NWP SAF

AAPP Documentation:
OPS-LRS User Manual

Version 1.5

24th January 2014

OPS-LRS User Manual

This documentation was developed within the context of the EUMETSAT Satellite Application Facility on Numerical Weather Prediction (NWP SAF), under the Cooperation Agreement dated 16th December 2003, between EUMETSAT and the Met Office, UK, by one or more partners within the NWP SAF. The partners in the NWP SAF are the Met Office, ECMWF, KNMI and Météo France.

Change record				
Version	**Date**	**Author**	**Approved**	**Remarks**
0.1	21.03.06	P Marguinaud	-	Initial draft
0.2	29.03.06	N C Atkinson	-	Expand and transfer to NWP SAF template
0.3	26.04.06	N C Atkinson	-	Update for OPS v3-5
0.4	26.06.06	PM and NCA	-	Add section on perl scripts and new test case
1.0	27.09.06	N C Atkinson	R A Francis	Initial approved version
1.1	13.03.07	PM and NCA	R A Francis	OPS-LRS updated for post-launch IASI data
1.2	01.06.07	PM		Add explanations about OPS_TIMEOUT
1.3	16/06/10	P Brunel and NCA	R A Francis	Add FFTPack and NumRec capability. Update for OPS v5-0. Update section on test cases
1.4	07/11/11	P Brunel and NCA	S J Keogh	Update for OPS v6-0, patch 1
1.5	24/01/14	NCA		Expand section on environment variables.

Table of Contents

1. INTRODUCTION

This document is the user manual for the IASI "Operation Software – Local Reception Station", or OPS-LRS. The software is based on the original OPS software that was supplied to EUMETSAT by CNES, in association with Thales Information Systems, for use in the EPS Core Ground Segment. It has been modified by the NWP SAF for portability and to allow distribution with AAPP.

The OPS-LRS processes IASI instrument data from Level 0 (raw instrument data) through to level 1c (calibrated, geolocated, Gaussian-apodised radiances).

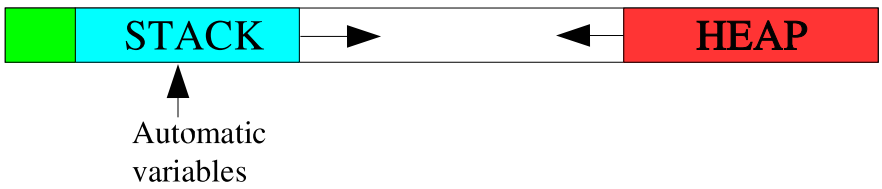
In this document, section 2 gives some technical background while section 3 explains how to build the software. Users who just want to use the supplied top-level scripts may on first reading skip over the following sections on software design (section 4) and using OPS-LRS (section 5). Instructions on running the top-level scripts and running the supplied test case are in sections 6, 7 and 8.

This document release refers to OPS-LRS version 6.0, patch 1. The main issue of this release is that the IASI configuration files no longer need to be converted to local machine endianness; all configuration files should be Big Endian as delivered by EUMETSAT.

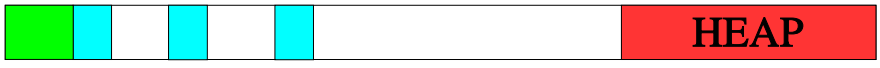
2. TECHNICAL BACKGROUND

2.1 Multithreading

Memory layout - single-threaded



Memory layout – multi-threaded



OPS-LRS makes use of multithreading; in such an environment, the layout of the memory is different from what it is in a single threaded environment: several threads share global static data (green area), and dynamically allocated data (red area). Each thread has its own private stack (blue

areas), with local variables. Obviously, there is a risk one stack grows and step onto its neighbour's stack, therefore, a multithreaded program must allocate dynamically the biggest data structures, and avoid allocating large amounts of data on the stack. The size of threads stacks in OPS-LRS is controlled by the macro `OPS_PTHREAD_STACK_SIZE`.

The right compiler options should be provided for multithreading; the following output from g77/objdump for the subroutine `FUNC` shows that the use of -fno-automatic should be avoided for g77. This means the compiler has to issue thread-safe code, otherwise, the software will crash without giving much explanation.

```
SUBROUTINE FUNC()
REAL X, Y
...
END SUBROUTINE
```

g77 -fno-automatic

```
[philou@kaitain tmp]$ objdump -t func.o

func.o:     file format elf32-i386

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 ccO4ddge.f
00000000 l    d  .text  00000000
00000000 l    d  .data  00000000
00000000 l    d  .bss   00000000
00000000 l     O .bss   00000004 x.0
00000004 l     O .bss   00000004 y.1
00000000 l    d  .note.GNU-stack        00000000
00000000 l    d  .comment       00000000
00000000 g     F .text  00000005 func_
```

```
[philou@kaitain tmp]$ objdump -t func.o

func.o:     file format elf32-i386

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 ccyMXVWw.f
00000000 l    d  .text  00000000
00000000 l    d  .data  00000000
00000000 l    d  .bss   00000000
00000000 l    d  .note.GNU-stack
00000000 l    d  .comment       00000000
00000000 g     F .text  00000008 func_
```

Therefore, it is essential to **read your compiler manpage**, focussing on thread-related issues.

Multithreading is nice, because it allows several threads of execution within the same program, but at the same time, it introduces new problems: what if a variable is accessed for read/write by several threads of execution? This can lead to (mysterious) crashes; for instance:

```
SUBROUTINE FUNC()
COMMON / GLOB / N
IF( N .EQ. 0 ) THEN
  N = N + 1
ENDIF
PRINT *, N
END SUBROUTINE
```

If two threads enter simultaneously the FUNC routine, then the value of N is undefined, because these two threads could have both incremented the N variable, or just one of them could have done so.

Therefore, global variables shall be protected by locks.

2.2 XML

OPS-LRS uses the XML language to communicate with the outside; it takes input written in this format, and generates reports in XML format too.

XML is a markup language with a very strict syntax; non-authorized characters, non-closed markups, etc., are prohibited, and will result in an error.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PPF_Work_Order SYSTEM "PPF_Work_Order.dtd">
<PPF_Work_Order>
<ProcessingType>L1a</ProcessingType>
<SensingStart>20020808181256Z</SensingStart>
<SensingStop>20020808181530Z</SensingStop>
<TimeIntervalFlag>First</TimeIntervalFlag>
```

<TimeIntervalCounter>1</TimeIntervalCounter>
<UnProcData>input/unproc_data/IASI_xxx_00_M01_20020808181248Z_20020808181530Z_B_O_20020810170509Z</UnProcData>
<AuxData>input/aux_data/IASI_BRD_xx_M01_20020101181957Z_20100101000000Z_20020910125223Z_IAST_00000000000</AuxData>
<AuxData>input/aux_data/IASI_BRD_xx_M01_20020101181957Z_20100101000000Z_20020910125223Z_IAST_00000000001</AuxData>
<AuxData>input/aux_data/IASI_CTX_xx_M01_20011008025258Z_20100101000000Z_20021008165703Z_IAST_xxxxxx01001</AuxData>
<AuxData>input/aux_data/IASI_CTX_xx_M01_20020808181452Z_xxxxxxxxxxxxxxx_20020808184711Z_IAST_xxxxxx00002</AuxData>
...

2.3 FORTRAN77 / C

OPS-LRS is written in C, C++ and Fortran 90 compatible Fortran 77.

This means Fortran is called from within C, and we had to deduce the names the F77 compiler gives to external symbols.

Here is what the g77 compiler does with subroutine names:

- `SUBROUTINE FUNC   -> func_`
- `SUBROUTINE FU_NC  -> fu_nc__`

We give below some examples of function prototyping in C and F77:

```
SUBROUTINE FUNC( X, I )          void func_( real* x, int* i );
REAL X, I
...
END SUBROUTINE

SUBROUTINE FUNC( X )             void func_( real x[20][10] );
REAL X( 10, 20 )
...
END SUBROUTINE

SUBROUTINE FUNC( S )             void func_( char* s, int len );
CHARACTER*(*) S
...
END SUBROUTINE
```

We have tried to reduce as much as possible the interaction between C and F77, but pre-existing AAPP software, and free math libraries were implemented in F77, so we had no choice; however, we do not encourage the reader to do that.

Fortran and C interfacing is controlled by the FORTRAN_UDC environment variable:

- U stands for underscore; most Fortran compilers append an underscore to external symbols.
- D stands for double underscore; the g77, notably appends a second trailing underscore to external symbols which already contain an underscore.

- C stands for case; some compilers gives upper case names to external symbols.

The g77 compiler, for instance gets a FORTRAN_UDC=110 (underscore, double underscore, lower case). For some other examples, see the “config” directory of AAPP v6.

2.4 Endianness

It is well known that all computers do not represent numeric data using the same byte order. Since version 6.0 OPS-LRS binary configuration files do not have to be converted to the local endianness of the platform it runs on.

3. BUILDING THE SOFTWARE

3.1 Prerequisites

We assume you have a UNIX or Linux workstation, with C, C++, F90 or F77 compilers, lex, gmake, yacc, bison, perl5 and that you have already installed AAPP v6/v7. We also assume that your host
machine can run multithreaded applications, using pthreads (POSIX threads).

The basic installation with the test data set requires about 1Gb of free disk space. Running the OPS-LRS requires about 2Gb of memory.

3.2 External libraries

OPS-LRS requires two externals libraries to be installed:

- XERCES-1.7.0 XML library

```
$ tar zxvf xerces-c-src1_7_0.tar.gz
$ cd xerces-c-src1_7_0/
$ export XERCESCROOT=$PWD
$ cd src/xercesc
$ sh runConfigure  -p linux -c gcc -x g++ -r pthread -P install_directory
$ make ; make install
```

- FFTW v3

```
$ tar zxvf fftw-3.0.1.tar.gz
$ cd fftw-3.0.1
$ ./configure --prefix=install_directory
$ make
```

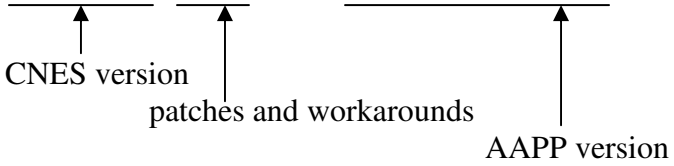
These two packages are bundled with OPS-LRS, and have embedded documentation. Installing them should be straightforward. It is recommended that you install each to a separate directory, e.g. `.../install/fftw-3.0.1` and `.../install/xerces-c-1.7.0`.

FFTW-3 version 3.3 has been successfuly tested with OPS-LRS v6.

3.3 OPS-LRS version numbers

The OPS-LRS package contains source code, compilation scripts, runtime scripts and environment, in a single version-specific tar archive. For example,

```
OPS_V3-5-p12-AAPP-slai-20060623.tgz
```



Also delivered are tar archives containing test data (*iasi-test-dump.tgz* – see section 88) and the archives containing XERCES-1.7.0 and FFTW v3.

3.4 Unpack and configure

This is how you typically install OPS-LRS.

1. Unpack the source code
   ```
   $ tar zxvf OPS_V3-5-p12-AAPP-slai-20060623.tgz
   ```

2. Run the "configure" script. For example:
   ```
   $ cd OPS_V3-5-p12-AAPP-slai-20060623
   $ ./configure --aapp-prefix=/soft/AAPP_6 \
    --xrcs-prefix=/opt/xerces-c-1.7.0 --fftw-prefix=/opt/fftw-3.0.1 \
    --arch=Linux-g77 --prefix=/soft/OPS_V3-5-p1-AAPP-slai-20060623 \
    --site-id=CMS --nthreads=2
   $ make ; make install
   ```

   In the above,
   - "aapp-prefix" is the top directory of AAPP v6 /v7(i.e. the directory containing *AAPP*, *metop-tools* and *iasi-tools*)
   - "xrcs-prefix" is the installation directory for XERCES (the directory containing *lib*)
   - "fftw-prefix" is the installation directory for FFTW
   - "arch" is one of the supported architectures of OPS-LRS (e.g. Aix, Irix-g++-g77, Linux-g77, Solaris, Solaris-10-gcc, Irix, Linux-Intel, Linux-g95, Solaris-10 – see list in the *config* sub-directory).
   - "prefix" (optional) is the install path of OPS-LRS. The contents of the sub-directory "run" will be copied to the location <prefix>/*OPS-LRS-run*
   - "site-id" is a site identifier of up to 4 characters.
   - "nthreads" is the number of active threads to be used in OPS-LRS

   Other options are:
   - --metoplib-prefix= EUMETSAT MetopLib installation prefix (external library)
   - --use-fft-numrec
     Use Numerical Recipies FFT library (source code provided with OPS-LRS)
   - --use-fft-fftpack Use FFTPack library (source code provided with OPS-LRS)
   - --use-essl Use default IBM ESSL library (external library)
   - --optimize= Optimization level (normal / debug)
   - --extra-libs= Libraries options to be passed to the linker

   Type "configure" without argument for the complete and up-to-date list of options

Note that fftw-prefix, use-fft-numrec, use-fft-fftpack and use-essl are exclusive.

3. OPS-LRS is supplied with Makefiles installed. Normally you should not need to change them. However, as with AAPP, if you need to re-generate them you can do so using the command
   `$ perl Makefile.PL`

The "configure" script sets up two configuration files: Makefile.ARCH and Makefile.local. These may be edited by hand if required (e.g. to change optimisation settings). It also sets up the files example.env and OPS.env in directory *run/OPS/bin*.

We list here the macros you will find in **Makefile.ARCH**

`ARCH_CFLAGS, ARCH_CPPFLAGS, ARCH_FFLAGS :`

- thread-safe flags
- `-DHASNT_UNION_SEMUN` for architectures which do not define `union semun`
- `-DOPS_BYTE_ORDER=  OPS_LITTLE_ENDIAN / OPS_BIG_ENDIAN`
- `-DHASNT_SETENV` for architectures which do not implement setenv
- `-DHASNT_INADDR_NONE` for architectures without the macro `INADDR_NONE`
- `-DFORTRAN_NOT_THREADSAFE` for non threadsafe fortran libraries
- `-DLINUX` for Linux platforms
- `-DAIX` for IBM/AIX platform

`ARCH_LIBS :`

fortran runtime libraries + pthread library

Makefile.local contains settings for the specific installation; here you will see the path of the AAPP package you want to link with, and the installation directories of fftw3 and xerces 1.7.0.

As you can see in the config directory, there are a number of predefined architectures (OS+compiler); if you want to compile OPS on a different platform, you have to create a new config Makefile.

3.5 Note for Linux 64bits, using MetOpLib

The Eumetsat MetopLib navigation library is only available for Linux in binary and 32bits. The 32bits compatibility is achieved by `-m32` compiler/linker option. In that case FFTW and Xerces should also be compiled with the same option.

3.6 Note for MacOSX

MacOSX operating system is not supported. The issue concerns the Xerces v1.7.0 library which fails at the compilation step.

3.7 Note for IRIX users

IRIX operating system is no longer supported

3.8 F77 runtime libraries

OPS-LRS is written in C, C++ and Fortran. The linker is invoked by the C++ compiler, which means that you have to explicitly set the F77 runtime libraries `ARCH_LIBS` in Makefile.ARCH. If your system is non-standard you may need to modify the default settings. How to guess them? You can use a little test program, and compile it using the -v option:

```
$ f77 -v main.F
...
 /usr/lib/gcc/i586-mandrake-linux-gnu/3.4.3/collect2 --eh-frame-hdr -m
elf_i386 -dynamic-linker /lib/ld-linux.so.2 /usr/lib/gcc/i586-mandrake-
linux-gnu/3.4.3/../../../crt1.o /usr/lib/gcc/i586-mandrake-linux-
gnu/3.4.3/../../../crti.o /usr/lib/gcc/i586-mandrake-linux-
gnu/3.4.3/crtbegin.o -L/usr/lib/gcc/i586-mandrake-linux-gnu/3.4.3 -
L/usr/lib/gcc/i586-mandrake-linux-gnu/3.4.3 -L/usr/lib/gcc/i586-mandrake-
linux-gnu/3.4.3/../../.. /home/philou/tmp/ccxwJr0H.o -lfrtbegin -lg2c -lm
-lgcc_s -lgcc -lc -lgcc_s -lgcc /usr/lib/gcc/i586-mandrake-linux-
gnu/3.4.3/crtend.o /usr/lib/gcc/i586-mandrake-linux-
gnu/3.4.3/../../../crtn.o
```

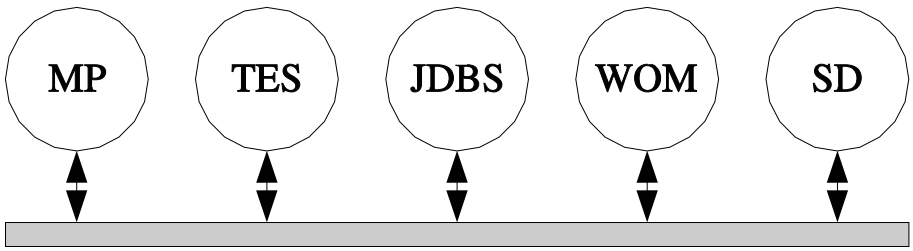
Your F77 compiler will then tell you what libraries your test program is linked against (in bold in the example above)

4. SOFTWARE DESIGN

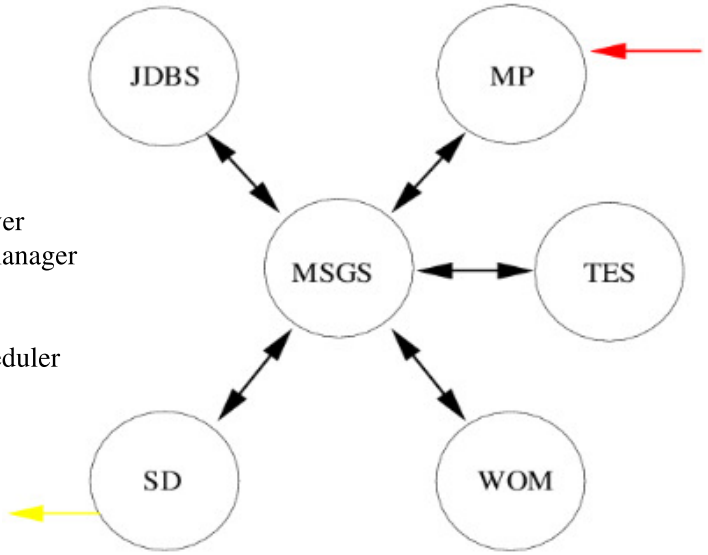
4.1 Multi-process software

OPS-LRS is a multi-process application. An instance of a running OPS-LRS creates six processes, which run in parallel and communicate with each other using a “software bus”, implemented by the MSGS process.

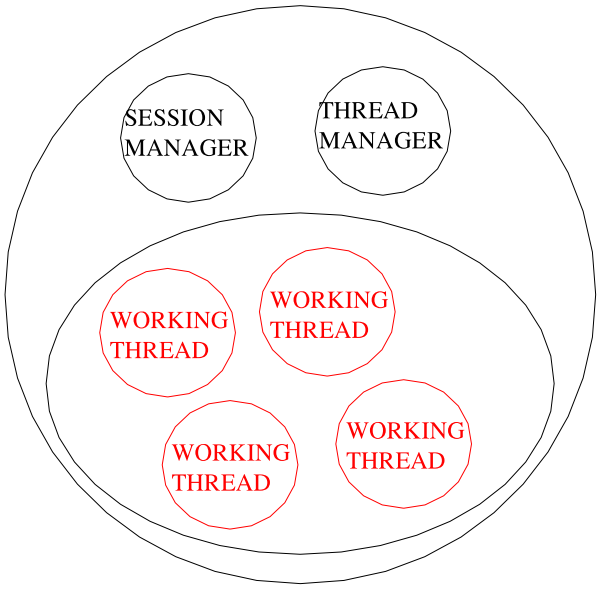
Of these processes, the MP starts the application and creates the other processes, the TES schedules timeouts and events, the SD is the multithreaded data server.



MP : main process
MSGS : message server
WOM : work order manager
SD : data server
JDBS : log server
TES : time event scheduler



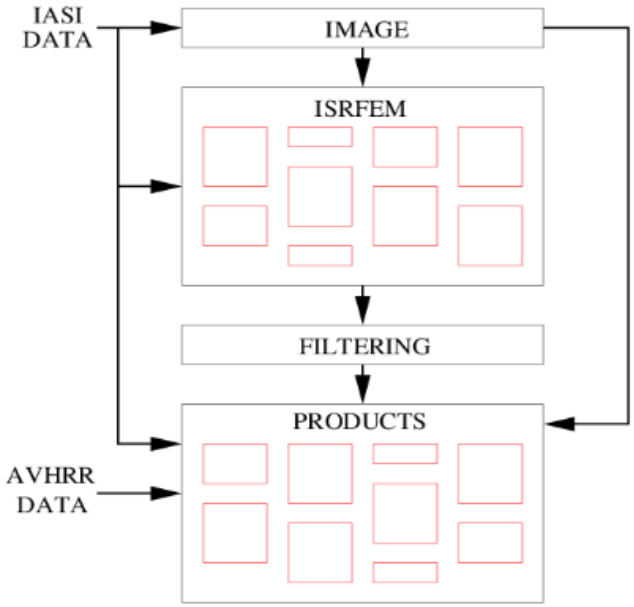
4.2 Multithreaded data server



The data server process of OPS-LRS is multithreaded and performs the processing of IASI L0 data; here is how this process is organized, in terms of threads and tasks:

- Session manager handles incoming messages and subdivides them into tasks
- Thread manager dispatches tasks to working threads
- Tasks fall into two distinct categories :
  - *Line*: such tasks can be executed concurrently
  - *Rendez-vous*: such a task must be run separately

4.3 Parallelisation



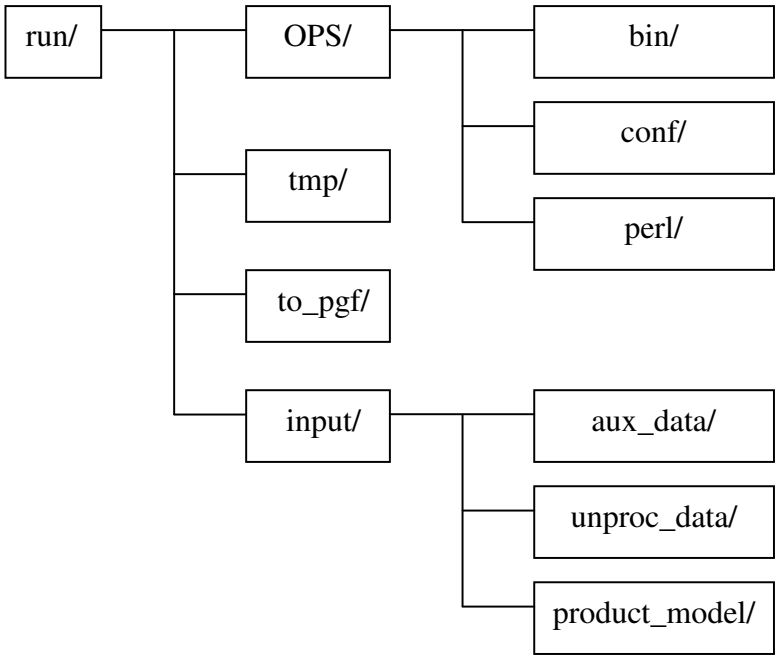
OPS-LRS is able to process several lines of data in parallel; here are the four kinds of tasks it can handle:

- IMAGE : radiometric calibration
- ISRFEM : interferometer axis position / spectral calibration / apodisation functions
- FILTERING : of the interferometer axis position
- PRODUCTS : 1A / 1B / 1C

Only tasks of types ISRFEM and PRODUCT are run in parallel. The most time consuming part of the processing is the generation of 1A/1B/1C products.

5. USING OPS-LRS

5.1 Runtime environment



Above is a diagram describing the layout of an OPS-LRS installation tree. (If you have used the "prefix" option then the directory named "run" is the installation directory).

- **`OPS/bin`**: binaries & scripts, OPS.env (OPS environment variables), example.env, log files (one per process), pid files (one per process).
- **`OPS/conf`**: configuration files, OPS_SD.cfg (number of active threads in the SD process).
- **`tmp`**: log & HKTM files, cmd_fromPGF (pipe from which OPS reads commands), temporary data.
- **`to_pgf`**: reports & product files.
- **`OPS/perl:`** Perl scripts for running OPS in DUMP mode.
- **`input`**: input data for the OPS (see below).

The input directory contains work-orders, and three other directories:

- **`aux_data`**: IASI auxiliary data (cold start context file, spectral database, stable and other configuration files), and AVHRR 1B data.
- **`unproc_data`**: level 0 IASI data
- **`product_model`**: template headers for created files

5.2 Starting OPS-LRS

Starting OPS-LRS requires setting the following environment variables:

- `REP_WORKING_ROOT` **:** the prefix where OPS is installed.
- `SATPOS_PATH` **:** path of the satpos file valid for processing.
- `SPACECRAFT_ID` **:** M01/M02/M03/M04.
- `METOP_ENV` **:** attitude of the satellite; TEST for NOAA data (including test data), NORMAL for METOP data.
- `CONTEXT_SOURCE` **:** processing center (4 characters max)

The first four of these are set up in `example.env`, which may be edited and sourced as required. The last is in `OPS.env`, which should be sourced before starting the MP__MainProcess.sh script. You may also wish to check the contents of `OPS/conf/OPS_SD.cfg` which specifies the number of threads to use.

When the OPS starts, the MP process will create five other processes. It is possible to check whether all of them are running using the ps command.

