HIRDLS

SW-HIR-2011

Originator: Charles Cavanaugh

Date: 10 Dec. 2012

Subject/Title:

The User's Guide of the HIRDLS Level 2 Preprocessor

Description/Summary/Contents:

The purpose of this document is to describe the building, running and validating of the High Resolution Dynamics Limb Sounder (HIRDLS) Level 2 Preprocessing System. The goal of this document is to detail the description sufficiently, such that the user requires no more guidance than that what is listed in this document.

Keywords:

Purpose of this Document:

Oxford University
Atmospheric, Oceanic &
Planetary Physics
Parks Road
OXFORD OXI 3PU
United Kingdom

University of Colorado, Boulder
Center for Limb Atmospheric Sounding
3450 Mitchell Lane, Bldg. FL-0
Boulder, CO 80301

EOS

The User's Guide of the HIRDLS Level 2 Preprocessor

Charles Cavanaugh

Table of Contents

List of Figures and Tables

1 Document Purpose and Goal

The purpose of this document is to describe the building, running and validating of the High Resolution Dynamics Limb Sounder (HIRDLS) Level 2 Preprocessing System. The goal of this document is to detail the description sufficiently, such that the user requires no more guidance than that what is listed in this document.

2 References

The HIRDLS Level 2 (L2) Preprocessing System, hereby known as L2 Preprocessor, has had its requirements, architecture and design already detailed in previous documents. It is not necessary to read those documents before using L2 Preprocessor, as those documents detail the construction of the system. However, it would be helpful to read those documents before using L2 Preprocessor, as those documents also contain useful information on system input and output files, and also contain information on the purpose of the system.

3 Creating the System

L2 Preprocessor is included in the DAP (Delivered Algorithm Package) as a gzipped file, called "L2Preprocessor.gz". This section will detail how to create the system.

3.1 Extracting the System

To build L2 Preprocessor, copy the gzipped L2 Preprocessor file to the directory in which you want the system to be built, and then extract the files (thereby building the system) by using the Unix "tar" utility. This will create a directory that includes all the source code, make files, and control files that you will need. However, an input HIRDLS standard data product file (HIRDLS1C) will be necessary to run L2 Preprocessor, and that is described in Section 5.3. Figure 1 shows a practical example of the steps in this section.

```
hal:/hal/cavanaug/L2Preprocessor> ls
-rw-r--r-- 1 cavanaug hirdls 2327617 Sep 24 11:11 L2Preprocessor.gz
hal:/hal/cavanaug/L1Processor> tar xf L2Preprocessor.gz
hal:/hal/cavanaug/L1Processor> ls
-rw-r--r-- 1 cavanaug hirdls 2327617 Sep 24 11:11 L2Preprocessor.gz
drwxr-xr-x 4 cavanaug hirdls      39 Apr  4 15:47 v6.2.9/
```

Figure 1 Extracted System

3.2 Platform Dependencies

L2 Preprocessor, overall, was developed and implemented to be Unix platform independent and ANSI compatible. L2 Preprocessor uses ASCII files and HDF5 files during processing, and both of these file structures are platform independent.

3.3 External Dependencies

In order to run, L2 Preprocessor needs access to the SDP Toolkit. L2 Preprocessor uses this toolkit to perform such tasks as time conversion and control file reading. As of this document version, the current toolkit version is 5.2.18. L2 Preprocessor should be built with this version, as it is not guaranteed that L2 Preprocessor will work correctly with previous versions. Also, the SDP Toolkit libraries that L2 Preprocessor accesses must be compiled with the same compiler as used on L2 Preprocessor (see Section 3.4). The SDP Toolkit also provides environment-creating scripts that must be run before

compiling or running L2 Preprocessor. These scripts are platform-dependent, and can be found in the toolkit directory, in the appropriate platform "bin" subdirectory. Figure 2 shows a listing that could be used, depending on the platform.

```
/usr/local/TOOLKIT5.2.18gnu/bin/linux/pgs-env.ksh
/usr/local/TOOLKIT5.2.18gnu/bin/linux/pgs-dev-env.ksh
/usr/local/TOOLKIT5.2.18gnu/hdfeos/bin/linux/hdfeos_env.ksh
```

Figure 2 External Scripts

3.4 Compiling the System

To create the L2 Preprocessor binary executable, go to the "driver" subdirectory of the "C++/packages" subdirectory of the build you created. In that subdirectory are 5 Makefiles that control compilation of L2 Preprocessor. At most, only two of them, "Makefile.exec" and "Makefile.global", should need editing.

"Makefile.global" contains compilation flag definitions, and these can be platform-specific. The default listings of "CFLAGS" and "EXECCFLAGS" are minimal and should work on all platforms, but you are certainly welcome to add your favorite compilation flags. The default for "FASTCFLAGS" turns optimization on, to level 2. Again, this should work for most cases. The defaults for "WARNCFLAGS" and "DEBUGCFLAGS" should also work for most cases. This file also contains a soft reference to the compiler to use to build L2 Preprocessor. The default is GNU's g++ compiler.

