

4.4.1 Purpose

Enables the modification/corruption of any input file/buffer.

4.4.2 Inputs

One or more input files

4.4.3 Outputs

A single output file consisting of the input files and any specified corruptions

4.4.4 Usage

buffer_corrupt_filter [-i|--in <filename>] [-o|--out <filename>] [-z|-size N] [-s|--skip <N>] [-c|--count <N>] [--for-byte-offset <N1|N1 N2>] [--set-byte <random|N1|N1 N2>] [--for-bit-offset <N1|N1 N2>] [--set-bit <random|N1|N1 N2|flip>] [--set-bit-width <N|random|N random>] [--set-increment <row|column>] [--set-max-corruptions <N>] [-m|--simple] [-h|--help] [-v|--version]

Note: To avoid unnecessary typing, most of the options above have short hand names by which they can be refereed to (-i, -o, -z, -s, -c, -m, -h and -v).

where:

--in <filename> : Allows an input file to be specified.

--out <filename> : Allows the name of an output file to be specified.

--size <N> : Allows users to split a file into smaller packets. If left undefined, a default size of 1024 bytes is used. If the default size of 1024 bytes is found to be larger than the size of the input file, the total size of the input file will be used instead.

--skip <N> : Allows N packets to be skipped before applying any corruptions.

--count <N> : Allows corruptions to be applied to only N packets starting at the first packet or after skipping a number of packets if the --skip option is used.

--for-byte-offset <N1|N1 N2> : Allows corruptions to be introduced to a single byte at byte offset N1 or to a range of bytes starting at byte offset N1 and ending at byte offset N2. Note that if the corruption is to be applied to the first byte, the byte offset must be set to zero. Hence, the byte offset must be greater or equal to zero and of course less than the packet length.

--set-byte <random|N1|N1 N2> : Allows a single byte per packet or a range of bytes per packet to be set to random values, to be set to a single value N1, or to be filled with a number pattern starting with a value N1 and incremented by a value N2. Note that N1 and N2 can not be set to anything greater than 255. While N1 can not be less than zero, the increment N2 can be assigned a negative value.

--for-bit-offset <N1|N1 N2> : Allows corruptions to be introduced to a single bit at bit offset N1 or to a range of bits starting at bit offset N1 and ending at bit offset N2. Note that if the corruption is to be applied to the first bit, the bit offset must be set to zero. Hence, the bit offset must be greater or equal to zero and of course less than the packet length. Further more, if the numbers N1 or N2 are preceded with a '-/+' symbol, they will be assumed to be relative offsets. This allows users to combine both the --for-byte-offset and the --for-bit-offset options in the following way:

If --for-byte-offset N1 --for-bit-offset = +N2
Then start offset = (N1*8)+N2 bits

If --for-byte-offset N1 and --for-bit-offset +N2 N3
Then start offset = (N1*8)+N2 bits and end offset = N3 bits

If --for-byte-offset N1 and --for-bit-offset +N2 +N3
Then start offset = (N1*8)+N2 bits and end offset = (N1*8)+N2+N3 bits

If --for-byte-offset N1 N2 and --for-bit-offset +N3 +N4
Then start offset = (N1*8)+N3 bits and end offset = (N2*8)+N4 bits

If --for-byte-offset N1 N2 and --for-bit-offset +N3
Then start offset = (N1*8)+N3 bits and end offset = N2*8 bits

--set-bit <random|N1|N1 N2|flip> : Allows a single bit per packet or a range of bits per packet to be set to random values, to be set to a single value N1(0/1), or to be filled with a number pattern starting with a value N1(0/1) and incremented by a value N2(0/1). Note that N1 and N2 can not be set to anything other than 0 and 1 and hence, only patterns such as 01010101 or 10101010 are possible. The final flip option can be used to change single or a range of bits from 0 to 1 or vice versa.

--set-bit-width <N|random|N random> : The --set-bit-width option will only work and can only be set if the --for-byte/bit-offset options are used to define a range of bytes/bits to corrupt. The width option can be set to a number N, the key word random or to both random and a numeric value at the same time. Assuming we want to modify bits starting at bit offset N1 and ending at bit offset N2:

If --set-bit-width N then the following bits will be modified
N1, N1+N, N1+2N, N1+3N ... N1+xN
as long as N1+xN is less than or equal to N2

If --set-bit-width random then the following bits will be modified
N1, N1+R1, N1+(R1+R2), N1+(R1+R2+R3), N1+(R1+R2+R3+...+Rx)
as long as N1+(R1+R2+R3+...+Rx) is less than or equal to N2. Note that R1, R2, Rx represents different random numbers determined by the code.

If --set-bit-width random N or --set-bit-width N random, then the value of N is simply used by the code to place a maximum limit on the random numbers so that they do not exceed the value of N.

Note that the width cannot be set to a value greater than the difference between offsets N1 and N2.

--set-increment <column|row> : This option only becomes relevant if one is attempting to assign a number pattern to a range of bits/bytes per packet. For example a user might want to corrupt byte offsets 101 to 105 (--for-byte-offset 101 105) with a number pattern: 1, 3, 5, 7, 9, ... (--set-byte 1 2). If the --set-increment option is left undefined, the resulting corruption per packet will be as follows:

Byte Offset	101	102	103	104	105
Packet 1	1	3	5	7	9
Packet 2	3	5	7	9	11
Packet 3	5	7	9	11	13
Packet 4	7	9	11	13	15
Packet 5	9	11	13	15	17
Packet 6	11	13	15	17	19

Note that the pattern has been increment over both column (Byte Offset) and row (Packet). Hence, if the --set-increment option is left undefined, the default option is to apply the pattern over both columns and rows. If however, the --set-increment is set to column, the pattern will only be incremented over the different byte offsets and will result into the following corruption:

Byte Offset	101	102	103	104	105
Packet 1	1	3	5	7	9
Packet 2	1	3	5	7	9
Packet 3	1	3	5	7	9
Packet 4	1	3	5	7	9
Packet 5	1	3	5	7	9
Packet 6	1	3	5	7	9

Similarly, if the --set-increment option is set to row, the pattern will only be incremented over the different packets and will result into the following corruption:

Byte Offset	101	102	103	104	105
Packet 1	1	1	1	1	1
Packet 2	3	3	3	3	3
Packet 3	5	5	5	5	5
Packet 4	7	7	7	7	7
Packet 5	9	9	9	9	9
Packet 6	11	11	11	11	11

Note that --set-increment will have a similar effect on the --set-bit option.

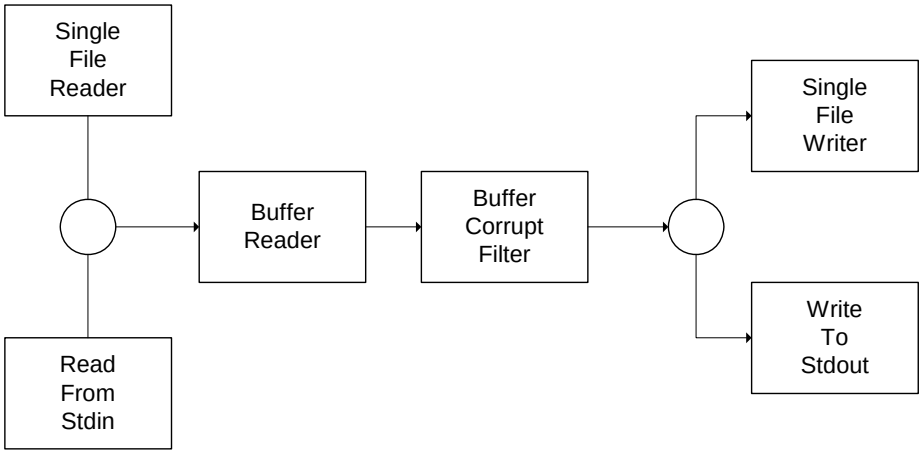
--set-max-corruptions <N> : Allows a limit to be set on the maximum number of corruptions that a user might like to introduce.

--simple : Use simple animations while processing.

--help : Shows usage information.

--version : Shows version information.

4.4.5 Functional Diagram



4.4.6 Description

The buffer_corrupt_filter allows a single or a range of bytes and bits to be modified in any input file. For example,

```
> buffer_corrupt_filter –i input –o output
   --for-byte-offset 0 3 --set-byte 255
```

The simple command above will set the value of bytes 0 to 3 to 255 (the maximum value a byte can be set to). Note that in this case a packet size has not been specified and the default packet size of 1024 bytes is used. If the input file is smaller than 1024 bytes, then the packet size is automatically set by the code to be the same size as the input file size.

```
> buffer_corrupt_filter –i input –o output –z 512
--for-byte-offset 4 15 --set-bit 1
```

The example above ensures that the packet size is set to 512 bytes. Note that in this case the --set-bit command is used instead of the --set-byte command. The --set-byte command must be used together with the --for-byte-offset option. However, the --set-bit command can be used with either the --for-byte-offset or the --for-bit-offset options.

```
> buffer_corrupt_filter –i input –o output –-size 512
```

```
--for-byte-offset 4 15 --for-bit-offset +2 –1 --set-bit flip
```

The --for-byte-offset and the --for-bit-offset options can be combined. If the +/- symbols are used, the bit range is defined relative to the byte range. In this example the bit range will start at 34 (4*8 + 2) and end at 119 (15*8 - 1). Of course if an absolute bit range is used, it will override any byte range that may also be defined.

```
> buffer_corrupt_filter –i input –o output –z 2048
--for-byte-offset 4 15 --for-bit-offset 5 23 --set-bit 0
--set-bit-width 7
```

The last example illustrates the use of the --set-bit-width option. This option enables users to set a bit to 0/1 or flip it at random intervals. In this particular case bit 5 will be set to 0 and also every 7th bit up until bit 23 is reached will also be set to 0. If the width value is set to 0, every bit up to and including bit 23 will be set to 0. The width value can also be set to random or random and a max-value. If set to random, the code will generate a random width by itself. If a max-value is specified, then the random values will not be able to exceed it.

4.5 cadu_corrupt_filter

4.5.1 Purpose

Enables the modification and corruption of CADU packets.

4.5.2 Inputs

One or more streams of CADU packets.

4.5.3 Outputs

CADU packets with applied corruptions.

4.5.4 Usage

cadu_corrupt_filter [-i|--in <filename(s)>] [-o|--out <filename>] [-s|--skip <N>] [-c|--count <N>] [--for-byte-offset <N1|N1 N2>] [--set-byte <random|N1|N1 N2>] [--for-bit-offset <N1|N1 N2>] [--set-bit <random|N1|N1 N2|flip>] [--set-bit-width <N|random|N random>] [--set-increment <row|column>] [--set-max-corruptions <N>] [-t|--show-cadu-table] [-m|--simple] [-h|--help] [-v|--version]

Note: To avoid unnecessary typing, most of the options above have short hand names by which they can be refereed to (-i, -o, -s, -c, -t, -m, -h and -v).

where:

--in <filename(s)> : Allows multiple input files to be specified. Note that multiple input file names should be delimited by spaces. If the '--in' option is omitted, stdin is used by default.

--out <filename> : Allows the name of an output file to be specified. If omitted stdout is used.

--skip <N> : Allows N packets to be skipped before applying any corruptions.

--count <N> : Allows corruptions to be applied to only N packets starting at the first packet or after skipping a number of packets if the --skip option is used.

--for-byte-offset <N1|N1 N2> : Allows corruptions to be introduced to a single byte at byte offset N1 or to a range of bytes starting at byte offset N1 and ending at byte offset N2. Note that if the corruption is to be applied to the first byte, the byte offset must be set to zero. Hence, the byte offset must be greater or equal to zero and of course less than the packet length.

--set-byte <random|N1|N1 N2> : Allows a single byte per packet or a range of bytes per packet to be set to random values, to be set to a single value N1, or to be filled with a

number pattern starting with a value N1 and incremented by a value N2. Note that N1 and N2 can not be set to anything greater than 255. While N1 can not be less than zero, the increment N2 can be assigned a negative value.

--for-bit-offset <N1|N1 N2> : Allows corruptions to be introduced to a single bit at bit offset N1 or to a range of bits starting at bit offset N1 and ending at bit offset N2. Note that if the corruption is to be applied to the first bit, the bit offset must be set to zero. Hence, the bit offset must be greater or equal to zero and of course less than the packet length. Further more, if the numbers N1 or N2 are preceded with a '-/+' symbol, they will be assumed to be relative offsets. This allows users to combine both the --for-byte-offset and the --for-bit-offset options in the following way:

If --for-byte-offset N1 --for-bit-offset = +N2
Then start offset = (N1*8)+N2 bits

If --for-byte-offset N1 and --for-bit-offset +N2 N3
Then start offset = (N1*8)+N2 bits and end offset = N3 bits

If --for-byte-offset N1 and --for-bit-offset +N2 +N3
Then start offset = (N1*8)+N2 bits and end offset = (N1*8)+N2+N3 bits

If --for-byte-offset N1 N2 and --for-bit-offset +N3 +N4
Then start offset = (N1*8)+N3 bits and end offset = (N2*8)+N4 bits

If --for-byte-offset N1 N2 and --for-bit-offset +N3
Then start offset = (N1*8)+N3 bits and end offset = N2*8 bits

--set-bit <random|N1|N1 N2|flip> : Allows a single bit per packet or a range of bits per packet to be set to random values, to be set to a single value N1(0/1), or to be filled with a number pattern starting with a value N1(0/1) and incremented by a value N2(0/1). Note that N1 and N2 can not be set to anything other than 0 and 1 and hence, only patterns such as 01010101 or 10101010 are possible. The final flip option can be used to change single or a range of bits from 0 to 1 or vice versa.

--set-bit-width <N|random|N random> : The --set-bit-width option will only work and can only be set if the --for-byte/bit-offset options are used to define a range of bytes/bits to corrupt. The width option can be set to a number N, the key word random or to both random and a numeric value at the same time. Assuming we want to modify bits starting at bit offset N1 and ending at bit offset N2:

If --set-bit-width N then the following bits will be modified
N1, N1+N, N1+2N, N1+3N ... N1+xN
as long as N1+xN is less than or equal to N2

If --set-bit-width random then the following bits will be modified
N1, N1+R1, N1+(R1+R2), N1+(R1+R2+R3), N1+(R1+R2+R3+...+Rx)
as long as N1+(R1+R2+R3+...+Rx) is less than or equal to N2. Note that R1, R2, Rx represents different random numbers determined by the code.

If --set-bit-width random N or --set-bit-width N random, then the value of N is simply used by the code to place a maximum limit on the random numbers so that they do not exceed the value of N.

Note that the width cannot be set to a value greater than the difference between offsets N1 and N2.

--set-increment <column|row> : This option only becomes relevant if one is attempting to assign a number pattern to a range of bits/bytes per packet. For example a user might want to corrupt byte offsets 101 to 105 (--for-byte-offset 101 105) with a number pattern: 1, 3, 5, 7, 9, ... (--set-byte 1 2). If the --set-increment option is left undefined, the resulting corruption per packet will be as follows:

Byte Offset	101	102	103	104	105
Packet 1	1	3	5	7	9
Packet 2	3	5	7	9	11
Packet 3	5	7	9	11	13
Packet 4	7	9	11	13	15
Packet 5	9	11	13	15	17
Packet 6	11	13	15	17	19

Note that the pattern has been increment over both column (Byte Offset) and row (Packet). Hence, if the --set-increment option is left undefined, the default option is to apply the pattern over both columns and rows. If however, the --set-increment is set to column, the pattern will only be incremented over the different byte offsets and will result into the following corruption:

Byte Offset	101	102	103	104	105
Packet 1	1	3	5	7	9
Packet 2	1	3	5	7	9
Packet 3	1	3	5	7	9
Packet 4	1	3	5	7	9
Packet 5	1	3	5	7	9
Packet 6	1	3	5	7	9

Similarly, if the --set-increment option is set to row, the pattern will only be incremented over the different packets and will result into the following corruption:

Byte Offset	101	102	103	104	105
Packet 1	1	1	1	1	1
Packet 2	3	3	3	3	3
Packet 3	5	5	5	5	5
Packet 4	7	7	7	7	7
Packet 5	9	9	9	9	9
Packet 6	11	11	11	11	11

Note that --set-increment will have a similar effect on the --set-bit option.

--set-max-corruptions <N> : Allows a limit to be set on the maximum number of corruptions that a user might like to introduce.

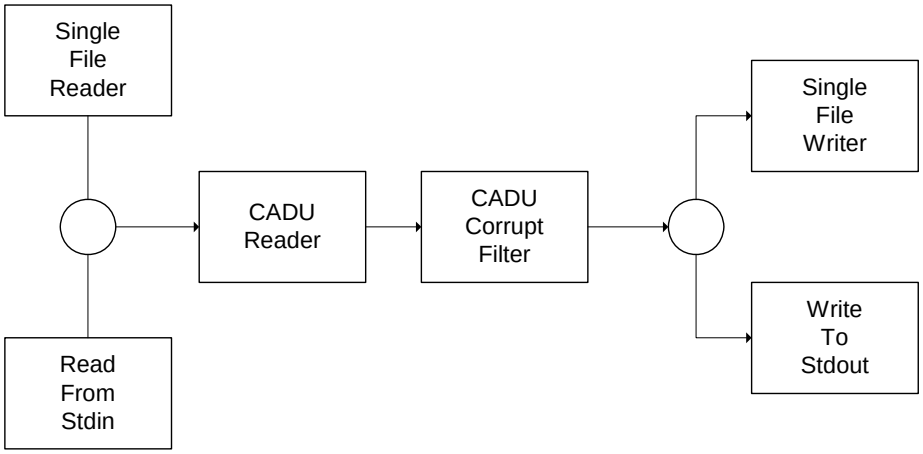
--show-cadu-table : Displays a table showing the structure of CADU packets (Note that the screen must be wide enough for the table to be displayed properly).

--simple : Use simple animations while processing.

--help : Shows usage information.

--version : Shows version information.

4.5.5 Functional Diagram



4.5.6 Description

The cadu_corrupt_filter allows a single or a range of bytes and bits to be modified per CADU packet. For example,

```
> cadu_corrupt_filter –i input.cadu –o output.cadu
  --for-byte-offset 0 3 --set-byte 255
```

The simple command above will set the value of bytes 0 to 3 to 255 (the maximum value a byte can be set to). Note that all CADUs start with a 4-byte sync marker that is always set to 0x1ACFFC1D. If the above command is carried out, this sync marker will be corrupted and thus other MetOpizer code will not be able to find any CADU packets in the output file produced.

```
> cadu_corrupt_filter –i input.cadu –o output.cadu
--for-byte-offset 4 15 --set-bit 1
```

The above command leaves the sync marker uncorrupted by modifying bytes 4 to 15. Note that in this case the --set-bit command is used instead of the --set-byte command. The --set-byte command must be used together with the --for-byte-offset option. However, the --set-bit command can be used with either the --for-byte-offset or the --for-bit-offset options.

```
> cadu_corrupt_filter –i input.cadu –o output.cadu
--for-byte-offset 4 15 --for-bit-offset +2 –1 --set-bit flip
```

The --for-byte-offset and the --for-bit-offset options can be combined. If the +/- symbols are used, the bit range is defined relative to the byte range. In this example the bit range will start at 34 (4*8 + 2) and end at 119 (15*8 - 1). Of course if an absolute bit range is used, it will override any byte range that may also be defined.

```
> cadu_corrupt_filter –i input.cadu –o output.cadu
--for-byte-offset 4 15 --for-bit-offset 5 23 --set-bit 0
--set-bit-width 7
```

The last example illustrates the use of the --set-bit-width option. This option enables users to set a bit to 0/1 or flip it at random intervals. In this particular case bit 5 will be set to 0 and also every 7th bit up until bit 23 is reached will also be set to 0. If the width value is set to 0, every bit up to and including bit 23 will be set to 0. The width value can also be set to random or random and a max-value. If set to random, the code will generate a random width by itself. If a max-value is specified, then the random values will not be able to exceed it.

4.6 cadu_displayer

4.6.1 Purpose

To read CADU packets and display in brief text format.

4.6.2 Inputs

File containing a list of concatenated CADU packets, or CADU packets on standard input.

4.6.3 Outputs

Either text display to console or a file of CADU packets.

4.6.4 Usage

cadu_displayer <input filename>] [--compact] [--help] [--version]

where:

<input filename> : Optional name of input file. If omitted stdin is used. Multiple files can be given by separating them with spaces.

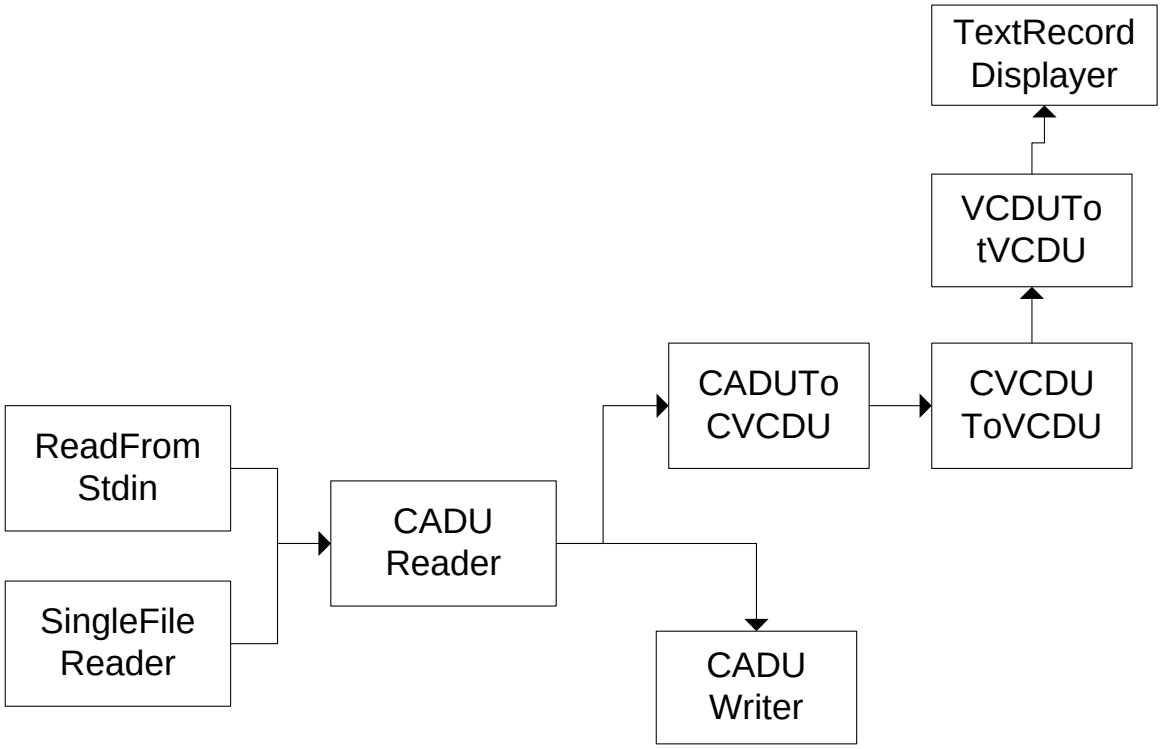
--compact : Use a compact layout with one line per t-VCDU packet.

--write-cadus : Writes the input CADU packets back to disk as a new file.

--help : show usage information

--version : show program version

4.6.5 Functional diagram



4.6.6 Description

The output text shows information taken from the header section of each CADU packet (and its enclosed VCDU packet). In addition fields if the CADU contains a complete CCSDS Packet Primary Header it is also shown.

This tool can also be used to clean up CADU files using the –write-cadus option, since only the CADU packets themselves are written to the output file. Any gaps or padding present in the input file will be stripped out.

4.7 cadu_orbit_splitter

4.7.1 Purpose

To read in CADU packets split them into one file per dump, with simulated overlaps.

4.7.2 Inputs

File containing a list of concatenated CADU packets.

4.7.3 Outputs

One file of CADU packets per dump.

4.7.4 Usage

cadu_orbit_splitter <input filename>] [--compact] [--help] [--version]

where:

<input filename> : Name of input file. If omitted standard input is used.

-o <filename> : Filename prefix to use for output files.

--overlap <n> : Duration of dump overlap. This should be a number followed by ‘m’ or ‘s’ for minutes or seconds, ie. ‘—overlap 8m’.

--duration <n> : Nominal dump duration.

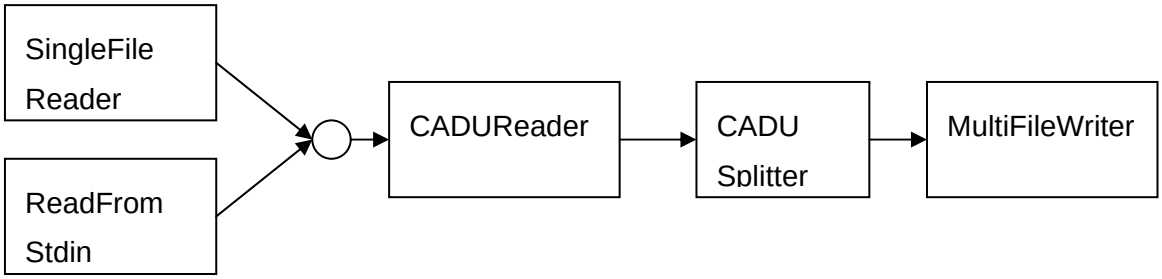
--shift-gras-obt : Assume times in GRAS ISPs are stored in shifted OBT notation (see ccsds_displayer).

--apid1-only : Only use APID 1 packets to determine dump boundaries.

--help : show usage information

--version : show program version

4.7.5 Functional diagram



4.7.6 Description

The default dump duration is 102 minutes with an 8 minute overlap.

To find dump boundaries the tool looks for CADU packets containing a CCSDS Packet Primary Header to read the OBT time.

4.8 cadu_to_ccsds

4.8.1 Purpose

To convert one or more files in the PDA-PDU format (a concatenated list of CADU packets) into CCSDS packets.

4.8.2 Inputs

List of input files containing CADU packets.

4.8.3 Outputs

One or more files containing lists of CCSDS packets

4.8.4 Usage

cadu_to_ccsds [--in <input filename(s)>] [--out <filename prefix>] [--filter <APID>] [--test-filter] [--show-apids] [--vcfilter <vcid>] [--show-mpdus] [--help] [--version]

where:

--in : Gives an optional list of input files. If omitted stdin is used. Multiple input filenames should be delimited by spaces.

--out : Gives an optional output filename prefix. For each output file a numerical application ID is appended to the filename prefix (the program always generates one file per APID) and a file extension of ".ccsds". If omitted all packets go to stdout in the same order as found in the input stream.

--filter x : Specify a list of filters based on CCSDS Application ID (space delimited). Only packets matching the filter are allowed through. Filters can be given by name or number, and if a name is given it may represent multiple APIDs.

--test-filter : Show the results of the specified --filter as a list of numeric APIDs. No other processing is performed.

--show-apids : Display list of allowable APID names against numbers. No other processing is performed.

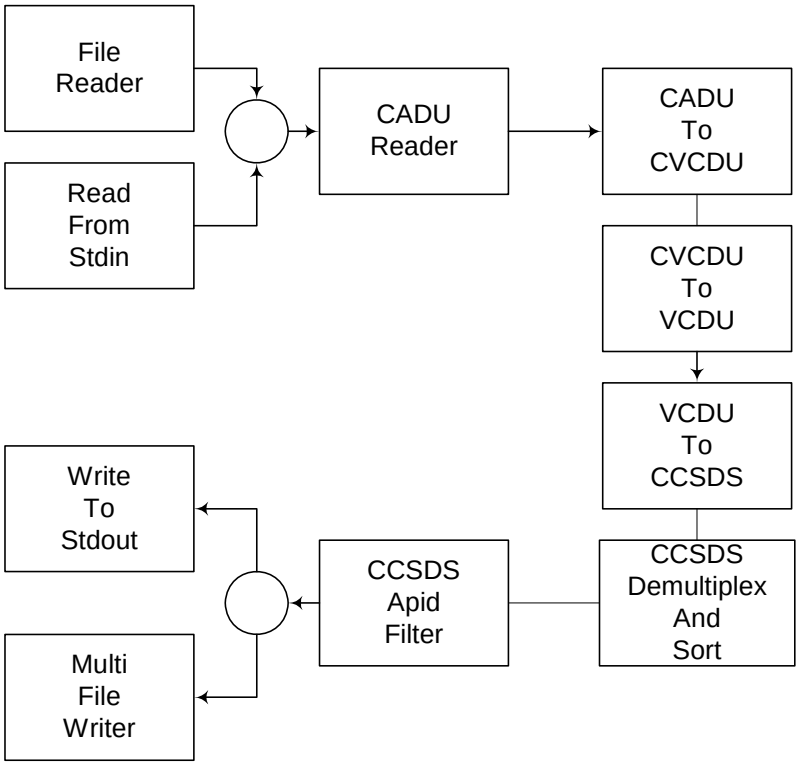
--vcfilter <vcid> : where vcid is a single number. This causes all virtual channel packets not matching vcid to be removed early on in processing. This optional parameter is a performance optimisation (its not required for full functionality as the CCSDS APID filter option gives more control) but can eliminate several time consuming operations. Not currently implemented.

--show-mpdu : Display a summary (First Header Pointer and a calculated checksum) of the payload M-PDU packet of each VCDU packet. No other processing is performed.

--help : Show usage information.

--version : Show program version number.

4.8.5 Functional diagram



4.8.6 Description

The input stream is examined for CADU packets by searching for the embedded synchronisation markers.

M-PDU packets are extracted from the CADU stream, demultiplexed and sorted into Virtual Channels.

Each Virtual Channel is then examined for embedded CCSDS packets which are picked out. They must then be demultiplexed since more than one type of CCSDS packet can share a single Virtual Channel, then packets of each APID are sorted by sequence number.

The final CCSDS packet streams are either written to disk or stdout.

4.9 cadu_to_tvcdu

4.9.1 Purpose

To convert a stream of CADU packets into t-VCDU packets.

4.9.2 Inputs

One or more files containing CADU packets.

4.9.3 Outputs

A single file of concatenated t-VCDU packets.

4.9.4 Usage

cadu_to_tvcdu [--in <input filename(s)>] [--out <filename>] [--help] [--version]

where:

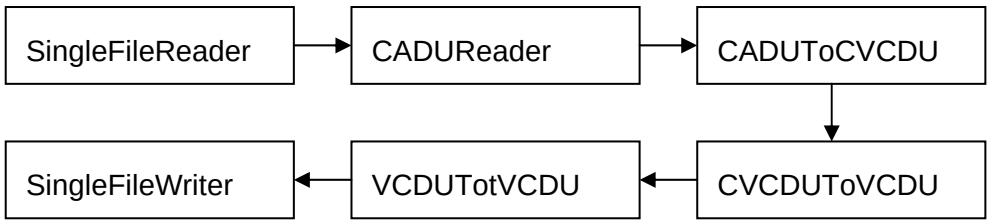
--in : Gives an optional list of input files. If omitted stdin is used. Multiple input filenames should be delimited by spaces.

--out : Gives an optional output filename. If omitted all packets go to stdout in the same order as found in the input stream.

--help : Show usage information.

--version : Show program version number.

4.9.5 Functional diagram



4.9.6 Description

The format of the tVCDU packet is given in [RD.12] Section 5.12.

The configuration file tvcdu.config is used by this program for filling in some t-VCDU header fields.

4.10 ccsds_apid_filter

4.10.1 Purpose

To iterate over a list of CCSDS packets and remove those whose Application IDs do not match the users filter.

4.10.2 Inputs

One or more files containing a number of concatenated CCSDS packets.

4.10.3 Outputs

A single file of concatenated CCSDS packets.

4.10.4 Usage

ccsds_apid_filter [--in <filename(s)>] [--out <filename>] [--filter <APID(s)>] [--test-filter] [--show-apids] [--prepacket-header x] [--help] [--version]

where

--in : specifies a list of filenames, delimited by spaces, to be read in. If omitted stdin is used.

--out : give filename to write to. If omitted stdout is used.

--filter <APID(s)> : Specify a list of Application Identifiers to be allowed through. Filters can be given by name of number. If a name is given it may match to more than one APID number.

--test-filter : Show the results of the specified filter as a list of numeric APIDs. No processing is performed.

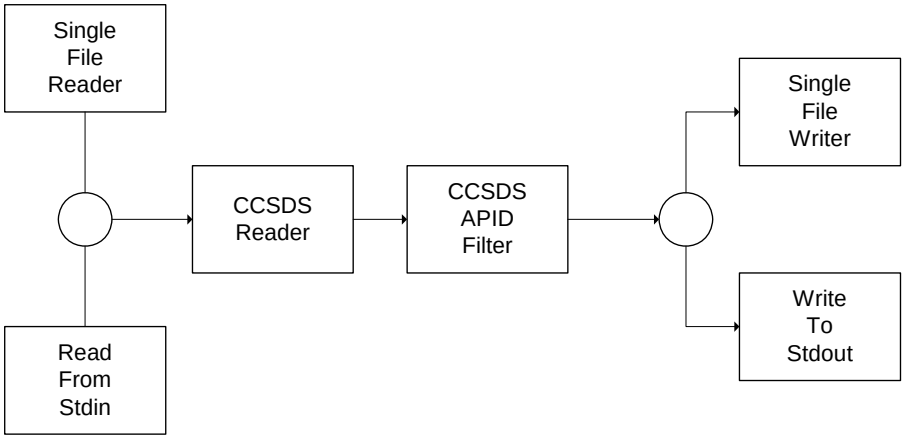
--show-apids : Display list of allowable APID names.

--prepacket-header x : see below

--help : Show usage information.

--version : Show program version number.

4.10.5 Functional Diagram



4.10.6 Description

The --prepacket-header x flag is used to specify that each packet in the input stream has a fixed length header of size ‘x’ bytes preceding it. This header is removed by the CCSDSReader module.

4.11 ccsds_corrupt_filter

4.11.1 Purpose

To enable the corruption of CCSDS packet common fields. Apart from the variable ancillary and application data sections, the remaining sections of CCSDS packets are common to all packet types (see the structure of CCSDS packets below). Additional functionality to corrupt any part of the packet is provided using the --set-byte option (this option must be used wisely to avoid introducing unwanted corruptions).