```
$ cd OPS/bin
$ export REP_WORKING_ROOT=... ; \
  export SATPOS_PATH=... ;      \
  export SPACECRAFT_ID=...;     \
  export METOP_ENV=...            #or use . ./example.env
$ . ./OPS.env
$ ./MP__MainProcess.sh
...
$ ps -u metop
  PID TTY          TIME CMD
 9371 pts/12   00:00:00 bash
23717 pts/13   00:00:00 bash
23969 pts/13   00:00:00 MP__MainProcess
23980 pts/13   00:00:00 MSGS__Serveur
24003 pts/13   00:00:00 TES__ServeurTem
24029 pts/13   00:00:00 JDBS__Serveur
24057 pts/13   00:00:00 WOM__ServeurWor
24065 pts/13   00:00:00 SD_FRW__Serveur
24120 pts/13   00:00:00 ps
```

The application will then create .pid and .eo files containing standard error and output of the six processes of the OPS:

```
[metop@kaitain bin]$ ls -lrt *.eo
-rw-r-----  1 metop metop     0 fév 27 16:43 WOM__ServeurWorkOrder.eo
-rw-r-----  1 metop metop     0 fév 27 16:43 TES__ServeurTemps.eo
-rw-r-----  1 metop metop 38684 fév 27 16:43 SD_FRW__ServeurDonnees.eo
-rw-r-----  1 metop metop     0 fév 27 16:43 MSGS__Serveur.eo
-rw-rw-r--  1 metop metop   751 fév 27 16:43 MP__MainProcess.eo
-rw-r-----  1 metop metop     0 fév 27 16:43 JDBS__Serveur.eo
```

In cases of emergency you can kill all OPS-LRS processes by issuing the command
`kill -9 $(cat *.pid)`

5.3 Passing commands to OPS-LRS

In order to send command to OPS-LRS, one has to prepare a work order file; here is a sample work order:

```
$ cat ../../input/IASI_9_wo_001
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PPF_Work_Order SYSTEM "PPF_Work_Order.dtd">
<PPF_Work_Order>
<ProcessingType>L1a</ProcessingType>
<SensingStart>20020808181256Z</SensingStart>
<SensingStop>20020808181530Z</SensingStop>
<TimeIntervalFlag>First</TimeIntervalFlag>
<TimeIntervalCounter>1</TimeIntervalCounter>
<UnProcData>input/unproc_data/IASI_xxx_00_M01_20020808181248Z_20020808181530Z_B_
O_20020810170509Z</UnProcData>
<AuxData>input/aux_data/IASI_BRD_xx_M01_20020101181957Z_20100101000000Z_20020910
125223Z_IAST_0000000000</AuxData>
...
```

Then, the command has to be written to the pipe file located in tmp:

```
$ echo "STEP 1 input/IASI_9_wo_001" > ../../tmp/cmd_fromPGF
$ ls -lrt
-rw-r-----  1 metop 502        6 Sep 29 06:59 TES__ServeurTemps.pid
-rw-r-----  1 metop 502        0 Sep 29 06:59 TES__ServeurTemps.eo
-rw-r-----  1 metop 502        6 Sep 29 06:59 JDBS__Serveur.pid
-rw-r-----  1 metop 502        0 Sep 29 06:59 JDBS__Serveur.eo
-rw-r-----  1 metop 502        6 Sep 29 07:00 WOM__ServeurWorkOrder.pid
-rw-r-----  1 metop 502        0 Sep 29 07:00 WOM__ServeurWorkOrder.eo
-rw-r-----  1 metop 502        6 Sep 29 07:00 SD_FRW__ServeurDonnees.pid
-rw-r--r--  1 metop 502       91 Sep 29 07:02 MP__MainProcess.eo
-rw-r-----  1 metop 502    33600 Sep 29 07:04 SD_FRW__ServeurDonnees.eo
$
```

On the previous file list, we see that the processing has begun, since the `SD_FRW__ServeurDonnees.eo` is not empty.

Other work orders may be sent as required, numbering the `STEP` commands sequentially.

When all processing is complete (check for products in the `to_pgf` directory), issue the `STOP` command, e.g.

