SPIRou Data Reduction Software

Developer Guide

0.3.004

For DRS SPIRou 0.2.058 (alpha pre-release)

N. Cook, F. Bouchy, E. Artigau, , M. Hobson, C. Moutou, I. Boisse, E. Martioli

2018-07-03

Abstract

This is the guide to coding the DRS (including installation, running, rules and standardisation approaches). This document is not intended for the general used of the DRS, instead it is intended for those who wish to develop the software further and understand the changes between this version and previous versions.

Contents

Chapter 1

Introduction

This documentation will cover the installation, data architecture, the changes between the previous versions and this version, using the DRS (with a working example), descriptions of the variables and keywords for input and output FITS rec headers , and the recipes and module code .

Variables are defined in detail in section 10 and will be defined throughout via the following syntax: VARIABLE. When referred to, one should take it as using the value set in section 10 by default or in the file described in the variables description 'Defined in' section. Clicking these variables will go to the appropriate variable description.

Certain sections will be written in code blocks, these imply text that is written into a text editor, the command shell console, or a python terminal/script. Below explains how one can distinguish these in this document.

The following denotes a line of text (or lines of text) that are to be edited in a text editor.

Generic text file

```
# A variable name that can be changes to a specific value
VARIABLE_NAME = "Variable Value"
```

These can also be shell scripts in a certain language:

For example in ~/.bashrc

```
#!/usr/bin/bash
# Find out which console you are using
echo $0
# Set environment Hello
export Hello="Hello"
```

For example in ~/.tcshrc

```
#!/usr/bin/tcsh
# Find out which console you are using
echo $0
# Set environment Hello
setenv Hello "Hello"
```

The following denotes a command to run in the command shell console

```
>> cd ~/Downloads
```

The following denotes a command line print out

```
This is a print out in the command line
produced by using the echo command
```

The following denotes a python terminal or python script

Python/Ipython

```
import numpy as np
print("Hello world")
print("{0} seconds".format(np.sqrt(25)))
```

The following denotes LaTeXcode (in raw form and then compiled form) - this is used in Section 8.

LaTeX

```
This is my \LaTeX code.
```

This is my LaTeXcode.

Chapter 2

Quick Install Guide

2.1 Linux

This is a quick guide to installation, for a more full description please see Chapter 3 . This assumes you have the latest version of Anaconda for python 3 (or python 2) and are using BASH.

1. Get the latest version of the DRS (for SPIRou version 0.2.058 (alpha pre-release)) from here: https://github.com/njcuk9999/spirou_py3
2. Download the test data from here: http://genesis.astro.umontreal.ca/neil/spirou_test_data_alpha0.1.003.zip (if required).
3. Extract the DRS (make a note of the path, hereinafter DRS_DIR).
4. Copy DRS_DIR /INTROOT/config/config.py to a user directory (e.g. /home/user/spirou/) hereinafter USER_DIR .
5. Add the following paths to your PATH and PYTHON PATH environmental variables

   e.g. in ~/.bashrc

   ```
   export PATH="DRS_DIR/INTROOT/bin/":{$PATH}
   export PYTHONPATH="DRS_DIR/INTROOT/":"DRS_DIR/INTROOT/bin/":{$PYTHONPATH}
   export DRS_UCONFIG="USER_DIR"
   ```

6. make sure your paths are set

   ```
   >>  source ~/.bashrc
   >>  echo $PATH
   ```

7. Make recipes executable (found in the DRS_DIR /bin folder) - to use from the command line.
8. Setup the DRS paths (edit the file: 'USER_DIR /config /config.py '):

TDATA	=	/drs/data/	/	Define the DATA directory
DRS_ROOT	=	/drs/INTROOT/	/	Define the installation directory
DRS_DATA_RAW	=	/drs/data/raw	/	Define the folder with the raw data files in
DRS_DATA_REDUC	=	/drs/data/reduced	/	Define the directory that the reduced data should be saved to/read from
DRS_CALIB_DB	=	/drs/data/calibDB	/	Define the directory that the calibration files should be saved to/read from
DRS_DATA_MSG	=	/drs/data/msg	/	Define the directory that the log messages are stored in
DRS_DATA_WORKING	=	/drs/data/tmp/	/	Define the working directory

   where DRS_ROOT should be "DRS_DIR /INTROOT".
9. validate the DRS installation:

   ```
   >> cal_validate_spirou.py
   ```

   Select "Yes" when prompted to setup the calibration database.

The DRS is now installed and set-up. To run see section 5 .

2.2 Mac

This is a quick guide to installation, for a more full description please see Chapter 3 . This assumes you have the latest version of Anaconda for python 3 (or python 2) and are using BASH.

1. Get the latest version of the DRS (for SPIRou version 0.2.058 (alpha pre-release)). from here: https://github.com/njcuk9999/spirou_py3
2. Download the test data from here: http://genesis.astro.umontreal.ca/neil/spirou_test_data_alpha0.1.003.zip (if required).
3. Extract the DRS (make a note of the path, hereinafter DRS_DIR)
4. Copy DRS_DIR /INTROOT/config/config.py to a user directory (e.g. /Users/user/spirou/) hereinafter USER_DIR .
5. Add the following paths to your PATH and PYTHON PATH environmental variables

   e.g. in ~/.bashrc or ~/.bash_profile

   ```
   export PATH=.:"DRS_DIR/INTROOT/bin/":{$PATH}
   export PYTHONPATH=.:"DRS_DIR/INTROOT/":"DRS_DIR/INTROOT/bin/":{$PYTHONPATH}
   export DRS_UCONFIG="USER_DIR"
   ```

6. make sure your paths are set

   ```
   >>  source ~/.bashrc
   >>  echo $PATH
   ```

7. Make recipes executable (found in the DRS_DIR /bin folder) - to use from the command line.
8. Setup the DRS paths (edit the file: 'USER_DIR /config /config.py '):

TDATA	=	/drs/data/	/	Define the DATA directory
DRS_ROOT	=	/drs/INTROOT/	/	Define the installation directory
DRS_DATA_RAW	=	/drs/data/raw	/	Define the folder with the raw data files in
DRS_DATA_REDUC	=	/drs/data/reduced	/	Define the directory that the reduced data should be saved to/read from
DRS_CALIB_DB	=	/drs/data/calibDB	/	Define the directory that the calibration files should be saved to/read from
DRS_DATA_MSG	=	/drs/data/msg	/	Define the directory that the log messages are stored in
DRS_DATA_WORKING	=	/drs/data/tmp/	/	Define the working directory

   where DRS_ROOT should be "DRS_DIR /INTROOT".
9. validate the DRS installation:

   ```
   >> cal_validate_spirou.py
   ```

   Select "Yes" when prompted to setup the calibration database.

The DRS is now installed and set-up. To run see section 5 .

2.3 Windows

This is a quick guide to installation, for a more full description please see Chapter 3 . This assumes you have the latest version of Anaconda for python 3 (or python 2)

1. Get the latest version of the DRS (for SPIRou version 0.2.058 (alpha pre-release)). from here: https://github.com/njcuk9999/spirou_py3
2. Download the test data from here: http://genesis.astro.umontreal.ca/neil/spirou_test_data_alpha0.1.003.zip (if required).
3. Extract the DRS (make a note of the path, hereinafter DRS_DIR)
4. Copy DRS_DIR \INTROOT\config\config.py to a user directory (e.g. C:\Users\user\spirou\) hereinafter USER_DIR .
5. Add the following paths to your PATH environmental variable

   In "Environmental Variables"

   ```
   DRS_DIR\INTROOT\bin\;
   ```

6. Add the following paths to your PYTHONPATH environmental variable

   In "Environmental Variables"

   ```
   %PYTHONPATH%;DRS_DIR\INTROOT\;DRS_DIR\INTROOT\bin\;
   ```

7. Add the following path to a new environmental variable

   In "Environmental Variables"

   ```
   %DRS_UCONFIG%;USER_DIR;
   ```

8. Setup the DRS paths (edit the file: 'USER_DIR /config /config.py '):

   ```
   TDATA          = C:\\Users\\User\Documents\    / Define the DATA directory
                    \drs\\data
   DRS_ROOT       = C:\\Users\\User\              / Define the installation directory
                    \Documents\\drs\\INTROOT
   DRS_DATA_RAW   = C:\\Users\\User\              / Define the folder with the raw data files
                    \Documents\\drs\\data\\raw      in
   DRS_DATA_REDUC = C:\\Users\\User\              / Define the directory that the reduced
                    \Documents\\drs\\data\          data should be saved to/read from
                    \reduced
   DRS_CALIB_DB   = C:\\Users\\User\              / Define the directory that the calibra-
                    \Documents\\drs\\data\          tion files should be saved to/read from
                    \calibDB
   DRS_DATA_MSG   = C:\\Users\\User\              / Define the directory that the log mes-
                    \Documents\\drs\\data\\msg      sages are stored in
   DRS_DATA_WORKING= C:\\Users\\User\             / Define the working directory
                    \Documents\\drs\\data\\tmp
   ```

   Note: Note paths in windows must have a '\\' also the python files must be open with a valid editor such as Sublime Text, Notepad++, Spyder or Pycharm for example

   where DRS_ROOT should be "DRS_DIR \INTROOT".

9. validate the DRS installation:

```
>> python cal_validate_spirou
```

Select "Yes" when prompted to setup the calibration database.

The DRS is now installed and set-up. To run see section 5 .

Chapter 3

Installation

3.1 Introduction

Once finalized the installation should just be a download, run set-up.py and configure the DRS directories, however, during development the following stages are required.

> **Note:** Currently the download repository on git-hub is private and requires a git-hub account, and the user to be added to the list of collaborators. To be added to the collaborators please email neil.james.cook@gmail.com with your git-hub user name.

3.2 Download

Get the latest version of the DRS (for SPIRou version 0.2.058 (alpha pre-release)). Use any of the following ways:

- manually download from here: https://github.com/njcuk9999/spirou_py3\protect\kern+.1667em\relax
- use Git:

```
>> git checkout https://github.com/njcuk9999/spirou_py3.git
```

- use SVN:

```
>> svn checkout  https://github.com/njcuk9999/spirou_py3.git
```

- use ssh:

```
>> scp -r git@github.com:njcuk9999/spirou_py3.git
```

3.3 Prerequisites

It is recommended to install the latest version of Anaconda python distribution, available for Windows, macOS and Linux (here: https://www.anaconda.com/download/). However one can run the DRS on a native python installation.

We recommend python 3 over python 2 for long term continued support (however the latest version of the DRS supports the newest versions of python 2.7).

Note: Before installing the DRS you must have one of the following:

- Latest version of Anaconda (for python 2 or python 3) — RECOMMENDED
- An Up-to-date version of python (python 2 or python 3)

3.3.1 Anaconda python distribution

A valid version of the Anaconda python distribution (for python 2 or python 3) Currently tested version of python are:

- Python 2.7.13 and Anaconda 4.4.0 (or later)
- Python 3.6.3 and Anaconda 5.0.1 (or later) — RECOMMENDED

3.3.2 Separate python installation

An up-to-date version of python (either python 2 or python 3) and the following python modules (with version of python they were tested with).

- Python 3.6.4
  - ASTROPY (tested with version 2.0.3)
  - MATPLOTLIB (tested with version 2.1.2)
  - NUMPY (tested with version 1.14.0)
  - SCIPY (tested with version 1.0.0)
  - and the following built-in modules (comes with python): __FUTURE__, CALENDER, CODE, COLLECTIONS, DATETIME, FILECMP, GLOB, OS, PDB, PKG_RESOURCES, SHUTIL, STRING, SYS, TIME, WARNINGS.
- Python 2.7.14
  - astropy (tested with version 2.0.3)
  - matplotlib (tested with version 2.1.2)
  - numpy (tested with version 1.14.0)
  - scipy (tested with version 1.0.0)
  - and the following built-in modules (comes with python): __FUTURE__, CALENDER, CODE, COLLECTIONS, DATETIME, FILECMP, GLOB, OS, PDB, PKG_RESOURCES, SHUTIL, STRING, SYS, TIME, WARNINGS.

3.4 Installation Linux and macOS

Currently the DRS has to be installed manually. This involves the following steps:

1. Extraction (Section 3.4.1)
2. Copy the config file to a user defined path (Section 3.4.2)
3. Modify environmental settings (Section 3.4.3)
4. Make recipes executable (Section 3.4.4)

3.4.1 Extraction

The first step is to extract the DRS into a folder (the DRS_DIR).
Do this by using the following commands:

```
>> cd  DRS_DIR
>> unzip DRS.zip
```

3.4.2 Copy the config file to a user defined path

Copy the config file (DRS_DIR`/INTROOT/config/config.py`) to a user directory (e.g. `/home/user/spirou/` or `/Users/user/spirou/`) hereinafter USER_DIR. One can also copy the constants file (constants_SPIROU_H4RG.py). This config file (and constants file if copied) will overwrite any default values in the DRS folder (DRS_DIR`/INTROOT/config/`).