PACKET PRIMARY HEADER (6 octets)							PACKET DATA FIELD (max: 65536 octets)			
								SOURCE DATA		
Packet Identification				Packet Sequence Control		Packet length	Sec. hdr	User Data		Error Contrl
Version number	Type indic.	Second hdr flag	AP ID	Sequ. flags	Packet Sequ. count		UTC Time stamp	Ancill. Data	Applic. data	PEC
3 bits "000"	1 bit "0"	1 bit "x"	11 bits "xxx"	2 bits "11"	14 bits "xx"	16 bits "xx"	64 bits UTC "xx"	even nb of octets "xx"	even nb of octets "xx"	16 bits "xx"
16 bits				16 bits		16 bits	64 bits	SBT 48 bits / Variable		Variable	16 bits

4.11.2 Inputs

One or more streams of CCSDS packets.

4.11.3 Outputs

CCSDS packets with applied corruptions.

4.11.4 Usage

ccsds_corrupt_filter [-i|--in <filename(s)>] [-o|--out <filename>] [-s|--skip <N>] [-c|--count <N>] [For Options] [Set Options] [Byte Options] [-l|--collapse-length] [-a|--show-apids] [-t|--show-ccsds-table] [-h|--help] [-v|--version]

where:

--in <filename(s)> : Allows multiple input files to be specified. Note that multiple input file names should be delimited by spaces. If the "--in" option is omitted, stdin is used by default.

--out <filename> : Allows the name of an output file to be specified. If omitted stdout is used.

--skip <N> : Allows N packets to be skipped before applying any corruptions.

--count <N> : Allows corruptions to be applied to only N packets starting at the first packet or after skipping a number of packets if the --skip option is used.

For Options : The "--for-" options discussed below allow the corruption constraints to be defined and thus allow single or a group of packets to be isolated and corrupted while leaving any packets that do not meet the constraints untouched.

Set Options : The "--set-" options discussed below allow the exact nature of the corruptions to be defined.

Byte Options: The byte options discussed below are a special case that allow corruptions to be applied to any part of a packet.

--collapse-length : Only applicable if used together with the --set-packet-length option and otherwise ignored. If the --set-packet-length option is used with out this option, the value of the packet length is simply corrupted without effecting the original physical size of the input packet. If however, this option is used, the physical size of the input packet will be either increased or collapsed so that it matches any value the packet length may be set to (this option must be used wisely as it effects the structure of the input packet and will probably make the packet impossible to read or decode using existing tools such EPSView or the MetOpizer ccsds_displayer).

--show-apids : Shows a list of recognised APID types and the APID numbers that they represent. The recognised types or their numeric values can be used with the --for-apid option to only apply corruptions to a select number of packets that match the specified APID constraints.

--show-ccsds-table : Displays a table similar to the table shown in section 4.11.1 above. (Note that the screen must be wide enough for the table to be displayed properly.)

--help : Shows usage information.

--version : Shows version information.

To avoid unnecessary typing, all the options discussed above have short hand names by which they can be refereed to (-i, -o, -s, -c, -l, -a, -t, -h and -v).

For Options

--for-apid <apid(s)> : Allows single or multiple APIDs to be specified either by name or number. Only packets with APIDs matching the ones specified will be corrupted. To obtain a full list of APID names and numbers use the --show-apids option.

--for-sequence-counter <start_ctr> <end_ctr> : Only packets with sequence counters greater or equal to start_ctr and less than or equal to end_ctr will be corrupted.

--for-utc-time <start_time> <end_time> : Only packets with UTC-times greater or equal to start_time and less than or equal to end_time will be corrupted. Start and end times can be specified in two ways: a relative time and an absolute time. The absolute time must be in the format YYYYMMDDhhmmssZ, where the letters respectively represent Years, Months, Days, hours, minutes and seconds (e.g. 20041223151139Z). The relative time can be defined as an offset of +N(ms|s|m), where N represents a number and must be

followed by units of millisecond(ms), seconds(s) or minutes(m) (e.g. +2m). The relative start time is always relative to the UTC-time of the first packet. The relative end time is a little more complicated. If an absolute start time is followed by a relative end time, then the relative end time will be relative to the absolute start time. If a relative start time is followed by a relative end time, then the relative end time will be relative to the UTC time of the first packet offset by the relative start time. It might sound a little complicated in words, but mathematically its quite simple:

if start_time = A then start_time = A

if end_time = +5m then end_time = A + 5m

if start_time = +2m then start_time = T1 + 2m

if end_time = +5m then end_time = T1 + 2m + 5m

Where A represents a specified absolute time, T1 represents the UTC-time of the first CCSDS packet read and +2m and +5m represent 2 and 5 minute relative time offsets.

Set Options

--set-version-number <random|N1|N1 N2>

--set-type <random|N1|N1 N2>

--set-secondary-header <random|N1|N1 N2>

--set-apid <random|N1|N1 N2>

--set-sequence-flag <random|N1|N1 N2>

--set-sequence-counter <random|N1|N1 N2>

--set-packet-length <random|N1|N1 N2>

--set-utc-day <random|N1|N1 N2>

--set-utc-millisecond <random|N1|N1 N2>

--set-utc-microsecond <random|N1|N1 N2>

--set-obt-coarse <random|N1|N1 N2>

--set-obt-fine <random|N1|N1 N2>

--set-pec <off|random|N1|N1 N2>

The functions of the above set options are obvious and do not need to be individually explained. Each set option allows one of the CCSDS packet common fields to be corrupted (see the structure of CCSDS packets in section 4.11.1 above). Note that three types of corruptions can be introduced:

- The first type of corruption allows the fields of consecutive CCSDS packets to be set to an appropriate* random number.
- The second type of corruption allows the fields of consecutive CCSDS packets to be set to an appropriate* single number (N1).

- The third type of corruption allows the fields of consecutive CCSDS packets to be filled with an appropriate* number series or patter starting with the number N1 and incremented by the number N2. For example if N1=0 and N2=2, then consecutive packets will be filled with the numbers 0, 2, 4, 6, … If the field has a maximum value that is reached, the pattern or series will simply wrap around and continue. For example if the maximum acceptable value is 7, then the pattern will be: 0, 2, 4, 6, 0, 2, 4, 6, 0, … Note that N2 can also be assigned a negative value and thus produce increasing and decreasing number patterns.

The term appropriate* has been used in the above paragraph to imply that the corruptions applied must be greater or equal to the minimum value the field can accept and of course less than or equal to the maximum value the field can accept. The code attempts to handle all this automatically. If however, the user does input values that are out of range, they will be prompted with error messages and advised to revise their inputs.

The only exception to the above set options is the final --set-pec option. This option has an additional off mode. The Packet Error Control (PEC) field is automatically calculated and updated whenever a corruption is introduced. Of course it can itself be corrupted in the usual three ways above. However, should a need arise to apply some corruptions without updating the PEC, the off mode can be used to simply do nothing and keep the PEC of the initial input CCSDS packet.

Byte Options

Unlike the set options described above, the byte options allow corruptions to be applied to any part of a CCSDS packet.

--for-byte-offset <N1|N1 N2> : Corruptions can be introduced to a single byte at byte offset N1 or to a range of bytes starting at byte offset N1 and ending at byte offset N2. Note that if the corruption is to be applied to the first byte, the byte offset must be set to zero. Hence, the byte offset must be greater or equal to zero and of course less than the packet length.

--set-byte <random|N1|N1 N2> : Similar to the set options discussed above, the --set-byte option allows a single byte per packet or a range of bytes per packet to be set to random values, to be set to a single value N1, or to be filled with a number pattern starting with a value N1 and incremented by a value N2. Note that N1 and N2 can not be set to anything greater than 255. While N1 can not be less than zero, the increment N2 can be assigned a negative value.

--set-byte-increment <column|row> : This option only becomes relevant if one is attempting to assign a number pattern to a range of bytes per packet. For example a user might want to corrupt byte offsets 101 to 105 (--for-byte-offset 101 105) with a number pattern: 1, 3, 5, 7, 9, … (--set-byte 1 2). If the --set-byte-increment option is left undefined, the resulting corruption per packet will be as follows:

Byte Offset	101	102	103	104	105
Packet 1	1	3	5	7	9
Packet 2	3	5	7	9	11
Packet 3	5	7	9	11	13
Packet 4	7	9	11	13	15
Packet 5	9	11	13	15	17
Packet 6	11	13	15	17	19

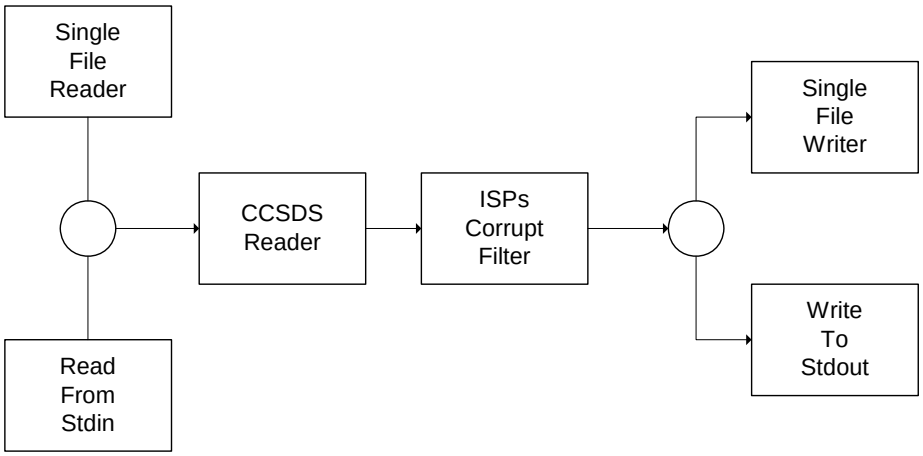
Note that the pattern has been increment over both column (Byte Offset) and row (Packet). Hence, if the --set-byte-increment option is left undefined, the default option is to apply the pattern over both columns and rows. If however, the --set-byte-increment is set to column, the pattern will only be incremented over the different byte offsets and will result into the following corruption:

Byte Offset	101	102	103	104	105
Packet 1	1	3	5	7	9
Packet 2	1	3	5	7	9
Packet 3	1	3	5	7	9
Packet 4	1	3	5	7	9
Packet 5	1	3	5	7	9
Packet 6	1	3	5	7	9

Similarly, if the --set-byte-increment option is set to row, the pattern will only be incremented over the different packets and will result into the following corruption:

Byte Offset	101	102	103	104	105
Packet 1	1	1	1	1	1
Packet 2	3	3	3	3	3
Packet 3	5	5	5	5	5
Packet 4	7	7	7	7	7
Packet 5	9	9	9	9	9
Packet 6	11	11	11	11	11

4.11.5 Functional Diagram



4.11.6 Description

The idea of separating all the options into two groups (*for* and *set*) aims to help the user understand the options logically. For example if we wanted to set the version number to 2 for a range of sequence counters starting at 3 and ending at 15, we can express the command in words as: For sequence counters starting at 3 and ending at 15, set version number to 2. Applying this command to the ccsds_corrupt_filter works the same way:

```
> ccsds_corrupt_filter –i amsua.ccsds –o amsua.ccsds.corrupt
  --for-sequence-counter 3 15 --set-version-number 2
```

We can take things a step further and specify that we want the version number to be set to a random number for sequence counters 3 to 15, but only if the APID matches that of AMSU-A1. We know that AMSU-A1 has an APID value of 39, so we can specify the APID in two ways:

```
> ccsds_corrupt_filter –i amsua.ccsds –o amsua.ccsds.corrupt
  --for-apid 39 --for-sequence-counter 3 15
  --set-version-number random
> ccsds_corrupt_filter –i amsua.ccsds –o amsua.ccsds.corrupt
  --for-apid amsu-a1 --for-sequence-counter 3 15
  --set-version-number random
```

We have a third option where we can start our corruption with an initial start value and increment further corruptions by a fixed amount. For example we can start our corruption with a value of 1 and increment subsequent corruptions by a value of 2 (i.e. introduce a corruption pattern of 1, 3, 5, …). Further more, AMSU-A has two APIDs associated with it:

39 and 40 corresponding to AMSU-A1 and AMSU-A2. If we wanted to apply the corruption to both APIDs, we can do it in a number of ways:

```
> ccsds_corrupt_filter –i amsua.ccsds –o amsua.ccsds.corrupt
  --for-apid 39 40 --for-sequence-counter 3 15
  --set-version-number 1 2
> ccsds_corrupt_filter –i amsua.ccsds –o amsua.ccsds.corrupt
  --for-apid amsu-a1 amsu-a2 --for-sequence-counter 3 15
  --set-version-number 1 2
> ccsds_corrupt_filter –i amsua.ccsds –o amsua.ccsds.corrupt
  --for-apid amsu-a1 40 --for-sequence-counter 3 15
  --set-version-number 1 2
> ccsds_corrupt_filter –i amsua.ccsds –o amsua.ccsds.corrupt
  --for-apid 39 amsu-a2 --for-sequence-counter 3 15
  --set-version-number 1 2
> ccsds_corrupt_filter –i amsua.ccsds –o amsua.ccsds.corrupt
  --for-apid amsu-a --for-sequence-counter 3 15
  --set-version-number 1 2 --set-sequence-flag random
```

Note that the last example uses the APID tag AMSU-A. This tag has multiple APIDs associated with it, which in the case of AMSU-A includes both APID 39 and 40.

The final example above shows multiple fields (version number and sequence flag) being corrupted at the same time. Hence, one is not limited to corrupting one field at a time.

As illustrated the *for* and *set* options work well together and allow multiple corruptions to be introduced to any subset of packets.

4.12 ccsds_count_filter

4.12.1 Purpose

To remove a selected range of CCSDS packets from a stream.

4.12.2 Inputs

One or more files containing a number of concatenated CCSDS packets.

4.12.3 Outputs

A single file of concatenated CCSDS packets.

4.12.4 Usage

ccsds_count_filter [--in <filename(s)>] [--out <filename>] [--skip x] [--count x] [--prepacket-header x] [--help] [--version]

where

--in : specifies a list of filenames, delimited by spaces, to be read in. If omitted stdin is used.

--out : give filename to write to. If omitted stdout is used.

--skip x : The initial x packets of the input stream will be stripped out. Defaults to ‘0’ if not specified.

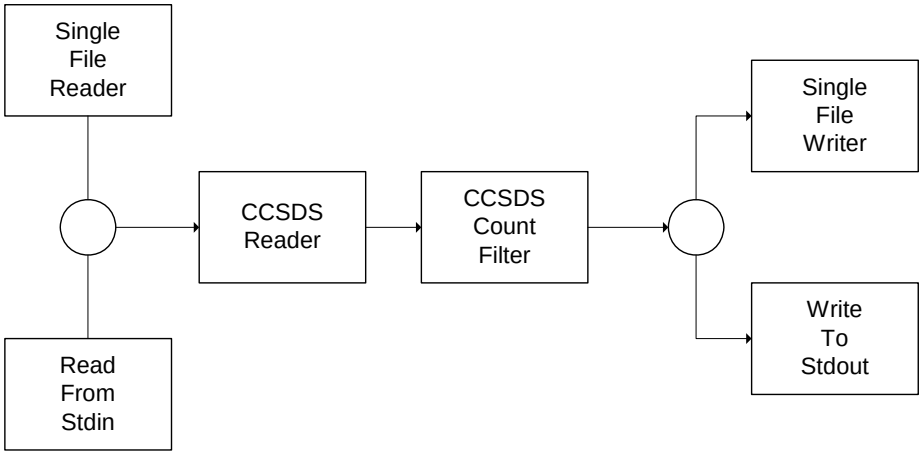
--count : The number of input packets to allow through (after --skip have been skipped over). If this parameter is omitted all packets after --skip are passed through.

--prepacket-header x : see below

--help : Show usage information.

--version : Show program version number.

4.12.5 Functional Diagram



4.12.6 Description

Note the input packets are not sorted or demultiplexed, and all operations are performed on the packets in the order they are read in.

The –prepacket-header x flag is used to specify that each packet in the input stream has a fixed length header before it. This header is removed by the CCSDSReader module.

4.13 ccsds_displayer

4.13.1 Purpose

To open a file containing one or more CCSDS packets and display information about the entire file or individual packets.

4.13.2 Inputs

File(s) containing CCSDS packets, or EPS L0 products, or Reduced MHS packets (1300 bytes with UTC time omitted).

4.13.3 Output

Text description to the terminal.

4.13.4 Usage

ccsds_displayer [--in <filename(s)>] [--stream] [--headers] [--packets] [--prepacket-header <n>] [--decode] [--reference] [--show-stats] [--show-gaps] [--show-seq-diffs] [--shift-gras-obt] [--show-viadr] [--count <n>] [--skip <n>] [--select-utc-gaps <arg>] [--select-grh-gaps <arg>] [--select-obt-gaps <arg>] [--config <filename>] [--check-config] [--iasi-config <filename>] [--check-iasi-config] [--image-avhrr <filename>] [--image-iasi-iis <filename>] [--jpeg] [--hflip] [--vflip] [--invert] [--quiet] [-z] [--version] [--help] [--long-help]

where:

Input settings

--in : Specify input filename. If omitted stdin is used. The “--in” keyword is optional.

--prepacket-header <n> : see below

--decode : Attempt to interpret the contents of the packet application data section.

--reference : Used in conjunction with the –decode option, restricts the output displayed to those fields common to both source packets and NOAA Level 1b products.

Text settings

--stream : see below.

--headers : see below.

--packets : see below.

--show-stats : Display channel count statistics instead of normal output.

--show-gaps : Display time difference between each packet and the next (implies --headers if neither --headers nor --packets is specified.

--show-seq-diffs : Display sequence counter deltas from each packet to the next.

--shift-gras-obt : Interpret GRAS OBT header times using the correct notation, shifted 8 bits upwards relative to other instruments (APID 480-511).

--show-viadr : Display content of the viadr-l0-obt2utc records as they are found.

--shift-gras-obt : Interpret OBT times in GRAS packet headers using the correct shifted notation.

Packet selection

--count <n> : Display only <n> packets.

--skip <n> : Skip <n> packets before processing.

--select-utc-gaps <arg> : Display packet number(s) which have UTC time differences of <n> ms.

--select-grh-gaps <arg> : Display packet number(s) which have GRH time differences of <n> ms (L0 products only).

--select-obt-gaps <arg> : Display packet number(s) which have OBT time differences of <n> s^-16.

Config settings

--config <filename> : Specify name of the options file as detailed in Section 9.6. If the parameter is omitted the file “ccsdsdisplayer.config” is looked for first in the directory $METOPIZER_CONFIG_DIR, then “$HOME/.metopizer”. If no configuration file is found a set of default values are used.

--check-config : Show the current configuration options. No other processing is performed.

--iasi-config <filename> : Specify name of the IASI specific configuration file as detailed in Section 9.7. If the parameter is omitted the file “iasi_spectrum.config” is looked for first in the directory $METOPIZER_CONFIG_DIR, then “$HOME/.metopizer”. If no configuration file is found, the IASI spectrum application data field can not be decoded and a warning message is displayed.

--check-iasi-config : Show the current IASI specific configuration options. No other processing is performed.

--quiet : Omit additional messages.

-z : Use simple busy animations.

Image settings

--image-avhrr <filename> : For AVHRR HR packets only, generate a series of images, per default in TIFF format.

--image-iasi-iis <filename> : For IASI HR packets only, generate a series of images, per default in TIFF format.

--jpeg : Output JPEG format images instead of TIFF.

--hflip : Flip images horizontally (around the vertical axis) (IASI only).

--vflip : Flip images vertically (around the horizontal axis) (IASI only).

--invert : Invert image colours (IASI only).

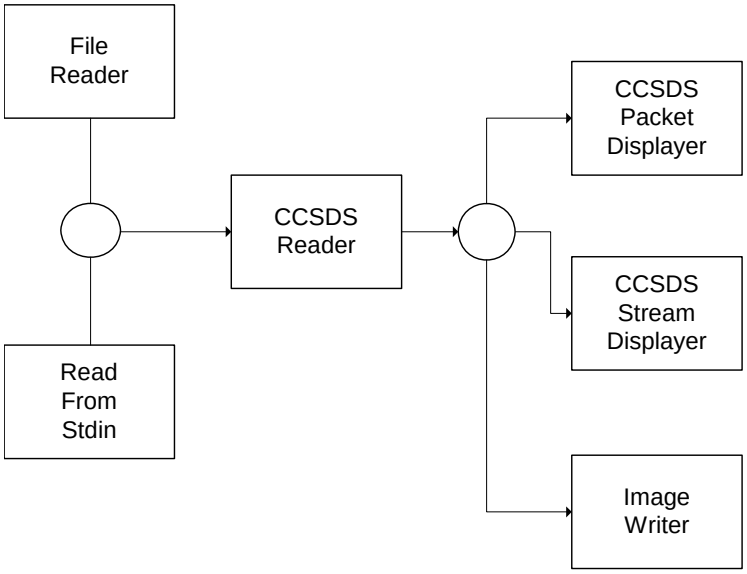
Program information

--version : Show program version number.

--help : Show usage information.

--long-help : Show extended help and examples.

4.13.5 Functional diagram



4.13.6 Description

The full packet contents, or optionally only the packet header information, of the input file (a L0 product or a set of individual CCSDS packets) is displayed. This includes the OBT time converted UTC packet time based on the VIADR used, while a warning is raised in case of an offset with the GRH time.

When multiple packets are found in the input stream, the program shows an analysis of the whole stream including total number of packets, which Application IDs were found and ranges of sequence count values.

If the input is a single packet then by default the program displays the header section followed by a hex dump of the application section. The only exception is for AMSU packets for which a raw hex dump is displayed followed by a hex dump with the filler words removed.

The –decode flag instructs the program to interpret the application data section and show it as a list of labelled tables, so the packet's contents can be read easily. This is only available for ATOVS, IASI, ASCAT, GRAS and GOME instrument packets.

The –reference option restricts the output from a –decode display to only show fields that appear in both Level 1b products and in source packets. In this case the format used is identical to that of the noaa_l1b_displayer program with it's –reference option, allowing the two outputs to be compared. This option is only applicable to the ATOVS instruments.

The flags –stream, --header and –packet can be used to force the output mode used.

The --prepacket-header x flag is used to specify that each packet in the input stream has a fixed length header before it. This header is removed by the CCSDSReader module.

4.14 ccsds_iasi_flags_fix

4.14.1 Purpose

To correct IASI CCSDS packet flag settings.

4.14.2 Inputs

One or more streams of IASI CCSDS packets.

4.14.3 Outputs

IASI CCSDS packets with corrected flag settings.

4.14.4 Usage

ccsds_iasi_flags_fix [--in|-i <filename(s)>] [--out|-o <filename>] [--type-only] [--sequence-flag-only] [--help|-h] [--version|-v]

where:

--in <filename(s)> : Allows multiple input files to be specified. Note that multiple input file names should be delimited by spaces. If the “--in” option is omitted, stdin is used by default.

--out <filename> : Specify name of output file. If omitted stdout is used.

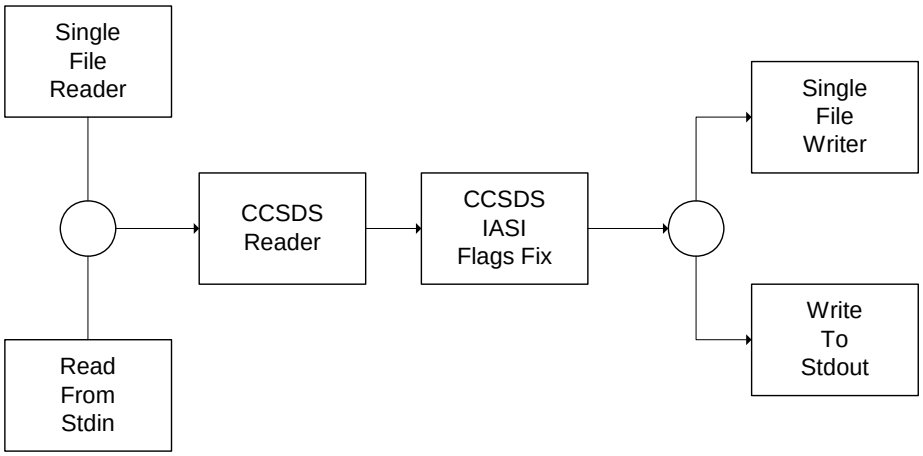
--type-only : This option can be used to just invert the type indicator flag in the CCSDS packet primary header and not invert any other flags. If this option is not used, then by default the type indicator flag and all the flags listed in Appendix D will be inverted.

--sequence-only : This option can be used to just set the sequence flag in the CCSDS packet primary header to 3 and not invert any other flags. If this option is not used, then by default all the flags listed in Appendix D will be inverted as well as the sequence flag being set to 3.

--help : Show usage information.

--version : Show program version.

4.14.5 Functional Diagram



4.14.6 Description

The simulated IASI CCSDS packets produced by Noveltis had various flags set incorrectly. This code simply reads input single or multiple IASI CCSDS packets and inverts the flags that were incorrectly set. A full listing of the flags that the code inverts is given in Appendix D.

4.15 ccsds_splitter

4.15.1 Purpose

To read in a number of files containing concatenated CCSDS packets and output them as individual files.

4.15.2 Inputs

One or more streams of CCSDS packets.

4.15.3 Outputs

CCSDS packets in separate files

4.15.4 Usage

ccsds_splitter [--in <filename(s)>] [--out <filename prefix>] [--prepacket-header x] [--version] [--help]

where:

--in <filename(s)> : One or more files to be read for packets. stdin is used if this option is not specified.

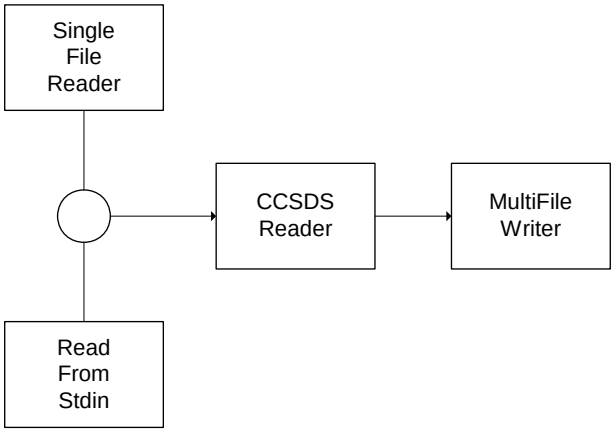
--out <filename prefix> : Base name to use when writing packets.

--prepacket-header x : see below

--help : Show usage information.

--version : Show program version number.

4.15.5 Functional Diagram



4.15.6 Description

Each packet is written in the form <filename prefix>.<apid>.<sequence count>.ccsds

where <apid> is the numerical CCSDS Application ID field and <sequence count> is that packets CCSDS packet sequence count.

The –prepacket-header x flag is used to specify that each packet in the input stream has a fixed length header before it. This header is removed by the CCSDSReader module.

4.16 ccsds_strip_filter

4.16.1 Purpose

To remove CCSDS packets that do not satisfy specified UTC start and end times.

4.16.2 Inputs

One or more streams of CCSDS packets originating from the same instrument.

4.16.3 Outputs

CCSDS packets arranged in an increasing time order that start and end at UTC times specified by the user.

4.16.4 Usage

ccsds_strip_filter [--in <filename(s)>] [--out <filename>] [--config <filename>] [--check-config] [--help] [--version]

where:

--in <filename(s)> : Allows multiple input files to be specified. Note that multiple input file names should be delimited by spaces. If the “--in” option is omitted, stdin is used by default.

--out <filename> : Specify name of output file. If omitted stdout is used.

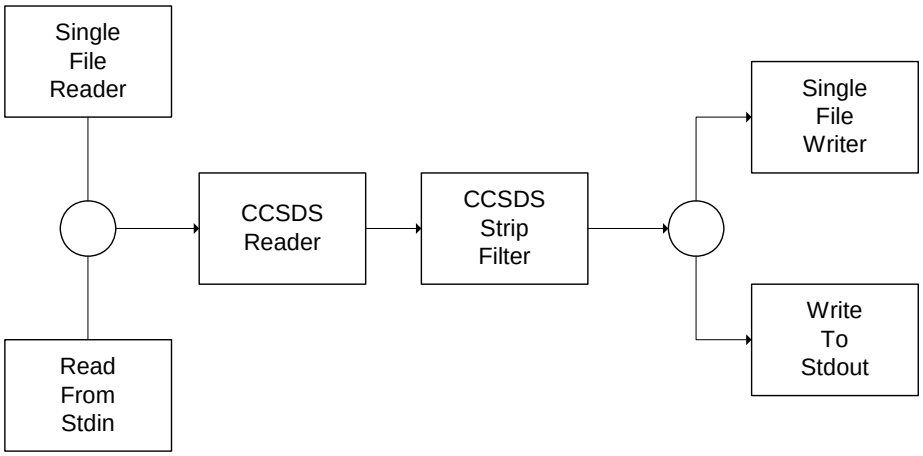
--config <filename> : Specify the configuration file used to strip the input CCSDS packets as detailed in Section 9.9. If the switch is absent the filename “$METOPIZER_CONFIG_DIR/ccsds_strip_filter.config” will be searched, otherwise “$HOME/.metopizer/ccsds_strip_filter.config”. If no configuration file can be found a set of default values will be used.

--check-config : The configuration files are read in and their contents displayed. No other processing is performed.

--help : Show usage information.

0--version : Show program version.

4.16.5 Functional Diagram



4.16.6 Description

Given single or multiple CCSDS packets, this code checks to ensure that the UTC time of the packets are greater or equal to the start UTC time and less than or equal to the end UTC time defined in a configurable file similar in structure to the one shown in Section 9.9. Only CCSDS packets within the time range are kept and used to produce the output file.

The concatenated CCSDS packets produced as output are expected to be arranged in chronological order where the UTC time is expected to either remain unchanged or increase from one CCSDS packet to the next. CCSDS packets that do not satisfy this condition and hence, have an UTC time that is less than that of the previous packet are removed.

A final check is also made to ensure that the UTC time gap between consecutive CCSDS packets is not too large. If the time difference is larger than the set configurable time_gap_limit field, the CCSDS packet is not included in the output and error/warning messages are displayed on screen.

Note that multiple input CCSDS packets fed to this code must all originate from the same instrument. For example they must all be IASI CCSDS packets, or they must all be ASCAT CCSDS packets. They cannot be a combination of IASI and ASCAT packets.

4.17 ccsds_seq_fix

4.17.1 Purpose

To modify/offset the sequence counter in the CCSDS primary header and the OBT time field of the CCSDS ancillary data packet.

4.17.2 Inputs

One or more streams of CCSDS packets originating from the same instrument.

4.17.3 Outputs

CCSDS packets with modified sequence counters and OBT times.

4.17.4 Usage

ccsds_seq_fix [--in filename(s)] [--out filename] [--config filename] [--check-config] [--accumulate] [--single] [--help] [--version]

where:

--in <filename(s)> : Allows multiple input files to be specified. Note that multiple input file names should be delimited by spaces. If the “--in” option is omitted, stdin is used by default.

--out <filename> : Specify name of output file. If omitted stdout is used.

--config <filename> : Specify the configuration file used to set the initial time and sequence counter values as detailed in Section 9.11. If the switch is absent the filename “$METOPIZER_CONFIG_DIR/ccsds_seq_fix.config” will be searched, otherwise “$HOME/.metopizer/ccsds_seq_fix.config”. If no configuration file can be found a set of default values will be used.

--check-config : The configuration files are read in and their contents displayed. No other processing is performed.

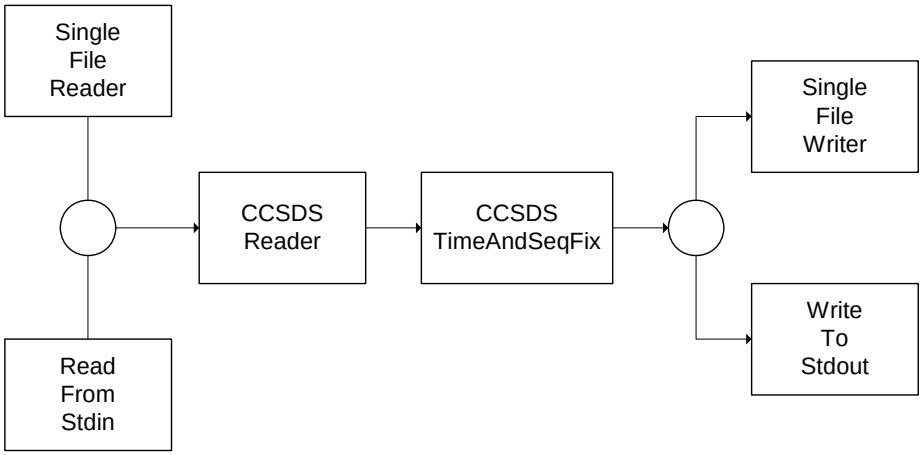
--accumulate : See description below.

--single : See description below.

--help : Show usage information.

--version : Show program version.

4.17.5 Functional Diagram



4.17.6 Description

This code, designed to handle both single and multiple input CCSDS packets, simply replaces the value of the sequence counter of the first input CCSDS packet with the value of the configurable parameter COUNTER_INITIAL. It will also replace the OBT time fields of the ancillary data part of the first input CCSDS packet with values obtained from the configurable parameters OBT_COARSE_INITIAL and OBT_FINE_INITIAL (see Section 9.11 for further details of the configurable parameters). Consecutive CCSDS packets are updated in accordance to the equations below:

First CCSDS Packet:

```
CCSDS[0].new_sequence_counter = COUNTER_INITIAL

CCSDS[0].new_obt_coarse = OBT_COARSE_INITIAL

CCSDS[0].new_obt_fine = OBT_FINE_INITIAL
```

Consecutive CCSDS packets:

```
CCSDS[n+1].new_sequence_counter =
      COUNTER_INITIAL + CCSDS[n+1].sequence_counter - CCSDS[n].sequence_counter

CCSDS[n+1].new_obt_coarse =
      OBT_COARSE_INITIAL + CCSDS[n+1].obt_coarse - CCSDS[n].obt_coarse

CCSDS[n+1].new_obt_fine =
      OBT_FINE_INITIAL + CCSDS[n+1].obt_fine - CCSDS[n].obt_fine
```

Where n is a counter starting at the first CCSDS packet (n=0). Note that OBT coarse and OBT fine are related. Hence, when OBT fine exceeds it maximum value of 65535, it is

reset and OBT coarse is updated to account for the excess. If OBT coarse exceeds its maximum value of 16777215, it is reset to start counting again from zero.