```
$ echo "STOP 2" > ../../tmp/cmd_fromPGF
```

Note that OPS-LRS creates OPC semaphores (you can list them with `ipcs`). You might need to do some cleanup from time to time (using `ipcrm`).

5.4 Getting feedback

Knowing what the OPS is doing, whether it has finished its processing, etc., requires looking at the log files (with a human eye, or a script). Here is what shall be written in `MP__MainProcess.eo` (in OPS/bin) when the processing of the previous granule has finished:

```
$ cat MP__MainProcess.eo
Ouverture du pipe d'entree : /metop/app/opsiasi/tmp/cmd_fromPGF
ACK START 0 0
ACK STEP 1 0
STAGE L1a 1 R to_pgf/IASI_9_wo_001_001.rpt
STAGE L1a 1 P
to_pgf/IASI_xxx_1B_M01_20020808181253Z_20020808181524Z_N_O_20050929070224Z
STAGE L1a 1 P
to_pgf/IASI_xxx_1C_M01_20020808181253Z_20020808181524Z_N_O_20050929070224Z
STAGE L1a 1 P
to_pgf/IASI_xxx_1A_M01_20020808181253Z_20020808181524Z_N_O_20050929070224Z
STAGE L1a 1 P
to_pgf/IASI_ENG_01_M01_20020808181253Z_20020808181524Z_N_O_20050929070224Z
STAGE L1a 1 P
to_pgf/IASI_VER_01_M01_20020808181253Z_20020808181524Z_N_O_20050929070224Z
STAGE L1a 1 W
to_pgf/IASI_CTX_xx_M01_20020808181524Z_xxxxxxxxxxxxxxx_20050929070745Z_IAST_xxxx
x00002
ACK STEP 1 1
```

On the previous listing, we have highlighted **`to_pgf/IASI_9_wo_001_001.rpt`** because it is an important file to look at, in order to have detailed information about what the processing has produced.

The above listing tells us that the OPS has started successfully "`ACK START 0 0`", that it has received a STEP command "`STEP 1 0`", and that L1a processing was successful "`STEP 1 1`". Here is how it works :

- `ACK STEP 1 0` -> STEP 1 received
- `ACK STEP 1 1` -> STEP 1 completed successfully
- `ACK STEP 1 2` -> STEP 1 rejected

- etc...

Now here is what the **`to_pgf/IASI_9_wo_001_001.rpt`** file looks like (we call that a report file):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE PPF_report>
<PPF_report>
<WorkOrder>
<ProcessingType>L1a</ProcessingType>
<SensingStart>20020808181256Z</SensingStart>
<SensingStop>20020808181530Z</SensingStop>
<TimeIntervalFlag>First</TimeIntervalFlag>
<TimeIntervalCounter>1</TimeIntervalCounter>
<UnProcData>input/unproc_data/IASI_xxx_00_M01_20020808181248Z_20020808181530Z_B_
O_20020810170509Z
</UnProcData>
<StageInfo>
<StageName>L1a</StageName>
...
<GeneratedFile>
...
<FileName>IASI_xxx_1C_M01_20020808181253Z_20020808181524Z_N_O_20050929070224Z</F
ileName>
<MdrTotalCount>16</MdrTotalCount>
<MdrDegraded>1</MdrDegraded>
<DataGap>
<GapReason>(05) processing impossible</GapReason>
<GapCount>1</GapCount></DataGap></GeneratedFile>

<MdrTotalCount>16</MdrTotalCount>
<MdrDegraded>1</MdrDegraded>

<ProcessingInfo>
<ProcessingStart>20050929070223.000Z</ProcessingStart>
<ProcessingStop>20050929070745.000Z</ProcessingStop>
<ElapsedTime>321.939</ElapsedTime>
<UserTime>277.160</UserTime></ProcessingInfo>
<LogMessage>End of treatment: L1a which lasts: +00321.939 (ELAPSED); +00277.160
(USER)</LogMessage></StageInfo></PPF_report>
```

5.5 Dump / Pipeline mode

OPS-LRS can ingest data in two modes:

- Dump mode: the L0 data is provided in a single data file.
- Granule mode: the L0 is provided in fixed size chunks (the length should not be smaller than 3 minutes).

Here are the characteristics of the Dump mode:

- `<TimeIntervalFlag>Full</TimeIntervalFlag>` this flag should be set in the work order.
- Single IASI-L0 file
- AVHRR-1B file(s)
- Single output L1C file

Now, we describe the Pipeline mode:

- `TimeIntervalFlag = First, Middle, ..., Last` – these flags should be set for the subsequent IASI L0 files.
- IASI L0 files, with an overlap of at least 8 seconds.
- AVHRR files; AVHRR data provided for each work order shall cover the IASI L0 granule with 100 extra AVHRR lines at the beginning and at the end.
- One output L1C file per L0 file.

5.6 Input / Output

We describe here what the OPS takes as input, and what it produces.

OPS-LRS inputs:

- Level 0 products (from HRPT), PFS L0
- AVHRR 1B (from HRPT), PFS L1B
- Context file (recursive data), binary
- Spectral database (EUMETSAT, TBD), binary
- Configuration files (EUMETSAT, TBD), binary
- Command + Work-Order, xml

OPS-LRS products:

- Report file, xml
- Log/HKTM files
- Context file, binary
- Engineering data, PFS
- Verification data, PFS
- 1A, 1B, 1C products, PFS

Note that in the OPS implementation at EUMETSAT the Context file is designed to be recursive, i.e. it is continually updated. However, for local reception *OPS-LRS should always be run with a "cold start" context file*, i.e. you should ignore the CTX product file. You may also ignore the ENG and VER product files.

6. RUNNING OPS-LRS WITH THE OPS_PROCESS SCRIPT

OPS-LRS is delivered with a set of standalone Perl scripts to run the software either in DUMP mode or in GRANULE mode. These scripts will set-up the environment, start OPS, prepare OBT, SVM, OSV files, select valid configuration files, prepare work-orders, monitor the software, and stop the OPS when the processing is done.

Environment variables of interest for `ops_process` :

- `DIR_NAVIGATION` : navigation directory, containing messages and satpos files valid for the data to be processed.
- `DIR_IASICONFIG` : directory containing the configuration files for IASI, encoded in Big Endian : ODB, CTX, BRD, GRD, and product models.
- `METOP_ENV` : this variable shall be set to `TEST` for processing M04 data (based on NOAA17); this variable is automatically set to `NORMAL` if not defined.
- `OPS_TIMEOUT` : If set, this variable should contain the number of seconds the software is expected to take to process 1 minute of data. If the processing goes beyond this limit, the software will abort. This is used to catch situations where the OPS fails and hangs instead of returning an error code.
- Example: `export OPS_TIMEOUT=100` will cause the OPS to abort if processing of 60 s of data takes more than 100 s.
- `ADR_MSG_SERVER_PORT_BASE` : Base port for MSG server, defaults to 4000. The port number used by the OPS is the base number plus the satellite number.
- `APP_OPSIASI_PORT_BASE` : Base port for ops_process, defaults to 6000.