> **Note:** One does not need every value that is present in the default files only those values which are to be changed (this is a good way to keep track on non-default values that the DRS will use).

3.4.3 Modify environmental settings

The next step is to modify your PATH and PYTHONPATH environmental variables (to include the DRS_DIR) and add a new environmental variable 'DRS_UCONFIG'. This depends which shell you are using (type '`echo $0`' to find out which).

- In bash open the '.bashrc' or '.bash_profile' text file in your home (~) directory (or create it if it doesn't exist)

  e.g. in ~/.bashrc or ~/.bash_profile

  ```
  export PATH="DRS_DIR/INTROOT/bin/":{$PATH}
  export PYTHONPATH="DRS_DIR/INTROOT/":"DRS_DIR/INTROOT/bin/":{$PYTHONPATH}
  export DRS_UCONFIG="USER_DIR"
  ```

- In csh/tcsh open the '.cshrc' or '.tcshrc' text file in your home (~) directory (or create it if it doesn't exist).

  e.g. in ~/.tcshrc

  ```
  setenv PATH ".:DRS_DIR/INTROOT/bin/":{${PATH}}"
  ```

  In csh/tcsh one has to check whether '$PYTHONPATH' exists before adding to the PYTHONPATH. To check type:

```
>>  echo $PYTHONPATH
```

If the response is:

```
PYTHONPATH: Undefined variable.
```

Then in the '.cshrc' or '.tcshrc' add:

e.g. in ~/.tcshrc

```
setenv PYTHONPATH ".:DRS_DIR:DRS_DIR/bin/"
```

If the response is some paths, then in the '.cshrc' or '.tcshrc' add:

e.g. in ~/.tcshrc

```
setenv PYTHONPATH ".:DRS_DIR:DRS_DIR/bin/:${PYTHONPATH}"
```

3.4.4 Make recipes executable

To run the recipes from the command line (without starting python) one must make them executable. Do this by using the following command:

```
>> chmod +x DRS_DIR/bin/*.py
```

3.5 Setting up the DRS on Linux and macOS

Before running the DRS one must set the data paths.

The 'USER_DIR /config /config.py ' file should be located in the USER_DIR directory.
The following keywords **must** be changed (and must be a valid path):

TDATA	= /drs/data/	/	Define the DATA directory
DRS_ROOT	= /drs/INTROOT/	/	Define the installation directory
DRS_DATA_RAW	= /drs/data/raw	/	Define the folder with the raw data files in
DRS_DATA_REDUC	= /drs/data/reduced	/	Define the directory that the reduced data should be saved to/read from
DRS_CALIB_DB	= /drs/data/calibDB	/	Define the directory that the calibration files should be saved to/read from
DRS_DATA_MSG	= /drs/data/msg	/	Define the directory that the log messages are stored in
DRS_DATA_WORKING	= /drs/data/tmp/	/	Define the working directory

Note: Some text editors (such as TextEdit on macOS) use 'smart speech-marks' these characters are invalid for python and thus care must be taken when editing the configuration files. The cal_validate_spirou recipe (Section 3.6) will check for invalid characters and show the lines which do not have valid characters (valid characters are ASCII letters, digits, punctuation and white spaces as defined by pythons 'string' module). We suggest using text editors such as sublime-text or a python IDE such as Pycharm.

where DRS_ROOT should be "DRS_DIR /INTROOT".

3.5.1 Other configuration parameters

If one is using the DRS in the default setup one needs to make sure the following parameters are set correctly:

- USER_CONFIG is set to a value of 0 (or False)
- DRS_DEBUG is set to a value of 0 (or False)
- ICDP_NAME is set to 'constants_SPIROU_H4RG.py'

The following keywords can be changed:

DRS_PLOT	=	1	/	Whether to show plots
DRS_INTERACTIVE	=	1	/	Whether to run in interactive mode
COLOURED_LOG	=	True	/	Whether to have coloured text.
PRINT_LEVEL	=	"all"	/	Level at which to print
LOG_LEVEL	=	"all"	/	Level at which to log in log file

For the 'PRINT_LEVEL and LOG_LEVEL keywords the values are set as follows:

- "all" – prints all events
- "info" – prints info, warning and error events
- "warning" – prints warning and error events
- "error" – print only error events

3.6 Validating Installation on Linux and macOS

Note: One must install the DRS (Section 3.4) AND set up the DRS (Section 3.5) before validation will be successful.

There are four ways to run the DRS in Linux and macOS (thus four ways to verify installation was correct).

- To validate running from command line type:

```
>> cal_validate_spirou.py
```

- To validate running from python/ipython from the command line type:

```
>> python cal_validate_spirou
>> ipython cal_validate_spirou
```

- To validate running from ipython, open ipython and type:

Python/Ipython

```
run cal_validate_spirou.py
```

- To validate running from import from python/ipython, open python/ipython and type:

Python/Ipython

```
import cal_validate_spirou
cal_validate_spirou.main()
```

If validation is successful the following should appear:

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (0.0.1)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp/
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||
HH:MM:SS.S -   ||Validation successful. DRS installed corrected.
```

One will then be asked whether to set-up the calibration database (for first time use only). The following should appear:

```
First time installation?
Add required files to DRS?
  Setup calibration database? [Y]es or [N]o
```

to which the answer "Yes" will copy the calibration database files into the ic_calibDB_filename and the fits files into the calibDB directory.

3.7 Installation Windows

This is very similar currently to the Linux/macOS installation (in the future a '.exe' file will be given).

1. Extract to DRS_DIR with your favourite unzipping software (Section 3.7.1).
2. Copy the config file to a user defined path (Section 3.7.2)
3. Add DRS_DIR to your PYTHONPATH (Section 3.7.3)

3.7.1 Extraction

The first step is to extract the DRS into a folder (the DRS_DIR).

3.7.2 Copy the config file to a user defined path

Copy the config file (DRS_DIR `/INTROOT/config/config.py`) to a user directory (e.g. `C:\Users\user\spirou\`) hereinafter USER_DIR. One can also copy the constants file (constants_SPIROU_H4RG.py). This config file (and constants file if copied) will overwrite any default values in the DRS folder (DRS_DIR `/INTROOT/config/`).

> **Note:** One does not need every value that is present in the default files only those values which are to be changed (this is a good way to keep track on non-default values that the DRS will use).

3.7.3 How to modify environmental settings in windows

This process is a little more convoluted than on Linux or macOS system.

1. Go to 'My computer > Properties > Advanced System Settings > Environmental Variables'. Note in windows 10 you can also click the windows icon and type 'Advanced System Settings' then click 'Environment Variables'.
2. under system variable 'Path' click edit and add:

   In "Environmental Variables"
   ```
   DRS_DIR;DRS_DIR\INTROOT\bin;
   ```
3. if under system variable 'PYTHONPATH' exists click edit and add 'DRS_DIR;' to the end.

   i.e.

   In "Environmental Variables"
   ```
   DRS_DIR;DRS_DIR\INTROOT\bin;
   ```
4. if under system variables 'PYTHONPATH' does not exist create a new variable called 'PYTHONPATH' and add:

   In "Environmental Variables"
   ```
   %PYTHONPATH%;DRS_DIR\INTROOT\;DRS_DIR\INTROOT\bin\;
   ```
5. Add the following path to a new environmental variable

   In "Environmental Variables"
   ```
   %DRS_UCONFIG%;USER_DIR;
   ```

Figure 3.1 shows screen-grabs of the various steps above to aid in updating PATH and PYTHONPATH. For problems/troubleshooting see here: https://stackoverflow.com/questions/3701646.

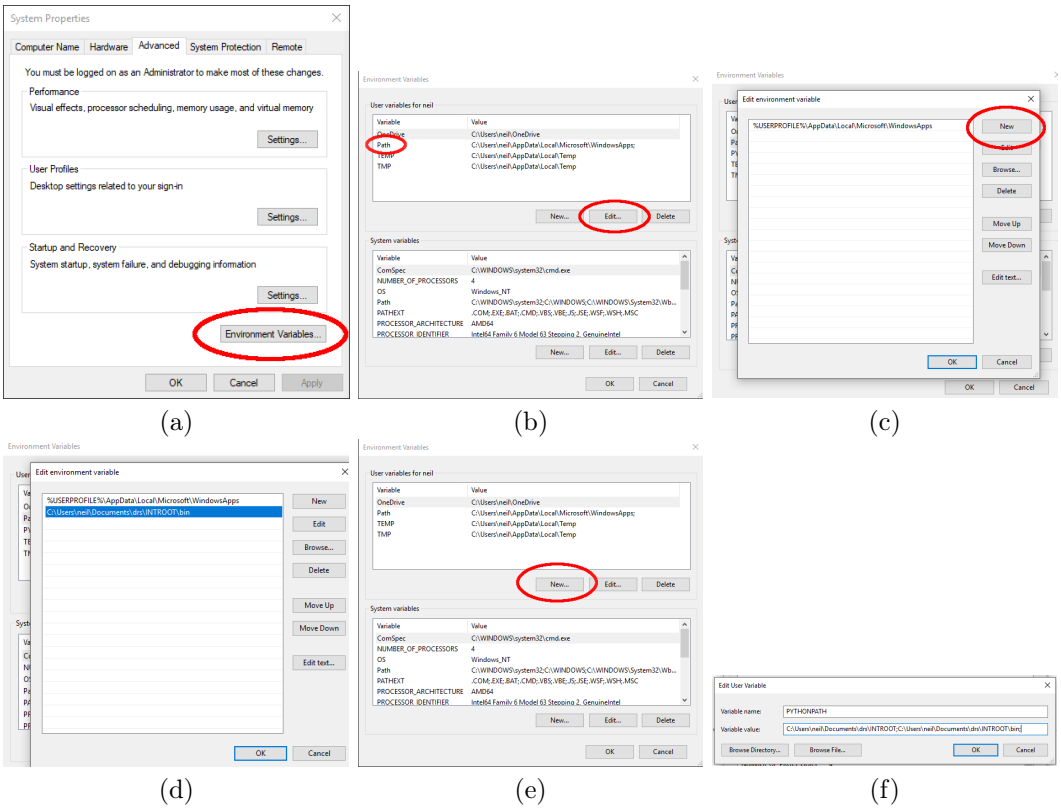


Figure 3.1 (a) Once in "Advanced system properties" click "Environment Variables" (b) Click "Path" and click "Edit..." to edit the "Path" environmental variable (c) Once in the "Path" environmental variable click "New" to add a new path (d) Type in the new line to add variable and click "OK" (e) Once back in the Environmental variable page click "New" to add 'PYTHONPATH' (f) Set the variable name to "PYTHONPATH" and edit the variable value accordingly.

3.8 Setting up the DRS (Windows)

Before running the DRS one must set the data paths.

The 'USER_DIR /config /config.py' file should be located in the USER_DIR directory.
The following keywords **must** be changed (and must be a valid path):

TDATA	= `C:\\Users\\User\Documents\\drs\\data`	/	Define the DATA directory
DRS_ROOT	= `C:\\Users\\User\\Documents\\drs\\INTROOT`	/	Define the installation directory
DRS_DATA_RAW	= `C:\\Users\\User\\Documents\\drs\\data\\raw`	/	Define the folder with the raw data files in
DRS_DATA_REDUC	= `C:\\Users\\User\\Documents\\drs\\data\\reduced`	/	Define the directory that the reduced data should be saved to/read from
DRS_CALIB_DB	= `C:\\Users\\User\\Documents\\drs\\data\\calibDB`	/	Define the directory that the calibration files should be saved to/read from
DRS_DATA_MSG	= `C:\\Users\\User\\Documents\\drs\\data\\msg`	/	Define the directory that the log messages are stored in
DRS_DATA_WORKING	= `C:\\Users\\User\\Documents\\drs\\data\\tmp`	/	Define the working directory

Note: Note: On windows paths in windows must have a '\\' also the python files must be open with a valid editor such as Sublime Text, Notepad++, Spyder or Pycharm for example

3.8.1 Other configuration parameters

If one is using the DRS in the default setup one needs to make sure the following parameters are set correctly:

- USER_CONFIG is set to a value of 0 (or False)
- DRS_DEBUG is set to a value of 0 (or False)
- ICDP_NAME is set to 'constants_SPIROU_H4RG.py'

The following keywords can be changed:

DRS_PLOT	=	1	/	Whether to show plots
DRS_INTERACTIVE	=	1	/	Whether to run in interactive mode
COLOURED_LOG	=	True	/	Whether to have coloured text.
PRINT_LEVEL	=	"all"	/	Level at which to print
LOG_LEVEL	=	"all"	/	Level at which to log in log file

For the 'PRINT_LEVEL and LOG_LEVEL keywords the values are set as follows:

- "all" – prints all events
- "info" – prints info, warning and error events
- "warning" – prints warning and error events
- "error" – print only error events

3.9 Validating Installation on Windows

> **Note:** One must install the DRS (Section 3.7) AND set up the DRS (Section 3.5) before validation will be successful.

In windows there are currently 3 ways to run the RS (running in python/ipython).

- To validate running from python/ipython from the command line type:

```
>> python cal_validate_spirou
>> ipython cal_validate_spirou
```

- To validate running from ipython, open ipython and type:

Python/Ipython

```
run cal_validate_spirou.py
```

- To validate running from import from python/ipython, open python/ipython and type:

Python/Ipython

```
import cal_validate_spirou
cal_validate_spirou.main()
```

If validation is successful the following should appear:

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (0.0.1)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp/
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||
HH:MM:SS.S -   ||Validation successful. DRS installed corrected.
```

One will then be asked whether to set-up the calibration database (for first time use only). The following should appear:

```
First time installation?
Add required files to DRS?
  Setup calibration database? [Y]es or [N]o
```

to which the answer "Yes" will copy the calibration database files into the ic_calibDB_filename and the fits files into the calibDB directory.