The configurable parameters COUNTER_INITIAL_1, COUNTER_INITIAL_2, …, COUNTER_INITIAL_n can be used to set the sequence counter of the second, third and n^{th}+1 CCSDS packets to specific values. Note that the definition of these configurable fields is optional. If left undefined, the sequence counter of consecutive CCSDS packets will continue to be filled in accordance to the above equation. However, should there be a need to assign specific values to more than one CCSDS packet sequence counter, the option is available.

According to the above equation, if the difference between the sequence counter of one CCSDS packet and the next is greater than one, it follows through and results in the values of the newly calculated sequence counters also being different by the same amount. By using the “--single” option, the code overrides the above equation and ensures that the sequence counter is always incremented by one.

Note that multiple input CCSDS packets fed to this code must all originate from the same instrument. For example they must all be IASI CCSDS packets, or they must all be ASCAT CCSDS packets. They cannot be a combination of IASI and ASCAT packets.

Some instruments such as ASCAT or IASI have more than one APID assigned to them (see Section 5.15.4). In some cases (for example ASCAT), the sequence counter is incremented irrespective of the different APIDS. In other cases (for example IASI), the sequence counter is incremented only if another CCSDS packet with the same APID is detected. In fact, with the exception of ASCAT, all other instruments with multiple APIDs assigned to them have their sequence counter incremented only if another CCSDS packet with the same APID is detected. The “--accumulate” option can be used to treat all instruments like the ASCAT case and to increment the sequence counter irrespective of the different APIDs.

If some of the CCSDS packet terminology used is unclear, further details can be found in Sections 5.23 and 5.24 of this document and in RD10.

4.18 ccsds_time _fix

4.18.1 Purpose

To set/modify CCSDS packet UTC and OBT times.

4.18.2 Inputs

One or more streams of CCSDS packets originating from the same instrument.

4.18.3 Outputs

CCSDS packets with modified time values.

4.18.4 Usage

ccsds_time_fix [--in filename(s)] [--out filename] [--config filename] [--check-config] [--accumulate] [--utc <UTC time>] [--obt <OBT time>] [--help] [--version]

where:

--in <filename(s)> : Allows multiple input files to be specified. Note that multiple input file names should be delimited by spaces. If the “--in” option is omitted, stdin is used by default.

--out <filename> : Specify name of output file. If omitted stdout is used.

--config <filename> : Specify the configuration file used to set the initial time values as detailed in Section 9.10. If the switch is absent the filename “$METOPIZER_CONFIG_DIR/ccsds_time_fix.config” will be searched, otherwise “$HOME/.metopizer/ccsds_time_fix.config”. If no configuration file can be found a set of default values will be used.

--check-config : The configuration files are read in and their contents displayed. No other processing is performed.

--accumulate : See description below.

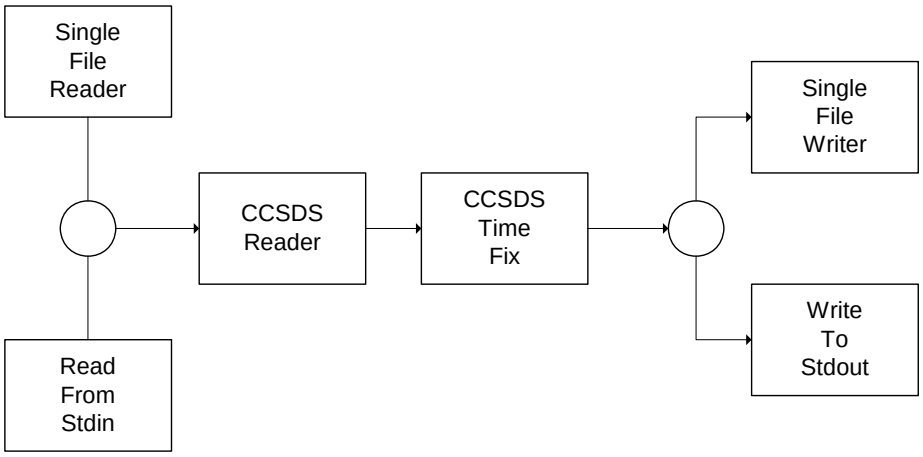
--utc <UTC time>: Set the UTC time of the first packet of the output stream. This should be in the form YYYYMMDDhhmmssZ or YYYYMMDDhhmmssuuuZ (YYYY=year, MM=month, DD=day, hh=hour, mm=minute, ss=second, uuu=millisecond, Z=”Z”).

--obt <OBT time>: Set the OBT time of the first packet of the output stream. The format must be “C,F” (C=coarse time, F=fine time) ie. “—obt 50000, 20”.

--help : Show usage information.

--version : Show program version.

4.18.5 Functional Diagram



4.18.6 Description

The tool can either be configured using a text configuration file or command line arguments.

This code, designed to handle both single and multiple input CCSDS packets, can be used to replaced the CCSDS UTC time tags with ones specified by the user. As described in Section 9.10, the UTC time of the first input CCDS packet (UTC_start) can be set using the configurable fields: utc_days_start, utc_milliseconds_start and utc_microseconds_start.

As shown by the equation below, the UTC time of consecutive input CCSDS packets are determined by adding a constant amount of time (K) to the user specified start time of the first input CCSDS packet (UTC_start).

```
CCSDS[0].UTC = UTC_start
CCSDS[n].UTC = UTC_start + nK
```

Where n represents the number of CCSDS packets and n=0 represents the first CCSDS packet. The value of the constant time increment, K, is user configurable and can be set by assigning values to the configurable fields: utc_days_increment, utc_milliseconds_increment and utc_microseconds_increment.

Some instruments such as ASCAT or IASI have more than one APID assigned to them (see Section 5.15.4). The code monitors the change in APIDs and only increments the UTC time if another CCSDS packet with the same APID is detected. The "--accumulate" option can be used to ignore the APID differences of the same instrument and thus, increment the UTC time irrespective of the different APIDs. Note that multiple input CCSDS packets fed to this code must all originate from the same instrument. For example

they must all be IASI CCSDS packets, or they must all be ASCAT CCSDS packets. They cannot be a combination of IASI and ASCAT packets.

If some of the CCSDS packet terminology used is unclear, further details can be found in Sections 5.23 and 5.24 of this document and in RD10.

4.19 ccsds_to_l0

4.19.1 Purpose

Convert a stream of ISPs to an EPS L0 product.

4.19.2 Inputs

One or more streams of CCSDS packets.

4.19.3 Outputs

One EPS L0 product.

4.19.4 Usage

ccsds_to_l0 [--in filename(s)] [--out filename] [--utc <UTC time>] [--obt <OBT time>] [--clock-drift <number>] [--rollovers <number>] [--viadr <VIADR filename>] [--help] [--version]

where:

--in <filename(s)> : Allows input files to be specified. Multiple input files should be delimited by spaces. If the --in option is omitted, stdin is used by default.

--out <filename> : Specify name of output file.

--viadr <filename> : Specify a filename containing a viadr-l0-obtutc record to take the parameters for the OBT->UTC correlation from. The file can be a complete L0 product or just a single record.

--utc-obt <UTC time> : Set the UTC-OBT (or UTC0) parameter for the OBT->UTC equation to either an absolute time or time relative to the start of the input data. Absolute times are given as either YYYYMMDDhhmmssZ or YYYYMMDDhhmmssuuuZ where YYYY=year, MM=month, DD=day, hh=hour, mm=minute, ss=second, uuu=milliseconds, and the final Z to show the format is EPS generalised time. Relative times are given as -<number><units> where <units> can be o=orbits, h=hours, m=minutes, s=seconds, or ms=milliseconds. The default value is -31o.

--obt-utc <OBT time> : Set the OBT-UTC (or OBT0) parameter for the OBT->UTC equation. The value is specified as C,F (C=coarse time, F=fine time). The value defaults to 0,0.

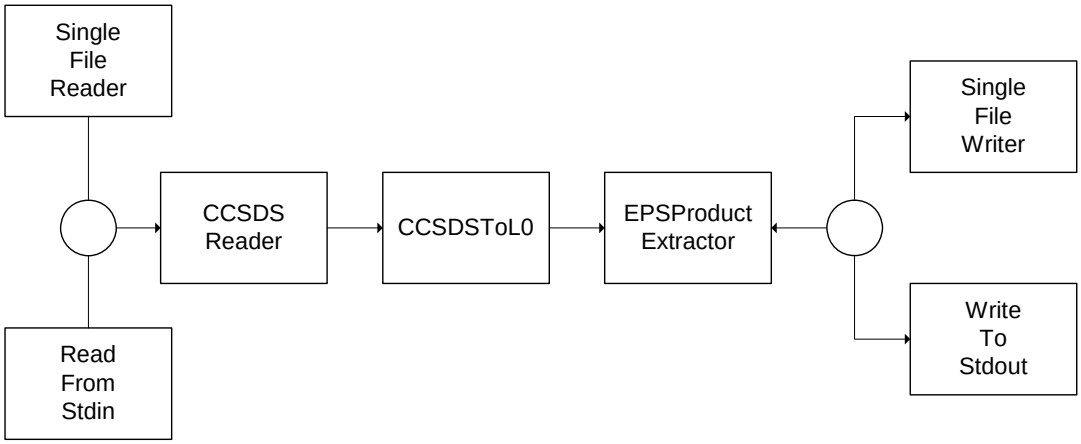
--clock-drift <number> : The value for clock drift (or A_STEP) for the OBT->UTC equation. The default value is 3906249305.

--rollovers <number> : See below.

--help : Show usage information.

--version : Show program version.

4.19.5 Functional Diagram



4.19.6 Description

A L0 product is built using all input CCSDS packets in the order they are stored in the input files. The input files are processed as given on the command line.

The source packets are never modified and the only processing performed is to create MPHR, viadr-l0-utc2obt and IPR records and to form the L0 MDR header information including GRH RECORD_START_TIME and RECORD_STOP_TIME fields for each packet. The RECORD_STOP_TIME field is set to the same value as RECORD_START_TIME.

The OBT->UTC equation used to calculate the MDR time values is:

GRH time = ((OBT - OBT0)*CLOCK_DRIFT) + UTC0.

If no OBT->UTC parameters are given then a realistic set of defaults is be generated, with UTC0 set 31 orbits before the UTC time of the first input packet, and OBT0 based on the OBT time of the first packet.

Alternatively any of the 3 values (UTC0, OBT0, CLOCK_DRIFT) can be supplied by the user.

If the rollovers command line argument is used then the algorithm will assume that number of OBT clock rollovers have occurred between the UTC0 time and the current

time, and this will be shown in the generated MDR record headers. These will be increased by approx. 49 days per increment of the rollovers parameter above zero.

To help fill the MPHR record, ccsds_to_l0 can try to read in OSVS values and subsatellite positions from the Eugene home directory, which is found by reading the EUGENE_HOME environment variable if defined, otherwise by running eugene --show-dir-home. If not found then default values will be placed into the product.

The MPHR field PRODUCT_NAME will be set correctly in the output product regardless of the actual name of the file. This can be renamed using the eps_renamer command from Eugene.

4.20 hirs_to_ccsds

4.20.1 Purpose

To convert a NOAA L1b format HIRS product into a stream of CCSDS packets, or a series of pictures.

4.20.2 Inputs

NOAA HIRS Level 1b product.

4.20.3 Outputs

Depending on parameters used:

-A file of concatenated CCSDS packets

-A series of images, one per radiometric channel.

4.20.4 Usage

hirs_to_ccsds [--in <filename>] [--out <filename>] [--config <filename>] [--show-config] [--check-config] [--image <filename prefix>] [--help] [--version] [--jpeg]

where:

--in <filename> : Specify name of input file. If omitted stdin is used.

--out <filename> : Specify name of output file. If omitted stdout is used.

--config <filename> : Specify name of CCSDS options file as detailed in Section 9.4. If the parameter is omitted the file “hirs.config” is looked for first in the directory $METOPIZER_CONFIG_DIR, then “$HOME/.metopizer”. If no configuration file is found a set of default values are used.

--show-config : After processing display a list of the configuration parameters used.

--check-config : Only show the configuration parameters. No other processing is performed.

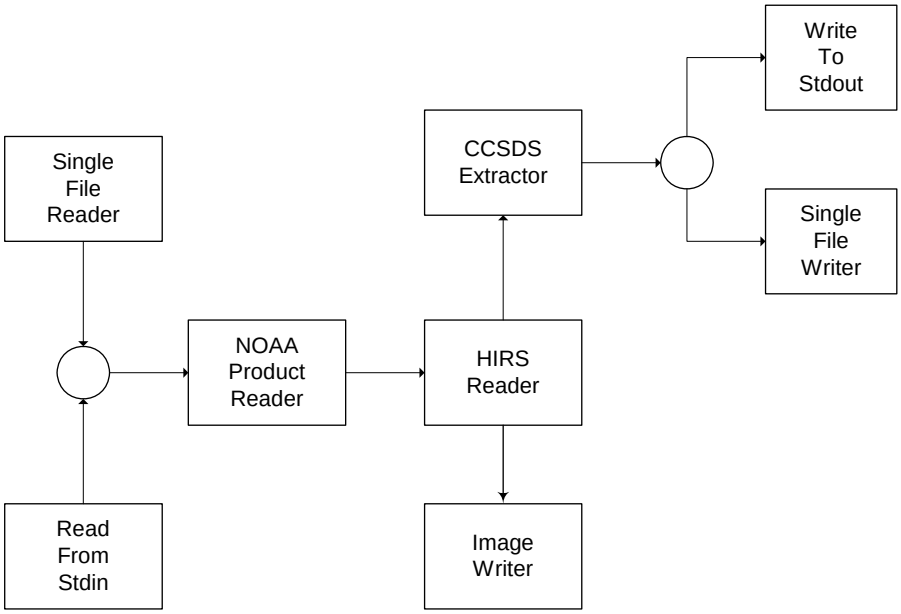
--image <filename prefix> : Generate a sequence of images in TIFF format based on the input file. The output consists of 20 monochrome images. No CCSDS packets are generated.

--help : show usage information

--version : show program version

--jpeg : Output JPEG format images instead of TIFF.

4.20.5 Functional diagram



4.20.6 Description

Converts a NOAA L1b HIRS product into a stream of source packets, one per scan line, or to a series of 20 graphics files, one per instrument channel.

4.21 l0_to_reduced_ccsds

4.21.1 Purpose

Convert an EPS L0 product into a hierarchy of Reduced CCSDS packets.

4.21.2 Inputs

EPS L0 product.

4.21.3 Outputs

-Reduced MHS packets, one per file, sorted into 1 directory per hour.

4.21.4 Usage

usage: l0_to_reduced_ccsds OPTIONS

Process either a stream of CCSDS packets to a L0 product and convert it to Reduced CCSDS packets suitable for MHS analysis tools.

Where OPTIONS can include:

File selection:

[-i, --in] <arg...>

Select input file(s)

Output settings:

-o, --output <arg>

Set output directory

Misc:

-q, --quiet Omit additional messages

-z Use simple busy animations

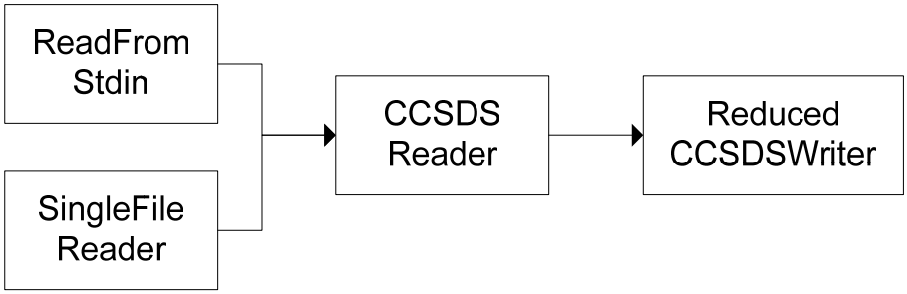
Program information:

--version show version information and exit

-?, --help show help message and exit

--long-help show extended help and examples

4.21.5 Functional diagram



4.21.6 Description

The Reduced MHS format is compatible with that used by the MHS SD Analysis software.

4.22 mhs_to_ccsds

4.22.1 Purpose

To convert a NOAA L1b format MHS product into a stream of CCSDS packets, or a directory hierarchy of Reduced MHS packets, or a series of pictures.

4.22.2 Inputs

NOAA MHS Level 1b product.

4.22.3 Outputs

Depending on parameters used:

-A file of concatenated CCSDS packets

-A series of images, one per radiometric channel.

-Reduced MHS packets, one per file, sorted into 1 directory per hour.

4.22.4 Usage

usage: mhs_to_ccsds OPTIONS

Converts a NOAA L1b v3 MHS product to MHS CCSDS packets.

Where OPTIONS can include:

File selection:

[-i, --in] <file> Select input file

-o, --out <file> Select output file

-c, --config <file>

Use a config file

--image <file> Output 5 channel images in TIFF format

Settings:

-j, --jpeg Use Jpeg output instead of TIFF

--mhs Force the input file to be read as MHS

--reduced-mhs <file>

Output as Reduced MHS packets

-q, --quiet Suppress most output messages

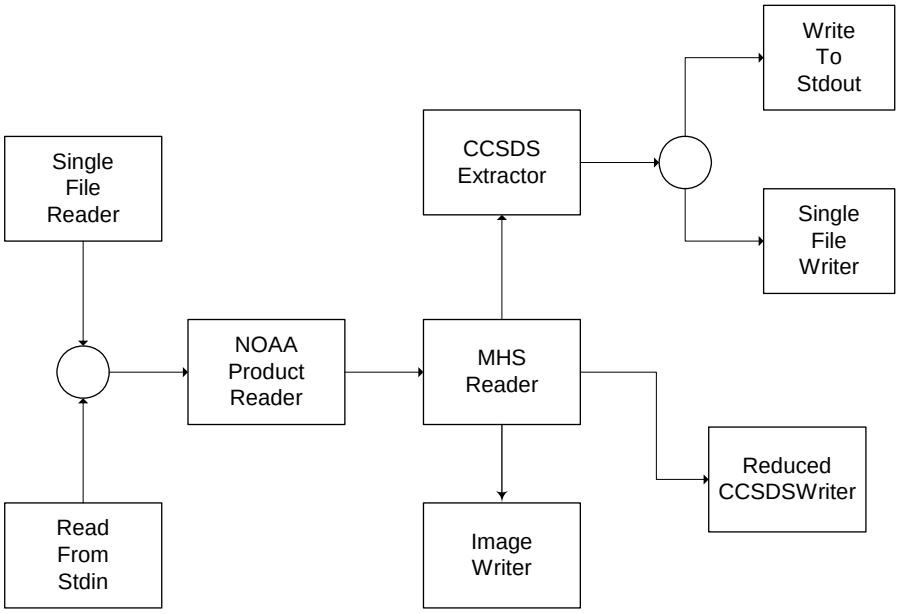
Program information:

--version show version information and exit

-?, --help show help message and exit

--long-help show extended help and examples

4.22.5 Functional diagram



4.22.6 Description

The Reduced MHS format is compatible with that used by the MHS SD Analysis software.

4.23 metopizer_viewer

4.23.1 Purpose

To use the Metopizer Python extension and read CCSDS Packets or L0 products for the extraction of sample data or date values. The resultant output is either an image or a graph depending on specified options.

4.23.2 Inputs

File containing either CCSDS Packets or L0 Products

4.23.3 Outputs

Graph for samples and times data and images for sample data, however image data can be displayed on screen or saved to disk for later viewing.

4.23.4 Usage

metopizer_displayer <input filename>] [--filter] [--skip] [--number] [--type] [--in] [--out] [--modulus] [--width] [--quiet] [--help] [--version]

where:

<input filename> or --in <filename> : Required name of input file

--filter : Applied APID filter to the product reader for selective packet reading.

--skip : Skip fixed number of packets from start of stream.

--number : Channel number to extract based upon required instrument sample data.

--type : Type of data to extract [IMAGE|GRAPH] to plot samples or produce images and [UTC|GRH|OBT] to plot times as a graph.

--in : Input file to read either as CCSDS Packets or L0 Product, however not that CCSDS stream does not contain GRH times.

--out : Output file which is only used when generating images, currently only the PNG file format is supported.

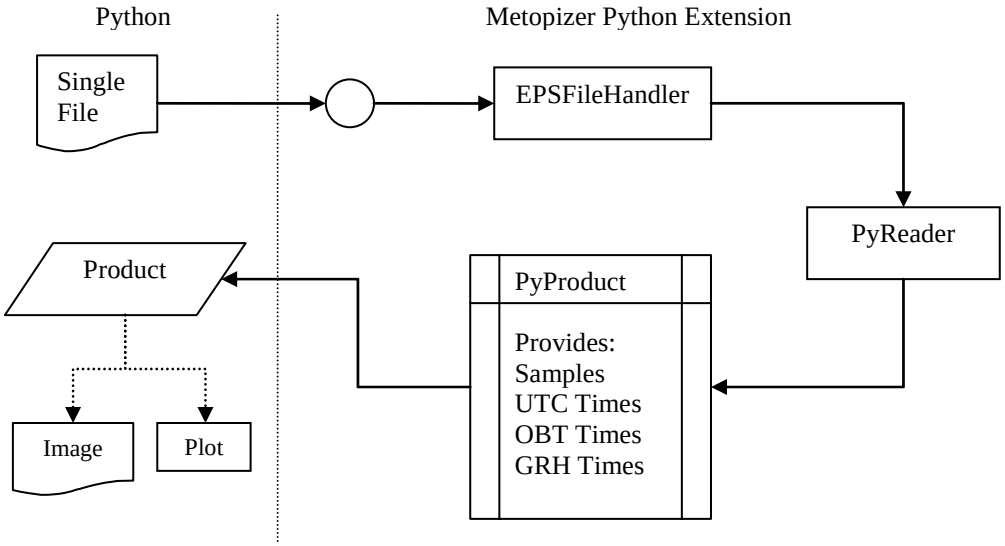
--modulus : Apply modulus filter across all packets, i.e. skip n number of subsequent packets after reading one.

--width : Apply sub sampling to each scan line, i.e. skip n number of samples after reading single sample value

--help : show usage information

--version : show program version

4.23.5 Functional diagram



4.23.6 Description

The input can only be a CCSDS Stream of L0 Product. The Metopizer Python interface is used to open the input file and read samples data from the stream. The resultant data returned is lists containing sample values for samples of doubles containing time values depending on the required output. All data is either plot onto a graph using Matplotlib or output to a file when generating images. Note that graphs will be shown with an appropriate title and x,y headings.

4.23.7 Usage Examples

Metopizer_viewer is a python script used to quickly visualise L0 products.

All information about the tool and its available commands are shown by running the following command:

```
metopizer_viewer --help
```

This will show all available commands and a simple example. Typical usage for the tools is as follows:

```
metopizer_viewer <product name>
```

where <product name> is the file to open and the default behaviour is to show the image contained within the product for channel 0 and for all sample values.

4.24 noaa_l1b_displayer

4.24.1 Purpose

To read in Level1b products and display in decoded text format, as a series of tables.

4.24.2 Inputs

Single NOAA L1b product.

4.24.3 Outputs

Text display.

4.24.4 Usage

noaa_l1b_displayer <filename> <record number> [--reference] [--amsu-a1] [--amsu-a2] [--help]

where:

<filename> : Specify name of input file.

<record number> : The program only displays a single block from the input product. Block 0 always shows the header information, 1 and above display a data record. The header display includes the number of data record present.

--reference : This switch modifies the output format to only display those settings common to source packets and L1b products (the format is identical to that of the ccsds_displayer program using its –reference option).

--amsu-a1 : Can only be used when the input is an AMSU-A product, this switch restricts output to the AMSU-A1 instrument data.

--amsu-a2 : As above but displays the AMSU-A2 data.

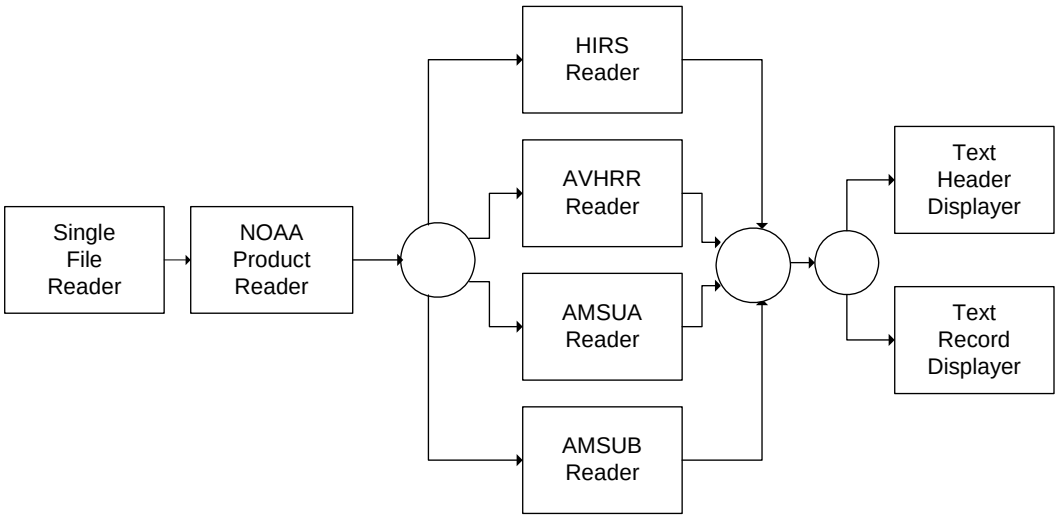
--bps x : for AVHRR products only, specify the product uses x bits per sample format. Not required when a NOAA archive header is present. Default value is 10bps.

--channels xxxxx : Each x specifies whether a particular channel 1-5 is present in the input product. Not required if a NOAA archive header is present.

--help : show usage information

--version : show program version

4.24.5 Functional Diagram



4.24.6 Description

This is not a generic product displayer, and must be hard coded to handle each input type. The program only supports the ATOVS file formats (AVHRR, AMSU-A, AMSU-B, HIRS). Also this program was developed in order to test the MetOpizer software, so the results do not show all data from the input product but concentrate on the fields used by the MetOpizer.

Due to the number of variants of AVHRR formats, there are some cases where they cannot be identified. In particular if the input product does not have an SAA Archive Header attached, the AVHRR record must be in 10 bits per sample format or it will not be recognised correctly. Also note if a file is in the wrong format the noaa_l1b_displayer program will not always detect the problem.

The switches –bps and –channels can be used to tell the program what the actual format is.

For products with the SAA Archive Header there are no problems identifying products. This issue only applies to AVHRR data.

4.25 sband_tm_displayer

4.25.1 Purpose

To open a file containing one or more CCSDS packets and display information about the entire file or individual packets.

4.25.2 Inputs

Files containing concatenated S-band frames.

4.25.3 Output

Text description to the terminal.

4.25.4 Usage

usage: sband_tm_displayer OPTIONS

Decode and display the contents of single/multiple s-band telemetry packets.

Where OPTIONS can include:

File selection:

[-i, --in] <arg...>

Select input file(s)

Packet selection:

-c, --count <n> Display only <n> packets

-k, --skip <n> Skip <n> packets before processing

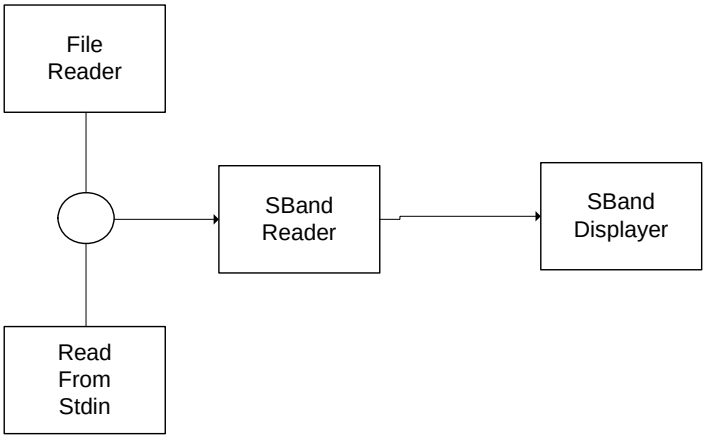
Program information:

--version show version information and exit

-?, --help show help message and exit

--long-help show extended help and examples

4.25.5 Functional diagram



4.25.6 Description

4.26 tvcdu_corrupt_filter

4.26.1 Purpose

The tVCDU packet structure is shown in the table below.

Version number	Space Craft ID	Virtual Channel ID	VCDU counter	Replay flag	Spare	Insert zone	Payload	UTC time stamp	Reed Solomon Status
2 bits	8 bits	6 bits	24 bits	1 bit	7 bits	2 bytes	884 bytes	8 bytes	8 bits
1 (v2 packet format)	11, 12, 13, 14 for METOP 1, 2, 3 and Simulator.	(See section 5.43.4)	Incrementing counter unique to each VCID.	0 for real time data	Zero fill	Encrypt_flag key (1 byte) Encrypt_key (1 byte)	M-PDU packet (See section 5.43.4)	Days (2 bytes) Milliseconds (4 bytes) Microseconds (2 bytes)	(See section 5.44.4)

As described in the remainder of this section, the tvcdu_corrupt_filter allows all the fields shown in the table above, including the M-PDU packet, to be modified and hence, corrupted.

4.26.2 Inputs

One or more streams of tVCDU packets.

4.26.3 Outputs

tVCDU packets with applied corruptions.

4.26.4 Usage

tvcdu_corrupt_filter [-i|--in <filename(s)>] [-o|--out <filename>] [-s|--skip <N>] [-c|--count <N>] [For Options] [Set Options] [Byte Options] [-a|--show-vcids] [-t|--show-tvcdu-table] [-m|--show-mpdu-table] [-r|--show-rss-table] [-h|--help] [-v|--version]

where:

--in <filename(s)> : Allows multiple input files to be specified. Note that multiple input file names should be delimited by spaces. If the “--in” option is omitted, stdin is used by default.

--out <filename> : Allows the name of an output file to be specified. If omitted stdout is used.

--skip <N> : Allows N packets to be skipped before applying any corruptions.

--count <N> : Allows corruptions to be applied to only N packets starting at the first packet or after skipping a number of packets if the --skip option is used.

For Options : The "--for-" options discussed below allow the corruption constraints to be defined and thus allow single or a group of packets to be isolated and corrupted while leaving any packets that do not meet the constraints untouched.

Set Options : The "--set-" options discussed below allow the exact nature of the corruptions to be defined.

Byte Options: The byte options discussed below are a special case that allow corruptions to be applied to any part of a packet.

--show-vcids : Shows a list (similar to that in section 5.43.4) of recognised Virtual Channel IDs (VCIDs). The recognised VCID numbers can be used with the --for-vcid option to only apply corruptions to a select number of packets that match the specified VCID constraints.

--show-tvcdu-table : Displays a table similar to the table shown in section 4.26.1 above. (Note that the screen must be wide enough for the table to be displayed properly.)

--show-mpdu-table : Displays a table, similar to that in section 5.43.4, showing the structure of M-PDU packets. (Note that the screen must be wide enough for the table to be displayed properly.)

--show-rss-table : Displays a table, similar to that in section 5.44.4, showing the structure of the Reed Solomon Status bitfield. (Note that the screen must be wide enough for the table to be displayed properly.)

--help : Shows usage information.

--version : Shows version information.

To avoid unnecessary typing, all the options discussed above have short hand names by which they can be refereed to (-i, -o, -s, -c, -l, -a, -t, -m, -r, -h and -v).

For Options

--for-virtual-channel-id (--for-vcid) <vcid(s)> : Allows single or multiple numeric VCIDs to be specified. Only packets with VCIDs matching the ones specified will be corrupted. To obtain a full list of acceptable VCID numbers use the --show-vcids option.

--for-vcdu-counter (--for-vcnt) <start_ctr> <end_ctr> : Only packets with VCDU counters greater or equal to start_ctr and less than or equal to end_ctr will be corrupted.

--for-utc-time (--for-time) <start_time> <end_time> : Only packets with UTC-times greater or equal to start_time and less than or equal to end_time will be corrupted. Start and end times can be specified in two ways: a relative time and an absolute time. The absolute time must be in the format YYYYMMDDhhmmssZ, where the letters respectively represent Years, Months, Days, hours, minutes and seconds (e.g. 20041223151139Z). The relative time can be defined as an offset of +N(ms|s|m), where N represents a number and must be followed by units of millisecond(ms), seconds(s) or minutes(m) (e.g. +2m). The relative start time is always relative to the UTC-time of the first packet. The relative end time is a little more complicated. If an absolute start time is followed by a relative end time, then the relative end time will be relative to the absolute start time. If a relative start time is followed by a relative end time, then the relative end time will be

relative to the UTC time of the first packet offset by the relative start time. It might sound a little complicated in words, but mathematically its quite simple:

if start_time = A then start_time = A

if end_time = +5m then end_time = A + 5m

if start_time = +2m then start_time = T1 + 2m

if end_time = +5m then end_time = T1 + 2m + 5m

Where A represents a specified absolute time, T1 represents the UTC-time of the first tVCDU packet read and +2m and +5m represent 2 and 5 minute relative time offsets.

Set Options

--set-version-number (--set-vn) <random|N1|N1 N2>

--set-space-craft-id (--set-scid) <random|N1|N1 N2>

--set-virtual-channel-id (--set-vcid) <random|N1|N1 N2>

--set-vcdu-counter (--set-vcnt) <random|N1|N1 N2>

--set-replay-flag (--set-rf) <random|N1|N1 N2>

--set-spare (--set-sp) <random|N1|N1 N2>

--set-encrypt-flag (--set-ef) <random|N1|N1 N2>

--set-encrypt-key (--set-ek) <random|N1|N1 N2>

--set-mpdu-spare (--set-msp) <random|N1|N1 N2>

--set-mpdu-header (--set-mh) <random|N1|N1 N2>

--set-utc-day (--set-day) <random|N1|N1 N2>

--set-utc-millisecond (--set-mil) <random|N1|N1 N2>

--set-utc-microsecond (--set—mic) <random|N1|N1 N2>

--set-reed-solomon-status (--set-rss) <random|N1|N1 N2>

The functions of the above set options are obvious and do not need to be individually explained. Each set option allows one of the tVCDU fields shown in section 4.26.1 to be corrupted. Note that three types of corruptions can be introduced:

- The first type of corruption allows the fields of consecutive tVCDU packets to be set to an appropriate* random number.
- The second type of corruption allows the fields of consecutive tVCDU packets to be set to an appropriate* single number (N1).
- The third type of corruption allows the fields of consecutive tVCDU packets to be filled with an appropriate* number series or patter starting with the number N1 and incremented by the number N2. For example if N1=0 and N2=2, then consecutive packets will be filled with the numbers 0, 2, 4, 6, … If the field has a maximum value that is reached, the pattern or series will simply wrap around and continue. For example if the maximum acceptable value is 7, then the pattern will be: 0, 2, 4, 6, 0,

2, 4, 6, 0, … Note that N2 can also be assigned a negative value and thus produce increasing and decreasing number patterns.