To run more than one instance of OPS-LRS sumltaneously (for a given satellite), you can set the `ADR_MSG_SERVER_PORT_BASE` and `APP_OPSIASI_PORT_BASE` variables, e.g.:

```
i=$(date +%N | cut -c1-3)
export ADR_MSG_SERVER_PORT_BASE=$((4000+$i))
export APP_OPSIASI_PORT_BASE=$((6000+$i))
```

`ops_process` requires the AAPP_6/7 environment and utilities.

Using `ops_process` is simple; assuming you have some data to process in dump mode, for instance:

```
[iasi@kaitain data]$ ls
AVHR_xxx_1B_M04_20020808181200Z_20020808181300Z_x_x_20060410082149Z
AVHR_xxx_1B_M04_20020808181300Z_20020808181400Z_x_x_20060410082258Z
AVHR_xxx_1B_M04_20020808181400Z_20020808181500Z_x_x_20060410082348Z
AVHR_xxx_1B_M04_20020808181501Z_20020808181601Z_x_x_20060410082459Z
AVHR_xxx_1B_M04_20020808181601Z_20020808181701Z_x_x_20060410082559Z
AVHR_xxx_1B_M04_20020808181701Z_20020808181801Z_x_x_20060410082702Z
AVHR_xxx_1B_M04_20020808181802Z_20020808181902Z_x_x_20060410082756Z
AVHR_xxx_1B_M04_20020808181902Z_20020808182002Z_x_x_20060410082859Z
IASI_xxx_00_M04_20020808181248Z_20020808182130Z_B_O_20020810170509Z
```

Some auxiliary data located in `$DIR_IASICONFIG` :

```
[iasi@kaitain data]$ ls $DIR_IASICONFIG
IASI_BRD_xx_M04_20020101181957Z_20100101000000Z_20020910125223Z_IAST_0000000000
IASI_BRD_xx_M04_20020101181957Z_20100101000000Z_20020910125223Z_IAST_0000000001
IASI_CTX_xx_M04_20020101181957Z_xxxxxxxxxxxxxx_20021008165703Z_IAST_xxxxx01001
IASI_ENG_01_M04_xxxxxxxxxxxxxxZ_xxxxxxxxxxxxxxZ_N_O_xxxxxxxxxxxxxZ
IASI_GRD_xx_M04_20020101181957Z_xxxxxxxxxxxxxx_xxxxxxxxxxxxxx_IAST_0000000000
IASI_ODB_xx_M04_20020101181957Z_20100101000000Z_20021008165703Z_IAST_0000000000
IASI_VER_01_M04_xxxxxxxxxxxxxxZ_xxxxxxxxxxxxxxZ_N_O_xxxxxxxxxxxxxZ
IASI_xxx_1A_M04_xxxxxxxxxxxxxxZ_xxxxxxxxxxxxxxZ_N_O_xxxxxxxxxxxxxZ
IASI_xxx_1B_M04_xxxxxxxxxxxxxxZ_xxxxxxxxxxxxxxZ_N_O_xxxxxxxxxxxxxZ
IASI_xxx_1C_M04_xxxxxxxxxxxxxxZ_xxxxxxxxxxxxxxZ_N_O_xxxxxxxxxxxxxZ
```

And valid navigation data for this case:

```
[iasi@kaitain data]$ find $DIR_NAVIGATION/spm_db/
spm_db/
spm_db/spm_metop04.index
spm_db/2006-05
spm_db/2006-05/admin-20020808.dat_1148974380.txt
```

You can start `ops_process` like this:

```
[iasi@kaitain data]$ ops_process --processing=DUMP --satellite=metop04  \
   IASI_xxx_00_M04_20020808181248Z_20020808182130Z_B_O_20020810170509Z AVHR*
```

When this script returns, a file called `metop04-pfsIASI1C.txt`is created and contains the names of IASI 1C products created by the OPS. It is then possible to save those files:

```
$ cp $(cat metop04-pfsIASI1C.txt) /somewhere/i/keep/iasi/files/
```

And then to stop the OPS:

```
$ ops_process --satellite=metop04 --stop
```

A directory named `metop04` contains log files and the data used during the processing. You can safely remove it.

It also possible to run the software in GRANULE mode; one shall also issue commands like the following each time data comes in (it is assumed that data is fed in chronological order):

```
$ ops_process --satellite=metop02 --processing=GRANULE IASI_xxx_00_...
$ ops_process --satellite=metop02 AVHR_xxx_00_...
  ...
$ ops_process --satellite=metop02 IASI_xxx_00_...
$ ops_process --satellite=metop02 --finish
  ... # save IASI 1C files
$ ops_process --satellite=metop02 --stop
```

The `ops_process` script contains both client and server code, which means a copy of itself will remain in memory and monitor all OPS processes; after half an hour of inactivity, it will stop the OPS. If the OPS crashes it will detect it and clean up everything, including semaphores created by the OPS. If it receives a SIGTERM signal, it will abort the processing, and kill other processes; this gives an efficient mean for stopping the software in case of emergency.

Multiple `ops_process` can run on the same machine; but caution is required:

- any number of different METOP satellite IASI data may be processed in parallel on the same machine.
- processing two dumps of metop02 is possible, but it is necessary to change the environment variables `ADR_MSG_SERVER_PORT_BASE` and `APP_OPSIASI_PORT_BASE` which define the port ranges used by the OPS. These defaults to 4000 and 6000, which means that metop02 will use the ports 4002 and 6002, metop04, 4004 and 6004, etc...

7. RUNNING OPS-LRS WITH AAPP_RUN_METOP

The AAPP_RUN_METOP script can be used to process EPS level 0 data for any of the following instruments: AMSU, MHS, HIRS, AVHRR and IASI. It includes all of the AAPP calibration and pre-processing steps, i.e. including the module that maps AMSU and MHS to the IASI grid.

To process IASI level 0 data, the script makes use of `ops_process`, as described in the previous section. The IASI data are processed in dump mode. Before running the script you need to have the environment variable `DIR_IASI_CONFIG` defined. Also, be aware that processing will take place in your `WRK` directory.

Assuming that you have some level 0 files (one file per instrument) in a directory *indir*, and you wish to send products to a directory *outdir*, you would call AAPP_RUN_METOP as follows:

```
AAPP_RUN_METOP -i "AMSU-A MHS IASI AVHRR" -g IASI -d indir -o outdir
```

If you just want to generate the IASI level 1c file, then the command would be

```
AAPP_RUN_METOP -i "IASI AVHRR" -g " " -d indir -o outdir
```

If you want to generate ATOVS products as quickly as possible, then IASI products afterwards you can do it as follows:

```
AAPP_RUN_METOP -i "AMSU-A MHS HIRS AVHRR" -g IASI -d indir -o outdir -c
AAPP_RUN_METOP -i "AMSU-A MHS IASI AVHRR" -g IASI -d indir -o outdir -b
```

where the "-c" flag means keep the level 1b files for a second run, while "-b" re-uses any level 1b files that are present in the WRK directory.