3.10 Having multiple set-ups

During development there is a period when both the H2RG and H4RG will be needed. As well as this there may be circumstances where one wants to switch between sets of constants.
This is a simple guide to switch between config and constant files.

> **Note:** The best and cleanest way to do this is to set up a custom directory for each detectors settings (i.e. see Section **??**)

the steps to change detector are as follows:

- Change ICDP_NAME to 'constants_SPIROU_H2RG.py' for the H2RG detector and 'constants_SPIROU_H4RG.py' for the H4RG detector.
- if using custom directories change DRS_UCONFIG to the required path.

An example custom 'config.py' is shown below:

In "/home/user/spirouH2RG/config.py"

```
DRS_ROOT = '/DATA/H2RG/'
DRS_DATA_RAW = '/DATA/H2RG/RAW'
DRS_DATA_REDIC = '/DATA/H2RG/RED'
DRS_CALIB_DB = '/DATA/H2RG/CALIBDB'
DRS_DATA_MSG = '/DATA/H2RG/MSG'
DRS_DATA_WORKING = '/DATA/H2RG/TMP'
```

In "/home/user/spirouH4RG/config.py"

```
DRS_ROOT = '/DATA/H4RG/'
DRS_DATA_RAW = '/DATA/H4RG/RAW'
DRS_DATA_REDIC = '/DATA/H4RG/RED'
DRS_CALIB_DB = '/DATA/H4RG/CALIBDB'
DRS_DATA_MSG = '/DATA/H4RG/MSG'
DRS_DATA_WORKING = '/DATA/H4RG/TMP'
```

Chapter 4

Data Architecture

Described below is the file structure, after correct installation (Chapter 3).

4.1 Installed file structure

The file structure should look as follows:

```
{dir}
├── {DRS_ROOT}
│   ├── bin ..................................................Recipes
│   ├── config ...............................................Configuration files
│   ├── documentation ........................................Documentation files
│   ├── man ..................................................Manual files
│   ├── misc .................................................Other scripts
│   └── SpirouDRS ............................................The DRS Module
└── {TDATA}*
    ├── calibDB
    ├── msg .2 raw ...........................................Observation directories
    │   └── YYYYMMDD .........................................Raw observation files
    ├── reduced
    └── tmp
```

* This is the recommended file structure and raw, reduced, calibDB, msg and tmp can be changed using the DRS_DATA_RAW, DRS_DATA_REDUC, DRS_CALIB_DB, DRS_DATA_MSG, and DRS_DATA_WORKING variables in Section 3.5.

i.e. for the paths given in Section 3.5 this would be:

```
drs
├── INTROOT
│   ├── bin
│   ├── config
│   ├── documentation
│   ├── man
│   ├── misc
│   └── SpirouDRS
└── data
    ├── calibDB
    ├── msg
    ├── raw
    │   └── YYYYMMDD
    ├── reduced
    └── tmp
```

4.2 The Installation root directory

The DRS_ROOT contains all the installed recipes, modules functions, documentation and configuration files needed to run the DRS. The file structure is set up as below:

```
{dir}
└─{DRS_ROOT}
   ├─bin ..........................................................Recipes
   ├─config .....................................................Configuration files
   ├─documentation ..............................................Documentation files
   └─SpirouDRS ..................................................The DRS Module
```

4.2.1 The bin directory

The bin directory is located in the DRS_ROOT directory. This contains all the recipes that can be used. A detailed description of all recipes can be found in Chapter 12 but are listed here for completeness.

```
{dir}
└─{DRS_ROOT}
   └─bin ..........................................................Recipes
      ├─cal_BADPIX_spirou ..................................See Section 12.3
      ├─cal_CCF_E2DS_spirou ................................See Section 12.9
      ├─cal_DARK_spirou ....................................See Section 12.2
      ├─cal_DRIFT_E2DS_spirou ..............................See Section 12.8
      ├─cal_DRIFTPEAK_E2DS_spirou ..........................See Section 12.8
      ├─cal_exposure_meter
      ├─cal_extract_RAW_spirou .............................See Section 12.7
      ├─cal_extract_RAW_spirouAB ...........................See Section 12.7
      ├─cal_extract_RAW_spirouC ............................See Section 12.7
      ├─cal_FF_RAW_spirou ..................................See Section 12.6
      ├─cal_HC_E2DS_spirou .................................See Section 12.11
      ├─cal_loc_RAW_spirou .................................See Section 12.4
      ├─cal_preprocess_spirou
      ├─cal_reset
      ├─cal_SLIT_spirou ....................................See Section 12.5
      ├─cal_validate_spirou ................................See Section 12.10
      ├─cal_WAVE_E2DS_spirou ...............................See Section 12.12
      ├─off_listing_RAW_spioru
      ├─pol_spirou
      ├─visu_E2DS_spirou
      ├─visu_RAW_spirou
      └─visu_WAVE_spirou
```

4.2.2 The SPIROU module directory

The SpirouDRS directory is the SPIROU DRS package, it contains all sub-packages that contain all the worker functions and code associated with the recipes. The modules are described in detail in Chapter 13.

The file structure is as follows:

```
SpirouDRS
├── data ..................................................The DRS data directory
├── fortran ..........................................The Fortran script directory
├── spirouBACK .....................The SPIRou background module (Section 13.2)
├── spirouCDB .............The SPIRou calibration database module (Section 13.3)
├── spirouConfig ..........The SPIRou configuration tools module (Section 13.4)
├── spirouCore ............................The SPIRou core modules (Section 13.5)
├── spirouEXTOR .....................The SPIRou extraction module (Section 13.6)
├── spirouFLAT ......................The SPIRou Flat field module (Section 13.7)
├── spirouImage ...........................The SPIRou image module (Section 13.8)
├── spirouLOCOR ..................The SPIRou localization module (Section 13.9)
├── spirouPOLAR ..................................The SPIRou polarization module
├── spirouRV .................The SPIRou radial velocity module (Section 13.11)
├── spirouStartup .............The SPIRou start up tools module (Section 13.12)
├── spirouTHORCA .......................The SPIRou THORCA module (Section 13.13)
├── spirouTools ..........................................The SPIRou tools module
└── spirouUnitTests ..............................The SPIRou unit tests module
```

4.3 The data root directory

This is the directory where all the data should be stored. The default and recommended design is to haveDRS_DATA_RAW, DRS_DATA_REDUC, DRS_CALIB_DB, DRS_DATA_MSG, and DRS_DATA_WORKING as sub-directories of DRS_ROOT. However as in Section 3.5. these sub-directories can be defined elsewhere.

4.3.1 The raw and reduced data directories

The raw observed data is stored under the DRS_DATA_RAW path, the files are stored by night in the form YYYYMMDD .

The file structure can be seen below:

```
{DRS_DATA_RAW}
└── YYYYMMDD ..................................................night_repository
    ├── ..................................................Raw observation files
    ├── {odometer number}d.fits
    ├── {odometer number}f.fits
    ├── {odometer number}a.fits
    ├── {odometer number}c.fits
    └── {odometer number}o.fits
```

where d=dark, f=flat, a=align, c=comparison and o=object

4.4 The calibration database directory

```
{TDATA}
└── calibDB or {DRS_CALIB_DB}
    ├── master_calib_SPIROU.txt
    └── ..................................................The calibration fits files
```

The calibDB contains all the calibration files that pass the quality tests and a test file ic_calibDB_filename. It is located at DRS_CALIB_DB or if this is not defined is located by default at the TDATA directory. Each line in this file is a unique calibration file and lines are formatted in the following manner:

In calibration database file

```
{key} {night_repository} {filename} {human readable date} {unix time}
```

where

- key is a code assigned for each type of calibration file. Currently accepted keys are:
  - DARK - Created from cal_DARK_spirou
  - ORDER_PROFIL_fiber - Created in cal_loc_RAW_spirou
  - LOC_C - Created in cal_loc_RAW_spirou
  - TILT - Created in cal_SLIT_spirou
  - FLAT_fiber - Created in cal_FF_RAW_spirou
  - WAVE - Currently manually added
- night_repository is the raw data observation directory (in DRS_DATA_RAW) normally in the form YYYYMMDD.
- filename is the filename of the calibration file (located in the calibDB).
- human readable date is the date in DD/MM/YY/HH:MM:SS.ss format taken from the header keyword 'ACQTIME1' of the file that created the calibration file.
- unix time is the time (as in human readable date) but in unix time (in seconds).

An example working ic_calibDB_filename is shown below (assuming the listed files are present in DRS_CALIB_DB)

In calibration database file

```
DARK 20170710 dark_dark02d406.fits 07/10/17/16:37:48 1499704668.0
ORDER_PROFIL_C 20170710 dark_flat02f10_order_profil_C.fits 07/10/17/17:03:50 1499706230.0
LOC_C 20170710 dark_flat02f10_loco_C.fits 07/10/17/17:03:50 1499706230.0
ORDER_PROFIL_AB 20170710 flat_dark02f10_order_profil_AB.fits 07/10/17/17:07:08 1499706428.0
LOC_AB 20170710 flat_dark02f10_loco_AB.fits 07/10/17/17:07:08 1499706428.0
TILT 20170710 fp_fp02a203_tilt.fits 07/10/17/17:25:15 1499705515.0
FLAT_C 20170710 dark_flat02f10_flat_C.fits 07/10/17/17:03:50 1499706230.0
WAVE 20170710 spirou_wave_ini3.fits 07/10/17/17:03:50 1499706230.0
```

Chapter 5

Using the DRS

There are two ways to run the DRS recipes. The first (described in Section 5.1) directly calls the code and inputs arguments (either from the command line or from python), the second way is to import the recipes in a python script and define arguments in a call to a function (see Section 5.2).

5.1 Running the DRS recipes directly

As in Chapter 3, using Linux or macOS one can run DRS recipes from the command line or from python, in windows one is required to be in python before running the scripts. Below we use cal_DARK_spirou as an example:

- To run from command line type:

```
>> cal_DARK_spirou YYMMDD Filenames
```

- To run from python/ipython from the command line type:

```
>> python cal_DARK_spirou YYMMDD Filenames
>> ipython cal_DARK_spirou YYMMDD Filenames
```

- To run from ipython, open ipython and type:

Python/Ipython

```
run cal_DARK_spirou YYMMDD Filenames
```

5.2 Running the DRS recipes from a python script

In any operating system one can also import a recipe and call a function to run the code. This is useful in batch operations, timing tests and unit tests for example. Below we use cal_DARK_spirou as an example:

Python/Ipython

```
# import the recipe
import cal_DARK_spirou
# define the night folder name
night_name = "20170710"
# define the file(s) to run through the code
files = ['dark_dark02d406.fits']
# run code
cal_validate_spirou.main(night_name=night_name, files=files)
```

5.3 Working example of the code for SPIRou

5.3.1 Overview

For this example all files are from 2018-04-09.
following our example data architecture (from Section 3.5 and shown explicitly in Section 4.1) all files should be places in the DRS_DATA_RAW (/drs/data/raw in our case) and placed into a night directory: '20180409'.
Starting with RAMP files and ending with extracted orders and calculated drifts we need to run six codes:

1. cal_DARK_spirou (See Section 12.2)
2. cal_BADPIX_spirou (See Section 12.3)
3. cal_loc_RAW_spirou (×2) (See Section 12.4)
4. cal_SLIT_spirou (See Section 12.5)
5. cal_FF_RAW_spirou (×2) (See Section 12.6)
6. cal_extract_RAW_spirou (See Section 12.7)
7. cal_DRIFT_E2DS_spirou (See Section 12.8)
8. cal_DRIFTPEAK_E2DS_spirou (See Section 12.8)
9. cal_CCF_E2DS_spirou (See Section 12.9)

5.3.2 Run through from command line/python shell (Linux and macOS)

As long as all codes are executable (see Section 3.4.4) one can run all codes from the command line or if not executable or one has a preference for python one can run the following with 'python {command}', 'ipython {command}' or indeed through an interactive ipython session using 'run {command}'.