The term appropriate* has been used in the above paragraph to imply that the corruptions applied must be greater or equal to the minimum value the field can accept and of course less than or equal to the maximum value the field can accept. The code attempts to handle all this automatically. If however, the user does input values that are out of range, they will be prompted with error messages and advised to revise their inputs.

Byte Options

Unlike the set options, described above, that only act to modify specific fields, the byte options allow corruptions to be applied to any part of a tVCDU packet.

--for-byte-offset <N1|N1 N2> : Corruptions can be introduced to a single byte at byte offset N1 or to a range of bytes starting at byte offset N1 and ending at byte offset N2. Note that if the corruption is to be applied to the first byte, the byte offset must be set to zero. Hence, the byte offset must be greater or equal to zero and of course less than the packet length.

--set-byte <random|N1|N1 N2> : Similar to the set options discussed above, the --set-byte option allows a single byte per packet or a range of bytes per packet to be set to random values, to be set to a single value N1, or to be filled with a number pattern starting with a value N1 and incremented by a value N2. Note that N1 and N2 can not be set to anything greater than 255. While N1 can not be less than zero, the increment N2 can be assigned a negative value.

--set-byte-increment <column|row> : This option only becomes relevant if one is attempting to assign a number pattern to a range of bytes per packet. For example a user might want to corrupt byte offsets 101 to 105 (--for-byte-offset 101 105) with a number pattern: 1, 3, 5, 7, 9, … (--set-byte 1 2). If the --set-byte-increment option is left undefined, the resulting corruption per packet will be as follows:

Byte Offset	**101**	**102**	**103**	**104**	**105**
Packet 1	1	3	5	7	9
Packet 2	3	5	7	9	11
Packet 3	5	7	9	11	13
Packet 4	7	9	11	13	15
Packet 5	9	11	13	15	17
Packet 6	11	13	15	17	19

Note that the pattern has been increment over both column (Byte Offset) and row (Packet). Hence, if the --set-byte-increment option is left undefined, the default option is to apply the pattern over both columns and rows. If however, the --set-byte-increment is set to column, the pattern will only be incremented over the different byte offsets and will result into the following corruption:

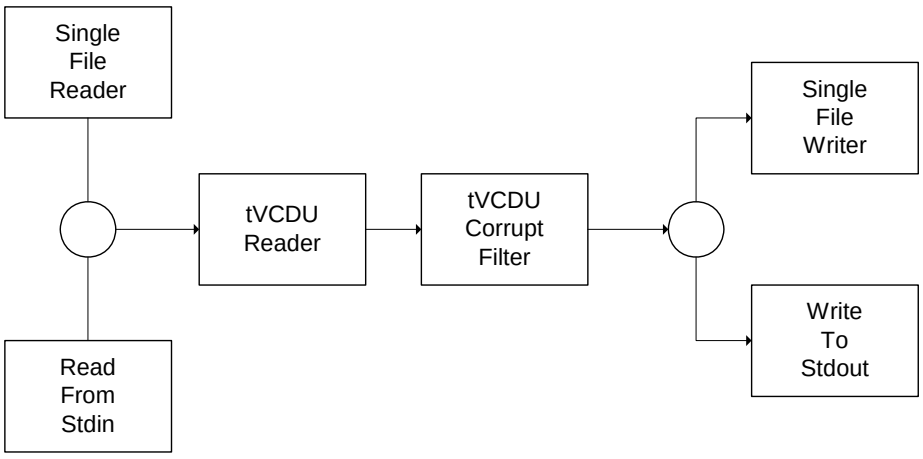
Byte Offset	**101**	**102**	**103**	**104**	**105**
Packet 1	1	3	5	7	9

Packet 2	1	3	5	7	9
Packet 3	1	3	5	7	9
Packet 4	1	3	5	7	9
Packet 5	1	3	5	7	9
Packet 6	1	3	5	7	9

Similarly, if the --set-byte-increment option is set to row, the pattern will only be incremented over the different packets and will result into the following corruption:

Byte Offset	**101**	**102**	**103**	**104**	**105**
Packet 1	1	1	1	1	1
Packet 2	3	3	3	3	3
Packet 3	5	5	5	5	5
Packet 4	7	7	7	7	7
Packet 5	9	9	9	9	9
Packet 6	11	11	11	11	11

4.26.5 Functional Diagram



4.26.6 Description

The idea of separating all the options into two groups (*for* and *set*) aims to help the user understand the options logically. For example if we wanted to set the version number to 2 for a range of VCDU counters starting at 3 and ending at 15, we can express the command in words as: For VCDU counters starting at 3 and ending at 15, set version number to 2. Applying this command to the tvcdu_corrupt_filter works the same way:

```
> tvcdu_corrupt_filter –i in.tvcdu –o out.tvcdu
```

```
  --for-vcdu-counter 3 15 --set-version-number 2
```

We can take things a step further and specify that we want the version number to be set to a random number for VCDU counters 3 to 15, but only if the VCID is equal to 9 and 10:

```
> tvcdu_corrupt_filter –i in.tvcdu –o out.tvcdu --for-vcid 9 10
  --for-vcdu-counter 3 15 --set-version-number random
```

We have a third option where we can start our corruption with an initial start value and increment further corruptions by a fixed amount. For example we can start our corruption with a value of 1 and increment subsequent corruptions by a value of 2 (i.e. introduce a corruption pattern of 1, 3, 5, …).

```
> tvcdu_corrupt_filter –i in.tvcdu –o out.tvcdu --for-vcid 9 10
  --for-vcdu-counter 3 15 --set-version-number 1 2
```

Further more, time constraints can also be set and multiple fields (in this case Version Number, Spacecraft-ID and Reed Solomon Status) can be corrupted at the same time. Hence, one is not limited to corrupting one field at a time.

```
> tvcdu_corrupt_filter –i in.tvcdu –o out.tvcdu --for-vcid 9 10
  --for-vcdu-counter 3 15 --for-utc-time 20050211095233Z +5m
  --set-version-number 1 2 –-set-scid 12 –-set-rss random
```

As illustrated the *for* and *set* options work well together and allow multiple corruptions to be introduced to any subset of packets.

4.27 tvcdu_summarizer

4.27.1 Purpose

To show a statistical display of the ranges of t-VCDU packets split per VCID.

4.27.2 Inputs

A single file containing concatenated t-VCDU packets, or standard input.

4.27.3 Outputs

Text display to console.

4.27.4 Usage

tvcdu_summarizer [filename] [--no-times]

where:

filename: the name of an input file to read. If omitted standard input is used.

--no-times: If specified the columns containing the t-VCDU transmission timestamps are hidden.

4.27.5 Description

Internally this Perl script calls tvcdu_displayer, captures the output and processes as text.

Here is a sample of the output of the program:

```
-------------------------------------------------------------------------------------------------------------
VCID   Occurrence    Start Count  End Count  Start Time                  End Time                    RSS
-------------------------------------------------------------------------------------------------------------
3      22            0            21         08-Aug-2002,20:02:00.054800   08-Aug-2002,20:02:00.67384    128 (22)
5      137           0            136        08-Aug-2002,20:02:00.043040   08-Aug-2002,20:02:00.76232    128 (137)
9      2352          0            2351       08-Aug-2002,20:02:00.006720   08-Aug-2002,20:02:00.79992    128 (2352)
12     14            0            13         08-Aug-2002,20:02:00.035200   08-Aug-2002,20:02:00.75392    128 (14)
15     148           0            147        08-Aug-2002,20:02:00.048160   08-Aug-2002,20:02:00.78880    128 (148)
24     1508          0            1507       08-Aug-2002,20:02:00.004080   08-Aug-2002,20:02:00.79264    128 (1508)
27     33            0            32         08-Aug-2002,20:02:00.000720   08-Aug-2002,20:02:00.72232    128 (33)
29     83            0            82         08-Aug-2002,20:02:00.122800   08-Aug-2002,20:02:00.72168    128 (83)
34     26            0            25         08-Aug-2002,20:02:00.000000   08-Aug-2002,20:02:00.78232    128 (26)
63     1             0            0          08-Aug-2002,20:02:00.001360   08-Aug-2002,20:02:00.00136    128 (1)
```

```
1           0           0          08-Aug-2002,20:02:00.001440   08-Aug-2002,20:02:00.00144    128 (1)
1           0           0          08-Aug-2002,20:02:00.001520   08-Aug-2002,20:02:00.00152    128 (1)
```

The table format is designed to mimic the output of a similar tool present in the FEP RUS.

4.28 tvcdu_to_ccsds

4.28.1 Purpose

To convert one or more files of concatenated t-VCDU packets into CCSDS packets.

4.28.2 Inputs

List of input files containing t-VCDU packets.

4.28.3 Outputs

One of more files containing lists of CCSDS packets

4.28.4 Usage

tvcdu_to_ccsds [--in <input filename(s)>] [--out <filename prefix>] [--filter <APID>] [--test-filter] [--show-apids] [--vcfilter <vcid>] [--show-mpdus] [--help] [--version]

where:

--in : Gives an optional list of input files. If omitted stdin is used. Multiple input filenames should be delimited by spaces.

--out : Gives an optional output filename prefix. For each output file a numerical application ID is appended to the filename prefix (the program always generates one file per APID) and a file extension of ".ccsds". If omitted all packets go to stdout in the same order as found in the input stream.

--filter x : Specify a list of filters based on CCSDS Application ID (space delimited). Only packets matching the filter are allowed through. Filters can be given by name or number, and if a name is given it may represent multiple APIDs.

--test-filter : Show the results of the specified --filter as a list of numeric APIDs. No other processing is performed.

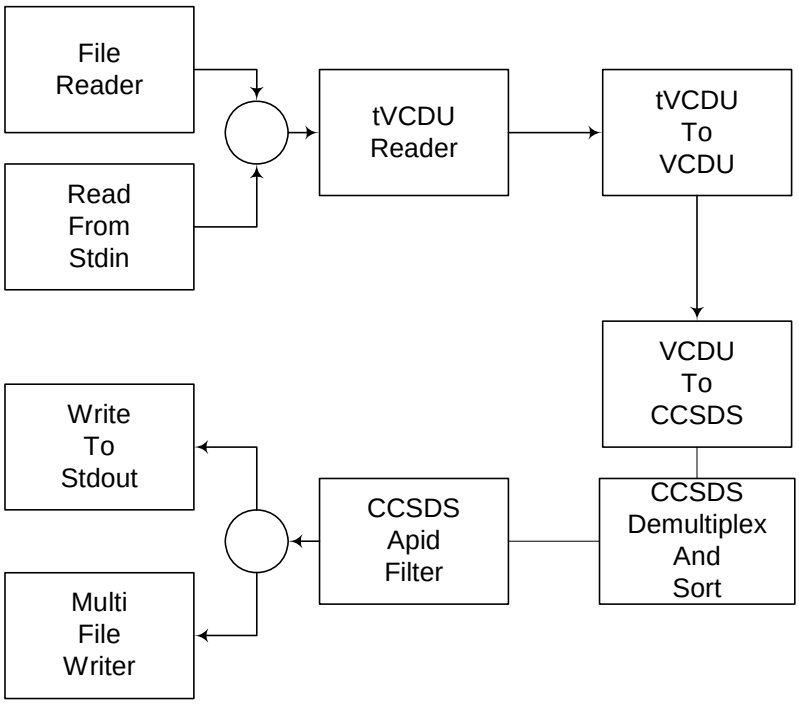
--show-apids : Display list of allowable APID names against numbers. No other processing is performed.

--show-mpdu : Display a summary (First Header Pointer and a calculated checksum) of the payload M-PDU packet of each VCDU packet. No other processing is performed.

--help : Show usage information.

--version : Show program version number.

4.28.5 Functional diagram



4.28.6 Description

M-PDU packets are extracted from the input stream, demultiplexed and sorted into Virtual Channels.

Each Virtual Channel is then examined for embedded CCSDS packets which are picked out. They must then be demultiplexed since more than one type of CCSDS packet can share a single Virtual Channel, then packets of each APID are sorted by sequence number.

The final CCSDS packet streams are either written to disk or stdout.

4.29 tvcdu_displayer

4.29.1 Purpose

To read in Level1b products and display in brief text format.

4.29.2 Inputs

File containing a list of concatenated t-VCDU packets, as generated [by cadu_to_tvcdu.

4.29.3 Outputs

Text display.

4.29.4 Usage

tvcdu_displayer <input filename>] [--compact] [--help] [--version]

where:

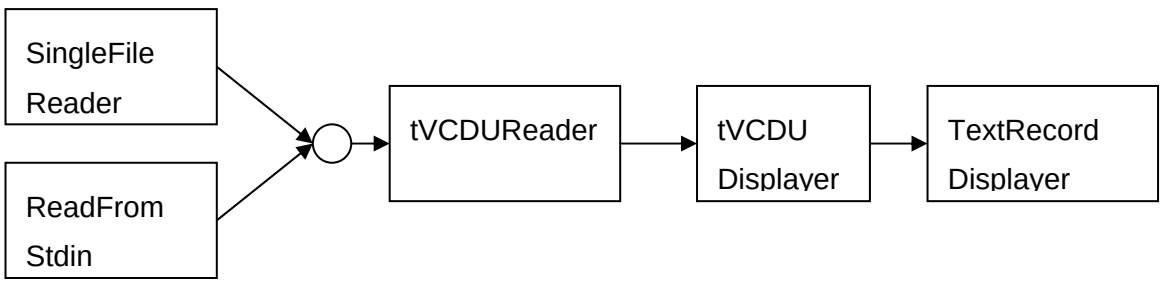
<input filename> : Optional name of input file. If omitted stdin is used.

--compact : Use a compact layout with one line per t-VCDU packet.

--help : show usage information

--version : show program version

4.29.5 Functional diagram



4.29.6 Description

The output includes t-VCDU header fields and the first header pointer and a calculated checksum per M-PDU packet.

5 FILTERS DETAIL

A MetOpizer program is constructed by building a network of filter objects, where each filter accepts inputs from a previous filter and outputs to one or more successor filters. At the beginning and endings of a chain will be filters that handle I/O such as reading from disk or printing to screen.

One feature common to almost all filters is an implementation for TextSummaryProducer, which generates a few lines of text describing what the filter has done.

In the following sections the filters that form MetOpizer are examined in detail.

5.1 AMSUAReader

5.1.1 Purpose

To read in and interpret a NOAA AMSU-A Level 1b product. The outputs are either a text description of the header and individual records, or a set of images.

Note for CCSDS packet output the filters AMSUA1Reader and AMSUA2Reader should be used, since these focus on specific instruments.

In addition this filter implements the TextDetailsProducer which outputs the complete input product in a readable text format. This can be used to check the outputs by comparison against the TextDetailsProducer output from the CCSDSReader filter.

5.1.2 Constructor

ProductProducer object

5.1.3 Implements

TextHeaderProducer, TextRecordProducer, ImageProducer, TextSummaryProducer

5.1.4 Method

A NOAA AMSU-A L1b file consists of a header structure followed by a number of data structures, one per scan line.

A complication for the AMSU-A instrument is that the Application Data field of the source packets is several hundred bytes longer than the space required for actual data, with the data and special filler codes intermingled through the data section. An algorithm is used to control how this scattering is done using some parameters read from the configuration file.

The implementation of TextRecordDisplayer shows both AMSU-A1 and AMSU-A2 data.

5.2 AMSUA1Reader

5.2.1 Purpose

To read in a NOAA AMSU-A Level 1b product and convert it to a stream of MetOp format CCSDS packets for the AMSU-A1 instrument.

5.2.2 Constructor

ProductProducer object

(optional) filename of configuration file

5.2.3 Implements

CCSDSProducer, TextHeaderProducer, TextRecordProducer, TextConfigProducer, TextSummaryProducer

5.2.4 Method

A NOAA AMSU-A L1b file consists of a header structure followed by a number of data structures, one per scan line. This filter extracts the channels and telemetry for the AMSU-A1 part and converts them to CCSDS packets.

The optional configuration file allows user defined parameters to be read in to modify some details for the output packets. This includes the CCSDS initial sequence number, time stamp offsets and instrument ID codes.

An additional complication for the AMSU-A instrument is that the Application Data field of the source packets is several hundred bytes longer than the space required for actual data, with the data and special filler codes intermingled through the data section. An algorithm is used to control how this scattering is done using some parameters read from the configuration file.

See Appendix A for a table showing the exact structure of the CCSDS packets produced.

The implementation of TextRecordDisplayer differs from AMSUAReader in that it only shows the AMSU-A1 instrument data.

5.3 AMSUA2Reader

5.3.1 Purpose

To read in a NOAA AMSU-A Level 1b product and convert it to a stream of MetOp format CCSDS packets for the AMSU-A2 instrument.

5.3.2 Constructor

ProductProducer object

(optional) filename of configuration file

5.3.3 Implements

CCSDSProducer, TextHeaderProducer, TextRecordProducer, TextConfigProducer, TextSummaryProducer

5.3.4 Method

A NOAA AMSU-A L1b file consists of a header structure followed by a number of data structures, one per scan line. This filter extracts the channels and telemetry for the AMSU-A2 part and converts them to CCSDS packets.

The optional configuration file allows user defined parameters to be read in to modify some details for the output packets. This includes the CCSDS initial sequence number, time stamp offsets and instrument ID codes.

An additional complication for the AMSU-A instrument is that the Application Data field of the source packets is several hundred bytes longer than the space required for actual data, with the data and special filler codes intermingled through the data section. An algorithm is used to control how this scattering is done using some parameters read from the configuration file.

See Appendix A for a table showing the exact structure of the CCSDS packets produced.

The implementation of TextRecordDisplayer differs from AMSUAReader in that it only shows the AMSU-A2 instrument data.

5.4 AMSUBReader

5.4.1 Purpose

To read in a NOAA AMSU-B format Level 1b product and display text information about the file, or to produce a series of images.

5.4.2 Constructor

ProductProducer object

5.4.3 Implements

TextHeaderProducer, TextRecordProducer, ImageProducer

5.4.4 Method

This filter is used by programs to display information about the AMSU-B product, without using the additional MHS conversion features the AMSUBToMHS filter uses.

5.5 AMSUBToMHS

5.5.1 Purpose

To read in a NOAA AMSU-B format NOAA Level 1b product and generate a stream of MetOp format MHS source packets based on the science measurements from the input product combined with simulated telemetry information.

5.5.2 Constructor

ProductProducer object

(optional) filename of configuration file

5.5.3 Implements

CCSDSProducer, ImageProducer , TextHeaderProducer, TextRecordProducer, TextConfigProducer, TextSummaryProducer

5.5.4 Method

A NOAA AMSU-B L1b file consists of a header structure followed by a number of data structures, one per scan line. The science data returned by the AMSU-B is similar enough to the MHS instrument that it can be used as MHS test data with no further processing.

The optional configuration file allows user defined parameters to be read in to modify some details for the output packets. This includes the CCSDS initial sequence number, time stamp offsets and instrument ID codes.

See Appendix A for a table showing the exact structure of the CCSDS packets produced.

Due to hardware differences between the AMSU-B and MHS units, a large number of configuration parameters can be supplied to this filter. Appendix B lists these fields and Appendix A shows how they are used. In addition the document RD.2 describes the method used to generate simulated telemetry.

This filter also implements the TextSummaryProducer interface to give an ASCII description of the output data (if a configuration file is used) or the input product (if no configuration file is used).

5.6 AVHRRInterpolator

5.6.1 Purpose

To read in a NOAA Level 1b AVHRR GAC format file and convert it to LAC format.

5.6.2 Constructor

ProductProducer object

5.6.3 Implements

ProductProducer, TextSummaryProducer

5.6.4 Method

The NOAA LAC format stores data at a resolution of 1.1 km/pixel, in contrast to 4km/pixel used in GAC data.

In all cases the AVHRR initially samples data at the full resolution, which is then reduced to low resolution in software. The method used is to divide each scan line into blocks of 5 pixels. The first four pixels are averaged together to give a single value which becomes the corresponding pixel in the low resolution product. The fifth pixel of every block is discarded.

In addition two of every three scan lines are discarded.

This filter uses a simple linear interpolation to produce output data that conforms to the high resolution format For example, if the input consisted of the grid below:

X1	X2
X3	X4

The output would be:

X1				X2
X3				X4

The formula used is:

$$S(i,j) = \frac{X1*(4-i)*(2-j) + X2*i*(2-j) + X3*(4-i)*j + X4*i*j}{8}$$

Where S(i,j) means the value of the sample at output co-ordinates i,j and X1 to X4 are the input samples. An identical calculation is performed on each of the five channels of each sample.

If the input product is already in GAC format it is passed through unmodified.

5.7 AVHRRReader

5.7.1 Purpose

To read in a NOAA AVHRR LAC format (1.1 km/pixel) file and convert it to a stream of MetOp format CCSDS packets or a series of images.

The filter can also display sections of the product in text format.

5.7.2 Constructor

ProductProducer object

(optional) filename of configuration file

5.7.3 Implements

CCSDSProducer, ImageProducer, TextHeaderProducer, TextRecordProducer, TextConfigProducer, TextSummaryProducer

5.7.4 Method

Each record in the input product is translated into a single source packet.

For source packet output, the input data must be high resolution AVHRR format.

An optional configuration file allows the initial CCSDS sequence count and a time offset value to be specified.

See Appendix A for a table showing the exact structure of the CCSDS packets produced.

5.8 BufferCorruptFilter

5.8.1 Purpose

To enable the modification/corruption of any input files.

5.8.2 Constructor

BufferProducer object

Integer size

Integer skip

Integer count

2 structures of type INFO specifying the exact nature of Byte and Bit corruptions to be implemented

2 Integers Byte offset start and Byte offset end

String increment (both/row/column)

2 Integers Bit offset start and Bit offset end

2 Integers Bit offset start relative and Bit offset end relative

Integer Bit width value

String Bit width key (random/blank)

Integer Maximum number of corruptions

5.8.3 Implements

BufferProducer, TextSummaryProducer, FileProducer

5.8.4 Method

The BufferCorruptFilter reads packets of any specified size (greater than zero bytes) one at a time and applies corruptions to them by allowing a single or a range of bytes or bits to be modified. The modifications can consist of setting bytes/bits to be filled with:

1. Random values
2. A constant value
3. A number pattern (1,2,3,… or 1,3,5,,… or 1,0,1,0,… etc.)

A more in depth discussion of the functionality and the nature of corruptions that can be introduced is given in section 4.4.

5.9 BufferReader filter

5.9.1 Purpose

To read in binary data and convert it to a stream of packets.

5.9.2 Constructor

FileProducer object

5.9.3 Implements

BufferProducer, TextSummaryProducer

5.9.4 Method

The size of the packets is user configurable. A default packet size of 1024 bytes is used if no size is specified. If however, the size of the input file is less than 1024 bytes, a packet size equal to the total size of the input file is used.

5.10 CADUCorruptFilter

5.10.1 Purpose

To enable the corruption of CADU packets.

5.10.2 Constructor

CADUProducer object

Integer skip

Integer count

2 structures of type INFO specifying the exact nature of Byte and Bit corruptions to be implemented

2 Integers Byte offset start and Byte offset end

String increment (both/row/column)

2 Integers Bit offset start and Bit offset end

2 Integers Bit offset start relative and Bit offset end relative

Integer Bit width value

String Bit width key (random/blank)

Integer Maximum number of corruptions

5.10.3 Implements

CADUProducer, TextSummaryProducer, FileProducer

5.10.4 Method

The CADUCorruptFilter reads CADU packets one at a time and applies corruptions to them by allowing a single or a range of bytes or bits to be modified. The modifications can consist of setting bytes/bits to be filled with:

4. Random values
5. A constant value
6. A number pattern (1,2,3,… or 1,3,5,,… or 1,0,1,0,… etc.)

A more in depth discussion of the functionality and the nature of corruptions that can be introduced is given in section 4.5.

5.11 CADUSplitter

5.11.1 Purpose

Split a stream of CADU packets into dumps.

5.11.2 Constructor

CADUProducer source,

Int Duration,

Int Dump_overlap,

Bool shift_gras_obt,

Bool apid1_only

5.11.3 Implements

FileProducer,

CADUProducer

5.11.4 Method

The basic operation of the filter is that CADU packets are read from a single input stream, and written unmodified to the current output stream.

On construction the filter calculates 2 OBTG timestamps: begin_overlap and end_dump. When a CADU packet width an OBT time greater than begin_overlap is found the filter begins storing copies of all CADUs in an internal queue holding the dump overlap period When end_dump is reached the queue is flushed into a new file and future CADU packets go to the new file. At this point begin_overlap and end_dump are incremented ready for the new dump boundaries.

5.12 CADUReader filter

5.12.1 Purpose

To read in binary data and convert it to a stream of CADU packets

5.12.2 Constructor

FileProducer object

5.12.3 Implements

CADUProducer, TextSummaryProducer

5.12.4 Method

A CADU packet is defined as follows:

Sync Marker	**Payload**
4 byte fixed number.	1020 bytes data.

(the Sync Marker is always equal to 0x1ACFFC1D)

The filter scans the input files looking synchronisation markers. Each time one is found a CADU packet is formed from the marker plus the following 1020 bytes, giving the output packets.

The filter is written to handle gaps in between packets. In addition a warning is raised if a synchronisation marker is found within a packet payload, since this could mean data has been lost.

5.13 CADUToCVCDU filter

5.13.1 Purpose

To receive a stream of CADU packets and extract the embedded Coded VCDUs (CVCDUs).

5.13.2 Constructor

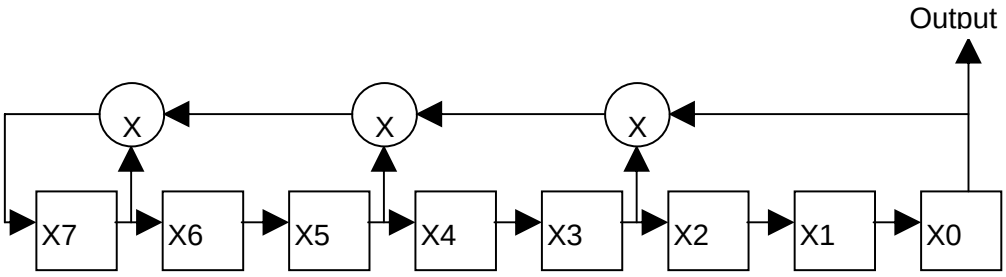
CADUProducer object

5.13.3 Implements

CVCDUProducer, TextSummaryProducer

5.13.4 Method

The CADU payload holds a CVCDU packet that has been exclusive-ored against a constant 1020 byte number, built using the following device:



In the diagram the boxes X0 to X7 are one bit registers each holding a single value, and the circles with crosses inside represent one bit adders with no carry.

Initially all eight registers are filled with ‘1’s. A clock is used to pulse the model, and on each beat a single bit of data flows through each line in the direction of the arrows.

Therefore each register forwards its bit of data to the next register in sequence and the output of the complete device, for that clock pulse, is read at the “Output” section.

Register X7 is filled with the sum of (X0 + X3 + X5 + X7) modulus 2. This value is computed at the start of the clock pulse, then inserted into X7 at the end of the pulse.

During initialisation the device is clocked 8160 times to generate 1020 bytes in total.

The device described above is sometimes also represented by the polynomial function

x8 + x7 + x5 + x3 + 1 (for additional information see RD.1, Section 3).

The decoded CADU payload is then output as a CVCDU packet.

5.14 CADUWriter filter

5.14.1 Purpose

Store CADU packets to a file on disk.

5.14.2 Constructor

CADUProducer object

Filename string

5.14.3 Implements

None.

5.14.4 Method

CADU packets are written to disk with no padding between them and no file header.

5.15 CCSDSAPIDFilter filter

5.15.1 Purpose

To examine an input stream of CCSDS packets and only allow through those that have an acceptable APID.

5.15.2 Constructor

CCSDSProducer object

APID list of strings

5.15.3 Implements

CCSDSProducer, TextSummaryProducer

5.15.4 Method

The APID can take values from the table below:

AVHRR_HR	103 and 104
AVHRR_HR_3A	103
AVHRR_HR_3B	104
AVHRR_LR	64-70
AVHRR_LR_1	64
AVHRR_LR_2	65
AVHRR_LR_3A	66
AVHRR_LR_3B	67
AVHRR_LR_4	68
AVHRR_LR_5	69
AVHRR_LR_CAL	70
AMSU_A	39 and 40
AMSU_A1	39
AMSU_A2	40
HIRS	38
SEM	37
A_DCS	35
MHS	34
IASI	130, 135, 140, 145, 150, 160, 180
ASCAT	192-255

GOME_2	384-477
GRAS_MEASURE	448-511
GRAS_NAV	2 and 3
SATELLITE or SL_PACKET	1
ADMIN	6

The filters constructor argument is a list of strings. Each can be either a name from the table above or a number. If a name is specified then all APID's corresponding to that name are allowed through.

The implementation of TextSummaryProducer normally outputs a list of the packets filtered out and allowed through. If used before the filter has processed any inputs, then the text output just consists of a list of numerical APIDs based on the constructor parameters.

5.16 CCSDSCorruptFilter

5.16.1 Purpose

To enable the corruption of CCSDS packet common fields.

5.16.2 Constructor

CCSDSProducer object
Integer skip
Integer count
StringList selected APIDs
ShortCDSTime absolute start time
ShortCDSTime absolute end time
Integer relative start time
Integer relative end time
Integer sequence counter start
Integer sequence counter end
13 Strings corresponding to each corruptible field containing keywords such as random
13 Integers corresponding to each corruptible field containing the actual corrupt values
String representing the byte keyword
Integer corrupt byte value
Integer byte increment
2 Integers to mark the start byte offset and the end byte offset
String representing the byte increment keyword
Boolean collapse flag

5.16.3 Implements

CCSDSProducer, TextSummaryProducer, FileProducer

5.16.4 Method

The CCSDSCorruptFilter reads CCSDS packets one at a time and applies corruptions to them only if they meet specific corruption constraints. If not, the packets are simply copied unaffected.

The corruption constraints allow a group of packets to be isolated by their APID, sequence counter and UTC-time. There is also a simple skip and count method of applying corruptions to a selection of packets. The code allows all the constraint options to be combined and used together to isolate groups of packets both elegantly and accurately.

The CCSDS packet common fields listed:

- Version Number
- Type Indicator
- Secondary Header Flag
- APID
- Sequence Flags
- Sequence Counter
- Packet Length
- UTC-Time
- OBT-Time
- Packet Error Control

can all be corrupted per packet in three different ways:

7. Set to a random value
8. Set to a constant value
9. Set to a number pattern (1,2,3,… or 1,3,5,,… or 0,4,8,… etc.)

These three types of corruption can also be applied to any part of a CCSDS packet by simply specifying the range and offset of bytes that one would like to corrupt. A more in depth discussion of the functionality and the nature of corruptions that can be introduced is given in section 4.11.

5.17 CCSDSCountFilter

5.17.1 Purpose

To receive a list of CCSDS packets and remove all except a range from middle of the list.

5.17.2 Constructor

CCSDSProducer object

skip integer

count integer

5.17.3 Implements

CCSDSProducer, TextSummaryProducer

5.17.4 Method

The first ‘skip’ packets from the input stream are discarded, then the following ‘count’ packets are passed through. The input stream is then terminated.

The ‘skip’ argument is optional and if omitted the remainder of the input stream is allowed through.

5.18 CCSDSDemultiplexAndSort filter

5.18.1 Purpose

To accept as input a multiplexed, unordered list of CCSDS packets and sort them by sequence number and APID.

The output consists of a separate stream for each APID present in the input stream and is sent via the FileConsumer interface, provided the output filter supports it (MultiFileWriter is currently the only filter that does).

5.18.2 Constructor

CCSDSProducer object

5.18.3 Implements

FileProducer, TextSummaryProducer

5.18.4 Method

Internally the filter holds a number of sorted lists (one per APID) to buffer the incoming packets. The filter attempts to output packets by incrementing sequence numbers, and if gaps or out of sequence packets are found they will be batched up until the packet with the correct sequence number arrives.

The CCSDS Application ID field runs from bits 5 through 15 of the CCSDS packet header and the sequence counter is stored in bits 18 through 31. These are the only parts of the packet examined by this filter.

The filter has a hard coded maximum buffer size, if this is exceeded the buffer will be flushed anyway with gaps in the sequence counts on the output.

5.19 CCSDSExtractor

5.19.1 Purpose

To read in a CCSDS packet stream and convert it to binary data.

5.19.2 Constructor

CCSDSProducer object

5.19.3 Implements

FileProducer

5.19.4 Method

The CCSDSExtractor is useful with objects that generate data in several different formats, such as a Reader filter that implements both ImageProducer and CCSDSProducer. The CCSDSExtractor reader in the source packet stream and can be connected to a Writer object to store files on disc.

5.20 CCSDSIASIFlagsFix

5.20.1 Purpose

Use to correct IASI CCSDS packet flag settings.

5.20.2 Constructor

CCSDSProducer object

5.20.3 Implements

CCSDSProducer, FileProducer, TextSummaryProducer

5.20.4 Method

The simulated IASI CCSDS packets produced by Noveltis had various flags set incorrectly. This code simply inverts the flags that were incorrectly set. A full listing of the flags that the code inverts is given in Appendix D.

5.21 CCSDSImageAvhrr

5.21.1 Purpose

To read a stream of AVHRR source packets and build a series of graphics images, one per channel plus false colour images.

5.21.2 Constructor

CCSDSProducer object

5.21.3 Implements

ImageProducer, TextSummaryProducer

5.21.4 Method

This filter derives from the same base class (AVHRRImageBase) as AVHRRReader, and uses the same functions for graphics output.

5.22 CCSDSMultiplexer

5.22.1 Purpose

To read in a number of input streams, and take one packet from each in a round-robin fashion. The packets are combined into a single output stream.