"Makefile.exec" contains the library listing to use for compilation. If these libraries are different on your machine, you will need to adjust this listing. And like "Makefile.global", "Makefile.exec" contains the same compiler soft link.

"Makefile.system" should not need editing, but contains the three included compilation targets: "systemfast", "systemdebug" and "systemwarn". These targets control which compilation flags are used during compilation. The default way to build L2 Preprocessor is to use the "systemfast" target. If you want to be able to access L2 Preprocessor with a debugger during a run, use the "systemdebug" target. If you want to see compilation warnings during compile, use the "systemwarn" target. It is recommended that for any extensive run jobs, you use the "systemfast" target to generate L2 Preprocessor. To actually build L2 Preprocessor, invoke the make command with one of these three targets, i.e., at the Unix prompt, enter "make systemfast" (without the parentheses). Upon successful completion, the system executable, called "L2PP", will have been created in the "run/sandbox" subdirectory, as shown in Figure 3.

4 Editing the System

It is strongly recommended that you successfully complete Section 3 before you do any editing, to make sure that any issues that might arise are not due to faulty instructions in this user's guide. If you edit the source code, you will of course need to recompile the system. If you edit any of the input files in the DAP, there is no need to recompile the system, unless of course you change the internal structure of a file. Any changes to the system should be documented in the "history" ancillary file.

```
hal:/hal/cavanaug/L2Preprocessor/v6.2.9/run/sandbox> ls
-rw-r--r-- 1 cavanaug hirdls     398 Jun 25 13:37 FilterConfig.txt
-rw-r--r-- 1 cavanaug hirdls    8782 Jun 25 13:42 HIRDLS1R.mcf
-rw-r--r-- 1 cavanaug hirdls  736387 Jun 25 13:37 HIRFOV_VerticalResponse.he5
-rw-r--r-- 1 cavanaug hirdls    3578 Jun 25 13:40 history
-rwxr-xr-x 1 cavanaug hirdls 5479965 Jun 25 13:37 L2PP*
-rw-r--r-- 1 cavanaug hirdls   26123 Jun 25 13:40 L2PP_pcf.txt
-rwxr-xr-x 1 cavanaug hirdls    2521 Jun 25 14:51 L2PP.py*
-rw-r--r-- 1 cavanaug hirdls   12485 Jun 25 13:37 proc.py
-rw-r--r-- 1 cavanaug hirdls   76968 Jun 25 13:37 rad_adjust_v27_c21-1_2006d138.txt
-rw-r--r-- 1 cavanaug hirdls   76968 Jun 25 13:37 rad_adjust_v27_c21-2_2006d138.txt
```

Figure 3 Sandbox Listing

5 Running the System

To run L2 Preprocessor, it is assumed you have followed the steps in Section 3 or Section 4, and have a compiled version of the system. Congratulations! That was the difficult step. This section will detail how to run the system

5.1 Building the Run

To build the run, you will first need to create a directory in which you want the run to happen. As detailed in the next section, you will need to consider how much free space you have in that directory. Once you have that directory established or newly created, copy all the files from the "run/sandbox" subdirectory, as listed in Figure 3, into the run directory. Also copy the "usfmask" file from the "run" subdirectory (one level up from the "sandbox" subdirectory).

5.2 Resource Dependencies

Running L2 Preprocessor requires at least 1.5 Gbytes of available memory and up to 270 Mbytes of available storage space (depending on the extent of the processing request, and assuming the input HIRDLS1C file is referenced via symbolic link). L2 Preprocessor requires only one thread.

5.3 Ancillary Files

Not included in L2 Preprocessor's build is the input HIRDLS L1C file necessary for a successful run. The specific file necessary is dependent on the processing request, and a DAP that included all HIRDLS L1C files would be impossibly wieldy. HIRDLS generates the L1C file as part of its standard processing run. If you are running L2PP Preprocessor only, and not also L1C Processor first, you will need to acquire the input L1C file from ESDIS. For a standard day's run, you will need only the L1C file for that respective day. Section 5.4 details how L2 Preprocessor is notified of the input L1C file.