1. run the pre-processing on all files:

```
>> cal_preprocess_spirou 20180409 "*.fits"
```

> **Note:** This adds the '_pp' suffix on the end of all pre-processed files.

2. run the dark extraction on the 'dark_dark' file:

```
>> cal_DARK_spirou.py 20180409 dark_dark_001_pp.fits
```

3. run the bad pixel identification on the 'flat_flat' and 'dark_dark' files:

```
>> cal_BADPIX_spirou.py 20180409 flat_flat_001_pp.fits dark_dark_001_pp.fits
```

4. run the order localisation on the 'dark_flat' files:

```
>> cal_loc_RAW_spirou.py 20180409 dark_flat_001_pp.fits
```

5. run the order localisation on the 'flat_dark' files:

```
>> cal_loc_RAW_spirou.py 20180409 flat_dark_001_pp.fits
```

6. run the slit calibration on the 'fp_fp' files.

```
>> cal_SLIT_spirou.py 20180409 fp_fp_001_pp.fits
```

7. run the flat field creation on the 'flat_flat' files:

```
>> cal_FF_RAW_spirou.py 20180409 flat_flat_001_pp.fits
```

8. run the extraction on the 'fp_fp' and 'hcone' files:

```
>> cal_extract_RAW_spirou.py 20180409 fp_fp_001_pp.fits
>> cal_extract_RAW_spirou.py 20180409 fp_fp_002_pp.fits
>> cal_extract_RAW_spirou.py 20180409 fp_fp_003_pp.fits
>> cal_extract_RAW_spirou.py 20180409 hcone_hcone_001_pp.fits
```

9. run the drift calculation on the 'fp_fp' files:

```
>> cal_DRIFT_E2DS_spirou.py 20180409 fp_fp_001_pp_e2ds_AB.fits
>> cal_DRIFTPEAK_E2DS_spirou.py 20180409 fp_fp_001_pp_e2ds_AB.fits
```

10. run the CCF calculation on an extracted file:

```
>> cal_CCF_E2DS_spirou 20180409 hcone_hcone_001_pp_e2ds_C.fits UrNe.mas 0 10 0.1
```

5.3.3 Run through python script

The process is in the same order as Section 5.3.2.

Python/Ipython

```
import cal_DARK_spirou, cal_loc_RAW_spirou
import cal_SLIT_spirou, cal_FF_RAW_spirou
import cal_extract_RAW_spirou, cal_DRIFT_RAW_spirou
import matplotlib.pyplot as plt
# define constants
NIGHT_NAME = '20180409'
# cal dark
files = ['dark_dark_001_pp.fits']
cal_DARK_spirou.main(NIGHT_NAME, files)
# cal badpix
flatfile = 'flat_flat_001_pp.fits'
darkfile = 'dark_dark_001_pp.fits'
cal_BADPIX_spirou.main(NIGHT_NAME, flatfile, darkfile)
# cal loc
files = ['dark_flat_001_pp.fits']
cal_loc_RAW_spirou.main(NIGHT_NAME, files)
# cal loc
files = ['flat_dark_001_pp.fits']
cal_loc_RAW_spirou.main(NIGHT_NAME, files)
# cal slit
files = ['fp_fp_001_pp.fits']
cal_SLIT_spirou.main(NIGHT_NAME, files)
# cal ff
files = ['flat_flat_001_pp.fits']
cal_FF_RAW_spirou.main(NIGHT_NAME, files)
# cal extract
files = ['fp_fp_001_pp.fits']
cal_extract_RAW_spirou.main(NIGHT_NAME, files)
files = ['fp_fp_002_pp.fits']
cal_extract_RAW_spirou.main(NIGHT_NAME, files)
files = ['fp_fp_003_pp.fits']
cal_extract_RAW_spirou.main(NIGHT_NAME, files)
files = ['hcone_hcone_001_pp.fits']
cal_extract_RAW_spirou.main(NIGHT_NAME, files)
# cal drift
reffile = ['fp_fp_001_pp_e2ds_AB.fits']
cal_DRIFT_E2DS_spirou.main(NIGHT_NAME, reffile)
cal_DRIFTPEAK_E2DS_spirou(NIGHT_NAME, reffile)
# cal ccf
reffile = 'hcone_hcone_001_pp_e2ds_C.fits'
mask = 'UrNe.mas'
cal_CCF_E2DS_spirou.main(NIGHT_NAME, reffile, mask, 0, 10, 0.1)
# close all the plots
plt.close('all')
```

Chapter 6

Current to do list

Below is the current to do list and any things that need to be addressed before release.

6.1 General

- Remove H2RG requirements
- Write help files for each recipe (In DRS_DIR /man filenames should be 'recipe name'.info)
- Need to sort out public and private variables (and keywords), some variables not needed to be changed by user – private and public configuration files
- Can we remove 'special_config_SPIROU' configuration file call as the file does not exist?
- fitsfilename is the last file in a group of files - is this correct or should it be the first (as initially defined)?
- Write a set-up.py installer / checker for prerequisites (Last step)
- Need axis labels for all plots

6.2 Documentation

- Write introduction (leading paragraph) ..... Dev
- Update output keywords
- Update recipes chapter ..... User + Dev

6.3 The cal_SLIT_spirou recipe

- fiber loop overwrites loc

6.4 The cal_FF_RAW_spirou recipe

- SpirouDRS.spirouBACK.spirouBACK .measure_background_flatfield() needs converting from C to python (interpol.c) - currently not used so not converted – background set to zero.
- SpirouDRS.spirouBACK.spirouBACK .measure_min_max() why is the max_signal the third biggest value and not a percentile?

6.5 The cal_extract_RAW_spirou recipes

- SpirouDRS.spirouBACK.spirouBACK .measure_background_flatfield() needs converting from C to python (interpol.c) - currently not used so not converted – background set to zero.
- SpirouDRS.spirouBACK.spirouBACK .measure_min_max() why is the max_signal the third biggest value and not a percentile?
- Quality control test QC_MAX_SIGNAL ignored for some reason - Why?

6.6 The cal_HC_E2DS_spirou recipe

- replace Fortran 'fitgaus.f' routine with python (including testing different versions)

6.7 The cal_WAVE_E2DS_spirou recipe

- compare cal_WAVE to cal_WAVE_NEW
- see Section 6.6

Chapter 7

Coding style and standardization

To keep the code neat, tidy, consistent and professional the following sections suggest guideline by which the DRS should conform to.

7.1 PEP 8 - A style guide for python code

PEP 8 is a style guide for python it lays out a specific way to format python code, a full guide can be found here: https://www.python.org/dev/peps/pep-0008/ but the following summarizes the main points used in the DRS.

- Code lay-out
  - 4 spaces per indentation level (spaces not tabs)
  - Continuation lines should align wrapped elements
  - Maximum line length of 79 characters
  - Surround top-level functions and class definitions with two blank lines (methods with one blank line and all other code with one blank line maximum)
  - imports should usually be on separate lines
- White space in expressions and statements
  - No white spaces immediately inside parentheses, brackets or braces
  - No white spaces immediately before a comma, semicolon, or colon (exception for slicing)
  - No white spaces immediately before the open parenthesis that starts the argument list of a function call
  - No white spaces immediately before the open parenthesis that starts an indexing or slicing
  - Exactly one white space around an assignment (or other) operator
  - No space around the = sign when used to indicate a keyword argument or a default parameter value
  - Avoid compound statements (multiple statements on the same line)
- Comments
  - Comments should start with a # and be followed by a single white space
  - In-line comments should be used sparingly
  - All functions, classes and methods should have a valid document string (see here: https://www.python.org/dev/peps/pep-0257)
- Naming conventions
  - Never use lower-case letter el 'l', upper-case letter 'oh' 'O', or upper-case letter 'eye' 'I' as single character variables names
  - Class Names should normally use CamelCase (words should be Capitalized)
  - Functions names should be lower-case with words separated by underscores as necessary (same is true for global variable names)
  - Constants defined on a module level should be written in capital letters with underscores separating words

7.2 DRS specific style and standardization

In addition to PEP-8 we stick to some extra style and standardization points, these include some custom objects to help the ease of development and user experience.

7.2.1 Functions from sub-modules

Unlike 'normal' functions these are written in CamelCase without underscores between words. This is done to distinguish them from standard functions. They are always defined in a module (or sub-modules) __init__() code and are essentially public aliases to module level code. An example is presented below.

Python/Ipython

```
# --------------------------
# in the module file spirouMath.py
# --------------------------
def add_x_to_y(x, y):
  """
  Returns the summation of x and y
  :param x: float, the first term to add
  :param y: float, the second term to add
  :return z: float, the summation of x and y
  """
  # add x to y
  z = x + y
  # return z
  return z

# --------------------------
# in the __init__ file for spirouCore
# --------------------------
# import from local code
from . import spirouMath
# publicly defined alias to local code function
AddXtoY = spirouMath.add_x_to_y

# --------------------------
# in the recipe
# --------------------------
# import sub-module
from SpirouDRS import spirouCore
# set up constants
x = 4.123
y = 5.234
# add via function
z = spirouCore.AddXtoY(x, y)
```

7.2.2 The logger (WLOG)

As in previous version of the DRS the printing and logging is controlled by a function. In this version of the DRS this is in SpirouDRS.spirouCore.spirouLog.logger but in most recipes/modules this is aliased to WLOG. The WLOG function controls both the printing to the screen (standard output) and to a log file. Where and how this is done is controlled by several variables.

The format of the log entry (whether it is printed to the standard output or to the logging file) is as follows:

Python/Ipython

```
WLOG(level, program, message)
```

and produces the following entry (in log or standard output)

```
HH:MM:SS.s - char | program | message
```

where the 'char' is dependent on the input level.
The 'char' and level are a dictionary pair in the form 'level = char' and is controlled by trig_key (see section 10.26) i.e. by default the level char pairs are:

Python/Ipython

```
dict(all=' ', error='!', warning='@', info='*', graph='~')
```

The level also determines whether or not a message is shown in the screen (standard output) or in the log. A log message will be shown if it has a numeric value (defined in write_level) higher than that set in PRINT_LEVEL for printing to the screen (standard output) or set in LOG_LEVEL for printing to the log.

i.e.:

Python/Ipython

```
write_level = dict(error=3, warning=2, info=1)
trig_key = dict(all=' ', error='!', warning='@', info='*', graph='~')
PRINT_LEVEL = 'warning'

WLOG('info', 'program', 'Info message')
WLOG('warning', 'program', 'Warning message')
WLOG('error', 'program', 'Error message')
```

returns

```
HH:MM:SS.s - @ |program|Warning message
HH:MM:SS.s - ! |program|Error message
```

Note: Note the info message was not shown as info=1 and PRINT_LEVEL is set to warning=2.

In addition to logging the certain levels can be set to exit the DRS recipe when they are used. They are defined in exit_levels and exiting python is controlled via exit and log_exit_type.

i.e.

Python/Ipython

```
write_level = dict(error=3, warning=2, info=1)
trig_key = dict(all=' ', error='!', warning='@', info='*', graph='~')
exit_levels = ['error']
PRINT_LEVEL = 'warning'

WLOG('error', 'program', 'Error message')
WLOG('info', 'program', 'Info message')
WLOG('warning', 'program', 'Warning message')
```

returns

```
HH:MM:SS.s - ! |program|Error message
```

Note: Note that 'WLOG('error')' triggered the recipe/module to exit python, thus no other logs were printed.

7.2.3 The coloured log

In addition to the features above the log can be coloured to aid usability. Currently errors are coloured red, warnings are coloured yellow and all other text is coloured green. These colours can be changed using clevels or turned on/off using COLOURED_LOG.

An example of each is shown below:

Python/Ipython

```
WLOG('all', 'program', 'All message')
WLOG('info', 'program', 'Info message')
WLOG('warning', 'program', 'Warning message')
WLOG('error', 'program', 'Error message')
WLOG('all', 'program', 'All message')
```

```
HH:MM:SS.s -   |program|All message
HH:MM:SS.s - * |program|Info message
HH:MM:SS.s - @ |program|Warning message
HH:MM:SS.s - ! |program|Error message
HH:MM:SS.s -   |program|All message
```

7.2.4 The Parameter Dictionary Object

While running the DRS there are many variables defined in many places that are used throughout the recipes, DRS module and sub-modules, defined in configuration files and from certain sub-modules and recipes. It is important as a developer (and for proper error handling) to keep track of where this variables are being defined and changed in the DRS.

For this reason, and for convenience for passing between functions and recipes, a new object, based on a dictionary has been defined to handle all variables defined throughout the DRS. This is the parameter dictionary (ParamDict) class (defined in SpirouDRS.spirouConfig.spirouConfig).