The AAPP_RUN_METOP script performs end-to-end processing, but is not as flexible as ops_process, e.g. it only works in DUMP mode. You may need to customise it for your application.

8. RUNNING THE SUPPLIED OPS-LRS TEST CASES

8.1 iasi-test-dump

Early releases of OPS-LRS (V3-5-p12 and V3-6-p23456) were supplied with a "dump mode" test case consisting of 10 minutes of HRPT data; an IASI level 1C file is supplied together with AVHRR 1B data encoded in PFS, as produced by `aapp-eps_avhrrl1b`.

Navigation data (SPOT bulletins) are also provided, as well as the IASI configuration data valid for this case.

It may be run as follows:

1. Install OPS-LRS as instructed in section 3.3.4
2. Unpack the supplied file `iasi-test-dump.tgz` (The name may include a version identifier)
   ```
   tar -xzf iasi-test-dump.tgz
   ```
3. Modify your PATH environment variable so that it includes the directory containing *ops_process_dump*:
   ```
   export PATH=${PATH}:prefix/OPS/perl
   ```
   where *prefix* is either the installation prefix you specified in step 1 or (*Installation directory*)/run if you did not specify a prefix.
4. Set appropriately your `AAPP_PREFIX` environment variable; it should point to the install path of your AAPP.
5. cd to `iasi-test-dump`
6. type
   ```
   ./test.sh
   ```

The script will set up the necessary environment variables (`DIR_NAVIGATION, METOP_ENV, DIR_IASICONFIG`), then call `ops_process` passing the names of the IASI and AVHRR data files.

If processing is successful the output files `metop02-pfsIASI1C.txt` and a IASI level 1C file will be found in the current directory. A copy of the OPS run-time tree (including log files) will be found in directory `metop02/.keep` .

If you wish to convert the IASI 1c output file to a format that can be ingested by ATOVPP (see main AAPP documents) you can then run `convert_iasi1c`, which is part of IASI tools (section 1010.1).

8.2 Test case based on AAPP_RUN_METOP

In 2010 a new test case was released, comprising locally-received MetOp level 0 files which can be processed using AAPP_RUN_METOP. The test case is available on the AAPP ftp server. It includes various test scripts; the script that exercises OPS-LRS is called run_iasi_OPS.sh.

To run it you first specify the following environment variables:

- AAPP_PREFIX

- PATH_OPS (i.e. the location of ops_process)
- DIR_IASICONFIG (i.e. the location of BRD, GRD, ODB, etc. files. They can be downloaded from the ftp server.)

Then run the script by typing

```
./run_iasi_OPS.sh
```

Output files (in AAPP level 1c format) will be generated in a directory "level1".

Note that if you are running OPS-LRS version 5-0 or later then you will need to make sure that there are no "old format" BRD/GRD files in `$DIR_IASICONFIG`. The BRD files should be 133200 bytes in size, the GRD files 3368760 bytes. See also section 10.3 which discusses the configuration files further.

9. OPS-LRS PORT

9.1 General remarks

We describe here what we have changed to port OPS to several UNIX architectures.
Here are listed the main issues we had to address:

- System dependent features (these were many, and related to the IBM power4).
- ESSL (IBM scientific library); we had to encapsulate calls to ESSL and allow calls to free software scientific libraries:
    - FFTW (C); independent package
    - LAPACK (F77); excerpt from LAPACK, we made it thread-safe and integrated it in OPS-LRS.
    - NCAR math library (F77); excerpt from NCAR math library, we made it thread-safe, double precision, and integrated it in OPS-LRS.
    - DFFTPACK (F77) version 1.0 (4 April 1985), package included in OPS-LRS
    - Numerical Recipes (C) package included in OPS-LRS

- Metop Lib: navigation routines. We had to encapsulate the Metop Lib and allow calls to AAPP navigation.

We have validated our work against various cases provided by CNES, and had very good results:

- Sounder and imager radiances reproduced with utmost accuracy
- Geolocation data reproduced with 1/1000 degree error

We have not to date been able to validate against a EUMETSAT test data set, because it has been impossible to retrieve a full and consistent test data set for IASI from EUMETSAT. However, we had very good results when we processed some data very close to EUMETSAT level 0 (provided by CNES), using AVHRR level 0 (coming from EUMETSAT), and configuration files from CNES.

9.2 OPS-LRS benchmark

We carried out many tests for OPS-LRS, in order to choose a machine capable of processing IASI L0 data:

Listed below is the number of seconds required to process a 3 minute granule, for different architectures:

Platform	Seconds / 3 minutes
altix-4cpu-linux-gcc-3.2	164
altix-4cpu-linu-icc	92
v40z-4cpu-linux-gcc-3.3	99
v40z-4cpu-linux-icc	105
v40z-4cpu-solaris-cc	123
v40z-4cpu-solaris-gcc-3.3	122
pc-2cpu-2GHz-linux-gcc-3.4	235
ibm-4cpu-power4	91

9.3 OPS-LRS V6-0

The version OPS-V6-0 provided by EUMETSAT is a first step in the Linux capability.

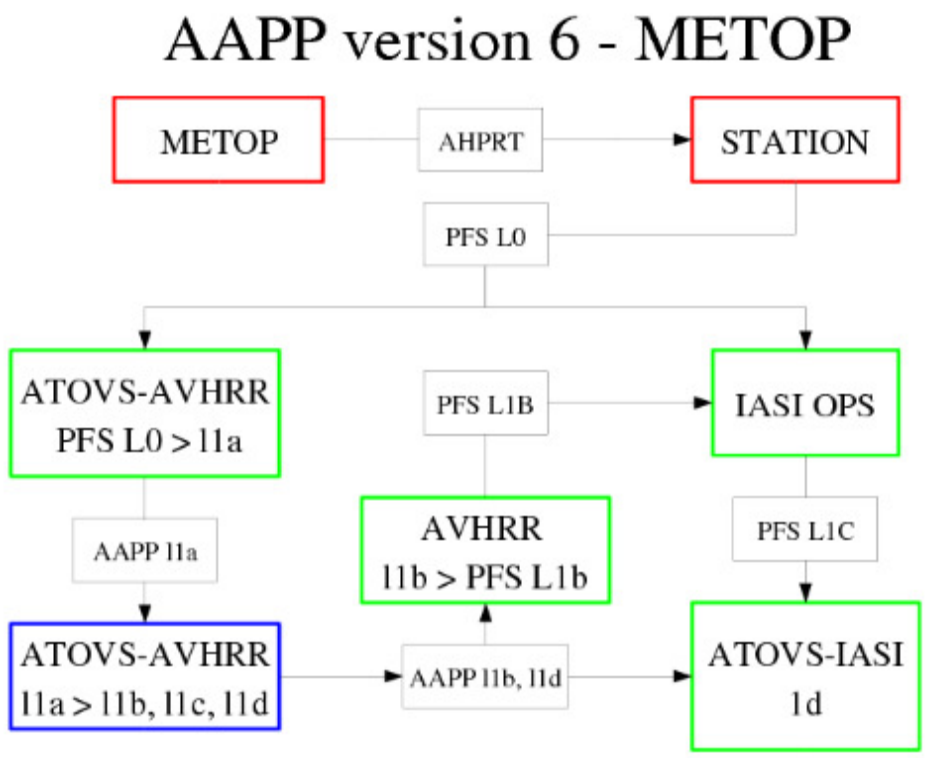
- Original Big Endian configuration files are read by OPS on Little Endian machines.
- It is officially Linux 32bits compatible on Red Hat Enterprise Linux Server 5.5
- Calls to ESSL have been encapsulated

Despite this improvement we still need to change things to make OPS-LRS running on generic Linux 64bits platforms and to make it compatible with AAPP navigation libraries. For the latest version 5 there were 96 different source code files between OPS and OPS-LRS; it is now 34 different files for version 6-0.