5.22.2 Constructor

No arguments.

5.22.3 Implements

CCSDSProducer, TextSummaryProducer

5.22.4 Method

After construction the filter's 'add_input()' method should be called once per input to be attached.

After all inputs have been connected the filter is ready to produce its output stream.

5.23 CCSDSReader filter

5.23.1 Purpose

Reads in binary data and converts it to one or more separate CCSDS packets.

5.23.2 Constructor

FileProducer object

5.23.3 Implements

CCSDSProducer, TextSummaryProducer

5.23.4 Method

Firstly the input stream is broken into individual CCSDS packets if more than one are found.

PACKET PRIMARY HEADER (6 octets)							PACKET DATA FIELD (max: 65536 octets)			
								SOURCE DATA		
Packet Identification				Packet Sequence Control		Packet length	Sec. hdr	User Data		Error Contrl
Version number	Type indic.	Second hdr flag	AP ID	Sequ. flags	Packet Sequ. count		UTC Time stamp	Ancill. Data	Applic. data	PEC
3 bits "000"	1 bit "0"	1 bit "x"	11 bits "xxx"	2 bits "11"	14 bits "xx"	16 bits "xx"	64 bits UTC "xx"	even nb of octets "xx"	even nb of octets "xx"	16 bits "xx"
16 bits				16 bits		16 bits	64 bits	SBT 48 bits	Variable	16 bits

The Packet Length field gives the size of the Packet Data Field section less 1. This means the total length of the packet is given by (Packet Length + 6 – 1) allowing the filter to break up concatenated packets.

5.24 CCSDSPacketDisplayer

5.24.1 Purpose

To read in one or more CCSDS packets and writes a text description of each.

5.24.2 Constructor

CCSDSProducer object

decode boolean

config_file string

iasi_config_file string

5.24.3 Implements

TextRecordProducer, HIRSBase, AMSUABase, TextConfigProducer

5.24.4 Method

The common fields of each input packet are decoded and displayed. This includes the Secondary Header field where the input is an instrument source packet, and also the Ancillary Data field when the input is an ATOVS, GRAS, IASI, ASCAT or GOME instrument source packet. These fields are output as raw hex for other types of packet.

The main Application Data section is not parsed but output as a hex dump.

The output template is :

```
CCSDS Packet number <count of packets produced in this run of the software>
        Packet Primary Header
                Packet Identification
                        Version number <format version number>
                        Type indicator <0/1>
                        Secondary header flag <0/1>
                        Application ID <APID> (<name of APID>)
                Packet Sequence Control
                        Flags <Packet sequence flags>
                        Counter <Packet sequence counter>
                        Packet length <Packet length field> (<actual full packet length>)
                Packet Data Field
                        Secondary Header (UTC time stamp)
                                Days <UTC days>
                                Seconds <UTC seconds>
                                Microseconds <UTC microseconds>
                        Source Data
                                User Data
                                        Ancillary Data (<OBT time>)
                                                OBT coarse
                                                OBT fine
```

```
                                    Application data
                                            (see below)
                          Error Control (<VPC / CRC>)

Main data payload:
        <hex dump>
```

When the input packet is from the AMSU instrument a second hex dump is shown with the filler words removed.

If the constructor flag “decode” is true and the input packet is one of the types recognised (ATOVS, IASI, GRAS, ASCAT and GOME) then instead of a hex dump, the application data will be fully decoded and shown as a list of tables.

If an IASI Spectrum CCSDS packet is expected to be decoded, an IASI specific configuration file needs to be specified (see Section 9.7 for further details).

5.25 CCSDSStreamDisplayer

5.25.1 Purpose

To count all input CCSDS packets and display some statistical information including numbers of packets with each APID and ranges of sequence counts.

5.25.2 Constructor

CCSDSProducer

5.25.3 Implements

TextRecordProducer

5.25.4 Method

The input stream is scanned and used to build up a list of sequences, where a sequence is a range of packets with the same CCSDS APID and sequential CCSDS sequence counts.

After reaching the end of the stream the list is drawn through the TextProducer functions.

A sample output would be:

```
Analysis of CCSDS packet stream:
    Received 211 packets in total
        106 had APID 39 (AMSU-A1)
            Sequence counts: 562-667
            Times:
                   01-Jan-2000,13:39:37.000000 to 01-Jan-2000,13:53:37.000000

        105 had APID 40 (AMSU-A2)
            Sequence counts: 562-666
            Times:
                   01-Jan-2000,13:39:37.000000 to 01-Jan-2000,13:53:29.000000
```

5.26 CCSDSStripFilter

5.26.1 Purpose

To remove CCSDS packets that do not satisfy specified UTC start and end times.

5.26.2 Constructor

CCSDSProducer object

5.26.3 Implements

CCSDSProducer, FileProducer, TextSummaryProducer, TextConfigProducer

5.26.4 Method

Given single or multiple CCSDS packets, this code checks to ensure that the UTC time of the packets are greater or equal to the start UTC time and less than or equal to the end UTC time defined in a configurable file similar in structure to the one shown in Section 7.9. Only CCSDS packets within the time range are kept and used to produce the output file.

The concatenated CCSDS packets produced as output are expected to be arranged in chronological order where the UTC time is expected to either remain unchanged or increase from one CCSDS packet to the next. CCSDS packets that do not satisfy this condition and hence, have an UTC time that is less than that of the previous packet are removed.

A final check is also made to ensure that the UTC time gap between consecutive CCSDS packets is not too large. If the time difference is larger than the set configurable time_gap_limit field, the CCSDS packet is not included in the output and error/warning messages are displayed on screen.

Note that multiple input CCSDS packets fed to this code must all originate from the same instrument. For example they must all be IASI CCSDS packets, or they must all be ASCAT CCSDS packets. They cannot be a combination of IASI and ASCAT packets.

5.27 CCSDSTimeAndSeqFix

5.27.1 Purpose

To modify/offset the sequence counter in the CCSDS primary header and the OBT time field of the CCSDS ancillary data packet.

5.27.2 Constructor

CCSDSProducer object

accumulate boolean

single boolean

5.27.3 Implements

CCSDSProducer, FileProducer, TextSummaryProducer, TextConfigProducer

5.27.4 Method

This code, designed to handle both single and multiple input CCSDS packets, simply replaces the value of the sequence counter of the first input CCSDS packet with the value of the configurable parameter COUNTER_INITIAL. It will also replace the OBT time fields of the ancillary data part of the first input CCSDS packet with values obtained from the configurable parameters OBT_COARSE_INITIAL and OBT_FINE_INITIAL (see Section 9.11 for further details of the configurable parameters). Consecutive CCSDS packets are updated in accordance to the equations below:

First CCSDS Packet:

```
CCSDS[0].new_sequence_counter = COUNTER_INITIAL

CCSDS[0].new_obt_coarse = OBT_COARSE_INITIAL

CCSDS[0].new_obt_fine = OBT_FINE_INITIAL
```

Consecutive CCSDS packets:

```
CCSDS[n+1].new_sequence_counter =
      COUNTER_INITIAL + CCSDS[n+1].sequence_counter – CCSDS[n].sequence_counter

CCSDS[n+1].new_obt_coarse =
      OBT_COARSE_INITIAL + CCSDS[n+1].obt_coarse – CCSDS[n].obt_coarse
```

```
CCSDS[n+1].new_obt_fine =
      OBT_FINE_INITIAL + CCSDS[n+1].obt_fine – CCSDS[n].obt_fine
```

Where n is a counter starting at the first CCSDS packet (n=0). Note that OBT coarse and OBT fine are related. Hence, when OBT fine exceeds it maximum value of 65535, it is reset and OBT coarse is updated to account for the excess. If OBT coarse exceeds its maximum value of 16777215, it is reset to start counting again from zero.

The configurable parameters COUNTER_INITIAL_1, COUNTER_INITIAL_2, …, COUNTER_INITIAL_n can be used to set the sequence counter of the second, third and n^{th}+1 CCSDS packets to specific values. Note that the definition of these configurable fields is optional. If left undefined, the sequence counter of consecutive CCSDS packets will continue to be filled in accordance to the above equation. However, should there be a need to assign specific values to more than one CCSDS packet sequence counter, the option is available.

According to the above equation, if the difference between the sequence counter of one CCSDS packet and the next is greater than one, it follows through and results in the values of the newly calculated sequence counters also being different by the same amount. By setting the “single” boolean flag, the code overrides the above equation and ensures that the sequence counter is always incremented by one.

Note that multiple input CCSDS packets fed to this code must all originate from the same instrument. For example they must all be IASI CCSDS packets, or they must all be ASCAT CCSDS packets. They cannot be a combination of IASI and ASCAT packets.

Some instruments such as ASCAT or IASI have more than one APID assigned to them (see Section 5.15.4). In some cases (for example ASCAT), the sequence counter is incremented irrespective of the different APIDS. In other cases (for example IASI), the sequence counter is incremented only if another CCSDS packet with the same APID is detected. In fact, with the exception of ASCAT, all other instruments with multiple APIDs assigned to them have their sequence counter incremented only if another CCSDS packet with the same APID is detected. The “accumulate” boolean flag can be used to treat all instruments like the ASCAT case and to increment the sequence counter irrespective of the different APIDs.

If some of the CCSDS packet terminology used is unclear, further details can be found in Sections 5.23 and 5.24 of this document and in RD10.

5.28 CCSDSTimeFix

5.28.1 Purpose

To set/modify CCSDS packet UTC times.

5.28.2 Constructor

CCSDSProducer object

accumulate boolean

5.28.3 Implements

CCSDSProducer, FileProducer, TextSummaryProducer, TextConfigProducer

5.28.4 Method

This code, designed to handle both single and multiple input CCSDS packets, can be used to replaces the CCSDS UTC time tags with ones specified by the user. As described in Section 9.10, the UTC time of the first input CCDS packet (UTC_start) can be set using the configurable fields: utc_days_start, utc_milliseconds_start and utc_microseconds_start.

As shown by the equation below, the UTC time of consecutive input CCSDS packets are determined by adding a constant amount of time (K) to the user specified start time of the first input CCSDS packet (UTC_start).

```
CCSDS[0].UTC = UTC_start
CCSDS[n].UTC = UTC_start + nK
```

Where n represents the number of CCSDS packets and n=0 represents the first CCSDS packet. The value of the constant time increment, K, is user configurable and can be set by assigning values to the configurable fields: utc_days_increment, utc_milliseconds_increment and utc_microseconds_increment.

Some instruments such as ASCAT or IASI have more than one APID assigned to them (see Section 5.15.4). The code monitors the change in APIDs and only increments the UTC time if another CCSDS packet with the same APID is detected. The "--accumulate" option can be used to ignore the APID differences of the same instrument and thus, increment the UTC time irrespective of the different APIDs. Note that multiple input CCSDS packets fed to this code must all originate from the same instrument. For example they must all be IASI CCSDS packets, or they must all be ASCAT CCSDS packets. They cannot be a combination of IASI and ASCAT packets.

If some of the CCSDS packet terminology used is unclear, further details can be found in Sections 5.23 and 5.24 of this document and in RD10.

5.29 CCSDSToL0

5.29.1 Purpose

Receives a stream of CCSDS packets together with optional parameters for performing OBT->UTC conversion, and generates an EPS MetOp L0 product.

5.29.2 Constructor

Config filename string, Subsat filename string, OSVSs filename string, UTC_OBT time string, OBT_UTC time string, clock drift string, VIADR filename string.

5.29.3 Implements

EPSProductProducer, TextSummaryProducer.

5.29.4 Method

All formatting of the product structure is handled by the EPSProductExtractor filter and the MPHR class, except for MDR and viadr-l0-obt2utc formatting which is handled by the make_mdr() and make_viadr() functions.

The OBT->UTC equation is performed by the obt_to_utc() function and used to generate the times for the MDR header.

See description for the ccsds_to_l0 executable for details of the processing.

5.30 CVCDUToVCDU filter

5.30.1 Purpose

To receive a stream of Coded VCDU (CVCDU) packets and verify the checksum block matches the data section, then output the VCDU packets.

5.30.2 Constructor

CVCDUProducer object

5.30.3 Implements

VCDUProducer

5.30.4 Method

A Coded VCDU has the following layout:

Payload	Checksum
892 bytes data	128 byte Reed-Solomon encoding

The payload is a complete VCDU packet.

There are three possibilities for dealing with the checksum block:

☐Ignore its contents and simply strip it out.

☐The filter can produce its own coding block using the Reed-Solomon encoding algorithm and check for an exact match against the supplied block. This will give a pass/fail test for the packet.

☐Perform a full Reed-Solomon decoding operation. This will allow the filter to decide it any data has been corrupted and also correct a certain number of errors in the data field.

Currently only the first option has been implemented.

In all cases the output is the 892 byte payload, or one VCDU packet.

5.31 EPSProductExtractor

5.31.1 Purpose

Read in records generated by a EPSProductProducer source and assemble them into an EPS product.

5.31.2 Constructor

EPSProductProducer object.

5.31.3 Implements

FileProducer.

5.31.4 Method

The source may implement any of the functions in EPSProductProducer depending on the type of records it generates.

It is up to the source object to create IPRs correctly.

The source does not have to produce an MPHR, but the output will be incorrect if this is not done.

No processing if performed by EPSProduceExtractor, its only role is to call the source object functions as needed.

5.32 ImageWriter

5.32.1 Purpose

To use an ImageProducer object to write a series of graphics images to separate files..

5.32.2 Constructor

ImageProducer object

Image::Type parameter

range integer (not implemented)

5.32.3 Implements

FileProducer

5.32.4 Method

The ImageWriter reads a series of images from an object that implements the ImageProducer interface and stores them to disk using an Image object. The read uses the get_image_count() and create_image() methods of the source object to read in all the images it can generate.

The Image::Type parameter is passed to the source object and potentially controls the filetype of the images produced. (TIFF and JPEG formats are implemented).

The filter uses the base class Image and its derived classes TiffImage and JpegImage to perform the actual image formatting.

5.33 HIRSReader

5.33.1 Purpose

To read in a NOAA HIRS Level 1b file and convert it to a stream of MetOp format CCSDS packets.

In addition this filter implements the TextDetailsProducer which outputs the complete input product in a readable text format. This can be used to check the outputs by comparison against the TextDetailsProducer output from the CCSDSReader filter.

5.33.2 Constructor

ProductProducer object

(optional) filename of configuration file

5.33.3 Implements

CCSDSProducer, TextSummaryProducer, ImageProducer

5.33.4 Method

A NOAA HIRS L1b file consists of a header structure followed by a number of data structures, one per scan line. Due to the similarities between the data stored in the NOAA HIRS product and the MetOp HIRS CCSDS format (since both come from identical instruments) it is possible to create complete CCSDS packets with all fields filled in according to real data.

See Appendix A for a table showing the exact structure of the CCSDS packets produced.

This filter also implements the TextSummaryProducer interface to give an ASCII description of the input product.

The optional configuration file allows user defined parameters to be read in to modify certain details of the output packets. In the case of HIRS the only settings are the initial CCSDS sequence number and an offset to be applied to the UTC and OBT time fields.

5.34 MHSReader

5.34.1 Purpose

To read in a NOAA HMHS Level 1b file and convert it to a stream of MetOp format CCSDS packets.

In addition this filter implements the TextDetailsProducer which outputs the complete input product in a readable text format. This can be used to check the outputs by comparison against the TextDetailsProducer output from the CCSDSReader filter.

5.34.2 Constructor

ProductProducer object

(optional) filename of configuration file

5.34.3 Implements

CCSDSProducer, TextSummaryProducer, ImageProducer

5.34.4 Method

A NOAA MHS L1b file consists of a header structure followed by a number of data structures, one per scan line. Due to the similarities between the data stored in the NOAA MHS product and the MetOp MHS CCSDS format (since both come from identical instruments) it is possible to create complete CCSDS packets with all fields filled in according to real data.

This filter also implements the TextSummaryProducer interface to give an ASCII description of the input product.

The optional configuration file allows user defined parameters to be read in to modify certain details of the output packets. In the case of MHS the only settings are the initial CCSDS sequence number and an offset to be applied to the UTC and OBT time fields.

5.35 MPDUDisplayer

5.35.1 Purpose

Write a brief summary of each MPDU packet in a stream.

5.35.2 Constructor

MPDUProducer object

5.35.3 Implements

TextRecordProducer

5.35.4 Method

This filter writes the First Header Pointer value and a calculated checksum to the output stream for each packet in the input stream.

5.36 MultiFileWriter

5.36.1 Purpose

To write a number of objects to files.

5.36.2 Constructor

FileProducer object

String “filename_prefix”

5.36.3 Implements

FileConsumer

5.36.4 Method

The filter reads in each input object or packet and writes it to a unique file, with one object in each separate file. The parameter “filename_prefix” gives the base for the filenames and an incrementing count is added to each one to form the complete name.

The implementation of MultiFileWriter calls the function “produce_file()” on its source object. The source can then call “get_named_output_stream()” back on the MultiFileWriter once per file it produces.

If the source does not implement “produce_files()” the MultiFileWriter will call the function “get_next_file()” on the source object instead, and stream the results into a single output file.

5.37 NOAAProductReader

5.37.1 Purpose

To read in NOAA Level 1b product files as binary inputs, test they have the correct header sections and remove the NOAA archive header block if present. The outputs have the data type of NOAA product files.

5.37.2 Constructor

FileProducer object

Optional block size parameter

5.37.3 Implements

ProductProducer

5.37.4 Method

NOAA L1b products have a header block followed by a series of data blocks, each one the same size. The filter needs to know the block size before it can read a file.

Products from the Satellite Active Archive may have a optional 512 byte long archive header attached to the front of the file. If present the filter will read the block size (and also the bits per sample for AVHRR products). Otherwise the filter can guess the block size from the data type from the header block of the product.

For AVHRR products the block size there are several formats available. The data can be in LAC or GAC resolution mode, stored in 8, 10 or 16 bits per pixel format, and for 8 and 16 bps products there can be between 1 and 5 instrument channels present. If the archive header is not present, the block size cannot be determined accurately just from the file contents so the filter will assume 10bps data unless the optional block size parameter is used.

5.38 ProductExtractor

5.38.1 Purpose

To read a stream of data through the ProductProducer interface and change its type to the FileProducer interface.

5.38.2 Constructor

ProductProducer object

5.38.3 Implements

FileProducer

5.38.4 Method

This filter is useful for storing product data streams on disk as product files.

5.39 ReducedCCSDSWriter

5.39.1 Purpose

Storage of CCSDS packets in a format used by the MHS SD Analysis software.

5.39.2 Constructor

CCSDSProducer object

String directory_name

5.39.3 Implements

TextSummaryProducer

5.39.4 Method

The output files are placed into directories with the naming convention YYMMDDhh.dd (where .dd is a fixed string) inside directory_name. Each source packet first has the UTC time removed, shrinking its size by 8 bytes, and is stored in a file with name mmcc.sd where cc is a counter within each minute, and .sd is a fixed string.

This filter should only be used with MHS packets.

5.40 tVCDUCorruptFilter

5.40.1 Purpose

To enable the modification and corruption of tVCDU packets.

5.40.2 Constructor

tVCDUProducer object

Integer skip

Integer count

deque<integer> list of selected VCIDs

ShortCDSTime absolute start time

ShortCDSTime absolute end time

Integer relative start time

Integer relative end time

Integer VCDU counter start

Integer VCDU counter end

15 structures of type INFO specifying the exact nature of corruptions to be implemented.

2 Integers to mark the start byte offset and the end byte offset

String representing the byte increment keyword

5.40.3 Implements

tVCDUProducer, TextSummaryProducer, FileProducer

5.40.4 Method

The tVCDUCorruptFilter reads tVCDU packets one at a time and applies corruptions to them only if they meet specific corruption constraints. If not, the packets are simply copied unaffected.

The corruption constraints allow a group of packets to be isolated by their VCID, VCDU counter and UTC-time. There is also a simple skip and count method of applying corruptions to a selection of packets. The code allows all the constraint options to be combined and used together to isolate groups of packets both elegantly and accurately.

The tVCDU packet fields listed:

- Version Number
- Spacecraft-ID (SCID)
- Virtual Channel-ID (VCID)
- VCDU Counter
- Replay Flag
- Spare
- Insert Zone - Encrypt Flag
- Insert Zone - Encrypt Key
- M-PDU packet – Spare
- M-PDU packet – Header
- UTC-Time Days, Milliseconds, Microseconds
- Reed Solomon Status

can all be corrupted per packet in three different ways:

1. Set to a random value
2. Set to a constant value
3. Set to a number pattern (1,2,3,… or 1,3,5,,… or 0,4,8,… etc.)

These three types of corruption can also be applied to any part of a tVCDU packet by simply specifying the range and offset of bytes that one would like to corrupt. A more in depth discussion of the functionality and the nature of corruptions that can be introduced is given in section 4.25.

5.41 tVCDUDisplayer

5.41.1 Purpose

Write a brief summary of each tVCDU packet in a stream.

5.41.2 Constructor

tVCDUProducer object

5.41.3 Implements

TextRecordProducer

5.41.4 Method

This filter writes the following fields from each input tVCDU packet to the output stream:

Version number

SCID

VCID

VCDU counter

Reed Solomon Status

(MPDU) First Header Pointer

(MPDU) Calculated checksum over payload section

5.42 VCDUFilter filter

5.42.1 Purpose

To remove VCDU packets whose virtual channel ID does not match a constructor argument.

5.42.2 Constructor

VCDUProducer object

VCID number

5.42.3 Implements

VCDUProducer

5.42.4 Method

Any incoming packets whose VCID field is not a match to the constructor argument is remove from the input stream.

This filter is intended mainly as a performance optimisation, to be used with the CCSDSFilter object. A program that is untimately looking for specific CCSDS Application IDs can filter by VCID first, allowing the majority of packets in a low level stream to be eliminated without having to process them up to CCSDS level.

5.43 VCDUToCCSDS filter

5.43.1 Purpose

To convert VCDU packets to CCSDS packets.

5.43.2 Constructor

VCDUProducer object

(optional) Strip fill packet flag, default=true

5.43.3 Implements

CCSDSProducer

5.43.4 Method

The satellite downlink transmits a stream of CCSDS packets to ground. These source packets are generated by several sources on board and prior to transmission, are multiplexed into virtual channels. Each virtual channel is a byte stream containing a series of CCSDS packets concatenated together. A single virtual channel can hold packets of several different types.

Each virtual channel is split into 882 byte chunks, and each chunk is given a header section to form an M-PDU packet. A second header is added around that to form a VCDU packet, and all virtual channels are multiplexed together to form a single stream of packets. This filter operates on the stream of VCDU packets.

An extra complication is that both CCSDS packets and VCDU packets may be transmitted out of sequence. Therefore the decoder software must use the sequence counter variable present in both packet types and perform a reordering if needed. The reordering of VCDU packets is handled by this filter and the reordering of CCSDS packets is handled by the CCSDSDemultiplexAndSort filter.

The VCDU packets are 892 bytes long and defined as follows:

Version number	Space Craft ID	Virtual Channel ID	VCDU counter	Replay flag	Spare	Encrypt flag	Encrypt key	Payload
2 bits	8 bits	6 bits	24 bits	1 bit	7 bits	8 bits	8 bits	884 bytes
1 (v2	See	See below	Increm-	0 for real	Zero fill	0x00 (no	0x00 (not	M-PDU

packet format)	below		enting counter unique to each VCID.	time data		encrypt)	used)	packet.

This filter cannot read encrypted data.

The values for the space craft ID (SCID) can be one of:

METOP 1	11
METOP 2	12
METOP 3	13
METOP simulator	14

(this filter will accept packets with any SCID)

The Virtual Channel ID (VCID) numbers are used as follows:

3	AMSU-A1 AMSU-A2 HIRS/4 SEM
5	AVHRR Low Rate
9	AVHRR High Rate
10	IASI
12	MHS
15	ASCAT
24	GOME-2
27	A-DCS
29	GRAS
34	GRAS NAV Satellite packet Admin packet
63	Fill VCDU

The actual values of the Virtual Channel numbers have no significance to the MetOpizer (except 63 Fill VCDU, which are stripped out) and there is no check that CCSDS packets are being sent down their assigned channels.

Each VCID has its own incrementing counter that resets after 2^24 counts. The filter maintains its own buffers for each VCID and reorders packets if needed. The Virtual Channel counters must be sequential, since if any sequence numbers are skipped this indicates a packet lost in transmission.

The structure of an M-PDU packet within the VCDU payload is:

Spare	First Header Pointer	Packet zone
5 bits	11 bits	882 bytes
"00000" if a CCSDS header is present in packet zone. "11111" if no header is in packet zone.	Index into the packet zone where the first CCSDS header begins.	Contains CCSDS packets or fragments of packets.

CCSDS packets can be very short with multiple packets per M-PDU, or long and spanned over a number of M-PDU's. There are no gaps or synchronisation marks between individual CCSDS packets, and their length has to be calculated from the CCSDS header.

When there are no CCSDS headers in the M-PDU the "First Header Pointer" and "Spare" fields are set to all ones.

If the M-PDU comes from a Fill VCDU the "First Header Pointer" and the ""Packet Zone" are set to all zeros.

The output from this filter is a single stream of multiplexed CCSDS source packets, in the same order as found in the inputs.

The optional constructor argument can be used to remove fill VCDU packets from the input stream. This action is carried out by default.

5.44 VCDUTotVCDU

5.44.1 Purpose

Read in VCDU packets and translate them into tVCDU packets. Each input packet is translated into exactly one output packet.

5.44.2 Constructor

VCDUProducer object

5.44.3 Implements

tVCDUProducer

5.44.4 Method

The tVCDU packet structure is shown below.

Version number	Space Craft ID	Virtual Channel ID	VCDU counter	Replay flag	Spare	Insert zone	Payload	UTC time stamp	Reed Solomon Status
2 bits	8 bits	6 bits	24 bits	1 bit	7 bits	2 bytes	884 bytes	8 bytes	8 bits
1 (v2 packet format)			Increm-enting counter unique to each VCID.	0 for real time data	Zero fill	Not used, zero fill	M-PDU packet	Long CCSDS Day Segmented (CDS)	See below.

This filter cannot read encrypted data.

The Version Number, Spacecraft ID, Virtual Channel ID, VCDU Counter, Replay flag and Spare fields are copied from the input VCDU packet. The Insert Zone field is set to zero.

The Reed Solomon bitfield is defined as follows:

Name	Length (bits)	Description	Default value
Status of SCLF state machine	2	00 – Search mode 01 – Check mode 10 – Lock mode	10

		11 – Flywheel mode	
Best match	1	0 – No 1 – Yes	0
Flush frame	1	0 – No 1 – Yes	0
Slip frame	1	0 – No 1 – Yes	0
Unroutable frame	1	0 – No 1 – Yes	0
Uncorrect frame	1	0 – No 1 – Yes	0
R-5 (255,233) found errors in frame	1	0 – No 1 – Yes	0

The default values can all be overridden by values from the tVCDU configuration file.

The UTC Time Stamp is stored in the same format as the CCSDS packet time stamp (2 bytes days since 1 Jan 2000, 4 bytes milliseconds, 2 bytes microseconds). For the first packet output this is set to the time stamp of the first CCSDS packet inside the M-PDU payloads, modified by an offset value. This offset is 110 minutes by default but can be changed using the UTCOffsetMilliseconds value in the tVCDU configuration file.

Subsequent tVCDU packets have a time stamps that increment by a fixed value each packet. By default the offset is 80 microseconds which can be changed using the UTCTimeDifference value from the configuration file.

5.45 SingleFileWriter

5.45.1 Purpose

To read in one or more input objects and write them to a single file.

5.45.2 Constructor

FileProducer object

String “filename”

5.45.3 Implements

None

5.45.4 Method

The filter concatenates all the input objects together with no gaps in between, opens a file with the given name and writes to it.

5.46 WriteToStdout

5.46.1 Purpose

To read in one or more input objects and display them on screen.

5.46.2 Constructor

FileProducer object

5.46.3 Implements

None

5.46.4 Method

The filter concatenates all the input objects together with no gaps in between, and copies them to standard output.

6 INTERFACES

The following section lists the interfaces used to link MetOpizer executables together. In all cases they are used to pass a variable number of objects from a single producer object to one or more consumer objects.

All transfers are 'pull' operations initiated by the consumer filter, except where the FileConsumer interface is used.

Where a function name begins with 'get_' and returns a pointer, the producer object allocates the memory and also frees it, usually on the next call to the same function. Where the function name begins with 'create_' and returns a pointer, the producer also allocates the memory but the client must free it.

6.1 BufferProducer

```
class BufferProducer
{
   virtual BUFFER* get_next_buffer(void) =0;
};
```

6.2 CADUProducer

```
class CADUProducer
{
   virtual CADU* get_next_cadu(void) =0;
};
```

6.3 CCSDSProducer

```
class CCSDSProducer
{
   virtual CCSDS* get_next_ccsds(void) =0;
};
```

6.4 CVCDUProducer

```
class CVCDUProducer
{
   virtual CVCDU* get_next_cvcdu(void) =0;
};
```

6.5 EPSProductProducer

```
class EPSProductProducer
{
public:
    virtual int get_ipr_count(void) const=0;
    virtual int get_sphr_size(void) const=0;
    virtual bool make_mphr(byte*& record, int& length)=0;
    virtual bool make_sphr(byte*& record, int& length)=0;
    virtual bool make_ipr(byte*& record, int& length)=0;
    virtual bool make_geadr(byte*& record, int& length)=0;
    virtual bool make_giadr(byte*& record, int& length)=0;
    virtual bool make_veadr(byte*& record, int& length)=0;
    virtual bool make_viadr(byte*& record, int& length)=0;
    virtual bool make_mdr(byte*& record, int& length)=0;
};
```

6.6 FileConsumer

```
class FileConsumer
{
  virtual ostream* get_output_stream(void) =0;
  virtual ostream* get_named_output_stream(const string&
     filename_suffix) =0;
  virtual string get_filename_base(void) =0;
};
```

The FileConsumer interface allows an object to receive multiple file streams. It is only used in conjunction with the FileProducer interface.

6.7 FileProducer

```
class FileProducer
{
  virtual istream* get_next_file(void) =0;
  virtual bool produce_files( FileConsumer& ) =0;
};
```

An object implementing FileProducer must implement either one or both its functions (the other can be stubbed). A client calling a FileProducer can use either produce_files() to request multiple named streams of data to be produced, or get_next_file() to request a flat series of files.

6.8 ImageProducer

```
class ImageProducer
{
  virtual void prepare_images(void) =0;
  virtual int get_image_count(void) =0;
  virtual Image* create_image(Image::Type type, int number) =0;
};
```

A client to ImageProducer always calls prepare_images() first to allow the source to carry out and initial processing, followed by get_image_count() and create_image() to request the number of images available and actual images.

6.9 MPDUProducer

```
class VCDUProducer
{
  virtual VCDU* get_next_vcdu(void) =0;
};
```

6.10 ProductProducer

```
class ProductProducer
{
    virtual NOAA_Archive_Header* get_archive_header(void) =0;
    virtual NOAA_L1b_GEN_Header* get_product_header(void) =0;
    virtual NOAA_L1b_GEN_Data* get_current_record(void) const=0;
    virtual void skip_records(unsigned int offset) =0;
    virtual int get_record_size(void) const=0;
    virtual void output_text_archive_header(ostream&) =0;
    virtual void output_text_gen_header(ostream&) =0;
    virtual void output_text_gen_record(ostream&,int) =0;
};
```

The client object navigates through an input product by calling skip_records() to advance through the file (random access is not allowed, as the source could be a pipe) and get_current_record() to test for end-of-file and to read data records.

The pointer returned by get_current_record() becomes invalid when skip_records() is next called.

6.11 VCDUProducer

```
class VCDUProducer
{
  virtual VCDU* get_next_vcdu(void) =0;
};
```

6.12 tVCDUProducer

```
class tVCDUProducer
{
  virtual tVCDU* get_next_tvcdu(void) =0;
};
```

6.13 TextSummaryProducer

```
class TextSummaryProducer
{
  virtual void output_text_summary( ostream& ) =0;
};
```

6.14 TextHeaderProducer

```
class TextHeaderProducer
{
  virtual void output_text_header( ostream& ) =0;
};
```

6.15 TextRecordProducer

```
class TextRecordProducer
{
  virtual void output_text_record( ostream&, int record_number,
    bool show_header ) =0;
};
```

6.16 TextConfigProducer

```
class TextConfigProducer
```

```
{
  virtual void output_text_config( ostream& ) =0;
};
```

7 METOPIZER PYTHON EXTENSION

The Metopizer tools mainly consist of C++ binary modules and shell scripts. In recent years Python has become more popular for quickly creating shell-like programs or complete GUIs, both of which Python supports. Python itself is a highly object orientated scripting language that is purely based on tabbed textual scripts. The Metopizer Python extension module is a shared object visible to the Python language which taps into the Metopizer product reader classes to extract data from ISPs. This chapter will introduce the Metopizer Python extension and give a brief look into how the underlying C++ code is used by the C exported functions used by Python.

7.1 Python C extensions

Python itself consists of C extension modules that provide the back bone for the user scripts when they are parsed by the scripting engine. The Metopizer Python module here on known as PyMetopizer is a further extension to this mechanism. PyMetopizer consists of a shared object that exports various C functions all made visible to the python interpreter. These functions are the entry point for the Python framework when using PyMetopizer features.

7.1.1 Python extensions

The Python support is initially provided through a C style interface consisting of exported function calls from a shared object. The PyMetopizer module consists of two primary classes that form the entry point for the C style interface required by Python.

The Python extension only requires a single entry point function consisting of the following C style function:

```
extern "C" PyMODINIT_FUNC initmetopizer(void)
```

The name must always start with “init” followed by the import module name which in this case is “metopizer”, therefore the Python script only needs to add the line “import metopizer” and the Python interpreter will load the shared object into memory and call this method. The purpose of this function is to register all global functions within this module scope and all Python objects that can be created within this scope.

For example to create a Product the syntax within python would be:

```
import metopizer
product = metopizer.Product("myfile.ccsds")
samples = product.get_samples(0)
```

The above Python code snippet will use the APID of the first ISP of the file to determine the instrument type to be read, and return all samples that correspond the primary APID extracting sample data for channel 0. The returned data structure is a list with one item per ISP, where each item is a list of integers with one item per sample. An optional parameter "modulus" can also be given to perform sub-sampling on the returned values.