The ParamDict is a custom dictionary class (that inherits all attributes and methods from the standard python dictionary object), with the ability to get and set a source for each key value pair. In addition to this all variables stored are **insensitive to case** (i.e. upper-case variables, lower-case variables and mixed case variables are stored as the **same** variable).

Construct/initiate the ParamDict in the same way one would a python dictionary:

Python/Ipython

```
# as an empty dictionary
p1 = ParamDict()
# from a list of keys and values (using zip)
p2 = ParamDict(zip(keys, values))
```

Once created key, value pairs are created the same way one would with a python dictionary.

Python/Ipython

```
# set a key, value pair
p1['test'] = 1
# ParamDict are case insensitive 'Test' overwrites 'test' and 'teST'
p1['Test'] = 99
```

After creating a key the source should be set. This can be done as follows:

Python/Ipython

```
# -----------------------
# Set a single source
# -----------------------
# set the key value pair
p1['TEST'] = 1
# set the source
p1.set_source('TEST', 'test.py/__main__()')
```

One can also add a set of sources (after creating multiple key value pairs)

Python/Ipython

```
# -----------------------
# Set a list of sources
# -----------------------
# set the key value pairs
p1['a'] = 1
p1['b'] = 2
p1['c'] = 3
# set the sources
p1.set_sources(['a', 'b', 'c'], 'SpirouConfig.DefineConstants()')
```

or one can set all sources in the ParamDict to a specific source

Note: Note set all sources will change every source in the ParamDict so should only be used after ParamDict created from a set of key value pairs

Python/Ipython

```
# -----------------------
# Set all sources
# -----------------------
# create ParamDict
keys = ['a', 'b', 'c', 'd', 'e']
values = [1, 2, 3, 4, 5]
p3 = ParamDict(zip(keys, values))
# set all sources
p3.set_all_sources('SpirouMath.LetterNumbers()')
```

The parameter dictionary also contains some helper functions (as the parameter dictionary can get quite large).

- startswith - This allows one to search for all keys that start with a defined string

  Python/Ipython

  ```
  p.startswith('DRS')
  ```

```
['DRS_PLOT',
 'DRS_DEBUG',
 'DRS_ROOT',
 'DRS_DATA_RAW',
 'DRS_DATA_REDUC',
 'DRS_CALIB_DB',
 'DRS_DATA_MSG',
 'DRS_DATA_WORKING',
 'DRS_NAME',
 'DRS_VERSION',
 'DRS_CONFIG',
 'DRS_MAN',
 'DRS_USED_DATE',
 'DRS_INTERACTIVE']
```

- contains - This allows one to search for all keys that contain a defined string

Python/Ipython

```
p.contains('calib')
```

```
['DRS_CALIB_DB', 'IC_CALIBDB_FILENAME', 'CALIB_MAX_WAIT', 'CALIB_DB_MATCH']
```

- endswith - This allows one to search for all keys that end with a defined string

Python/Ipython

```
p.endswith('ALL')
```

```
['NBFIB_FPALL',
 'IC_FIRST_ORDER_JUMP_FPALL',
 'IC_LOCNBMAXO_FPALL',
 'QC_LOC_NBO_FPALL',
 'FIB_TYPE_FPALL',
 'IC_EXT_RANGE1_FPALL',
 'IC_EXT_RANGE2_FPALL',
 'IC_EXT_RANGE_FPALL',
 'LOC_FILE_FPALL',
 'ORDERP_FILE_FPALL',
 'IC_EXT_D_RANGE_FPALL']
```

7.2.5 Configuration Error and Exception

As mentioned above in section 7.2.4 it is important to handle errors caused by variable definition. Included in the parameter dictionary definitions are a new set of exception handlers to be used with ParamDict and the SpirouDRS.spirouCore.spirouLog .logger (aliased to WLOG in most modules/recipes). It is very similar to standard python Exceptions but adds some new methods that can be accessed to be used with WLOG .

An example is below of the ConfigError exception (without using ParamDict)

Python/Ipython

```
def a_function():
    try:
        # some_code that causes an exception
        x = dict()
        y = x['a']
        return y
    except KeyError:
        # define a log message
        message = 'a was not found in dictionary x'
        raise ConfigError(message, level='error')

# Main code:
try:
    a_function()
except ConfigError as e:
    WLOG(e.level, 'program', e.message)
```

This functionality is coded into ParamDict (with a WLOG level set to 'error') thus one only needs the following code:

Python/Ipython

```
# set up the ParamDict
x = ParamDict()
# Main code:
try:
    y = x['add']
except ConfigError as e:
    WLOG(e.level, 'program', e.message)
```

and the result will be as follows:

```
HH:MM:SS.s - ! |program|Parameter "add" not found in parameter dictionary
```

Note: Due to WLOG 'error' currently meaning the code is exited a missing parameter will print the above message and then exit using the log_exit_type exit strategy (see section 7.2.2).

Chapter 8

Writing the documentation

8.1 Documentation Architecture

The documentation is written in LaTeXand to ease writing many packages and customizations are used. The documentation is located in the ./documentation folder. Both the user documentation and the developer documentation (this document) are written together.

The main .tex files are User_guide_spirou_drs.tex and Dev_guide_spirou_drs.tex these are the files that should be compiled by LaTeX. As well as this file there are four directories, the 'Chapters' directory (containing the content of each chapter), the 'Config' directory (containing custom commands, formatting and constants - see Section 8.4, Section 8.5, and Section 8.6), the 'Figures' directory (containing all figures and graphics) and the 'Tables' directory containing table .tex files.

The documentation is currently written using the 'memoir' class file (texdoc.net/texmf-dist/doc/latex/memoir/memman.pdf) and uses custom chapter styles from ctan.org/pkg/memoirchapterstyles (Contained within the .documentation/Config/preamble.tex file).

8.2 Required LaTeXpackages

To compile in LaTeXone needs the following document class:

- memoir ..... Typeset fiction, non-fiction and mathematical books

To compile in LaTeX one needs the following packages:

- inputenc ..... utf8 encoding
- fontenc ..... Standard package for selecting font encodings
- babel ..... Multilingual support for Plain TeXor LaTeX
- microtype ..... Subliminal refinements towards typographical perfection
- amsmath ..... AMS mathematical facilities for LaTeX
- amssymb ..... AMS symbols for LaTeX
- mathtools ..... Mathematical tools to use with amsmath
- memhfixc ..... Adjustment for using hyperref in memoir documents
- graphicx ..... Enhanced support for graphics
- listings ..... Typeset source code listings using LaTeX
- xcolor ..... Driver-independent colour extensions for LaTeX and pdfLaTeX
- hyperref ..... Extensive support for hypertext in LaTeX
- dirtree ..... Display trees in the style of windows explorer
- framed ..... Framed or shaded regions that can break across pages
- multirow ..... Create tabular cells spanning multiple rows
- float ..... Improved interface for floating objects

- background ..... Placement of background material on pages of a document
- tcolorbox ..... Coloured boxes, for LaTeX examples and theorems
- eso-pic ..... Add picture commands (or backgrounds) to every page
- ulem ..... Package for underlining
- tocloft ..... Control table of contents, figures, etc
- caption ..... Customising captions in floating environments
- pdflscape ..... Make landscape pages display as landscape
- xifthen ..... Extended conditional commands

8.3 Developer documentation content

As mentioned above we write the developer documentation and user guide using the same files, for this reason one needs a way to distinguish content that is unique to the user documentation or the developer documentation. This is done using the boolean statement '\ifdevguide' (defined in the main .tex files for the user documentation - \devguidefalse - and developer documentation \devguidetrue).
An example of a different content for each type of documentation is below:

LaTeX
```
\ifdevguide
This is the developer guide.
\else
This is the user guide.
\fi
```

This is the developer guide.

an example of content only for the developer guide is below:

LaTeX
```
\ifdevguide
This section is only for developers
\fi
```

This section is only for developers

an example of content only for the user guide is below:

LaTeX
```
\ifdevguide
\else
This section is only for developers
\fi
```

Note: As this is the developer guide the content for the user guide only will not be present.

Note: It is probably never the case where the user documentation will have content that the developer documentation does not need.

8.4 Custom Commands

To ease writing the documentation some custom commands are defined in ./documentation/Config/commands.tex. These include the following:

- \definevariable{label reference}{text} - used to create in-text variables (that link to the variables chapter) i.e.:

LaTeX
```
The variable \definevariable{ch:variables}{VARIABLE} can be used.
```
The variable VARIABLE can be used.

- \defineinkeyword{label reference}{text}, \defineoutkeyword{label reference}{text} - used to create in-text keywords (that link to the keywords chapter) i.e.:

LaTeX
```
The keyword \defineinkeyword{ch:input_keywords}{IN\_KEYWORD} can be used.
The keyword \defineoutkeyword{ch:output_keywords}{OUT\_KEYWORD} can be used.
```
The keyword IN_KEYWORD can be used. The keyword OUT_KEYWORD can be used.

- \Program - used to highligh a program (writen in small caps). i.e.:

LaTeX
```
The program \Program{AstroPy} can be used.
```
The program AstroPy can be used.

- \ParameterEntry - used to define a parameter entry. It requires 8 arguments (Variable title, Description, variable name, default value, which recipe it is used in, the place the variable is defined, the code/module/function it is used in – dev only, and the visibility level – dev only). i.e.:

LaTeX

```
\namedlabel{text:variablename1}
\ParameterEntry{Variable title}
{Description of the variable}
{VARIABLE\_NAME}
{Default Value}{The recipe used the variable is used in.}
{The place where the variable is defined.}
{The code (module + function) where variable is used.}
{
Who should be able to change this variable, levels are as follows:
\begin{itemize}
        \item Public: Everyone (including the user)
        \item Private: Only the developer
\end{itemize}
}
```

Variable title (VARIABLE_NAME)

Description of the variable

VARIABLE_NAME = Default Value

Used in:	The recipe used the variable is used in.
Defined in:	The place where the variable is defined.
Called in:	The code (module + function) where variable is used.
Level:	Who should be able to change this variable, levels are as follows:

- Public: Everyone (including the user)
- Private: Only the developer

Note: the \label here is used to link variables with this name (i.e. via definevariable)

- \PseudoParamEntry - used to define a pseudo parameter entry. It requires 6 arguments (Variable title, Description, variable name, which recipe it is used in, the place the variable is defined, the code/module/function it is used in). It is only available for the developer guide. i.e.:

LaTeX

```
\namedlabel{text:variablename2}
\PseudoParamEntry{Variable title}
{Description of the variable}
{VARIABLE\_NAME}
{The recipe used the variable is used in.}
{The place where the variable is defined.}
{The code (module + function) where variable is used.}
```

Variable title (VARIABLE_NAME)

Description of the variable

VARIABLE_NAME

Used in:	The recipe used the variable is used in.
Defined in:	The place where the variable is defined.
Called in:	The code (module + function) where variable is used.

Note: \PseudoParamEntry is identical to \ParameterEntry other than not requiring a default value and not requiring a visibility level (as it is generally used for code thus a simple value can not be given cleanly and will always be a private variable).

Note: the \label here is used to link variables with this name (i.e. via definevariable)

- \KeywordEntry - used to define a keyword entry. It requires 9 arguments (Keyword title, Description, keyword name, HEADER key name, default HEADER value, HEADER comment, which recipe it is used in, the place the variable is defined, the code/module/function it is used in). It is only available for the developer guide.

LaTeX

```
\namedlabel{text:keywordname}
\KeywordEntry{Keyword title}
{Description of the keyword}
{kw\_variable}{HEADER key}
{Default HEADER value}{HEADER comment}
{The recipe the keyword is used in}
{The place where the keyword is defined}
{The code where the keyword is used.}
```

Keyword title (kw_variable)

Description of the keyword

kw_variable = ["HEADER key", "Default HEADER value", "HEADER comment"]

HEADER file entry:

HEADER key = Default HEADER value \ HEADER comment

Used in: The recipe the keyword is used in
Defined in: The place where the keyword is defined
Called in: The code where the keyword is used.

Note: the \label here is used to link variables with this name (i.e. via definekeyword)

- \customdirtree - used to create a directory tree, so add background use with the tcustomdir environment (see 8.6). The format of each line is

```text
.{level} {directory}.
```

each line must start with a period and end with a period, comments can be added using the \DTcomment command.
an example is shown below:

LaTeX

```
        The file structure should look as follows:
\begin{tcustomdir}
\customdirtree{%
.1 home.
.2 user1.
.3 Downloads\DTcomment{User 1 downloads}.
.3 Documents.
.4 \DTcomment{Many documents in here}.
.2 user2.
.3 Downloads.
.3 Documents\DTcomment{User 2 documents}.
}
\end{tcustomdir}
```

The file structure should look as follows:

```
home
├── user1
│   ├── Downloads.....................................User 1 downloads
│   └── Documents
│       └── .....................................Many documents in here
└── user2
    ├── Downloads
    └── Documents.....................................User 2 documents
```

8.5 Constants

Many constants are setup to ease the writing of this documentation. These can be found in the ./documentation/Config/constants.tex file. These are defined and use in the form:

LaTeX
```
% define constant
\newcommand{\ConstantName}{ConstantName}
% user constant
The constant is called \ConstantName
```

The constant is called ConstantName

Please check out the constants.tex file for the list of which constants are currently defined.