10. OPS-LRS & AAPP, IASI TOOLS

10.1 IASI tools

The following chart shows the data flow in AAPP v6, and how ATOVS/AVHRR processing interacts with OPS-LRS.



In order to achieve such interactions, we had to develop a number of tools:

- Libraries & programs for reading, displaying, converting configuration files
- Library for reading, displaying IASI level 1C
- Library, programs to convert from AAPP 1B AVHRR to PFS 1B AVHRR

10.2 Format libraries

Here are the format libraries we implemented, in C language:

- PFS AVHRR L1B
- IASI 1A, 1B, 1C, VER, ENG
- IASI CTX, ODB, GRD, BRD
- Admin packets
- AVHRR AAPP

These libraries were all written in C language, and were generated from the XML description of PFS formats provided by EUMETSAT. Using these libraries, it is possible to read/write PFS files, and apply some of the scaling factors listed in XML description. AVHRR AAPP capability has also been written in C.

There is also a tool to convert IASI PFS 1C format to an internal AAPP IASI 1c format that can be readily ingested by ATOVPP.

10.3 IASI configuration files

We list here the characteristics of IASI configuration files:

- IASI CTX : context file
- IASI BRD : stable parameters
- IASI GRD : other parameters

- IASI ODB : spectral database
- Delivered in big-endian

For versions of OPS-LRS prior to 6.0 it was necessary to convert those files to local endianness with the IASI tools `cnes_iasi_brd-swapb`, `cnes_iasi_ctx-swapb`, `cnes_iasi_grd-swapb`, `cnes_iasi_odb-swapb`. Using these converters is fairly simple. Each of them takes two arguments: the name of the big-endian file, and the name of the little-endian file you want to create. Alternatively the tool `convert_config_files` can be used to check all the configuration files and convert as required.

Note that you may have to remove limits on stack size using `ulimit -s unlimited`, since these programs need quite a lot of memory.
Since version 6.0 OPS-LRS is able to read IASI configuration files in their native Big Endian format, even if the host machine is a Little Endian. Therefore routine convertion of configuration files is no longer needed. AAPP print utilities are still valid.

When transitioning to version 6.0 from an earlier version, you can use the tools `cnes_iasi_brd-swapb`, `cnes_iasi_ctx-swapb`, `cnes_iasi_grd-swapb`, `cnes_iasi_odb-swapb` to convert back to big-endian format. To simplify this, an updated version of `convert_config_files` is supplied with OPS-LRS version 6.0: the –b option creates big-endian output files. If you are using the AAPP_RUN_METOP script, you must either comment out the existing call to `convert_config files` or change it to `convert_config_files -b`.

10.4 Conversion of AVHRR

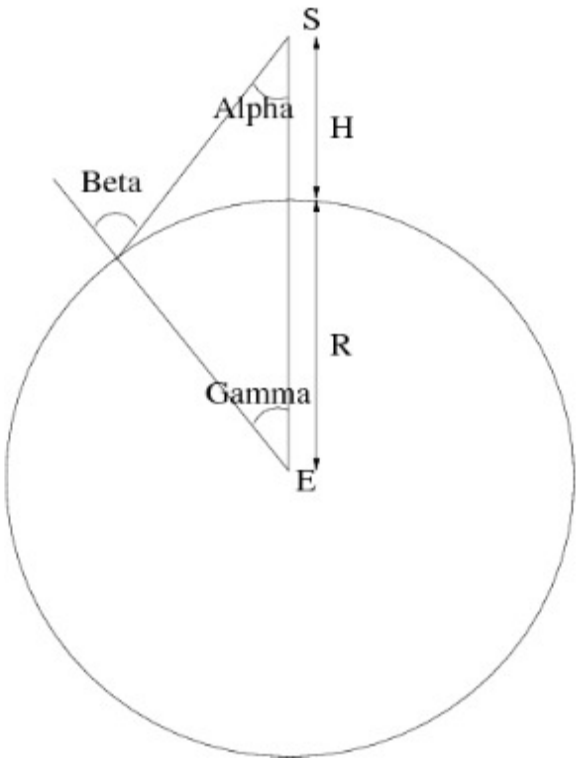
OPS-LRS requires AVHRR data to process IASI L0. Therefore, we had to find a means to convert AAPP L1B AVHRR to PFS 1B AVHRR. One critical issue was increasing the sampling of the navigation information in the PFS file (see next diagram for explanations on how it is implemented).

- (Partial) conversion from AAPP to PFS
  - Interpolation of geolocation data (51 to 103)
  - Instrument status, quality flags
  - Digital data –> radiances
- Autodetection of AAPP endianness

The user should be aware that this conversion is partial, and fields other than those required by the OPS-LRS may not be correctly filled.

Geolocation in PFS; going from 51 points to 103 points:
First, we assume that the Alpha angle is linear. From Alpha, we can deduce Beta, then Gamma. Assuming the earth is spherical locally, we get the latitude and longitude.

$$\text{Gamma} = \arcsin((R + H) / R * \sin(\text{Alpha})) - \text{Alpha} = \text{Beta} - \text{Alpha}$$



11. OTHER SOURCES FOR OPS-LRS DOCUMENTATION

Documents marked in bold type are available from the EUMETSAT web site www.eumetsat.int.

11.1 Scientific documentation

IA-SB-2100-9462_0503 (specification technique de besoin du logiciel operationnel IASI)
IA-DF-0000-2006-CNE (Dossier de definition des algorithmes IASI)

11.2 Technical documentation

IA-CP-2100-9552-THA 2R2 (dossier de conception preliminaire du logiciel IASI)

D18551.pdf (CGS contraints and rules for PPS) CGS, Work-order, report, aux data, PGF, PPF

D31307.pdf (IASI L1 Interface Control Document) Work-order, command, report, aux data, product model, trace, HKTM

IA-DD-2100-9564-THA-2R0.doc (Dossier de définition des services communs du logiciel IASI)

IA-DD-2100-9565-THA-2R2.doc (Dossier de définition du monitoring et controle du logiciel IASI), environment variables, HKTM, C++ classes

IA-DD-2100-9566-THA-2R0.doc (Dossier de définition du serveur de données IASI) C++ classes, threads management

IA-DLR-2100-9546-THA-2R2.doc (Dossier des logiciels réutilisés pour l'OPS IASI) multi-process layout

IA-ME-2100-9555-THA-2R0.doc (OPS IASI Level 1 software operational manual) multi-process layout

IA-MU-2100-9553-THA-2R7.doc (OPS IASI Level 1 user manual) error messages

IA-SL-2100-9547-THA-2R4.doc (spécifications logicielles du logiciel OPS-IASI) processing modules

11.3 Formats

A-NT-2100-9513-CNE (IASI Configfiles and database Format Spec V1.6)

EUM.EPS.SYS.SPE.990003 (EPS IASI Level 1 Product Format Specification)

EPS.MIS.SPE.97231 (EPS AVHRR/3 Level 1 Product Format Specification)

EPS.GGS.SPE.96167 (EPS Programme Generic Product Format Specification)