7.1.1.1 Internal Global Structures

Global functions within the "metopizer" Python scope

```
static PyMethodDef MyMetopizerMethods[] = {
  {"version", pymetopizer_version, METH_NOARGS,
   "Return PyMetopzer version"},
  {"set_time_type", pymetopizer_set_time_type, METH_VARARGS,
   "Set time format to return for individual products"},
  { NULL, NULL, 0, NULL }
};
```

The global Metopizer functions are called as follows:

Function	Purpose
`metopizer.version()`	Return metopizer version
`metopizer.set_time_type()`	Set time type used to determine type of values returned when asking for time values, parameter value: "gentime|matlab" as quoted string. Gentime time objects are 18 char strings formatted as Long CDS Time defined in the EPS GPFS, or YYYYMMDDhhmmssmmmZ. Matlab time objects are Python floats where the integer part gives days since 01/01/0000 and the fractional part gives fractions of a day. This is the same format used by the MatPlotLib (RD.18) plot_date() function.

Global functions within the “PyProduct” Python object scope abbreviated:

```
static PyMethodDef PyProductMethods[] = {
   {"name", …},
   {"count_filter", …},
   {"set_time_type", …},
   {"get_channels", …},
   {"get_samples", …},
   {"get_samples_count", …},
   {"get_packet_count", …},
   {"get_obt_times", …},
   {"get_utc_times", …},
   {"get_grh_times", …},
   {"get_apid_filter", …},
   {"get_instrument_name", …},
   {NULL}
};
```

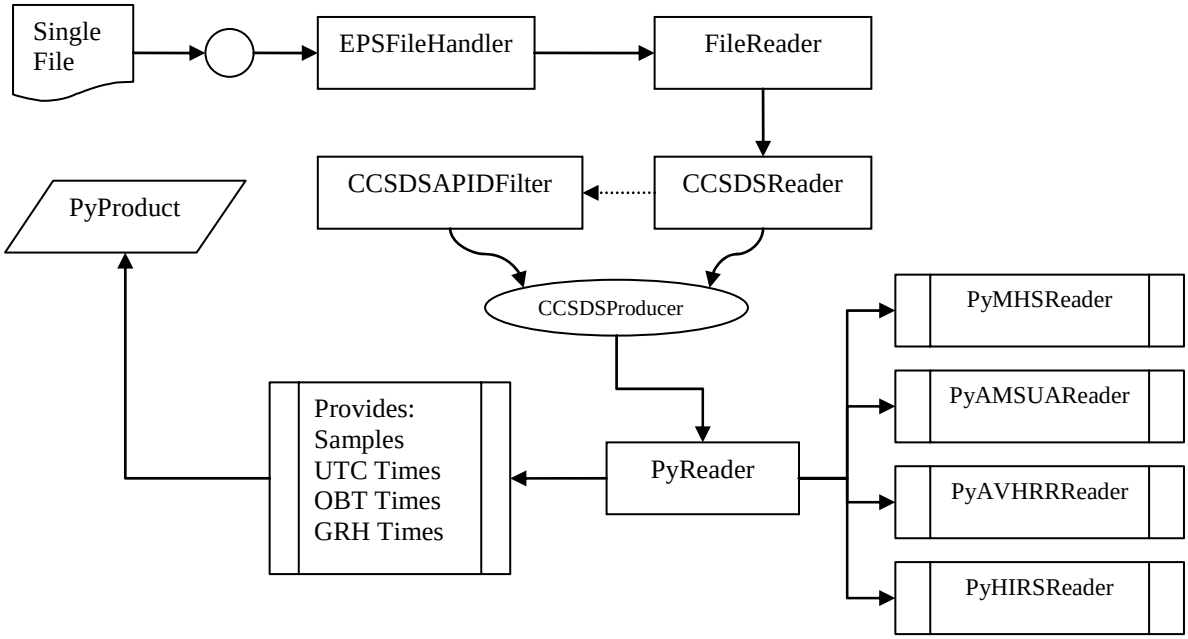
Metopizer Product functions are described as follows:

Function	Parameters	Purpose
`name()`	N/A	Returns internal class name
`count_filter()`	count skip modulus	All three parameters are integers that can be passed using keywords, i.e. count=1. count=number of packets to read, skip=number of packets to skip from start, modulus=sub sampling of packets.
`set_time_type()`	timetype	Set time type to control which type of time values are returned, timetype=gentime|matlab as quoted string
`get_channels()`	N/A	Return list of available channels as list of strings that can be used for get_samples()
`get_samples()`	channel modulus	Return a list of samples per scan line for given channel number and optionally sub sampling of a scan line using modulus parameter. The ‘channel’ parameter can either be an integer or a channel name.

`get_samples_count()`	N/A	Return number of available samples per scan line.
`get_packet_count()`	N/A	Return actual number of packets processed after making a get_samples() call
`get_obt_times()`	N/A	Return all pre-cached OBT times either as strings or doubles depending on set ***timetype***
`get_utc_times()`	N/A	Return all pre-cached UTC times either as strings or doubles
`get_grh_times()`	N/A	Return all pre-cached GRH times either as strings or doubles, however these values are only present when reading L0 products and not CCSDS streams.
`get_apid_filter()`	N/A	Return string containing the filter being applied to this reader if one was given when product was created.
`get_instrument_name()`	N/A	Return string containing acronym name of instrument being read.

The “Product” python object contains extensive methods to return samples and times data as necessary. Once the Product is no longer required the ***del*** python command should be used to release the product from memory.

7.2 Python Extension classes


Single File
EPSFileHandler
FileReader
CCSDSAPIDFilter
CCSDSReader
PyProduct
CCSDSProducer
PyMHSReader
PyAMSUAReader
Provides:
Samples
UTC Times
OBT Times
GRH Times
PyReader
PyAVHRRReader
PyHIRSReader

7.2.1 EPSFileHandler

The EPSFileHandler is a singleton class that has the responsibility of creating a PyReader corresponding to the given file type and instrument:

```
class EPSFileHandler
{
public:
  PyReader* open_file( const string& filename,
                       const string& apid_filter );
  void set_type_type(PyReader::TIME_TYPE atime_type);
private:
  PyReader::TIME_TYPE time_type;
};
```

The PyReader class is given ownership of the CCSDSProducer interface which is used to read CCSDS packets. The EPSFileHandler function open_file()creates the required PyReader class for the given product type and filter. The set_type_type() method is used to set a global time type that is applied to all new products.

7.2.2 PyReader

This abstract base class is used to encapsulate reading of CCSDS packets to extract both samples data and time values. Individual derived versions are used to handle extracting of instrument specific CCSDS packets. The PyReader class provides pure virtual methods that are required by derived variants to specialise this functionality as well as override default behaviour.

7.2.3 PyProduct

A PyProduct is a simple struct consisting of the following:

```
struct PyProduct
{
  PyReader* reader;
  string filename;
};
```

The PyProduct struct is created transparently by the Python framework using predefined calls from within the PyMetopizer extension.

7.3 Python Examples

The Metopizer extensions are only usable from within Python scripts therefore python must be installed on the local system before the Python extensions can be built.

A typical Python script that uses Metopizer would consist of the following initial lines:

```
import metopizer
metopizer.set_time_time("gentime")
product = metopizer.Product("<product file name>")
```

where <product file name> is file name of product to open either just file name or with relative or complete absolute path.

The product reference can then be used to obtain sample data as follows:

```
product.count_filter(count=100, skip=2, modulus=2)
product.set_time_type( timetype="gentime" )
```

The first line count_filter() sets the product to read exactly 100 CCSDS packets, to skip 2 CCSDS packets before starting the read process and to sub sample using modulus value of 2. The modulus value specified here is applied to CCSDS packets, i.e "y" or "vertical" direction sub sampling. The optional set_time_type() method can be called to define how time values will be read later on.

Now its possible to read the samples as follows:

```
samples = product.get_samples( channel=0, modulus=2 )
utc_times = product.get_utc_times()
```

The samples value returned is a list within a list, each list item is a list of sample values per scan line. The *channel* parameter indicates that channel 0 data is to be extracted. In this example a total of 100 samples will be extracted after skipping the first 2 packets. The optional *modulus* parameter is used to sub sample in the "x" or "horizontal" direction. The *utc_times* value is a list of strings in this example since the time type was set to "gentime".

8 APPENDIX A - NOAA LEVEL 1B TO CCSDS PACKET MAPPING

This section lists the fields of each ATOVS instruments CCSDS packet along with the source of each field.

8.1 AMSU-A1

The instrument can run in two basic modes - Measurement and Unique mode. The differences are:

Measurement mode
- Produces 1240 bytes per scan.
- Each scan consists of 30 Scene, 1 Cold Calibration and 1 Warm Calibration samples.
- Temperature sensor and housekeeping data also included.

Unique mode
- Produces 1120 bytes per scan.
- There are three types of Unique mode:
  - Warm Calibration
    - Each scan contains a Warm Calibration sample, repeated 30 times
  - Cold Calibration
    - Each scan contains a Cold Calibration sample, repeated 30 times
  - Nadir Position
    - The instrument sends a Scene measurement at position 15 (nadir), repeated 30 times
- Temperature sensor and housekeeping data also included.

The current mode for each scan is given in the AMSU-A Level1b Data record (Digital Housekeeping, octet 901, bits 1-4).
In all cases the CCSDS packet has a 2048 byte Application Data field which contains 1240 or 1120 bytes of data interleaved with filler (0x0001) 16-bit words.

The code 0x0001 will not appear in any valid data areas.

For the purposes of simulating this data stream the actual science data will always be placed at the end of the Application Data section with the filler codes appearing at the start.

Output Section (Measurement Mode) CCSDS Packet (2070 bytes)	**Input Section (AMSU-A Level 1b product)**
Packet Primary Header (6 bytes)	
Version number (3 bits)	0
Type indicator (1 bit)	0
Secondary Header Present (1 bit)	1
Application ID (11 bits)	39
Packet Sequence Flags (2 bits)	3
Packet Sequence Counter (14 bits)	Incrementing counter unique to each APID.
Packet Length (16 bits) (Size of Packet Data Field less 1)	2063
Packet Data Field (2064 bytes)	
Secondary Header (8 bytes) (Satellite UTC time)	
UTC Days since 1/1/2000 (2 bytes)	L1b Data octets 3-4 (year) and 5-6 (day of year)
UTC Milliseconds of day (4 bytes)	L1b Data octets 9-12
UTC Microseconds of millisecond (2 bytes)	Times above or below to be offset by Satellite Clock Drift Delta (octets 7-8) depending on whether the Clock Drift Correction has already been applied (octets 13-14 bit 14)
Ancillary Data, SBT time 6 bytes)	
Zero fill (1 byte)	Onboard time synthesised in software.
Seconds (3 bytes)	
Seconds / 65536 (2 bytes)	
Application Data (2048 bytes)	
Filler Section (808 bytes)	Set to 0x0001 repeating.
Science Data (1240 bytes)	
Sync Sequence (3 bytes)	Synchronisation Sequence (FF hex) (L1b octets 897-899)
Unit Identification and Serial Number (1 byte)	103 (Config file “Serial-Number”)
Digital Housekeeping Data (4 bytes)	Digital A Housekeeping
Unused (1 bit)	(bytes 0-2 copied from L1b octets 901-903)
Full Scan Mode (1 bit)	Full Scan Mode
Warm Cal. Mode (1 bit)	Warm Cal. Mode
Cold Cal. Mode (1 bit)	Cold Cal. Mode
Nadir Mode (1 bit)	Nadir Mode
Cold Cal. Position (2 bits)	Cold Cal. Position
Unused (2 bits)	0
Scanner AL1-1 Power (1 bit)	Scanner AL1-1 Power
Scanner AL1-2 Power (1 bit)	Scanner AL1-2 Power
PLL Power (1 bit)	PLL Power
Survival Heater Power (1bit)	Survival Heater Power
Unused (11 bits)	0
AMSU-A1 ID number (8 bits)	9 (Config file “AMSU-A1-ID”)
Position data (30 pixels * 34 bytes = 1020 bytes)	Scene Telemetry

Reflector 1, First reading (2 bytes) Reflector 2, First reading (2 bytes) Reflector 1, Second reading (2 bytes) Reflector 2, Second reading (2 bytes) Scene Radiometer Data (13 channels 3 thru 15 * 2 bytes = 26 bytes)	Binary copy (L1b octets 905-1924)
Cold Calibration (60 bytes) Reflector 1, Cold Cal. Position, First reading (2 bytes) Reflector 2, Cold Cal. Position, First reading (2 bytes) Reflector 1, Cold Cal. Position, Second reading (2 bytes) Reflector 2, Cold Cal. Position, Second reading (2 bytes) Cold Cal. 1 Radiometer data (13 channels 3 thru 15 * 2 bytes) = (26 bytes) Cold Cal. 2 Radiometer data (13 channels 3 thru 15 * 2 bytes) = (26 bytes)	Cold Calibration Telemetry Binary copy (L1b octets 1925-1984)
Temperature sensors (45 channels * 2 bytes) = (90 bytes)	Temperature Sensor Telemetry Binary copy (L1b octets 1985-2074)
Temperature Sensor Reference Voltage (2 bytes)	Reference voltage (L1b octets 2075-2076)
Warm Calibration (60 bytes) Reflector 1, Warm Cal. Position First Reading (2 bytes) Reflector 2, Warm Cal. Position First Reading (2 bytes) Reflector 1, Warm Cal. Position Second Reading (2 bytes) Reflector 2, Warm Cal. Position Second Reading (2 bytes) Warm Cal. 1 Radiometer data (13 channels 3 thru 15 * 2 bytes) = (26 bytes) Warm Cal. 2 Radiometer data (13 channels 3 thru 15 * 2 bytes) = (26 bytes)	Warm Calibration Telemetry Binary copy (L1b octets 2077-2136)
Error Control (2 bytes) Vertical Parity Checksum	Formed by EORing all byte pairs of the packet except the last pair

Output Section (Unique Mode) CCSDS Packet (2070 bytes)	**Input Section (AMSU-A Level 1b product)**
Packet Primary Header (6 bytes)	
Version number (3 bits)	0
Type indicator (1 bit)	0
Secondary Header Present (1 bit)	1
Application ID (11 bits)	39
Packet Sequence Flags (2 bits)	3
Packet Sequence Counter (14 bits)	Incrementing counter unique to each APID.
Packet Length (16 bits) (Size of Packet Data Field less 1)	2063
Packet Data Field (2064 bytes)	
Secondary Header (8 bytes) (Satellite UTC time)	
UTC Days since 1/1/2000 (2 bytes)	L1b Data octets 3-4 (year) and 5-6 (day of year)
UTC Milliseconds of day (4 bytes)	L1b Data octets 9-12
UTC Microseconds of millisecond (2 bytes)	Times above or below to be offset by Satellite Clock Drift Delta (octets 7-8) depending on whether the Clock Drift Correction has already been applied (octets 13-14 bit 14)
Ancillary Data, SBT time 6 bytes)	
Zero fill (1 byte)	Onboard time synthesised in software.
Seconds (3 bytes)	
Seconds / 65536 (2 bytes)	
Application Data (2048 bytes)	
Filler Section (928 bytes)	Set to 0x0001 repeating.
Science Data (1120 bytes)	
Sync Sequence (3 bytes)	Synchronisation Sequence (FF hex) (L1b octets 897-899)
Unit Identification and Serial Number (1 byte)	103 (Config file “Serial-Number”)
Digital Housekeeping Data (4 bytes)	Digital Housekeeping
Unused (1 bit)	(bytes 0-2 copied from L1b octets 901-903)
Full Scan Mode (1 bit)	Full Scan Mode
Warm Cal. Mode (1 bit)	Warm Cal. Mode
Cold Cal. Mode (1 bit)	Cold Cal. Mode
Nadir Mode (1 bit)	Nadir Mode
Cold Cal. Position (2 bits)	Cold Cal. Position
Unused (2 bits)	0
Scanner AL1-1 Power (1 bit)	Scanner AL1-1 Power
Scanner AL1-2 Power (1 bit)	Scanner AL1-2 Power
PLL Power (1 bit)	PLL Power
Survival Heater Power (1bit)	Survival Heater Power
Unused (11 bits)	0
AMSU-A1 ID number (8 bits)	9 (Config file “AMSU-A1-ID”)
Unique data repeated 30 times (34 bytes * 30) = (1020 bytes)	
Reflector 1, First reading (2 bytes)	See note below.
Reflector 2, First reading (2 bytes)	
Reflector 1, Second reading (2 bytes)	
Reflector 2, Second reading (2 bytes)	
Scene Radiometer Data (13 channels 3 thru 15 * 2 bytes) = (26 bytes)	
Temperature sensors (45 channels * 2 bytes) = (90 bytes)	Temperature Sensor Telemetry

Binary copy
(L1b octets 1985-2074)

Temperature Sensor Reference Voltage (2 bytes)

Reference voltage
(L1b octets 2075-2076)

Error Control (2 bytes)
Vertical Parity Checksum

Formed by EORing all byte pairs of the packet except the last pair (Error Control)

There are three types of Unique mode, each taking its "Unique data" from a different part of the L1b Data record.

Cold Calibration Unique Mode data is read from octets 905-1924 (ie. same area as Scene Data in Measurement Mode)
Warm Calibration Unique Mode data is read from octets 905-1924 (ie. same area as Scene Data in Measurement Mode)
Nadir Mode data is read from octets 905-1924 (ie. same area as Scene Data in Measurement Mode)

In this document the AMSU-A1 Digital A telemetry is used, not the Digital B telemetry from the TIP minor frame. AMSU-A1 ICD section 3.2 states the Digital B is sent down the S-band as telemetry data.

8.2 AMSU-A2

The instrument can run in two basic modes - Measurement and Unique mode. The differences are:

Measurement mode
- Produces 312 bytes per scan.
- Each scan consists of 30 Scene, 1 Cold Calibration and 1 Warm Calibration samples.
- Temperature sensor and housekeeping data also included.

Unique mode
- Produces 288 bytes per scan.
- There are three types of Unique mode:
  - Warm Calibration
    - Each scan contains a Warm Calibration sample, repeated 30 times
  - Cold Calibration
    - Each scan contains a Cold Calibration sample, repeated 30 times
  - Nadir Position
    - The instrument sends a Scene measurement at position 15 (nadir), repeated 30 times
- Temperature sensor and housekeeping data also included.

The current mode for each scan is given in the AMSU-A Level1b Data record (Digital A Housekeeping, octet 901, bits 1-4)

In all cases the CCSDS packet has a 1120 byte Application Data field which contains 312 or 288 bytes of data interleaved with filler (0x0001) 16-bit words.

The code 0x0001 will not appear in any valid data areas.
For the purposes of simulating this data stream the actual science data will always be placed at the end of the Application Data section with the filler codes appearing at the start.

Output Section (Measurement Mode) CCSDS Packet (1142 bytes)	**Input Section (AMSU-A Level 1b product)**
Packet Primary Header (6 bytes)	
Version number (3 bits)	0
Type indicator (1 bit)	0
Secondary Header Present (1 bit)	1
Application ID (11 bits)	40
Packet Sequence Flags (2 bits)	3
Packet Sequence Counter (14 bits)	Incrementing counter unique to each APID.
Packet Length (16 bits) (Size of Packet Data Field less 1)	1135
Packet Data Field (1136 bytes)	
Secondary Header (8 bytes) (Satellite UTC time)	
UTC Days since 1/1/2000 (2 bytes)	L1b Data octets 3-4 (year) and 5-6 (day of year)
UTC Milliseconds of day (4 bytes)	L1b Data octets 9-12
UTC Microseconds of millisecond (2 bytes)	Times above or below to be offset by Satellite Clock Drift Delta (octets 7-8) depending on whether the Clock Drift Correction has already been applied (octets 13-14 bit 14)
Ancillary Data, SBT time 6 bytes)	
Zero fill (1 byte)	Onboard time synthesised in software.
Seconds (3 bytes)	
Seconds / 65536 (2 bytes)	
Application Data (1120 bytes)	
Filler Section (808 bytes)	Set to 0x0001 repeating.
Science Data (312 bytes)	
Sync Sequence (3 bytes)	AMSU-A2 Digital A Telemetry Synchronisation Sequence (FF hex)
Unit Identification and Serial Number (1 byte)	103 (Config file “Serial-Number”)
Digital Housekeeping Data (4 bytes)	Digital Housekeeping
Unused (1 bit)	0
Full Scan Mode (1 bit)	Full Scan Mode
Warm Cal. Mode (1 bit)	Warm Cal. Mode
Cold Cal. Mode (1 bit)	Cold Cal. Mode
Nadir Mode (1 bit)	Nadir Mode
Cold Cal. Position (2 bits)	Cold Cal. Position
Unused (2 bits)	0
Scanner A2 Power (1 bit)	Scanner A2 Power
Scanner Compensation Power (1 bit)	Scanner Compensation Power
Unused (1bit)	0
Survival Heater Power (1bit)	Survival Heater Power
Unused (11 bits)	0
AMSU-A2 ID number (8 bits)	10 (Config file “AMSU-A2-ID”)
Scene data (30 * 8 bytes) = (240 bytes)	Scene Telemetry
"Reflector, First reading (2 bytes)"	Binary copy
"Reflector, Second reading (2 bytes)"	
Scene Radiometer Data Channel 1 (2 bytes)	
Scene Radiometer Data Channel 2 (2 bytes)	
Cold Calibration (12 bytes)	Cold Calibration Telemetry
Reflector, Cold Cal. Position, First reading (2 bytes)	Binary copy
Reflector, Cold Cal. Position, Second reading (2 bytes)	
Cold Cal. 1 Radiometer Data Channel 1 (2 bytes)	
Cold Cal. 1 Radiometer Data Channel 2 (2 bytes)	
Cold Cal. 2 Radiometer Data Channel 1 (2 bytes)	
Cold Cal. 2 Radiometer Data Channel 2 (2 bytes)	

Temperature sensors (19 channels * 2 bytes = 38 bytes) (format?)	Temperature Sensor Telemetry Binary copy
Temperature Sensor Reference Voltage (2 bytes)	Word 46 : Reference Voltage
Warm Calibration (12 bytes) Reflector, Warm Cal. Position First Reading (2 bytes) Reflector, Warm Cal. Position Second Reading (2 bytes) Warm Cal. 1 Radiometer Data Channel 1 (2 bytes) Warm Cal. 1 Radiometer Data Channel 2 (2 bytes) Warm Cal. 2 Radiometer Data Channel 1 (2 bytes) Warm Cal. 2 Radiometer Data Channel 2 (2 bytes)	Warm Calibration Telemetry Binary copy
Error Control (2 bytes) Vertical Parity Checksum	Formed by EORing all byte pairs of the packet except the last pair (Error Control)

Output Section (Unique Mode) CCSDS Packet (1142 bytes)	**Input Section (AMSU-A Level 1b product)**
Packet Primary Header (6 bytes)	
Version number (3 bits)	0
Type indicator (1 bit)	0
Secondary Header Present (1 bit)	1
Application ID (11 bits)	40
Packet Sequence Flags (2 bits)	3
Packet Sequence Counter (14 bits)	Incrementing counter unique to each APID.
Packet Length (16 bits) (Size of Packet Data Field less 1)	1135
Packet Data Field (1136 bytes)	
Secondary Header (8 bytes) (Satellite UTC time)	
UTC Days since 1/1/2000 (2 bytes)	L1b Data octets 3-4 (year) and 5-6 (day of year)
UTC Milliseconds of day (4 bytes)	L1b Data octets 9-12
UTC Microseconds of millisecond (2 bytes)	Times above or below to be offset by Satellite Clock Drift Delta (octets 7-8) depending on whether the Clock Drift Correction has already been applied (octets 13-14 bit 14)
Ancillary Data, SBT time 6 bytes)	
Zero fill (1 byte)	Onboard time synthesised in software.
Seconds (3 bytes)	
Seconds / 65536 (2 bytes)	
Application Data (1120 bytes)	
Filler Section (832 bytes)	Set to 0x0001 repeating.
Science Data (288 bytes)	
	AMSU-A2 Digital A Telemetry
Sync Sequence (3 bytes)	Synchronisation Sequence (FF hex)
Unit Identification and Serial Number (1 byte)	103 (Config file “Serial-Number”)
Digital Housekeeping Data (4 bytes)	Digital Housekeeping
Unused (1 bit)	0
Full Scan Mode (1 bit)	Full Scan Mode
Warm Cal. Mode (1 bit)	Warm Cal. Mode
Cold Cal. Mode (1 bit)	Cold Cal. Mode
Nadir Mode (1 bit)	Nadir Mode
Cold Cal. Position (2 bits)	Cold Cal. Position
Unused (2 bits)	0
Scanner A2 Power (1 bit)	Scanner A2 Power
Scanner Compensation Power (1 bit)	Scanner Compensation Power
Unused (1bit)	0
Survival Heater Power (1bit)	Survival Heater Power
Unused (11 bits)	0
AMSU-A2 ID number (8 bits)	10 (Config file “AMSU-A2-ID”)
Unique data repeated 30 times (30 * 8 bytes) = (240 bytes)	Scene Telemetry
Reflector, First reading (2 bytes)	Binary copy
Reflector, Second reading (2 bytes)	
Scene Radiometer Data Channel 1 (2 bytes)	
Scene Radiometer Data Channel 2 (2 bytes)	
Temperature sensors (19 channels * 2 bytes) = (38 bytes)	Temperature Sensor Telemetry Binary copy (L1b Data, octets x-x)
Temperature Sensor Reference Voltage (2 bytes)	Reference Voltage (L1b Data, octets x-x, (word 46))
Error Control (2 bytes)	
Vertical Parity Checksum	Formed by EORing all byte pairs of the packet except the last pair (Error Control)

There are three types of Unique mode, each with a different meaning for the "Unique data" section.

Cold Calibration Unique Mode data is read from L1b Data section octets 2193-2432 (same area as Scene Data in Measurement Mode).
Warm Calibration Unique Mode data is read from L1b Data section octets 2193-2432 (same area as Scene Data in Measurement Mode).
Nadir Mode data is read from L1b Data section octets 2193-2432 (same area as Scene Data in Measurement Mode)

Note the AMSU-A2 Digital A telemetry is used, not the Digital B telemetry from the TIP minor frame. AMSU-A2 ICD section 3.2 states the Digital B is sent down the S-band as telemetry data.

8.3 AVHRR

Output section (CCSDS packet, 12966 bytes)	**Input source (using NOAA AVHRR L1b product)**
Packet Primary Header (6 bytes)	
Version number (3 bits)	0
Type indicator (1 bit)	0
Secondary Header Present (1 bit)	1
Application ID (11 bits)	103 (AVHRR HR Channel 3A) or 104 (AVHRR HR Channel 3B). Using L1b Data octets 13-14 bits 0-1 "Channel 3 Select". Code 0=APID 104, otherwise APID 103
Packet Sequence Flags (2 bits)	3 (data not split into multiple packets)
Packet Sequence Counter (14 bits)	Incrementing counter unique to each APID.
Packet Length (16 bits) (Size of Packet Data Field less 1)	12959
Packet Data Field (12960 bytes)	
Secondary Header (8 bytes) (Satellite UTC time)	
UTC Days since 1/1/2000 (2 bytes)	L1b Data octets 3-4 (year) and 5-6 (day of year)
UTC Milliseconds of day (4 bytes)	L1b Data octets 9-12
UTC Microseconds of millisecond (2 bytes)	Times above or below to be offset by Satellite Clock Drift Delta (octets 7-8) depending on whether the Clock Drift Correction has already been applied (octets 13-14 bit 14)
Ancillary Data, SBT time 6 bytes)	
Zero fill (1 byte)	Onboard time synthesised in software.
Seconds (3 bytes)	
Seconds / 65536 (2 bytes)	
Application Data (12944 bytes)	Level 1b product, Data record
Detector : Space View / Deep Space (500 bits)	Space Data
10 Samples (50 bits / sample)	(octets 1161-1260)
Channel 1 (10 bits)	Channel 1 (lose 6 MSB)
Channel 2 (10 bits)	Channel 2 (lose 6 MSB)
Channel 3A/3B (10 bits)	Channel 3 (lose 6 MSB)
Channel 4 (10 bits)	Channel 4 (lose 6 MSB)
Channel 5 (10 bits)	Channel 5 (lose 6 MSB)
Electronic Ramp Calibration / Ramp Calibration (50 bits)	Ramp Calibration
1 Sample (50 bits)	(octets 1081-1090)
Channel 1 (10 bits)	Channel 1 (lose 6 MSB)
Channel 2 (10 bits)	Channel 2 (lose 6 MSB)
Channel 3A/3B (10 bits)	Channel 3 (lose 6 MSB)
Channel 4 (10 bits)	Channel 4 (lose 6 MSB)
Channel 5 (10 bits)	Channel 5 (lose 6 MSB)
Detector : Earth View / Video Data for 5 Channels (102400 bits)	Earth Data
2048 Samples (50 bits / sample)	(octets 1265-3992)
Channel 1 (10 bits)	We have a total of 409 FOV (pixels) equals 20% of High Rate.
Channel 2 (10 bits)	Each pixel has 5 x 10 bit channel values.
Channel 3A/3B (10 bits)	
Channel 4 (10 bits)	
Channel 5 (10 bits)	
TLM : IR Target Temp (50 bits)	Internal Target Temp
1 Sample (50 bits)	
Channel 1 (10 bits)	0
Channel 2 (10 bits)	0
Channel 3A/3B (10 bits)	We have three 16bit integers as source (octets 1091-1096).
Channel 4 (10 bits)	These represent readings from three sensors on one of the four
Channel 5 (10 bits)	PRTs and are copied to channels 3,4,5 respectively.
TLM : Patch Temp (50 bits)	Patch Temperature
1 Sample (50 bits)	
Channel 1 (10 bits)	We only have a single 16bit integer
Channel 2 (10 bits)	as source (octets 1097-1098, see also 4021-4022)
Channel 3A/3B (10 bits)	
Channel 4 (10 bits)	
Channel 5 (10 bits)	
Detector : IR Back Scan (500 bits)	Back Scan
10 Samples (50 bits / sample)	(octets 1101-1160)
Channel 1 (10 bits)	0
Channel 2 (10 bits)	0
Channel 3A/3B (10 bits)	Word (sample*3)
Channel 4 (10 bits)	Word (sample*3+1) (lose 6 MSB)
Channel 5 (10 bits)	Word (sample*3+2) (lose 6 MSB)
Unused (2 bits)	
Error Control (2 bytes)	
Vertical Parity Checksum	Formed by EORing all byte pairs of the packet except the last pair (Error Control)

8.4 HIRS/4

General Definitions:
1 cycle (256s) = 40 periods
1 period (6.4s) = 64 elements
1 element (0.1s?) = packet of 288 data bits (36 bytes)
Each CCSDS packet and each L1b data record contains one full period.

Output section (CCSDS packet, 2326 bytes)	**Input section (using NOAA HIRS/4 Level 1b product)**
Packet Primary Header (6 bytes)	
Version number (3 bits)	0
Type indicator (1 bit)	0
Secondary Header Present (1 bit)	1
Application ID (11 bits)	38
Packet Sequence Flags (2 bits)	3 (data not split into multiple packets)
Packet Sequence Counter (14 bits)	Incrementing counter unique to each APID.
Packet Length (16 bits) (Size of Packet Data Field less 1)	2319
Packet Data Field (2320 bytes)	
Secondary Header (8 bytes) (Satellite UTC time)	
UTC Days since 1/1/2000 (2 bytes)	L1b Data octets 3-4 (year) and 5-6 (day of year)
UTC Milliseconds of day (4 bytes)	L1b Data octets 9-12
UTC Microseconds of millisecond (2 bytes)	Source unknown (times above to be offset by Satellite Clock Drift Delta (octets 7-8) if Clock Drift Correction (octets 13-14 bit 14) shows times have been corrected)
Ancillary Data, SBT time 6 bytes)	Onboard time synthesised in software.
Zero fill (1 byte)	
Seconds (3 bytes)	
Seconds / 65536 (2 bytes)	
Application Data (2304 bytes)	Level 1b product, Data record, main data section octets 1457-4528"
Elements (0 thru 55) (56 elements * 36 bytes) = (2016 bytes)	Elements (0 thru 55) (56 elements * 48 bytes) = (2304 bytes)
Word 1 (bits 1-13) (13 bits)	Header (4 bytes)
Scan Encoder Position (8 bits)	Bits 31-24 Scan Encoder Position
Electronic Cal. Level Indicator (5 bits)	Bits 23-19 Electronic Cal Level Indicator
Word 2 (bits 14-26) (13 bits)	Bits 18-13 Zero fill
Channel 1 period monitor (6 bits)	Bits 12-7 Channel 1 period monitor
Element number (6 bits)	Bits 6-1 Element number
Filter Sync Designator (1 bit)	Bit 0 Filter Sync Designator
Data section (bits 27-286) (20 words * 13 bits each) = (260 bits)	Radiometric Data (20 words * 16 bits = 40 bytes)
Radiometric Channel 1	Radiometric Channel 1
13 bits Signed Radiant Signal outputs	Bits 15-13 Zero fill
Bit 0 sign	Bit 12 Inverted sign bit
Bits 1-12 data	Bits 11-0 Radiant Signal Amplitude
	(structure repeated over all channels)
Radiometric Channel 17	Radiometric Channel 17
Radiometric Channel 2	Radiometric Channel 2
Radiometric Channel 3	Radiometric Channel 3
Radiometric Channel 13	Radiometric Channel 13
Radiometric Channel 4	Radiometric Channel 4
Radiometric Channel 18	Radiometric Channel 18
Radiometric Channel 11	Radiometric Channel 11
Radiometric Channel 19	Radiometric Channel 19
Radiometric Channel 7	Radiometric Channel 7
Radiometric Channel 8	Radiometric Channel 8
Radiometric Channel 20	Radiometric Channel 20
Radiometric Channel 10	Radiometric Channel 10
Radiometric Channel 14	Radiometric Channel 14
Radiometric Channel 6	Radiometric Channel 6
Radiometric Channel 5	Radiometric Channel 5
Radiometric Channel 15	Radiometric Channel 15
Radiometric Channel 12	Radiometric Channel 12
Radiometric Channel 16	Radiometric Channel 16
Radiometric Channel 9	Radiometric Channel 9
	Bit flags (2 bytes)
Data Valid (bit 287) (1 bit)	Bit 15 Valid Data flag
Odd Parity (bit 288) (1 bit)	Bit 14 Odd Bit parity
	Bits 13-0 Zero fill
	Zero fill (2 bytes)
Element 56 Positive Electronics Calibration (288 bits / 36 bytes)	Element 56 (48 bytes) (structure as above)
Bits 1-13 (word 1)	Contents as given on the left.
As elements 0-55	
Bits 24-26 (word 2)	
As elements 0-55.	
Bits 27-286	
Signal output for all channels as a result of electronic calibration.	
Channel ordering as elements 0-55.	
Data Valid (bit 287) (1 bit)	
Odd Parity (bit 288) (1 bit)	
Element 57 Negative Electronics Calibration (288 bits / 36 bytes)	Element 57 (48 bytes) (structure as above)
Bits 1-13 (word 1)	Contents as given on the left.