8.6 Code formatting

This section deals with the textbox, cmdbox, pythonbox and LaTeXboxes seen throughout the documentation. These are defined in ./documentation/Config/code_format.tex along with many style definitions (that only need to be changed to change colours/box styles) - this is left out of this guide for berivity.

- A line/lines of text (that are to be edited in a text editor):

LaTeX
```
\begin{textbox}
<# A variable name that can be changes to a specific value>
@VARIABLE_NAME@ = "Variable Value"
\end{textbox}
```

text
```
# A variable name that can be changes to a specific value
VARIABLE_NAME = "Variable Value"
```

Note: A custom title can be added to any code box and any other tcolorbox parameters can be overwritten as follows:

LaTeX
```
\begin{textbox}[title={Custom title}, colback=blue!30!white]
<# A variable name that can be changes to a specific value>
@VARIABLE_NAME@ = "Variable Value"
\end{textbox}
```

Custom title
```
# A variable name that can be changes to a specific value
VARIABLE_NAME = "Variable Value"
```

For keywords and options see the TCOLORBOX documentation here: http://texdoc.net/texmf-dist/doc/latex/tcolorbox/tcolorbox.pdf.

- A line/lines of text (that are to be edited in bash):

LaTeX

```
\begin{bashbox}
#!/usr/bin/bash
# Find out which console you are using
echo $0
# Set environment Hello
export Hello="Hello"
\end{bashbox}
```

bash

```
#!/usr/bin/bash
# Find out which console you are using
echo $0
# Set environment Hello
export Hello="Hello"
```

- A line/lines of text (that are to be edited in bash):

LaTeX

```
\begin{cshbox}
#!/usr/bin/tcsh
# Find out which console you are using
echo $0
# Set environment Hello
setenv Hello "Hello"
\end{cshbox}
```

tcsh/csh

```
#!/usr/bin/tcsh
# Find out which console you are using
echo $0
# Set environment Hello
setenv Hello "Hello"
```

- A line/lines of text to be run in the command shell:

LaTeX

```
\begin{cmdbox}
cd (*$\sim$*)/Downloads
\end{cmdbox}
```

```
>> cd ~/Downloads
```

- A line/lines of text that is print out to the screen (standard output):

LaTeX
```
\begin{cmdboxprint}
 This is a print out in the command line
 produced by using the echo command
\end{cmdboxprint}
```

```
This is a print out in the command line
produced by using the echo command
```

- A line/lines of text in the python terminal or python script

LaTeX
```
\begin{pythonbox}
import numpy as np
print("Hello world")
print("{0} seconds".format(np.sqrt(25)))
\end{pythonbox}
```

Python/Ipython
```
import numpy as np
print("Hello world")
print("{0} seconds".format(np.sqrt(25)))
```

- A line/lines of text in LaTeXcode (in raw form and then compiled form).

LaTeX
```
\begin{latexbox}[colframe=blue!75!black,]
This is my \LaTeX code.
\end{latexbox}
```

LaTeX
```
This is my \LaTeX code.
```

This is my LaTeXcode.

- highlighted textbox:

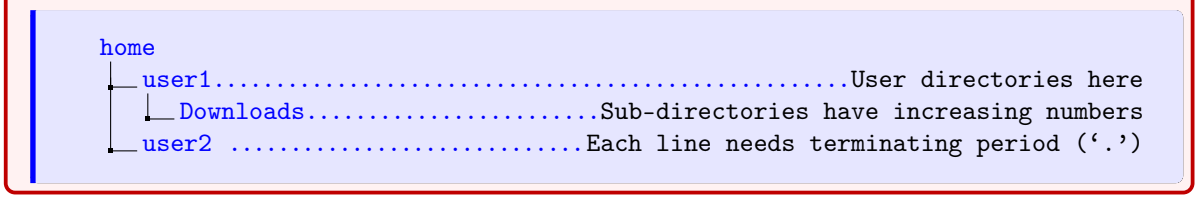
LaTeX
```
\begin{thighlight}
Highlighted section
\end{thighlight}
```

Highlighted section

- Custom directory tree (highlighted section):

LaTeX

```
\begin{tcustomdir}
\customdirtree{%
.1 home.
.2 user1\DTcomment{User directories here}.
.3 Downloads\DTcomment{Sub-directories have increasing numbers}.
.2 user2 \DTcomment{Each line needs terminating period (‘.’)}.
}
\end{tcustomdir}
```



- note box:

LaTeX

```
\begin{note}
This is a Note
\end{note}
```

Note: This is a Note

- todo box:

LaTeX

```
\begin{todo}
This is a to do statement (temporary)
\end{todo}
```

TODO: This is a to do statement (temporary)

Chapter 9

Required input header keywords

9.1 Required keywords

The following keywords are required by the current recipes to run.

- **Acquisition time (human readable) (kw_ACQTIME_KEY)**

  The acquisition time in format YYYY-mm-dd-HH-MM-SS.ss

  kw_ACQTIME_KEY = ["ACQTIME1", "YYYY-mm-dd-HH-MM-SS.ss", "Date at start of observation"]

  HEADER file entry:

  ```
  ACQTIME1  =  YYYY-mm-dd-HH-MM-SS.ss  \  Date at start of observation
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Acquisition time (unix time format) (kw_ACQTIME_KEY_UNIX)**

  The acquisition time in in unix time format (time since 1970-01-01-00-00-00)

  kw_ACQTIME_KEY_UNIX = ["ACQTIME", "000000000.00", "Date in unix time at start of observation"]

  HEADER file entry:

  ```
  ACQTIME  =  000000000.00  \  Date in unix time at start of observation
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Observation date (kw_DATE_OBS)**

  The observation date in format YYYY-mm-DD

  kw_DATE_OBS = ["DATE-OBS", "YYYY-mm-DD", "Date at start of observation (UTC)"]

  HEADER file entry:

  ```
  DATE-OBS  =  YYYY-mm-DD  \  Date at start of observation (UTC)
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Observation time (kw_UTC_OBS)**

  The observation time in format HH:MM:SS.SS

  kw_UTC_OBS = ["UTC-OBS", "HH:MM:SS.SS", "Time at start of observation (UTC)"]

  HEADER file entry:

  ```
  UTC-OBS  =  HH:MM:SS.SS  \  Time at start of observation (UTC)
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Object Right Ascension (kw_OBJRA)**

  The object Right Ascension in HH:MM:SS.SS

  kw_OBJRA = ["OBJRA", "HH:MM:SS.SS", "Target right ascension"]

  HEADER file entry:

  ```
  OBJRA  =  HH:MM:SS.SS  \  Target right ascension
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Object Declination (kw_OBJDEC)**

  The object Declination in DD:MM:SS.SS

  kw_OBJDEC = ["OBJDEC", "DD:MM:SS.SS", "Target declination"]

  HEADER file entry:

  ```
  OBJDEC  =  DD:MM:SS.SS  \  Target declination
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Object Name (kw_OBJNAME)**

The object name

kw_OBJNAME = ["OBJNAME", "Name", "Target name"]

HEADER file entry:

```
OBJNAME  =  Name  \  Target name
```

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Object Equinox (kw_OBJEQUIN)**

The object equinox

kw_OBJEQUIN = ["OBJEQUIN", "DDDD.D", "Target equinox"]

HEADER file entry:

```
OBJEQUIN  =  DDDD.D  \  Target equinox
```

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Object Right Ascension in proper motion (kw_OBJRAPM)**

The Object Right Ascension in proper motion [as/yr]

kw_OBJRAPM = ["OBJRAPM", "D.DD", "Target right ascension proper motion in as/yr"]

HEADER file entry:

```
OBJRAPM  =  D.DD  \  Target right ascension proper motion in as/yr
```

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Object Declination in proper motion (kw_OBJDECPM)**

The Object Declination in proper motion [as/yr]

kw_OBJDECPM = ["OBJDECPM", "D.DD", "Target declination proper motion in as/yr"]

HEADER file entry:

```
OBJDECPM  =  D.DD  \  Target declination proper motion in as/yr
```

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Read noise (kw_RDNOISE)**

The read noise (used for sigdet) [e-]

kw_RDNOISE = ["RDNOISE", "0.0", "read noise (electrons)"]

HEADER file entry:

```
RDNOISE  =  0.0  \  read noise (electrons)
```

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Gain (kw_GAIN)**

The gain [e-/ADU]

kw_GAIN = ["GAIN", "0.0", "gain (electrons/ADU)"]

HEADER file entry:

```
GAIN  =  0.0  \  gain (electrons/ADU)
```

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Exposure time (kw_EXPTIME)**

  The integration time in seconds

  kw_EXPTIME = ["EXPTIME", "0.0", "Integration time (seconds)"]

  HEADER file entry:

  ```
  EXPTIME  =  0.0  \  Integration time (seconds)
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Observation Type (kw_OBSTYPE)**

  The observation type

  kw_OBSTYPE = ["OBSTYPE", "Object", "Observation / Exposure type"]

  HEADER file entry:

  ```
  OBSTYPE  =  Object  \  Observation / Exposure type
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Cassegrain Fiber Position (kw_CCAS)**

  The cassegrain fiber position

  kw_CCAS = ["SBCCAS_P", "pos_pk", "SPIRou Cassegrain Fiber Position (predefined)"]

  HEADER file entry:

  ```
  SBCCAS_P  =  pos_pk  \  SPIRou Cassegrain Fiber Position (predefined)
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Reference Fiber Position (kw_CREF)**

  The reference fiber position

  kw_CREF = ["SBCREF_P", "pos_pk", "SPIRou Reference Fiber Position (predefined)"]

  HEADER file entry:

  ```
  SBCREF_P  =  pos_pk  \  SPIRou Reference Fiber Position (predefined)
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Calibration reference density (kw_CDEN)**

  The calibration reference density

  kw_CDEN = ["SBCDEN_P", "1.2", "SPIRou Calib-Reference density (0 to 3.3)"]

  HEADER file entry:

  ```
  SBCDEN_P  =  1.2  \  SPIRou Calib-Reference density (0 to 3.3)
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Exposure Sequence (kw_CMMTSEQ)**

  The exposure sequence description

  kw_CMMTSEQ = ["CMMTSEQ", "{X} exposure A of B", ""]

  HEADER file entry:

  ```
  CMMTSEQ  =  {X} exposure A of B  \
  ```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

  Note: where X = a Stoke parameter and A and B are integers.
  e.g. "V exposure 2, sequence 1 of 1"

9.2 Descriptions

The following FITS descriptors of the 2D raw frames are required for the DRS. Last updated version 21 Nov 2014.

9.2.1 Standard FITS Keywords

```
BITPIX   =                   16 / 16bit
NAXIS    =                    2 / Number of axes
NAXIS1   =                 4096 / Number of pixel columns
NAXIS2   =                 4096 / Number of pixel rows
BZERO    =              32768.0 / Zero factor
BSCALE   =                  1.0 / Scale factor
DATE     = '2013-11-26T09:06:14' / UTC Date of file creation
INSTRUME =              'SPIROU' / Instrument Name
```

9.2.2 FITS keywords related to the detector

```
EXPTIME  =                800.0 / Integration time (seconds)
DARKTIME =                800.0 / Dark current time (seconds)
GAIN     =                 1.30 / gain (electrons/ADU)
RDNOISE  =                 4.20 / read noise (electrons)
NSUBEXP  =                    4 / Total number of sub-exposures of 5.2s
OBSTYPE  =             'NORMAL' / Exposure type (DARK/NORMAL)
MIDEXPTM =                  400 / mid-exposure time (seconds)
EMCNTS   =               444578 / exposure meter counts at end
```

9.2.3 FITS keywords related to the target