5.4 Invoking the System

L2 Preprocessor is invoked by calling the python script "L2PP.py" with the input "usfmask" text file. This input text file, an example of which is shown in Figure 4, contains parameters read in by L2 Preprocessor. These parameters are then used to fill in the run file needed by the SDP Toolkit. The values of the parameters in the "Processor Specifics" and "Script Specifics" sections should remain as they are, except for "L2PPVERSION", whose value should be edited, if necessary, to reflect the current version. In the "Input Files" section, only the values of the parameter "HIRDLS1C" would need editing, as this value changes with each day's run. Use the sample listing in Figure 4 as a template for these values. All the values

for the parameters in the "Output Files" section can be changed to whatever you like, with the "HIRDLS1R" parameter value reflecting the name of the HIRDLS1R file that L2 Preprocessor will create. The values of the parameters in the "Processing Time Range" section should be ignored, as L2 Preprocessor does not currently use them. To reiterate, once the "usfmask" file contains valid values, invoke L2 Preprocessor by entering, at the Unix prompt, "L2PP.py usfmask" (without the parentheses).

```
# usfmask
# Mask file used by run script to fill L1X process control file
# Charles Cavanaugh

# Processor Specifics
L2PPEXEC     = L2PP
L2PPVERSION = v6.2.11
DORADIANCEADJUSTMENT = 1

# Script Specifics
L2PPPCF_MASK = L2PP_pcf.txt
L2PPPCF      = L2PP.pcf

# Input Files
HIRDLS1RMCF = HIRDLS1R.mcf
L2PPFOV     = HIRFOV_VerticalResponse.he5
L2PPADJUST1 = rad_adjust_v27_c19-1_2006d138.txt
L2PPADJUST2 = rad_adjust_v27_c19-2_2006d138.txt
L2PPFILTER  = FilterConfig.txt
HIRDLS1C    = HIRDLS1C_8011_2006d138.hdf

# Output Files
HIRDLS1R        = HIRDLS1R_629_2006d138.hdf
L2PPREPORT      = L2PPReport.txt
L2PPDIAGNOSTICS = L2PPDiagnostics.txt
L2PPSCREENOUT   = L2PP_stdout

# Processing time range
PROCESSINGSTARTTIME = ${data_start_time}
PROCESSINGSTOPTIME  = ${data_end_time}
```

Figure 4 Usfmask Parameter Value Listing

6 Validating the System

To validate the system, it is assumed you have followed the steps in Section 5, and have an output HIRDLS1R file. Congratulations! You are 2/3 the way to having a useful HIRDLS Level 2 Preprocessor file. This section will detail how to validate the system.

6.1 Toolkit Log Files

In the directory in which L2 Preprocessor was run, you should have 3 ASCII toolkit log files. These are named "LogReport", "LogStatus" and "LogUser". The "LogStatus" file is the useful one. This is the file into which SDP Toolkit writes its errors, warnings and reports. There is an endless permutation of these three types of messages, and it is impossible to detail them all, but if the only message is the ubiquitous "W A R N I N G" message about Toolkit version mismatch, then L2 Preprocessor has run successfully, in the view of SDP Toolkit.

6.2 System Log Files

Upon finishing, L2 Preprocessor generates an ASCII file called "L2PPReport.txt", which is a status report of the L2 Preprocessor run. L2 Preprocessor was designed to always create this file, no matter the status of the run. The only time this file won't be generated is if L2 Preprocessor dumps core. If this is the case, you need to go back to Section 4 and review why this happened.

Figure 5 details the formatted listing of "L2PPReport.txt", which are occurences that L2 Preprocessor deems worthy of special note. Figure 5 shows a typical, non-eventful run of L2 Preprocessor. Of special note is the last line. If the last line does not start with "Normal Termination", then L2 Preprocessor did not normally terminate. If this is the case, the message string after the colon will give a succinct reason why L2 Preprocessor terminated abnormally, and this can be used as a starting to point to determine what went wrong.

The first 3 lines of "L2PPReport.txt" should look very much like what is shown in Figure 5. The number of scans identified, filtered and adjusted should all match. If not, L2 Preprocessor detected anomalous scans, which should have been removed from processing view by upstream HIRDLS processors. The final number, that of the scans adjusted, is the number of scans in the HIRDLS1R file. It is important to note that this number will change from day to day. A typical non-eventful day will have many thousands of scans, depending on the HIRDLS scan table. There are, however, many "eventful" days, such as those with an Aura yaw maneuver, which will have less scans than expected, and at least one data "hole".

```
       5553  Scans identified
       5553  Scans filtered
       5553  Scans adjusted
Normal Termination : No anomalies during processing
```

Figure 5 L2PPReport.txt Listing

6.3 Metadata

A normally-terminated run of L2 Preprocessor will generate a metadata file for the output HIRDLS1R file, and the name of the file will be the same as the HIRDLS1R file, but with ".met" attached at the end of the name. The information in this file is used for ingestion of the HIRDLS1R file into the ESDIS system, and does not contain much useful information. The parameters that include "DATE" and/or "TIME" could be examined to check veracity of the run.

6.4 Graphical & System Tools

There are numerous graphical and/or system tools that will closely examine the fields inside the HIRDLS1R file. Included in this list are "h5ls" and "h5dump". The HIRDLS program relied heavily on Matlab and IDL programs to examine the contents of all the generated HIRDLS data products. These are not included in the DAP, as these were generated with minimal documentation and minimal notation, and were intended for in-house use only. If you are interested in using any of these, please contact your HIRDLS representative.