As elements 0-55.
Bits 24-26 (word 2)
As elements 0-55.
Bits 27-286
Signal output for all channels as a result of electronic calibration.
Channels ordered as elements 0-55.
Data Valid (bit 287) (1 bit)
Odd Parity (bit 288) (1 bit)

Element 58 (288 bits / 36 bytes)
Bits 1-13 (word 1)
As elements 0-55.
Bits 24-26 (word 2)
As elements 0-55.
Bits 27-91 Internal Warm Target Temperature Sensor #1
13 bit value repeated 5 times. Range 273 to 333K.
Bits 92-156 Internal Warm Target Temperature Sensor #2
13 bit value repeated 5 times. Range 273 to 333K.
Bits 157-221 Internal Warm Target Temperature Sensor #3
13 bit value repeated 5 times. Range 273 to 333K.
Bits 222-286 Internal Warm Target Temperature Sensor #4
13 bit value repeated 5 times. Range 273 to 333K.
Data Valid (bit 287) (1 bit)
Odd Parity (bit 288) (1 bit)

Element 58 (48 bytes) (structure as above)
Contents as given on the left.

Element 59 (288 bits / 36 bytes)
Bits 1-13 (word 1)
As elements 0-55.
Bits 24-26 (word 2)
As elements 0-55.
Bits 27-91 Internal Cold Target Temperature Sensor #1
13 bit value repeated 5 times. Range 243 to 303K.
Bits 92-156 Ground
13 bit value repeated 5 times.
Bits 157-221 Internal Warm Target Temperature Sensor #5.
13 bit value repeated 5 times. Range 273-333K
Bits 222-286 Telescope Temperature Sensor #3
13 bit value repeated 5 times. Range 260-300K.
Data Valid (bit 287) (1 bit)
Odd Parity (bit 288) (1 bit)

Element 59 (48 bytes) (structure as above)
Contents as given on the left.

Element 60 (288 bits / 36 bytes)
Bits 1-13 (word 1)
As elements 0-55.
Bits 24-26 (word 2)
As elements 0-55.
Bits 27-91 Filter Wheel Housing Temperature Sensor #1
13 bit value repeated 5 times. Range 273-333K.
Bits 92-156 Filter Wheel Housing Temperature Sensor #2
13 bit value repeated 5 times. Range 273-333K.
Bits 157-221 Filter Wheel Housing Temperature Sensor #3
13 bit value repeated 5 times. Range 273-333K.
Bits 222-286 Filter Wheel Housing Temperature Sensor #4
13 bit value repeated 5 times. Range 273-333K.
Data Valid (bit 287) (1 bit)
Odd Parity (bit 288) (1 bit)

Element 60 (48 bytes) (structure as above)
Contents as given on the left.

Element 61 (288 bits / 36 bytes)
Bits 1-13 (word 1)
As elements 0-55.
Bits 24-26 (word 2)
As elements 0-55.
Bits 27-91 Patch Temperature Expanded Scale
13 bit value repeated 5 times. Range 90-150K
Bits 92-156 First Stage Radiator Temperature Sensor
13 bit value repeated 5 times. Range 150-320K
Bits 157-221 Filter Wheel Housing Heater Current
13 bit value repeated 5 time. Range 0-500mA
Bits 222-286 Electronic Calibration Digital to Analog Converter
13 bit value repeated 5 times. Range 0-4V.
Data Valid (bit 287) (1 bit)
Odd Parity (bit 288) (1 bit)

Element 61 (48 bytes) (structure as above)
Contents as given on the left.

Element 62 (288 bits / 36 bytes)
Bits 1-13 (word 1)
As elements 0-55.
Bits 24-26 (word 2)
As elements 0-55.
Bits 27-39 Scan Mirror Temperature.
Range 260-320K
Bits 40-52 Primary Telescope Temperature
Range 260-320K
Bits 53-65 Secondary Telescope Temperature
Range 260-320K
Bits 66-78 Baseplate Temperature
Range 260-320K
Bits 79-91 Electronics Temperature
Range 260-320K
Bits 92-104 Patch Temperature - Full Range
Range 90-320K

Element 62 (48 bytes) (structure as above)
Contents as given on the left.

Bits 105-117 Scan Motor Temperature
Range 260-320K
Bits 118-130 Filter Motor Wheel Temperature
Range 260-320K
Bits 131-143 Cooler Housing Temperature
Range 260-320K
Bits 144-156 Patch Cooler Power
Range 0-80 mW
Bits 157-169 Scan Motor Current
Range 0.65-1.0 A
Bits 170-182 Filter Motor Current
Range 100-300 mA
Bits 183-195 +15 VDC
Range +15 +/- 0.2 V
Bits 196-208 -15 VDC
Range -15 +/- 0.2 V
Bits 209-221 +7.5 VDC
Range +7.5 +/- 0.05 V
Bits 222-234 -7.5 VDC
Range -7.5 +/- 0.05 V
Bits 235-247 +10 VDC
Range +10 +/- 0.2 V
Bits 248-260 +5 VDC
Range +5 +/- 0.2 V
Bits 261-273 Analog Ground
Range +/- 1 count (?)
Bits 274-286 Analog Ground (repeated?)
Range +/- 1 count (?)
Data Valid (bit 287) (1 bit)
Odd Parity (bit 288) (1 bit)

Element 63 (288 bits / 36 bytes) Bits 1-13 (word 1) As elements 0-55. Bits 24-26 (word 2) As elements 0-55. Bits 27-39 Line Counter (gives number of lines from the last auto-calibration sequence) 0-8191 unsigned. Bits 40-52 First Status Word Bit 40 Zero Fill Bits 41-44 Instrument Serial Number Bit 45 Command Status Bit - Instrument ON/OFF Bit 46 Command Status Bit - Scan Motor ON/OFF Bit 47 Command Status Bit - Filter Wheel ON/OFF Bit 48 Command Status Bit - Electronics ON/OFF Bit 49 Command Status Bit - Cooler Heat ON/OFF Bit 50 Command Status Bit - Internal Warm Target Position Bit 51 Command Status Bit - Internal Cold Target Position Bit 52 Command Status Bit - Space Position Bits 53-65 Second Status Word Bits 53-57 Zero Fill Bit 58 Command Status Bit - Nadir Position Bit 59 Command Status Bit - Calibration Enable/Disable Bit 60 Command Status Bit - Cooler Door Release Enable/Disable Bit 61 Command Status Bit - Cooler Door Open Bit 62 Command Status Bit - Cooler Door Closed Bit 63 Command Status Bit - Filter Housing Heater ON/OFF Bit 64 Command Status Bit - Patch Temperature Control ON/OFF Bit 65 Command Status Bit - Filter Motor Power HIGH	Element 63 (48 bytes) (structure as above) Contents as given on the left.
"Bits 66-286 Data Verification, 17 * signed 13 bit integers" 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	Data verification codes are supplied in L1b file. +1443 -1522 -1882 -1631 -1141 +1125 +3655 -2886 -3044 -3764 -3262 -2283 -2251 +3214 +1676 +1992
Error Control (2 bytes) Vertical Parity Checksum	Formed by EORing all byte pairs of the packet except the last pair (Error Control)

Notes:
- During a Space View Calibration period, elements 0-7 actually show scan mirror motion and elements 8-56 show space view dwell.
- During a Warm Target Calibration period, elements 0-56 show warm target dwell.
- The sign is described as "Inverted Sign Bit" in the L1b file, matching the format in the CCSDS packet (1=positive, 0=negative).
- The ordering of the bits in the CCSDS packet (MSB bit 11, LSB bit 0) is the same in the L1b file.
- The data supplied in the NOAA L1b file matches the elements array required for the CCSDS packets. The conversion of each element can be performed by mapping the 48 byte structure from the L1b file to the 36 byte CCSDS structure and looping around for all 64 elements.

8.5 MHS (based on NOAA AMSU-B inputs)

For additional information see RD.2.
The MHS packet generator uses a configuration file to allow some parameters to be tweaked. These are referred to below in the form "<default value> (Config File <parameter name>").

The layout of the configuration file is given in more detail, including permissible values for each parameter, in RD.2.

One scan is produced per 8/3 seconds.

Output section (CCSDS packet, 1308 bytes)	**Input source (using NOAA AMSU-B L1b product)**
Packet Primary Header (6 bytes)	
Version number (3 bits)	0
Type indicator (1 bit)	0
Secondary Header Present (1 bit)	1
Application ID (11 bits)	34 (MHS)
Packet Sequence Flags (2 bits)	3 (data not split into multiple packets)
Packet Sequence Counter (14 bits)	Incrementing counter unique to each APID.
Packet Length (16 bits) (Size of Packet Data Field less 1)	1301
Packet Data Field (1302 bytes)	
Secondary Header (8 bytes) (Satellite UTC time)	
UTC Days since 1/1/2000 (2 bytes)	L1b Data octets 3-4 (year) and 5-6 (day of year)
UTC Milliseconds of day (4 bytes)	L1b Data octets 9-12
UTC Microseconds of millisecond (2 bytes)	Times above or below to be offset by Satellite Clock Drift Delta (octets 7-8) depending on whether the Clock Drift Correction has already been applied (octets 13-14 bit 14)
Ancillary Data, SBT time 6 bytes)	
Seconds (4 bytes)	This is the onboard UTC time equal to the uncorrected UTC time given above.
Seconds / 65536 (2 bytes)	
Measurement Data Packet (1286 bytes)	
Full Housekeeping Data (39 bytes)	
Mode and sub-commutation code (1 byte)	
Mode code (4 bits)	3 (Config file "Scan-Mode")
PIE id (1 bit)	0 (Config file "PIE-ID")
Sub-commutation code (3 bits)	0 (Config file "Sub-Commutation-Code")
Telecommand Acknowledgement and fault code (5 bytes)	
TC Clean (1 bit)	0 (Config file "TC-clean")
TC Conforms (1 bit)	0 (Config file "TC-Conforms")
TC Recognised (1 bit)	0 (Config file "TC-Recognised")
TC Legal (1 bit)	0 (Config file "TC-Legal")
FDM Motor Current Trip Status (1 bit)	0 (Config file "FDM-Motor-Current-Trip-Status")
TC Application ID (3 bits)	0 (Config file "TC-Application--ID")
TC Application Process ID (8 bits)	0 (Config file "TC-Application-Process-ID")
TC Packet Sequence Count (8 bits)	0 (Config file "TC-Packet-Sequence-Count")
TC Packet Sequence (6 bits)	0 (Config file "TC-Packet-Sequence")
TC Received Count (2 bits)	0 (Config file "TC-Received-Count")
Monitor Current Fault (1 bit)	0 (Config file "Monitor-Current-Fault")
Thermistor Monitor Fault (1 bit)	0 (Config file "Thermistor-Monitor-Fault")
Switch Fault (1bit)	0 (Config file "Switch-Fault")
Processor Fault (1bit)	0 (Config file "Processor-Fault")
RDM Motor Current Trip Status (1bit)	0 (Config file "RDM-Motor-Current-Trip-Status")
DC Offset Error (1bit)	0 (Config file "DC-Offset-Error")
Scan Control Error (1bit)	0 (Config file "Scan-Control-Error")
REF CK Error (1bit)	0 (Config file "REF-CK-Error")
Switch Status (3 bytes)	
Receiver Channel H1 On/Off (1 bit)	1 (On) (Config file "Receiver-Channel-H1-On-Off")
Receiver Channel H1 Local Oscillator A/B selected (1 bit)	0 (A selected) (Config file "Receiver-Channel-H1-Local-Oscillator-A-B-Selected")
Receiver Channel H2 On/Off (1 bit)	1 (On) (Config file "Receiver Channel H2-On-Off")
Receiver Channel H2 Local Oscillator A/B selected (1 bit)	0 (A selected) (Config file "Receiver Channel-H2-Local-Oscillator-A-B-Selected")
Receiver Channels H3/H4 Frontend On/Off (1 bit)	1 (On) (Config file "Receiver-Channels-H3-H4-Frontend-On-Off")
Receiver Channels H3/H4 Local Oscillator A/B selected (1 bit)	0 (A selected) (Config file "Receiver-Channels-H3-H4-Local-oscillator-A-B-Selected")
Receiver Channel H3 Backend On/Off (1 bit)	1 (On) (Config file "Receiver-Channel-H3-Backend-On-Off")
Receiver Channel H4 Backend On/Off (1 bit)	1 (On) (Config file "Receiver-Channel-H4-Backend-On-Off")
Receiver Channel H5 On/Off (1 bit)	1 (On) (Config file "Receiver-Channel-H5-On-Off")
Receiver Channel H5 Local Oscillator A/B selected (1 bit)	0 (A selected) (Config file Receiver-Channel-H5-Local-Oscillator-A-B-Selected")
Rx CV On/Off (1 bit)	1 (On) (Config file "Rx-CV-On-Off")
Receiver Operational (1 bit)	1 (Operational) (Config file "Receiver-Operational")
Scan Mechanism Operational Heaters On/Off (1 bit)	1 (On) (Config file "Scan-Mechanism-Operational-Heaters-On-Off")
Auxiliary Operational Heaters On/Off (1 bit)	1 (On) (Config file "Auxiliary-Operational-Heaters-On-Off")
Signal Processing Electronics / Scan Control Electronics On/Off (1 bit)	1 (On) (Config file "Signal-Processing-Electronics-On-Off")
PROM failure (1 bit)	0 (No failure) (Config file "PROM-Failure")
RDM Zero Position Sensors A/B selected (1 bit)	0 (A selected) (Config file "RDM-Zero-Position-Sensors-A-B-Selected")
FDM Zero Position Sensors A/B selected (1 bit)	0 (A selected) (Config file

"FDM-Zero-Position-Sensors-A-B-Selected")

- RDM Motor Sensors A/B selected (1 bit) — 0 (A selected) (Config file "RDM-Motor-Sensors-A-B-Selected")
- FDM Motor Sensors A/B selected (1 bit) — 0 (A selected) (Config file "FDM-Motor-Sensors-A-B-Selected")
- RDM Motor Supply On/Off (1 bit) — 1 (On) (Config file "RDM-Motor-Supply-On-Off")
- FDM Motor Supply On/Off (1 bit) — 1 (On) (Config file "FDM-Motor-Supply-On-Off")
- RDM Motor Current Trip Enabled (1 bit) — 1 (On) (Config file "RDM-Motor-Supply-On-Off")
- FDM Motor Current Trip Enabled (1 bit) — 1 (On) (Config file "FDM-Motor-Current-Trip-Enabled")

Temperature Data (24 bytes)

- T1 (1 byte) — 133 (+25C) (from Config file "Resistance-T1")
- T2 (1 byte) — 133 (+25C) (from Config file "Resistance-T2")
- T3 (1 byte) — 133 (+25C) (from Config file "Resistance-T3")
- T4 (1 byte) — 133 (+25C) (from Config file "Resistance-T4")
- T5 (1 byte) — 133 (+25C) (from Config file "Resistance-T5")
- T6 (1 byte) — 133 (+25C) (from Config file "Resistance-T6")
- T7 (1 byte) — 133 (+25C) (from Config file "Resistance-T7")
- T8 (1 byte) — 133 (+25C) (from Config file "Resistance-T8")
- T9 (1 byte) — 133 (+25C) (from Config file "Resistance-T9")
- T10 (1 byte) — 133 (+25C) (from Config file "Resistance-T10")
- T11 (1 byte) — 133 (+25C) (from Config file "Resistance-T11")
- T12 (1 byte) — 133 (+25C) (from Config file "Resistance-T12")
- T13 (1 byte) — 147 (+20C) (from Config file "Resistance-T13")
- T14 (1 byte) — 147 (+20C) (from Config file "Resistance-T14")
- T15 (1 byte) — 147 (+20C) (from Config file "Resistance-T15")
- T16 (1 byte) — 147 (+20C) (from Config file "Resistance-T16")
- T17 (1 byte) — 174 (+10C) (from Config file "Resistance-T17")
- T18 (1 byte) — 174 (+10C) (from Config file "Resistance-T18")
- T19 (1 byte) — 174 (+10C) (from Config file "Resistance-T19")
- T20 (1 byte) — 120 (+30C) (from Config file "Resistance-T20")
- T21 (1 byte) — 117 (+31C) (from Config file "Resistance-T21")
- T23 (1 byte) — 120 (+30C) (from Config file "Resistance-T23")
- T22 (1 byte) — 120 (+30C) (from Config file "Resistance-T22")
- T24 (1 byte) — 130 (+26C) (from Config file "Resistance-T24")

Raw Current Consumption Data (6 bytes)

- +5V secondary current (1 byte) — 54 (900 mA) (Config file "5V-Secondary-Current")
- +8V receiver current (1 byte) — 72 (1444 mA) (Config file "8V-Receiver-Current")
- +15V receiver current (1 byte) — 114 (831 mA) (Config file "15V-Receiver-Current")
- -15V receiver current (1 byte) — 91 (169 mA) (Config file "N15V-Receiver-Current")
- RDM Motor Current (1 byte) — 50 (668 mA) (Config file "RDM-Motor-Current")
- FDM motor Current (1 byte) — 50 (668 mA) (Config file "FDM-Motor-Current")

Status word (1 byte)

- DC Offset Valid (1 bit) — 1 (Config file "DC-Offset-Valid")
- Scan Control Valid (1 bit) — 1 (Config file "Scan-Control-Valid")
- Profile (2 bits) — 0 (Config file "Profile").
- Unused (4 bits) — 0

Signal Processing Status (9 bytes)

- Channel H1 DC Offset Word (8 bits) — 0 (Config file "Channel-H1-DC-Offset-Word")
- Channel H2 DC Offset Word (8 bits) — 0 (Config file "Channel-H2-DC-Offset-Word")
- Channel H3 DC Offset Word (8 bits) — 0 (Config file "Channel-H3-DC-Offset-Word")
- Channel H4 DC Offset Word (8 bits) — 0 (Config file "Channel-H4-DC-Offset-Word")
- Channel H5 DC Offset Word (8 bits) — 0 (Config file "Channel-H5-DC-Offset-Word")
- H1 Valid (1 bit) — 1 (Config file "H1-Valid")
- H2 Valid (1 bit) — 1 (Config file "H2-Valid")
- H3 Valid (1 bit) — 1 (Config file "H3-Valid")
- H4 Valid (1 bit) — 1 (Config file "H4-Valid")
- H5 Valid (1 bit) — 1 (Config file "H5-Valid")
- SPE Mux Code (3 bits) — 3 (Config file "SPE-Mux-Code")
- H1 Gain (3 bits) — 0 (Config file "H1-Gain")
- H2 Gain (3 bits) — 0 (Config file "H2-Gain")
- Unused (2 bits) — 0
- H3 Gain (3 bits) — 0 (Config file "H3-Gain")
- H4 Gain (3 bits) — 0 (Config file "H4-Gain")
- Unused (2 bits) — 0
- H5 Gain (3 bits) — 0 (Config file "H5-Gain")
- Unused (4 bits)

Pixel Data (1176 bytes) — MHS 1b Data Record (Binary copy each pixel from L1b file)

- Earth (90 pixels * 12 bytes = 1080 bytes)
  - Mid-pixel position (2 bytes, MSB/LSB)
  - Channel H1 (2 bytes, MSB/LSB)
  - Channel H2 (2 bytes, MSB/LSB)
  - Channel H3 (2 bytes, MSB/LSB)
  - Channel H4 (2 bytes, MSB/LSB)
  - Channel H5 (2 bytes, MSB/LSB)
- Space (4 pixels * 12 bytes = 48 bytes)
  - Mid-pixel position (2 bytes, MSB/LSB)
  - Channel H1 (2 bytes, MSB/LSB)
  - Channel H2 (2 bytes, MSB/LSB)
  - Channel H3 (2 bytes, MSB/LSB)
  - Channel H4 (2 bytes, MSB/LSB)
  - Channel H5 (2 bytes, MSB/LSB)
- OBCT (4 pixels * 12 bytes = 48 bytes)
  - Mid-pixel position (2 bytes, MSB/LSB)
  - Channel H1 (2 bytes, MSB/LSB)
  - Channel H2 (2 bytes, MSB/LSB)
  - Channel H3 (2 bytes, MSB/LSB)
  - Channel H4 (2 bytes, MSB/LSB)

MHS 1b Data Record (Binary copy each pixel from L1b file)

- Scene Data
  - Shaft Position
  - Channel 16
  - Channel 17
  - Channel 18
  - Channel 19
  - Channel 20
- Space View Data
  - Shaft Position
  - Channel 16
  - Channel 17
  - Channel 18
  - Channel 19
  - Channel 20
- Target View Data
  - Shaft Position
  - Channel 16
  - Channel 17
  - Channel 18
  - Channel 19

Channel H5 (2 bytes, MSB/LSB)		Channel 20
OBCT Temperature Data (16 bytes)		
Unused (4 bits)		0
On-board Target Temperature PRT1 (12 bits)		See below for calculation. Temperature counts read from Data record Digital Data Word A11, octets 2701-2702 and calibration coeffs from the Header block Temperature 1 Coefficients 0-3, octets 561-568.
Unused (4 bits)	0	
On-board Target Temperature PRT2 (12 bits)		See below for calculation. Temperature counts read from Data record Digital Data Word A12, octets 2703-2704 and calibration coeffs from the Header block Temperature 2 Coefficients 0-3, octets 569-576.
Unused (4 bits)	0	
On-board Target Temperature PRT3 (12 bits)		See below for calculation. Temperature counts read from Data record Digital Data Word A13, octets 2704-2705 and calibration coeffs from the Header block Temperature 3 Coefficients 0-3, octets 577-584.
Unused (4 bits)	0	
On-board Target Temperature PRT4 (12 bits)		See below for calculation. Temperature counts read from Data record Digital Data Word A14, octets 2705-2706 and calibration coeffs from the Header block Temperature 4 Coefficients 0-3, octets 585-592.
Unused (4 bits)	0	
On-board Target Temperature PRT5 (12 bits)		See below for calculation. Temperature counts read from Data record Digital Data Word A15, octets 2707-2708 and calibration coeffs from the Header block Temperature 5 Coefficients 0-3, octets 593-600.
Unused (4 bits)		0
Calibration Channel 1 (PRT CAL 1 : 118 ohms) (12 bits)		Config file “PRT-CAL-1”
Unused (4 bits)		0
Calibration Channel 2 (PRT CAL 1 : 95.3 ohms) (12 bits)		Config file “PRT-CAL-2”
Unused (4 bits)		0
Calibration Channel 3 (PRT CAL 1 : 80.6 ohms) (12 bits)		Config file “PRT-CAL-3”
Spare (45 bytes)		0
Error Control (2 bytes)		
Vertical Parity Checksum		Formed by EORing all byte pairs of the packet except the last pair.

Notes:

-The OBCT PRT values are calculated by calibrating the PRT readings from the input product to produce a reading in degrees Kelvin, then converting back to MHS counts using the formula in [RD.2].

Calibrating NOAA PRT values:
Where V is the AMSU-B count value, T is the temperature in Kelvin and C0-3 are the calibration coefficients as stored in the product header.

$$T = C0/10^{2} + C1*V/10^{7} + C2*V^{2}/10^{12} + C3*V^{2}/10^{18}$$

MHS count values:
Where M is the MHS count and K is the temperature in Kelvin.

$$M = 1940.7024 + 34.744*(K - 273.15)$$

9 APPENDIX B - CONFIGURATION FILES

The MetOpizer collective uses a number of configuration files to generate CCSDS packets. These files are all self-describing, ie. the comments inside each configuration file are the documentation for each setting.

The default versions of all these files are listed in the following sections and are used if a particular setting is missing from the file. If no configuration file is specified when the executable is run all settings take default values.

In addition a warning message is generated if any values are out of range or missing or if an unrecognised parameter is found in the file.

All settings are case insensitive.

9.1 AMSU-A1

```
# MetOpizer AMSU-A1 Configuration File

# Serial Number (8 bit unsigned int) (default 101)
# Suitable values are:
#    1 - Engineering Model
#    9 - Flight Model 1
#   13 - Flight Model 2
#   17 - Flight Model 3
#   21 - Flight Model 4
#   25 - Flight Model 5
#   29 - Flight Model 6
#   33 - Flight Model 7
Serial-Number=29


# Unit ID (8 bit unsigned int) (default 0)
Unit-ID=0
# Note: This value should be zero.

# Note the total packet payload is 2048 bytes
# of which either 1120 or 1240 bytes are real data

# Filler Pattern (string) (default "x;x;x" ie. 60% data 40% filler)
# X : data
# x : 50% data, 50% skip this symbol
# . : filler
# ; : 50% filler, 50% skip this symbol
# ? : 50% filler, 50% data
Filler-Pattern=XXX.XXXXXx.XXXxx..;.

# Filler Random Speckle (unsigned integer) (default 50)
# count of words to be flipped at random
Filler-Random-Speckle=50
# (nb. the randomness is biased towards the 1120/2048 or 1240/2048 ratio)
# The algo makes three passes: pattern, speckle then  correct. The 'correct' pass
# is non-random and forces the right number of filler words.

# Scan Line Number as Sequence Count (boolean) (default yes)
#  When on, the Level 1b Scan Line Number field is used as the
#  CCSDS Sequence Count value.
#  Initial-Sequence-Count then becomes an offset.
```

```
Scan-Line-Number-As-Sequence-Count=yes


# CCSDS Initial Sequence Count (14 bit unsigned int) (default 0)
Initial-Sequence-Count=0


# UTC Ignore Input Time (boolean) (default false)
#  When switched on the times of an input product as modified to
#  begin at midnight 01/01/2000, with times throughout the product
#  all offset by the same value.
#  The three UTC time offset values below are applied afterwards.
UTC-Ignore-Input-Time=false


# UTC Time Offsets (int) (default 0,0,0)
#  Offsets applied to UTC time read from the Level1b UTC time
UTC-Days-Offset=0
UTC-Milliseconds-Offset=0
UTC-Microseconds-Offset=0


# OBT Time Offsets (int) (defaults 0,0)
#  The OBT time of the first packet in the product is set to 0,0.
#  Subsequent times are offset by the differences in the UTC times.
#  Finally the offsets below are applied.
OBT-Coarse-Offset=0
OBT-Fine-Offset=0


# OBT Fixed Fine Time (boolean) (default false)
#  When this option is used all OBT Fine time values will be set equal to
#  the OBT-Fine-Offset value given above.
OBT-Fixed-Fine-Time=false
```

9.2 AMSU-A2

```
# MetOpizer AMSU-A2 Configuration File

# Serial Number (8 bit unsigned int) (default 101)
# Suitable values are:
#    2 - Engineering Model
#   10 - Flight Model 1
#   14 - Flight Model 2
#   18 - Flight Model 3
#   22 - Flight Model 4
#   26 - Flight Model 5
#   30 - Flight Model 6
#   34 - Flight Model 7
Serial-Number=30


# Unit ID (8 bit unsigned int) (default 0)
Unit-ID=0
# Note: This value should be zero.

# Note the total packet payload is 1120 bytes
# of which either 312 or 288 bytes are real data

# Filler Pattern (string) (default ";;;x;;x;;x" ie. 30% data 70% filler)
# X : data
# x : 50% data, 50% skip this symbol
# . : filler
# ; : 50% filler, 50% skip this symbol
# ? : 50% filler, 50% data
Filler-Pattern=XXX.XXXXXx.XXXxx..;.

# Filler Random Speckle (unsigned integer) (default 50)
# count of words to be flipped at (pseudo)random
Filler-Random-Speckle=50
# (nb. the randomness is biased towards the 312/1120 or 288/1120 ratio)
# The algo makes three passes: pattern, speckle then  correct. The 'correct' pass
# is non-random and forces the right number of filler words.

# Scan Line Number as Sequence Count (boolean) (default false)
#  When on, the Level 1b Scan Line Number field is used as the
#  CCSDS Sequence Count value.
#  Note this means the Initial-Sequence-Count value is ignored.
Scan-Line-Number-As-Sequence-Count=yes


# CCSDS Initial Sequence Count (14 bit unsigned int) (default 0)
Initial-Sequence-Count=0


# UTC Ignore Input Time (boolean) (default false)
#  When switched on the times of an input product as modified to
#  begin at midnight 01/01/2000, with times throughout the product
#  all offset by the same value.
#  The three UTC time offset values below are applied afterwards.
UTC-Ignore-Input-Time=false


# UTC Time Offsets (int) (default 0,0,0)
#  Offsets applied to UTC time read from the Level1b UTC time
UTC-Days-Offset=0
UTC-Milliseconds-Offset=0
UTC-Microseconds-Offset=0
```

```
# OBT Time Offsets (int) (default 0,0)
#  Offsets applied to OBT calculated from L1b product.
# OBT equals UTC + Clock Drift value
OBT-Coarse-Offset=0
OBT-Fine-Offset=0


# OBT Fixed Fine Time (boolean) (default false)
#  When this option is used all OBT Fine time values will be set equal to
#  the OBT-Fine-Offset value given above.
OBT-Fixed-Fine-Time=false
```

9.3 AVHRR

```
# MetOpizer AVHRR Configuration File

# Scan Line Number as Sequence Count (boolean) (default yes)
#  When on, the Level 1b Scan Line Number field is used as the
#  CCSDS Sequence Count value.
#  Note this means the Initial-Sequence-Count value is ignored.
Scan-Line-Number-As-Sequence-Count=yes

# CCSDS Initial Sequence Count (14 bit unsigned int) (default 0)
Initial-Sequence-Count=0

# UTC Ignore Input Time (boolean) (default false)
#  When switched on the times of an input product as modified to
#  begin at midnight 01/01/2000, with times throughout the product
#  all offset by the same value.
#  The three UTC time offset values below are applied afterwards.
UTC-Ignore-Input-Time=false

# UTC Time Offsets (int) (default 0,0,0)
#  Offsets applied to UTC time read from the Level1b UTC time
UTC-Days-Offset=0
UTC-Milliseconds-Offset=0
UTC-Microseconds-Offset=0

# OBT Time Offsets (int) (default 0,0)
#  Offsets applied to OBT calculated from L1b product.
# OBT equals UTC + Clock Drift value
OBT-Coarse-Offset=0
OBT-Fine-Offset=0

# OBT Fixed Fine Time (boolean) (default true)
#  When this option is used all OBT Fine time values will be set equal to
#  the OBT-Fine-Offset value given above.
OBT-Fixed-Fine-Time=true
```

9.4 HIRS

```
# MetOpizer HIRS Configuration File

# Scan Line Number as Sequence Count (boolean) (default false)
#  When on, the Level 1b Scan Line Number field is used as the
#  CCSDS Sequence Count value.
#  Note this means the Initial-Sequence-Count value is ignored.
Scan-Line-Number-As-Sequence-Count=yes

# CCSDS Initial Sequence Count (14 bit unsigned int) (default 0)
Initial-Sequence-Count=0

# UTC Ignore Input Time (boolean) (default false)
#  When switched on the times of an input product as modified to
#  begin at midnight 01/01/2000, with times throughout the product
#  all offset by the same value.
#  The three UTC time offset values below are applied afterwards.
UTC-Ignore-Input-Time=false

# UTC Time Offsets (int) (default 0,0,0)
#  Offsets applied to UTC time read from the Level1b UTC time
UTC-Days-Offset=0
UTC-Milliseconds-Offset=0
UTC-Microseconds-Offset=0

# OBT Time Offsets (int) (default 0,0)
#  Offsets applied to OBT calculated from L1b product.
# OBT equals UTC + Clock Drift value
OBT-Coarse-Offset=0
OBT-Fine-Offset=0

# OBT Fixed Fine Time (boolean) (default true)
#  When this option is used all OBT Fine time values will be set equal to
#  the OBT-Fine-Offset value given above.
OBT-Fixed-Fine-Time=true
```