```
OBJNAME  =               Gl9999 / Target name
OBJRA    =        '5:35:09.87' / Target right ascension
OBJDEC   =        '-5:27:53.3' / Target declination
OBJRAPM  =              0.560 / Target right ascension proper motion in as/yr
OBJDECPM =              -0.33 / Target declination proper motion in as/yr
OBJEQUIN =             2000.0 / Target equinox
OBJRV    =              -30.0 / Target Radial velocity (km/s) (999 if unknown)
OBJTYPE  =               'M5' / Target spectral type
OBJJMAG  =                8.2 / Target J magnitude
OBJHMAG  =                9.2 / Target H magnitude
OBJKMAG  =               10.0 / Target K magnitude
```

9.2.4 FITS keywords related to the telescope

ACQTIME	=	2013-11-26T09 :06 :14.858	/	Date at start of observation
ACQTIME1	=	1385456774	/	Date in unix time at start of observation
DATE_OBS	=	'2013-11-26T09 :06 :14.858'	/	Date at start of observation (UTC)
EQUINOX	=	2000.0	/	Equinox of coordinates
EPOCH	=	2000.0	/	Epoch of coordinates
MJDATE	=	56622.3700212	/	Modified Julian Date at start of observation
MJEND	=	56622.3797593	/	Modified Julian Date at end of observation
AIRMASS	=	1.4	/	Airmass at start of observation
RA	=	'5:35:09.87'	/	Telescope right ascension
DEC	=	'-5:27:53.3'	/	Telescope declination
SEEING	=	1.0	/	Seeing at start of observation

9.2.5 FITS keywords related to the instrument

TPL_NAME	=	'SPIROU_POL_WAVE'	/	template Name
TPL_NEXP	=	1	/	# of exposure within template
TPL_EXPN	=	1	/	exposure # within template
INS_CAL	=	'WAVE'	/	Simultaneous calibration (WAVE/FP/NONE)
INS_LAMP	=	'UrAr'	/	Calibration lamp
INS_RHB1	=	90	/	SPIROU rhomb 1 position (deg)
INS_RHB2	=	180	/	SPIROU rhomb 2 position deg)

Chapter 10

Variables

To better understand the variables in the DRS we have laid out each variable in the following way:

- **Variable title (VARIABLE_NAME)**

  Description of the variable

  VARIABLE_NAME = Default Value

  Used in: The recipe used the variable is used in.
  Defined in: The place where the variable is defined.
  Called in: The code (module + function) where variable is used.
  Level: Who should be able to change this variable, levels are as follows:

  – Public: Everyone (including the user)

  – Private: Only the developer

Note: All variable from all configuration files are (and should be) loaded into the main parameter dictionary 'p' in all recipes and thus are accessed via:

Python/Ipython

```
variable = p["VARIABLE_NAME"]
```

10.1 Variable file locations

10.1.1 User modifiable variables

The variables are currently stored in two places. The first (USER_DIR/config/config.py) contains constants that deal with initial set up. These were mentioned in Section 3.5 and are located in DRS_DIR/config/USER_DIR/config/config.py.

The other variables modify how the DRS runs. These are located in constants_SPIROU_H4RG.py (located at DRS_DIR/config/constants_SPIROU_H4RG.py).

10.1.2 Private variables

In addition to the above (user modifiable public variable files) there are several files that will contain all constants that should not be changed by a user (i.e. static variables that are set and changed only in development). These are described below:

- **Keywords:** The keywords for header input and output are stored in SpirouDRS.spirouConfig.spirouKeywords. This contains keyword definitions in the form of a python list:

Python/Ipython

```
kw_VARIABLE = ['KEYWORD', 'Default value', 'Comment']
```

where the 'KEYWORD' is the key in the FITs REC header file, with the value and comment defined in the next positions. i.e. in a FITs REC header reader one would expect