9.5 MHS

```
#
# MetOpizer MHS Configuration File
#

#
# Mode and Sub-Commutation Code
#

# Scan Mode (4 bit unsigned int) (default 3)
#  Valid codes are:
#   0 - Power on
#   1 - Warm up
#   2 - Stand by
#   3 - Scan
#   4 - Fixed view
#   5 - Self Test
#   6 - Safeing
#   7 - Fault
#   8-14 - Unused
#   15 - Memory Data Packet ID

Scan-Mode=3

# PIE ID (1 bit) (default 0)
#  0=PIE-A in use, 1=PIE-B in use

PIE-ID=0

# Sub-Commutation Field (3 bit unsigned int) (defaults 1,1,1,7)
#  The first packet uses SubCommutationInitalValue as the
#  Sub-Commutation code. Subsequent packets add SubCommutationModValue
#  to the code from the previous packet.
#  If the code goes above SubCommutationMaxValue it is wrapped down to
#  SubCommutationMinValue and vice versa.

Sub-Commutation-Initial-Value=1
Sub-Commutation-Mod-Value=1
Sub-Commutation-Min-Value=1
Sub-Commutation-Max-Value=7


#
# Telecommand Acknowledgement And Fault Codes
#


# TC Clean (1 bit) (default 1)
TC-Clean=1

# TC Conforms (1 bit) (default 1)
TC-Conforms=1

# TC Recognised (1 bit) (default 1)
TC-Recognised=1

# TC Legal (1 bit) (default 1)
TC-Legal=1

# FDM Motor Current Trip Status (1 bit) (default 0)
FDM-Motor-Current-Trip-Status=0

# TC Application ID (3 bit unsigned int) (default 0)
TC-Application-ID=0

# TC Application Process ID (8 bit unsigned int) (default 34)
```

```
TC-Application-Process-ID=34

# TC Packet Sequence Count (8 bit unsigned int) (default=0)
TC-Packet-Sequence-Count=0

# TC Packet Sequence (6 bit unsigned int) (default=0)
TC-Packet-Sequence=0

# TC Received Count (2 bit unsigned int) (default=1)
TC-Received-Count=1

# Monitor Current Fault (1 bit) (default 0)
Monitor-Current-Fault=0

# Termistor Monitor Fault (1 bit) (default 0)
Thermistor-Monitor-Fault=0

# Switch Fault (1 bit) (default 0)
Switch-Fault=0

# Processor Fault (1 bit) (default 0)
Processor-Fault=0

# RDM Motor Current Trip Status (1 bit) (default 0)
RDM-Motor-Current-Trip-Status=0

# DC Offset Error (1 bit) (default 0)
DC-Offset-Error=0

# Scan Control Error (1 bit) (default 0)
Scan-Control-Error=0

# REF CK Error (1 bit) (default 0)
REF-CK-Error=0


#
# Switch Status
#


# Receiver Channel H1 On Off (1 bit) (default 1)
Receiver-Channel-H1-On-Off=1

# Receiver Channel H1 Local Oscillator A B Selected (1 bit) (default 0)
Receiver-Channel-H1-Local-Oscillator-A-B-Selected=0

# Receiver Channel H2 On Off (1 bit) (default 1)
Receiver-Channel-H2-On-Off=1

# Receiver Channel H2 Local Oscillator A B Selected (1 bit) (default 0)
Receiver-Channel-H2-Local-Oscillator-A-B-Selected=0

# Receiver Channels H3/H4 Frontend On Off (1 bit) (default 1)
Receiver-Channels-H3-H4-Frontend-On-Off=1

# Receiver Channels H3 H4 Local Oscillator A B Selected (1 bit) (default 0)
Receiver-Channels-H3-H4-Local-Oscillator-A-B-Selected=0

# Receiver Channel H3 Backend On Off (1 bit) (default 1)
Receiver-Channel-H3-Backend-On-Off=1

# Receiver Channel H4 Backend On Off (1 bit) (default 1)
Receiver-Channel-H4-Backend-On-Off=1

# Receiver Channel H5 On Off (1 bit) (default 1)
Receiver-Channel-H5-On-Off=1

# Receiver Channel H5 Local Oscillator A B Selected (1 bit) (default 0)
```

```
Receiver-Channel-H5-Local-Oscillator-A-B-Selected=0

# Rx CV On Off (1 bit) (default 1)
Rx-CV-On-Off=1

# Receiver Operational (1 bit) (default 1)
Receiver-Operational=1

# Scan Mechanism Operational Heaters On Off (1 bit) (default 1)
Scan-Mechanism-Operational-Heaters-On-Off=1

# Auxiliary Operational Heaters On Off (1 bit) (default 1)
Auxiliary-Operational-Heaters-On-Off=1

# Signal Processing Electronics On Off (1 bit) (default 1)
Signal-Processing-Electronics-On-Off=1

# PROM Failure (1 bit) (default 0)
PROM-Failure=0

# RDM Zero Positioning Sensors A/B Selected (1 bit) (default 0)
RDM-Zero-Position-Sensors-A-B-Selected=1

# FDM Zero Positioning Sensors A/B Selected (1 bit) (default 0)
FDM-Zero-Position-Sensors-A-B-Selected=1

# RDM Motor Sensors A/B Selected (1 bit) (default 0)
RDM-Motor-Sensors-A-B-Selected=0

# FDM Motor Sensors A/B Selected (1 bit) (default 0)
FDM-Motor-Sensors-A-B-Selected=0

# RDM Motor Supply On Off (1 bit) (default 1)
RDM-Motor-Supply-On-Off=1

# FDM Motor Supply On Off (1 bit) (default 1)
FDM-Motor-Supply-On-Off=1

# RDM Motor Current Trip Enabled (1 bit) (default 1)
RDM-Motor-Current-Trip-Enabled=1

# FDM Motor Current Trip Enabled (1 bit) (default 1)
FDM-Motor-Current-Trip-Enabled=1


#
# Temperature Data
#


# For the following 24 entries the value given is the thermistor resistance value
# as read from RD-4 Appendix A (A table giving mappings for temperature
# to resistance). RD.4 Section 3.1 gives more description of
# the algorithms used.

# The results of the calculation for each of the 24 values is a single 8 bit
# unsigned integer. This is clipped to 0 or 255 if required.

# The formula used to convert resistance value (Resistance-T* below)
# to code values (to be placed inside the source packet) is as follows
#
#  code = 266.175 * resistance / (resistance + 15000)
#
# where resistance is in ohms.
#
# The temperature data representation is not affected by different instrument
# models.
#
# See RD.2 for justification of the values selected as defaults.
```

```
# T1 (temperature in degC) (default 25.0)
Temperature-T1=25.0

# T2 (temperature in degC) (default 25.0)
Temperature-T2=25.0

# T3 (temperature in degC) (default 25.0)
Temperature-T3=25.0

# T4 (temperature in degC) (default 25.0)
Temperature-T4=25.0

# T5 (temperature in degC) (default 25.0)
Temperature-T5=25.0

# T6 (temperature in degC) (default 25.0)
Temperature-T6=25.0

# T7 (temperature in degC) (default 25.0)
Temperature-T7=25.0

# T8 (temperature in degC) (default 25.0)
Temperature-T8=25.0

# T9 (temperature in degC) (default 25.0)
Temperature-T9=25.0

# T10 (temperature in degC) (default 25.0)
Temperature-T10=25.0

# T11 (temperature in degC) (default 25.0)
Temperature-T11=25.0

# T12 (temperature in degC) (default 25.0)
Temperature-T12=25.0

# T13 (temperature in degC) (default 20.0)
Temperature-T13=20.0

# T14 (temperature in degC) (default 20.0)
Temperature-T14=20.0

# T15 (temperature in degC) (default 20.0)
Temperature-T15=20.0

# T16 (temperature in degC) (default 20.0)
Temperature-T16=20.0

# T17 (temperature in degC) (default 10.0)
Temperature-T17=10.0

# T18 (temperature in degC) (default 10.0)
Temperature-T18=10.0

# T19 (temperature in degC) (default 10.0)
Temperature-T19=10.0

# T20 (temperature in degC) (default 30.0)
Temperature-T20=30.0

# T21 (temperature in degC) (default 31.0)
Temperature-T21=31.0

# T22 (temperature in degC) (default 30.0)
Temperature-T22=30.0

# T23 (temperature in degC) (default 30.0)
Temperature-T23=30.0
```

```
# T24 (temperature in degC) (default 26.0)
Temperature-T24=26.0


#
# Raw Current Consumption Data
#


# The following 6 settings (Raw Current Consumption) all rely
# on one or two constants read from the following table.

# All current conversions rely on 9 constants from this section.
# Note each instrument model has its own set of constants.
# The constants for 5 models (EM1, PFM, FM2, FM3, FM4) are written
# out below.
# Only one set should be uncommented.
# If this configuration file cannot be read, the constants for FM2
# are used by default.

# EM1 constants begin
5V-Secondary-Current-k=0.01644
8V-Secondary-Current-k1=0.02105
8V-Secondary-Current-k2=-0.1067
15V-Receiver-Current-k1=0.007869
15V-Receiver-Current-k2=-0.07271
N15V-Receiver-Current-k1=0.001852
N15V-Receiver-Current-k2=-0.026218
RDM-Motor-Current-k=0.01307
FDM-Motor-Current-k=0.01307

# EM2 constants begin
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02105
#8V-Secondary-Current-k2=-0.1067
#15V-Receiver-Current-k1=0.007869
#15V-Receiver-Current-k2=-0.07271
#N15V-Receiver-Current-k1=0.001852
#N15V-Receiver-Current-k2=-0.026218
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337

# PFM constants begin
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02221
#8V-Secondary-Current-k2=-0.1364
#15V-Receiver-Current-k1=0.008207
#15V-Receiver-Current-k2=-0.09704
#N15V-Receiver-Current-k1=0.001875
#N15V-Receiver-Current-k2=-0.006094
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337

# FM2 constants begin
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02242
#8V-Secondary-Current-k2=-0.1371
#15V-Receiver-Current-k1=0.008054
#15V-Receiver-Current-k2=-0.10122
#N15V-Receiver-Current-k1=0.001893
#N15V-Receiver-Current-k2=-0.009060
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337

# FM3 constants begin
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02189
```

```
#8V-Secondary-Current-k2=-0.1277
#15V-Receiver-Current-k1=0.008110
#15V-Receiver-Current-k2=-0.10523
#N15V-Receiver-Current-k1=0.001882
#N15V-Receiver-Current-k2=-0.001686
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337

# FM4 constants begin
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02167
#8V-Secondary-Current-k2=-0.1226
#15V-Receiver-Current-k1=0.007898
#15V-Receiver-Current-k2=-0.07176
#N15V-Receiver-Current-k1=0.001883
#N15V-Receiver-Current-k2=-0.003277
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337

# The formulas used to convert amps to source packet codes are as
# follows:

# For channels 5.1, 5.2 and 5.3 (5V Secondary Current, 8V Receiver
# Current and 15V Receiver Current) :
#
#  code = current / k
#
# where code is the 8 bit unsigned int to be placed into the source packet
# clipped to 0 or 255 if needed, current is in milliamps and k is read
# from above.
#
# For channels 5.4, 5.5 and 5.6 (-15V Receiver Current, RDM Motor Current
# and FDM Motor Current) this formula is used:
#
#  code = (current - k2) / k1
#
# where code is the 8 bit unsigned int to be placed in the source packet,
# clipped to 0 or 255 as needed, current is in milliamps and k1 and k2
# are read from above.
#
# All formulas and constants taken from RD.4 Section 3.3.
#

# +5V Secondary Current (default 900mA)
5V-Secondary-Current=900

# +8V Receiver Current (default 1444mA)
8V-Secondary-Current=1444

# +15V Receiver Current (default 831mA)
15V-Receiver-Current=831

# -15V Receiver Current (default 169mA)
N15V-Receiver-Current=169

# RDM Motor Current (default 668mA)
RDM-Motor-Current=668

# FDM Motor Current (default 668mA)
FDM-Motor-Current=668


#
# Status Word
#


# DC Offset Valid (1 bit) (default 1)
DC-Offset-Valid=1
```

```
# Scan Control Valid (1 bit) (default 1)
Scan-Control-Valid=1

# Profile (2 bit unsigned int) (default 0)
Profile=0



#
# Signal Processing Status
#


# Channel H1 DC Offset Word (8 bit unsigned int) (default 0)
Channel-H1-DC-Offset-Word=0

# Channel H2 DC Offset Word (8 bit unsigned int) (default 0)
Channel-H2-DC-Offset-Word=0

# Channel H3 DC Offset Word (8 bit unsigned int) (default 0)
Channel-H3-DC-Offset-Word=0

# Channel H4 DC Offset Word (8 bit unsigned int) (default 0)
Channel-H4-DC-Offset-Word=0

# Channel H5 DC Offset Word (8 bit unsigned int) (default 0)
Channel-H5-DC-Offset-Word=0

# H1 Valid (1 bit) (default 1)
H1-Valid=1

# H2 Valid (1 bit) (default 1)
H2-Valid=1

# H3 Valid (1 bit) (default 1)
H3-Valid=1

# H4 Valid (1 bit) (default 1)
H4-Valid=1

# H5 Valid (1 bit) (default 1)
H5-Valid=1

# SPE Mux Code (3 bits) (default 3)
SPE-Mux-Code=3

# H1 Gain (3 bits) (default 0)
H1-Gain=0

# H2 Gain (3 bits) (default 0)
H2-Gain=0

# H3 Gain (3 bits) (default 0)
H3-Gain=0

# H4 Gain (3 bits) (default 0)
H4-Gain=0

# H5 Gain (3 bits) (default 0)
H5-Gain=0


#
# OBCT Temperature Data
#


# Note the following values are used to calibrate the OBCT thermistors
```

```
# resistance measurement device.

# In each case the resistance meter is connected to a fixed known resistance
# and the values sampled.

# The relevant formula is based on RD.4 Section 3.2 as:
#
#  code = (34.744 * resistance) + 21.84
#
# where code is a 12 bit unsigned value, clipped to 0 or 4095 if needed
# and resistance is in ohms.

# Note the formula above is for information only.
# The values listed below are copied unmodified by the MetOpizer into
# its source packets.

# Measurement of 118 ohm fixed resistance (12 bit unsigned int) (default 4122)
PRT-CAL-1=3540

# Measurement of 95.3 ohm fixed resistance (12 bit unsigned int) (default 3333)
PRT-CAL-2=1523

# Measurement of 80.6 ohm fixed resistance (12 bit unsigned int) (default 2822)
PRT-CAL-3=217


#
# Shaft Position
#

# Shaft Position Offset (integer) (default 18638)
#  This fixed value is added to each AMSU-B Shaft Position value
#  to form the MHS Mid-Pixel Position value
Shaft-Position-Offset=18638


#
# CCSDS Packet Generation
#


# Scan Line Number as Sequence Count (boolean) (default yes)
#  When on, the Level 1b Scan Line Number field is used as the
#  CCSDS Sequence Count value and Initial Sequence Count becomes
#  an offset to this value

#  When the Scan Line Number As Sequence Count flag is set the
#  Initial Sequence Count becomes instead an offset to the
#  scna line number.

Scan-Line-Number-As-Sequence-Count=yes

# CCSDS Initial Sequence Count (14 bit unsigned int) (default 0)

Initial-Sequence-Count=0

# UTC Ignore Input Time (boolean) (default false)
#  When switched on the times of an input product as modified to
#  begin at midnight 01/01/2000, with times throughout the product
#  all offset by the same value.
#  The three UTC time offset values below are applied afterwards.

UTC-Ignore-Input-Time=false


# UTC Time Offsets (int) (defaults 0,0,0)
#  Offsets applied to UTC time read from the Level1b UTC time

UTC-Days-Offset=0
```

```
UTC-Milliseconds-Offset=0
UTC-Microseconds-Offset=0

# OBT Starting Time (int) (defaults 0,0)
#  The first packet generated will be stamped with OBT as given below.
#  Subsequent packets times will be offset by comparing the scan line
#  UTC time values.

OBT-Coarse-Offset=0
OBT-Fine-Offset=100

# OBT Fixed Fine Time (boolean) (default false)
#  When this option is used all OBT Fine time values will be set equal to
#  the OBT-Fine-Offset value given above.

OBT-Fixed-Fine-Time=true
```

9.6 ccsds_displayer

This configuration file is not used for generating source packets, but is needed by the ccsds_displayer program to decode MHS electrical current values from instrument counts to amps.

```
# constants used by ccsds_displayer for converting
# MHS current values to engineering units.

# EM1 constants begin
Current-Constants-Name=EM1
5V-Secondary-Current-k=0.01644
8V-Secondary-Current-k1=0.02105
8V-Secondary-Current-k2=-0.1067
15V-Receiver-Current-k1=0.007869
15V-Receiver-Current-k2=-0.07271
N15V-Receiver-Current-k1=0.001852
N15V-Receiver-Current-k2=-0.026218
RDM-Motor-Current-k=0.01307
FDM-Motor-Current-k=0.01307

# EM2 constants begin
#Current-Constants-Name=EM2
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02105
#8V-Secondary-Current-k2=-0.1067
#15V-Receiver-Current-k1=0.007869
#15V-Receiver-Current-k2=-0.07271
#N15V-Receiver-Current-k1=0.001852
#N15V-Receiver-Current-k2=-0.026218
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337

# PFM constants begin
#Current-Constants-Name=PFM
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02221
#8V-Secondary-Current-k2=-0.1364
#15V-Receiver-Current-k1=0.008207
#15V-Receiver-Current-k2=-0.09704
#N15V-Receiver-Current-k1=0.001875
#N15V-Receiver-Current-k2=-0.006094
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337

# FM2 constants begin
#Current-Constants-Name=FM2
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02242
#8V-Secondary-Current-k2=-0.1371
#15V-Receiver-Current-k1=0.008054
#15V-Receiver-Current-k2=-0.10122
#N15V-Receiver-Current-k1=0.001893
#N15V-Receiver-Current-k2=-0.009060
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337
```

```
# FM3 constants begin
#Current-Constants-Name=FM3
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02189
#8V-Secondary-Current-k2=-0.1277
#15V-Receiver-Current-k1=0.008110
#15V-Receiver-Current-k2=-0.10523
#N15V-Receiver-Current-k1=0.001882
#N15V-Receiver-Current-k2=-0.001686
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337

# FM4 constants begin
#Current-Constants-Name=FM4
#5V-Secondary-Current-k=0.01681
#8V-Secondary-Current-k1=0.02167
#8V-Secondary-Current-k2=-0.1226
#15V-Receiver-Current-k1=0.007898
#15V-Receiver-Current-k2=-0.07176
#N15V-Receiver-Current-k1=0.001883
#N15V-Receiver-Current-k2=-0.003277
#RDM-Motor-Current-k=0.01337
#FDM-Motor-Current-k=0.01337
```

9.7 ccsds_displayer - IASI Spectrum

In order to decode the IASI spectrum application data field, the value of a number of parameters that are not included in the CCSDS packets need to be known. These parameters are:

Parameter	Description
IctcNbSeg	Number of coding regions
IdefCtcNbNsSeg	Number of samples per coding region
ICtcTabNbBit[*550][1]	Number of bits of each sample per coding region
ICtcTabNbBit[*550][2]	Offset of each sample per coding region
ICtcTabNbBit[*550][3]	Scale of each sample per coding region

*550 represents the maximum number of coding regions that can be used. Note that the number of coding regions can be reduced but not increased.

The IASI instrument stores its spectra measurements as 32-bit floating point values on board the satellite. However, due the large amounts of data involved, transmitting the data back to earth in such a format is difficult. Hence, the spectral data is divided into regions and for each region a number of bits, a scale and an offset is defined. The exact values of the number of bits, scale and offset are calculated by operation teams and uploaded to the satellite. In order to decode the spectral data received from the satellite, the same values initially sent up have to be used.

According to RD17, the COD algorithm implemented on board the satellite converts the measured spectral data, BMrgSpect into coded, Cod, values using the following equation:

$$Cod = Scale * BMrgSpect - Offset + 1$$

Knowing the values of Cod, Scale and Offset, the measured spectral data can be retrieved using the following equation:

```
BMrgSpect = (Cod –1 + Offset) / Scale
```

To decode the simulated IASI CCSDS packets produced by Noveltis, additional functionality has been added to the ccsds_displayer to allow it to extract and use the necessary fields described above from a Noveltis format configuration file. Note that the Noveltis configuration file contains many fields, the ccsds_dispalyer ignores all the other fields and only extracts the ones of interest. The listing below illustrates the basic format of a typical Noveltis configuration file, where the fields of interest have been highlighted in a bold font.

```
                S.IDefIssueIcd :          1
              S.IDefRevisionIcd :          5
                     S.IDefPTSI :          1
                   S.IDefIdConf :          9
                    S.IDefSdbID :          0
             S.IDefStableParamID :          0
             S.IDefParamDate.day :        800
              S.IDefParamDate.ms :          1
              S.IDefMETOPNumber :          1
                     S.ICtcNbSeg :     1        521
    S.IDefCtcNsegfirstNsfirst :     1       2536
           S.ICtcTabOffsetC M :     1       0.00       0.03
            S.ICtcTabScaleC M :     1 240078.36     274.38
S.ICtcTabNbBit Offset Scale :     1     1          8       3863        896
S.ICtcTabNbBit Offset Scale :     1     2          8       3477        697
S.ICtcTabNbBit Offset Scale :     1     3          9       3863        886
S.ICtcTabNbBit Offset Scale :     1     4          5       3670        873
S.ICtcTabNbBit Offset Scale :     1     5          4       3091        701
                                      .
                                      .
                                      .
S.ICtcTabNbBit Offset Scale :     1   517          8          0      41189
S.ICtcTabNbBit Offset Scale :     1   518          7          0      40288
S.ICtcTabNbBit Offset Scale :     1   519          8          0      42285
S.ICtcTabNbBit Offset Scale :     1   520          8          0      41357
S.ICtcTabNbBit Offset Scale :     1   521          8          0      40454
              S.IDefArcNbNsSeg :     1       400
                S.IDefArcNbSeg :     1         8
              S.IDefArcNbNsSeg :     2       400
                S.IDefArcNbSeg :     2        10
                               .
                               .
                               .
           S.IDefCcsSplitCutoff :  30.00000
    S.IDefCovarEigenValCalRad :   1.00000
     S.IDefCovarEigenValSpect :   1.00000
             S.IDefCtcNbNsSeg :        16
        S.IDefCovarEigenVal1b :    1   0.00000   0.00000
        S.IDefCovarEigenVal1b :   11   0.00000   0.00000
                               .
                               .
                               .
```

9.8 tVCDU

This configuration files is used by the cadu_to_tvcdu program.

```
# constants used by cadu_to_tvcdu for generating
# tVCDU packets

# Initial offset between time of first CCSDS packet in
# input file and time ofset for inital tVCDU packet
# (milliseconds)
# Default 110 minutes (110*60*1000)

UTCTimeOffset = 6600000

# Time difference between successive tVCDU packets
# (microseconds)
# Default 80us

UTCTimeDifference = 80

# Reed Solomon Status value
# (bitfield, 8 bits)
# Default 10000000 (128)

ReedSolomonStatus = 128

# String VCDU filler packets
# Default false

StripVCDUFillerPackets = false
```

9.9 ccsds_strip_filter

```
# Configurable parameters used by ccsds_strip_filter to
# removes CCSDS packets that do not satisfy the UTC start
# and end times.

# Only CCSDS packets with UTC times equal to or later than
# the specified start times below are kept.

# utc_days_start
# Default=0 MIN_Value=0 MAX_Value=65535
utc_days_start=0

# utc_milliseconds_start
# Default=0 MIN_Value=0 MAX_Value=86399999
utc_milliseconds_start=0

# utc_microseconds_start
# Default=0 MIN_Value=0 MAX_Value=999
utc_microseconds_start=0

# Only CCSDS packets with UTC times equal to or earlier than
# the specified end times below are kept.
```

```
# utc_days_end
# Default=0 MIN_Value=0 MAX_Value=65535
utc_days_end=65535

# utc_milliseconds_end
# Default=0 MIN_Value=0 MAX_Value=86399999
utc_milliseconds_end=86399999

# utc_microseconds_end
# Default=0 MIN_Value=0 MAX_Value=999
utc_microseconds_end=999

# time_gap_limit is used to set the acceptable difference between
# the UTC time of one CCSDS packet and the next packet in seconds.
# Default=600 MIN_Value=0
time_gap_limit=600.0
```

9.10 ccsds_time_fix

```
# Configurable parameters used by ccsds_time_fix to
# set the CCSDS packet UTC times.

# Use to set the UTC time of the first CCSDS packet.
# utc_days_start
# Default=0 MIN_Value=0 MAX_Value=65535
utc_days_start=0

# utc_milliseconds_start
# Default=0 MIN_Value=0 MAX_Value=86399999
utc_milliseconds_start=0

# utc_microseconds_start
# Default=0 MIN_Value=0 MAX_Value=999
utc_microseconds_start=0

# Use to define the time increment between the first
# CCSDS packet and the subsequent packets that follow.

# utc_days_increment
# Default=0 MIN_Value=0 MAX_Value=65535
utc_days_increment=0

# utc_milliseconds_increment
# Default=0 MIN_Value=0 MAX_Value=86399999
utc_milliseconds_increment=5000

# utc_microseconds_increment
# Default=0 MIN_Value=0 MAX_Value=999
utc_microseconds_increment=0
```

9.11 ccsds_seq_fix

```
# Configurable parameters used by ccsds_seq_fix for
```

```
# modifying/offseting the sequence counter in the CCSDS
# primary header and the OBT time field of the CCSDS
# ancillary data packet.

# COUNTER_INITIAL sets the first packet sequence counter value
# Default=1 MIN_Value=0 MAX_Value=16383
COUNTER_INITIAL=0
#COUNTER_INITIAL_1=1
#COUNTER_INITIAL_2=3
#COUNTER_INITIAL_3=4
#COUNTER_INITIAL_4=5


# OBT_COARSE_INITIAL sets the first packet OBT Coarse
# time (milliseconds) Default=0 MIN_Value=0 MAX_Value=16777215
OBT_COARSE_INITIAL=0

# OBT_FINE_INITIAL sets the first packet OBT Fine
# time (milliseconds) Default=0 MIN_Value=0 MAX_Value=65535
OBT_FINE_INITIAL=0
```

9.12 ccsds_to_l0

```
#
# MetOpizer L0 Production Configuration File
#

product_type=xxx
spacecraft_id=M01
instrument_model=1
processing_centre=CGS1
processing_mode=N
disposition_mode=O
processor_major_version=1
processor_minor_version=0
receiving_ground_station=SVL
```

10 APPENDIX C (REMOVED)

11 APPENDIX D – IASI FLAGS FIX LISTING

This section contains a full list of all the IASI CCSDS packet flags that the ccsds_iasi_flags_fix code inverts. Unless one is highly familiar with the IASI packet formats, the list below is best read and understood in combination with the IASI Measurement and Verification Data Document RD13.

- Packet Primary Header
  Word 1
  B3 – Type Indicator
  Word 2
  B0-1 – Sequence Flag (always = 3)

- Spectrum Ancillary Info Field
  Word 13
  Operational Data Word 3
  B1 - CSQ
  B2 - SQ1
  B3 - SQ2
  B4 - IEQ
  B8 - SN_NV
  B9 - CD_NV
  B10 - CSQ_NV
  B11 - SP_NV
  B12 - SQ1_NV
  B13 - SQ2_NV
  B14 - NS_NV
  B15 - IEQ_NV

- Spectrum Ancillary Info Field
  Word 150
  Status Area Word 1
  B0 - DPS LNR-Rx DVL flag
  B1 - DPS LNR-Rx VLN flag
  B2 - BBofFlagSpectNonQual
  B3 - BDcoFlagMasErrorPath_B1
  B4 - BDcoFlagMasErrorPath_B2
  B5 - BDcoFlagMasErrorPath_B3
  B6 - BDcoFlagMasOverflow_B1
  B7 - BDcoFlagMasOverflow_B2
  B8 - BDcoFlagMasOverflow_B3
  B9 - BDcoFlagMasEcret_B1
  B10 - BDcoFlagMasEcret_B2
  B11 - BDcoFlagMasEcret_B3
  B12 - BDcoFlagMasErrorNbWords

- Spectrum Ancillary Info Field

Word 151
Status Area Word 2
B0 - BSpkFlagSpik_B1
B1 - BSpkFlagSpik_B2
B2 - BSpkFlagSpik_B3
B3 - BzpdFlagNzpdNonQualEW
B4 - BIsiFlagErrorFft_B1
B5 - BIsiFlagErrorFft_B2
B6 - BIsiFlagErrorFft_B3
B7 - BArcFlagCalSpectNonQual_B1
B8 - BArcFlagCalSpectNonQual_B2
B9 - BArcFlagCalSpectNonQual_B3
B10 - BCodFlagFlood
B11 - BDcoFlagErrorInterf_B1
B12 - BDcoFlagErrorInterf_B2
B13 - BDcoFlagErrorInterf_B3

- Image Ancillary Info Field
  Word 13
  Operational Data Word 3
  B1 - CSQ
  B2 - SQ1
  B3 - SQ2
  B4 - IEQ
  B8 - SN_NV
  B9 - CD_NV
  B10 - CSQ_NV
  B11 - SP_NV
  B12 - SQ1_NV
  B13 - SQ2_NV
  B14 - NS_NV
  B15 - IEQ_NV

- Image Ancillary Info Field
  Word 21
  Image Status Area
  B8 - IIS ADC Overflow Flag
  B9 - No. of IIS Samples Flag

- Spectrum Verification Ancillary Info Field
  Word 13
  Operational Data Word 3
  B1 - CSQ
  B2 - SQ1
  B3 - SQ2
  B4 - IEQ
  B8 - SN_NV
  B9 - CD_NV
  B10 - CSQ_NV

B11 - SP_NV
B12 - SQ1_NV
B13 - SQ2_NV
B14 - NS_NV
B15 - IEQ_NV

- Spectrum Verification Ancillary Info Field
  Word 21
  Verification Status Area
  B8 - BZpdFlagNzpdNonQual
  B9 - BdcoFlagMasErrorNbWords

- Spectrum Verification Application Data Field Type A and B
  MAS Interferogram Sample
  B0 - Spike Detection Flag
  B1 - ADC Overflow Flag

- Auxiliary Ancillary Info Field
  Word 28
  OTM data, OTM_NV
  B0 - BBT Validity Flag
  B1 - FPT Validity Flag (always = 1)
  B2 - HAUT Validity Flag (always = 1)
  B3 - IFPT Validity Flag (always = 1)
  B4 - OBPT Validity Flag (always = 1)
  B5 - CCAT Validity Flag (always = 1)

- Auxiliary Application Data Field
  Word 73, 88, 103, 118 - Main Data Area for Pixel 1 (SN=32,33,35,36)
  Word 133, 148, 163, 178 - Main Data Area for Pixel 2 (SN=32,33,35,36)
  Word 193, 208, 223, 238 - Main Data Area for Pixel 3 (SN=32,33,35,36)
  Word 253, 268, 283, 298 - Main Data Area for Pixel 4 (SN=32,33,35,36)
  Operational Data Word 3
  B1 - CSQ
  B2 - SQ1
  B3 - SQ2
  B4 - IEQ
  B8 - SN_NV
  B9 - CD_NV
  B10 - CSQ_NV
  B11 - SP_NV
  B12 - SQ1_NV
  B13 - SQ2_NV
  B14 - NS_NV
  B15 - IEQ_NV

- Auxiliary Application Data Field
  Word 311, 315, 319, 323 - Main Status Area for Pixel 1 (SN=32,33,35,36)
  Word 331, 335, 339, 343 - Main Status Area for Pixel 2 (SN=32,33,35,36)

Word 351, 355, 359, 363 - Main Status Area for Pixel 3 (SN=32,33,35,36)
Word 371, 375, 379, 383 - Main Status Area for Pixel 4 (SN=32,33,35,36)
Status Area Word 1
B0 - DPS LNR-Rx DVL flag
B1 - DPS LNR-Rx VLN flag
B2 - BBofFlagSpectNonQual
B3 - BDcoFlagMasErrorPath_B1
B4 - BDcoFlagMasErrorPath_B2
B5 - BDcoFlagMasErrorPath_B3
B6 - BDcoFlagMasOverflow_B1
B7 - BDcoFlagMasOverflow_B2
B8 - BDcoFlagMasOverflow_B3
B9 - BDcoFlagMasEcret_B1
B10 - BDcoFlagMasEcret_B2
B11 - BDcoFlagMasEcret_B3
B12 - BdcoFlagMasErrorNbWords
B13 - BDcoFlagErrorInterf_B1
B14 - BDcoFlagErrorInterf_B2
B15 - BDcoFlagErrorInterf_B3

- Auxiliary Application Data Field
  Word 312, 316, 320, 324 - Main Status Area for Pixel 1 (SN=32,33,35,36)
  Word 332, 336, 340, 344 - Main Status Area for Pixel 2 (SN=32,33,35,36)
  Word 352, 356, 360, 364 - Main Status Area for Pixel 3 (SN=32,33,35,36)
  Word 372, 376, 380, 384 - Main Status Area for Pixel 4 (SN=32,33,35,36)
  Status Area Word 2
  B3 - BSpkFlagSpik_B1
  B4 - BSpkFlagSpik_B2
  B5 - BSpkFlagSpik_B3
  B6 - BzpdFlagNzpdNonQualEW
  B7 - BIrsFlagSrdNonIntegrity
  B8 - BIsiFlagErrorFft_B1
  B9 - BIsiFlagErrorFft_B2
  B10 - BIsiFlagErrorFft_B3

- Auxiliary Application Data Field
  Word 327 - Main Status Area for Pixel 1
  Word 347 - Main Status Area for Pixel 2
  Word 367 - Main Status Area for Pixel 3
  Word 387 - Main Status Area for Pixel 4
  Extra Status Area Word 1
  B0 - BBofFlagSrdInit_CD0
  B1 - BBofFlagSrdInit_CD1
  B2 - BBofFlagSrdNonUpdate_CD0
  B3 - BBofFlagSrdNonUpdate_CD1
  B4 - BBofFlagCoefCalInit_B1_CD0
  B5 - BBofFlagCoefCalInit_B2_CD0
  B6 - BBofFlagCoefCalInit_B3_CD0

- Auxiliary Application Data Field
  Word 328 - Main Status Area for Pixel 1
  Word 348 - Main Status Area for Pixel 2
  Word 368 - Main Status Area for Pixel 3
  Word 388 - Main Status Area for Pixel 4
  Extra Status Area Word 2
  B0 - BBofFlagCoefCalInit_B1_CD1
  B1 - BBofFlagCoefCalInit_B2_CD1
  B2 - BBofFlagCoefCalInit_B3_CD1
  B3 - BBofFlagCoefCalNonUpdate_B1_CD0
  B4 - BBofFlagCoefCalNonUpdate_B2_CD0
  B5 - BBofFlagCoefCalNonUpdate_B3_CD0
  B6 - BBofFlagCoefCalNonUpdate_B1_CD1
  B7 - BBofFlagCoefCalNonUpdate_B2_CD1
  B8 - BBofFlagCoefCalNonUpdate_B3_CD1
  B9 - BRciFlagNonIntegritySlope_B1_CD0
  B10 - BRciFlagNonIntegritySlope_B2_CD0
  B11 - BRciFlagNonIntegritySlope_B3_CD0
  B12 - BRciFlagNonIntegritySlope_B1_CD1
  B13 - BRciFlagNonIntegritySlope_B2_CD1
  B14 - BRciFlagNonIntegritySlope_B3_CD1

- Auxiliary Application Data Field
  Word 329 - Main Status Area for Pixel 1
  Word 349 - Main Status Area for Pixel 2
  Word 369 - Main Status Area for Pixel 3
  Word 389 - Main Status Area for Pixel 4
  Extra Status Area Word 3
  B0 - BRciFlagNonIntegrityOffset_B1_CD0
  B1 - BRciFlagNonIntegrityOffset_B2_CD0
  B2 - BRciFlagNonIntegrityOffset_B3_CD0
  B3 - BRciFlagNonIntegrityOffset_B1_CD1
  B4 - BRciFlagNonIntegrityOffset_B2_CD1
  B5 - BRciFlagNonIntegrityOffset_B3_CD1