```
KEYWORD = Default value / Comment
```

- **Constants and Pseudo-constants:** These are stored in SpirouDRS.spirouConfig.spirouConst , they range from simple objects (strings, integers, float, lists, python dictionaries etc.) to more complicated 'pseudo-constants' that are constructed themselves from other constants. These are kept private (i.e. no mentioned in the user manual) as they should not need be changed by the average user.

10.2 Global variables

- **DRS Name (DRS_NAME)**

Defines the data reduction software name. Value must be a valid string.

DRS_NAME = SPIROU

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouConst .NAME()
Called in: All Recipes
Level: Private

- **DRS Version (DRS_VERSION)**

Defines the data reduction software version. Value must be a valid string.

DRS_VERSION = 0.0.1

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouConst .VERSION()
Called in: All Recipes
Level: Private

- **DRS Detector Type (IC_IMAGE_TYPE)**

Defines the detector. Value must be a valid string. Currently supported values are: "H2RG" or "H4RG".

IC_IMAGE_TYPE = H4RG

Used in: All Recipes
Defined in: constants_SPIROU_H4RG.py
Called in: All Recipes
Level: Public

- **Release type (release)**

  Defines the current release type or state of the DRS. Value must be a valid string. This could explain the current state or just distinguish between alpha, beta and full releases.

  release = 'alpha'

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .RELEASE()
  Called in: All Recipes
  Level: Private

- **Package name (package)**

  Defines the name of the python package that all sub-modules are located in. Value must be a string and be the name of a valid python package.

  package = SpirouDRS

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .PACKAGE()
  Called in: All Recipes
  Level: Private

- **authors (authors)**

  Defines the authors of the DRS. Value must be a string, author names separated by a comma.

  authors = N. Cook, F. Bouchy, E. Artigau, I. Boisse, M. Hobson, C. Moutou

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .AUTHORS()
  Called in: All Recipes
  Level: Private

- **date (date)**

  Defines the last edited date for the DRS. Value must be a string in format YYYY-MM-DD format.

  date = 2017-11-17

  Used in: None
  Defined in: SpirouDRS.spirouConfig.spirouConst .LATEST_EDIT()
  Called in: None
  Level: Private

- **Plotting switch (DRS_PLOT)**

Defines whether to show plots (A value of 1 to show plots, a value of 0 to not show plots). Value must be an integer (0 or 1) or boolean (True or False)

DRS_PLOT = 1

Used in: All Recipes
Defined in: USER_DIR /config /config.py
Called in: All Recipes
Level: Public

- **Interactive switch (DRS_INTERACTIVE)**

Defines whether to run in interactive mode. If False or 0 will be set to non-interactive mode (i.e. DRS_PLOT will be set to 0). Will stop any user input at the end of recipes if set to 0.

DRS_INTERACTIVE = 1

Used in: All Recipes
Defined in: USER_DIR /config /config.py
Called in: All Recipes
Level: Public

- **Use matplotlib interactive plot environment (interactive_plots)**

Defines whether to use the matplotlib interactive plot environment. If True or 1 uses 'plot.ion()' and plots do not interrupt the running of code. If False or 0 all plots are run and 'plt.show(), plt.close()' is used after each plot (pausing the code and destroying the plots after they are manually closed). This is mostly useful for debugging.

interactive_plots = True

Used in: SpirouDRS.spirouCore.spirouPlot
Defined in: SpirouDRS.spirouConfig.spirouConst .INTERACITVE_PLOTS_ENABLED()
Called in: SpirouDRS.spirouCore.spirouPlot variable definition
Level: Private

- **Debug mode (DRS_DEBUG)**

  Defines whether we should run the DRS in debug mode. Certain print/log statements and certain graphs only plot in debug mode. On an error the option to enter the python debugger is asked (allows user to look into functions/current memory and see what variables are currently defined. Value must be an integer. Value must be an integer where:

  - 0 = No debug
  - 1 = basic debugging on errors (prompted to enter python debugger)
  - 2 = Same as 1 and recipes specific (plots and some code runs)

  DRS_DEBUG = 0

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: All Recipes
  Level: Public

- **Debugging mode controller (debug)**

  Controls the debug level (from DRS_DEBUG)

  debug

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .DEBUG()
  Called in: All Recipes

- **Enable use of custom configuration files (DRS_DEBUG)**

  Controls whether the DRS searches for custom configuration files (all configuration files). Value can be 1 (True) or 0 (False), True turns on the use of custom configuration files. Constants/variables set in the custom constant files **always** take precedence over the master files. Any constant/variable not set in a custom constant file is taken from the master files. Value must be 1 or True to switch on or 0 or False to switch off.

  DRS_DEBUG = 0

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: All Recipes
  Level: Public

- **Plot interval (constants_SPIROU.py)**

  Defines the main constants file for the DRS (not the directory just the file name). The directory is controlled internally or if USER_CONFIG is set by DRS_UCONFIG

  constants_SPIROU.py = All Recipes

  Used in: USER_DIR/config/config.py
  Defined in: All Recipes
  Called in: Public
  Level:

- **Plot interval (special_config_SPIROU.py)**

  Defines the a special constants file for the DRS (not the directory just the file name). The directory is controlled internally or if USER_CONFIG is set by DRS_UCONFIG

  special_config_SPIROU.py = All Recipes

  Used in: USER_DIR/config/config.py
  Defined in: All Recipes
  Called in: Public
  Level:

  Note:

  This file does not currently exist.

- **Plot interval (ic_display_timeout)**

  Set the interval between plots in seconds (for certain interactive graphs). Value must be a valid float larger than zero.

  ic_display_timeout = 0.5

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in:
  Level: Public

Note: Should this be public?

10.3 Directory variables

- **The data directory (TDATA)**

  Defines the path to the data directory. Value must be a string containing a valid file location.

  TDATA = /drs/data/

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: SpirouDRS.spirouConfig.spirouConst
  Level: Public

- **The installation directory (DRS_ROOT)**

  Defines the installation directory (DRS_DIR). Value must be a string containing a valid file location.

  DRS_ROOT = /drs/INTROOT/

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: SpirouDRS.spirouConfig.spirouConst
  Level: Public

- **The raw data directory (DRS_DATA_RAW)**

  Defines the directory that the reduced data will be saved to/read from. Value must be a string containing a valid file location.

  DRS_DATA_RAW = /drs/data/raw

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: SpirouDRS.spirouConfig.spirouConst
  Level: Public

- **The reduced data directory (DRS_DATA_REDUC)**

  Defines the directory that the reduced data will be saved to/read from. Value must be a string containing a valid file location.

  DRS_DATA_REDUC = /drs/data/reduced

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: SpirouDRS.spirouConfig.spirouConst
  Level: Public

- **The calibration database and calibration file directory (DRS_CALIB_DB)**

  Defines the directory that the calibration files and database will be saved to/read from. Value must be a string containing a valid file location.

  DRS_CALIB_DB = /drs/data/calibDB

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: SpirouDRS.spirouConfig.spirouConst
  Level: Public

- **The log directory (DRS_DATA_MSG)**

  Defines the directory that the log messages are stored in. Value must be a string containing a valid file location.

  DRS_DATA_MSG = /drs/data/msg

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: SpirouDRS.spirouConfig.spirouConst
  Level: Public

- **The working directory (DRS_DATA_WORKING)**

  Defines the working directory. Value must be a string containing a valid file location.

  DRS_DATA_WORKING = /drs/data/tmp/

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: SpirouDRS.spirouConfig.spirouConst
  Level: Public

- **The custom configuration directory (DRS_UCONFIG)**

  Defines the custom configuration directory. Value must be a string containing a valid file location.

  DRS_UCONFIG = ~/spirou_config/

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: SpirouDRS.spirouConfig.spirouConst
  Level: Public

10.4 Observatory variables

- **CFHT Longitude (IC_LONGIT_OBS)**

  Defines the CFHT longitude West (in degrees)

  IC_LONGIT_OBS = 155.468876

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV.earth_velocity_correction
  Level: Public

- **CFHT Longitude (IC_LATIT_OBS)**

  Defines the CFHT latitude North (in degrees)

  IC_LATIT_OBS = 19.825252

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV.earth_velocity_correction
  Level: Public

- **CFHT Longitude (IC_ALTIT_OBS)**

  Defines the CFHT altitude (in km)

  IC_ALTIT_OBS = 4.204

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV.earth_velocity_correction
  Level: Public

10.5 Image variables

- **Resizing blue window (ic_ccd{x/y}_blue_{low/high})**

  The blue window used in cal_DARK_spirou. Each value must be a integer between 0 and the maximum array size in each dimension.

  ic_ccdx_blue_low = 2048-200
  ic_ccdx_blue_high = 2048-1500
  ic_ccdy_blue_low = 2048-20
  ic_ccdy_blue_high = 2048-350

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DARK_spirou.main()
  Level: Public

- **Resizing red window (ic_ccd{x/y}_red_{low/high})**

  The blue window used in cal_DARK_spirou. Each value must be a integer between 0 and the maximum array size in each dimension.

  ic_ccdx_red_low = 2048-20
  ic_ccdx_red_high = 2048-1750
  ic_ccdy_red_low = 2048-1570
  ic_ccdy_red_high = 2048-1910

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DARK_spirou.main()
  Level: Public

- **Resizing red window (ic_ccd{x/y}_{low/high})**

  The blue window used in cal_DARK_spirou. Each value must be a integer between 0 and the maximum array size in each dimension.

  ic_ccdx_low = 5
  ic_ccdx_high = 2040
  ic_ccdy_low = 5
  ic_ccdy_high = 1935

  Used in: cal_loc_RAW_spirou, cal_SLIT_spirou, cal_FF_RAW_spirou, cal_extract_RAW_spirou, cal_DRIFT_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou main(), cal_SLIT_spirou main(), cal_FF_RAW_spirou main(), cal_extract_RAW_spirou main(), cal_DRIFT_RAW_spirou.main()
  Level: Public

- **Available fiber types (fiber_types)**

  Defines the type of fiber we have (used in various codes). Theses are define in a python list of string, where the earlier a fiber is in the list the more it takes priority in searches (i.e. AB over A or B if AB is first)

  fiber_types = ['AB', 'A', 'B', 'C']

  Used in: cal_extract_RAW_spirou, cal_DRIFT_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_extract_RAW_spirou.main(), SpirouDRS.spirouStartup.spirouStartup.get_fiber_type()
  Level: Public

10.6 Fiber variables

These variables are defined for each type of fiber and thus are defined as a python dictionary of values (read using the python 'eval' function) . As such they all must contain the same dictionary keys (currently 'AB', 'A', 'B' and 'C').

> **Note:** For python to combine these at run time the suffix '_fpall' must be used (thus once a fiber is defined the code will know to extract the key before the suffix). i.e. for variable 'nbfib_fpall' and a fiber 'AB' the extracted parameter will be 'nbfib' with the value in the dictionary corresponding to the 'AB' key.

- **Number of fibers (nbfib_fpall)**

  This describes the number of fibers of a given type. Must be a python dictionary with identical keys to all other fiber parameters (each value must be an integer).

  nbfib_fpall = `{'AB':2, 'A':1, 'B':1, 'C':1}`

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Order skip number (ic_first_order_jump_fpall)**

  Describes the number of orders to skip at the start of an image. Must be a python dictionary with identical keys to all other fiber parameters (each value must be an integer).

  ic_first_order_jump_fpall = `{'AB':2, 'A':0, 'B':0, 'C':0}`

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Maximum order numbers (ic_locnbmaxo_fpall)**

  Describes the maximum allowed number of orders. Must be a python dictionary with identical keys to all other fiber parameters (each value must be an integer).

  ic_locnbmaxo_fpall = `{'AB':72, 'A':36, 'B':36, 'C':36}`

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Number of orders to fit (QC) (qc_loc_nbo_fpall)**

  Quality control parameter for the number of orders on fiber to fit. Must be a python dictionary with identical keys to all other fiber parameters (each value must be an integer).

  qc_loc_nbo_fpall = `{'AB':72, 'A':36, 'B':36, 'C':36}`

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

  Note: Should this be merged with 'ic_locnbmaxo_fpall'?

- **Fiber types for this fiber (fib_type_fpall)**

  The fiber type(s) – as a list – for this fiber. Must be a python dictionary with identical keys to all other fiber parameters (each value must be a list of strings).

  fib_type_fpall = `{'AB':["AB"], 'A':["A"], 'B':["B"], 'C':["C"]}`

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_FF_RAW_spirou.main()
  Level: Public

  Note: This is not be needed but is in here due to a loop in cal_FF_RAW_spirou

- **Half-zone extraction width (left/top) (ic_ext_range1_fpall)**

  The pixels are extracted from the center of the order out to the edges in the row direction (y-axis), i.e. defines the illuminated part of the order - this number defines the **top** side (if one requires a symmetric extraction around the order fit both range 1 and range 2 – below – should be the same). This can also be used to extract A and B separately (where the fit order is defined at the centre of the AB pair). Must be a python dictionary with identical keys to all other fiber parameters.

  ic_ext_range1_fpall = `{'AB':14.5, 'A':0.0, 'B':14.5, 'C':7.5}`

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_extract_RAW_spirou.main(), SpirouDRS.spirouEXTOR.spirouEXTOR.extract_tilt_weight_order2(), SpirouDRS.spirouCore.spirouPlot.ff_sorder_fit_edges()
  Level: Public

  Note: Formally this was called 'plage1' in cal_FF_RAW_spirou

- **Half-zone extraction width (right/bottom) (ic_ext_range2_fpall)**

The pixels are extracted from the center of the order out to the edges in the row direction (y-axis), i.e. defines the illuminated part of the order - this number defines the **bottom** side (if one requires a symmetric extraction around the order fit both range 1 and range 2 – below – should be the same). This can also be used to extract A and B separately (where the fit order is defined at the centre of the AB pair). Must be a python dictionary with identical keys to all other fiber parameters.

ic_ext_range2_fpall = `{'AB':14.5, 'A':14.5, 'B':0.0, 'C':7.5}`

Used in:	cal_FF_RAW_spirou, cal_extract_RAW_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_FF_RAW_spirou.main(), cal_extract_RAW_spirou.main(), SpirouDRS.spirouEXTOR.spirouEXTOR.extract_tilt_weight_order2(), SpirouDRS.spirouCore.spirouPlot.ff_sorder_fit_edges()
Level:	Public

Note: Formally this was called 'plage2' in cal_FF_RAW_spirou

- **Half-zone extraction width for full extraction (ic_ext_range_fpall)**

The pixels are extracted from the centre of the order out to the edges in the row direction (y-axis), i.e. defines the illuminated part of the order. In cal_extract_RAW_spirou both sides of the fit order are extracted at with the same width (symmetric). Must be a python dictionary with identical keys to all other fiber parameters.

ic_ext_range_fpall = `{'AB':14.5, 'A':14.5, 'B':14.5, 'C':7.5}`

Used in:	cal_extract_RAW_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouEXTOR.spirouEXTOR.extract_order(), SpirouDRS.spirouEXTOR.spirouEXTOR.extract_tilt_order(), SpirouDRS.spirouEXTOR.spirouEXTOR.extract_tilt_weight_order(), SpirouDRS.spirouEXTOR.spirouEXTOR.extract_weight_order()
Level:	Public

Note: Formally this was called 'plage' in cal_extract_RAW_spirou

- **Localization fiber for extraction (loc_file_fpall)**

  Defines the localization fiber to use for each fiber type. This is the file in calibDB that is used i.e. the keyword ic_calibDB_filename used will be ÌLOC_{loc_file_fpall}' (e.g. for fiber='AB' use 'LOC_AB'). Must be a python dictionary with identical keys to all other fiber parameters.

  loc_file_fpall = `{'AB':'AB', 'A':'AB', 'B':'AB', 'C':'C'}`

  Used in: cal_extract_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.get_loc_coefficients()
  Level: Public

- **Order profile fiber for extraction (orderp_file_fpall)**

  Defines the order profile fiber to use for each fiber type. This is the file in calibDB that is used i.e. the keyword ic_calibDB_filename used will be ÒRDER_PROFILE_{orderp_file_fpall}' (e.g. for fiber='AB' use 'ORDER_PROFILE_AB'). Must be a python dictionary with identical keys to all other fiber parameters.

  orderp_file_fpall = `{'AB':'AB', 'A':'AB', 'B':'AB', 'C':'C'}`

  Used in: cal_extract_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouFITS.read_order_profile_superposition()
  Level: Public

- **Half-zone extract width cal_DRIFT_RAW_spirou (ic_ext_d_range_fpall)**

  The size in pixels of the extraction away from the order localization fit (to the top and bottom) - defines the illuminated area of the order for extraction. Must be a python dictionary with identical keys to all other fiber parameters.

  ic_ext_d_range_fpall = `{'AB':14.0, 'A':14.0, 'B':14.0, 'C':7.0}`

  Used in: cal_DRIFT_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DRIFT_RAW_spirou.main()
  Level: Public

  Note: Formally this was called 'ic_extnbsig' in cal_DRIFT_RAW_spirou

10.7 Pre-processing variables

- **Preprocessing Mode (PP_MODE)**

  Defines the preprocessing output type. Currently expected values are:

  - 0: Adds only the PROCESSED_SUFFIX
  - 1: Adds the PROCESSED_SUFFIX and an identifying suffix (i.e. `flat_dark`, `dark_dark` etc.)

  PP_MODE = 0

  Used in: cal_preprocess_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouFile.`id_mode`
  Level: Public

- **Force Pre-processing (IC_FORCE_PREPROCESS)**

  Defines whether the DRS should force pre-processed files only. If True (or 1) only files that are pre-processed will be allowed to be used in the DRS (an error will be generated on un-preprocessed files). If False (or 0) any file will be accepted (but un-preprocessed files will be assumed to be raw and hence rotated, as in pre-processing).

  IC_FORCE_PREPROCESS = 1

  Used in: All Recipes
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouFile.`check_file_id`, SpirouDRS.spirouImage.spirouFile.`check_preprocess`
  Level: Public

- **Pre-processing suffix (PROCESSED_SUFFIX)**

  Defines the suffix to apply to the pre-processed files. If "None" then no suffix is added (or checked for).

  PROCESSED_SUFFIX = "_pp.fits"

  Used in: All Recipes
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_preprocess_spirou.main, off_listing_RAW_spioru.main, SpirouDRS.spirouImage.spirouFile.`check_file_id`, SpirouDRS.spirouImage.spirouFile.`check_preprocess`, SpirouDRS.spirouImage.spirouImage.`fix_non_preprocessed`
  Level: Public

- **Number of dark amplifiers (NUMBER_DARK_AMP)**

  Defines the number of dark amplifiers on the detector

  NUMBER_DARK_AMP = 5

  Used in: cal_preprocess_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.median_filter_dark_amp
  Level: Public

- **Total number of amplifiers (TOTAL_AMP_NUM)**

  Defines the total number of amplifiers on the detector

  TOTAL_AMP_NUM = 32

  Used in: cal_preprocess_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.ref_top_bottom, SpirouDRS.spirouImage.spirouImage.median_filter_dark_amp
  Level: Public

- **Number of top reference pixels (NUMBER_REF_TOP)**

  Defines the number of un-illuminated reference pixels at the top of the image.

  NUMBER_REF_TOP = 4

  Used in: cal_preprocess_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.ref_top_bottom, SpirouDRS.spirouImage.spirouImage.median_one_over_f_noise
  Level: Public

- **Number of bottom reference pixels (NUMBER_REF_BOTTOM)**

  Defines the number of un-illuminated reference pixels at the bottom of the image.

  NUMBER_REF_BOTTOM = 4

  Used in: cal_preprocess_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.ref_top_bottom, SpirouDRS.spirouImage.spirouImage.median_one_over_f_noise
  Level: Public

- **Number of bins in dark median process (DARK_MED_BINNUM)**

  Define the number of bins used in the dark media process.

  DARK_MED_BINNUM = 32

  Used in: cal_preprocess_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.median_filter_dark_amp, SpirouDRS.spirouImage.spirouImage.median_one_over_f_noise
  Level: Public

- **Raw to Pre-processed rotation (RAW_TO_PP_ROTATION)**

  Defines the rotation angle of the raw images compared to that expected by the DRS. Value must be multiple of 90 degrees (in degrees counter-clockwise direction).

  RAW_TO_PP_ROTATION = -90

  Used in: All Recipes
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_preprocess_spirou.main, SpirouDRS.spirouImage.spirouImage.fix_non_preprocessed
  Level: Public

10.8 Dark calibration variables

- **Lower percentile for dead pixel stats (dark_qmin)**

  This defines the lower percentile to be logged for the fraction of dead pixels statistics. Value must be an integer between 0 and 100 (1 sigma below the mean is ~16).

  dark_qmin = 5

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.measure_dark()
  Level: Public

- **Upper percentile for dead pixel stats (dark_qmax)**

  This defines the upper percentile to be logged for the fraction of dead pixels statistics. Value must be an integer between 0 and 100 (1 sigma above the mean is ~84).

  dark_qmax = 95

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.measure_dark()
  Level: Public

- **Dark stat histogram bins (histo_bins)**

  Defines the number of bins to use in the dark histogram plot. Value must be a positive integer.

  histo_bins = 200

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.measure_dark()
  Level: Public

- **Lower bound for the Dark stat histogram (histo_range_low)**

  Defines the lower bound for the dark statistic histogram. Value must be a float less than (no equal to) the value of 'histo_range_high'

  histo_range_low = -0.5

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.measure_dark()
  Level: Public

- **Upper bound for the Dark stat histogram (histo_range_high)**

  Defines the upper bound for the dark statistic histogram. Value must be a float greater than (not equal to) the value of 'histo_range_low'

  histo_range_high = 5

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.measure_dark()
  Level: Public

- **Bad pixel cut limit (dark_cutlimit)**

  Defines the bad pixel cut limit in ADU/s.

  $$badpixels = (image > \text{dark\_cut\_limit}) \text{ OR (non-finite)} \quad (10.1)$$

  dark_cutlimit = 100.0

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DARK_spirou.main()
  Level: Public

10.9 Localization calibration variables

- **Order profile smoothed box size (loc_box_size)**

  Defines the size of the order profile smoothing box (from the central pixel minus size to the central pixel plus size). Value must be an integer larger than zero.

  loc_box_size = 10

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Image row offset (ic_offset)**

  The row number (y axis) of the image to start localization at (below this row orders will not be fit). Value must be an integer equal to or larger than zero.

  ic_offset = 40

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Central column of the image (ic_cent_col)**

  The column which is to be used as the central column (x-axis), this is the column that is initially used to find the order locations. Value must be an integer between 0 and the number of columns (x-axis dimension).

  ic_cent_col = 1000

  Used in: cal_loc_RAW_spirou, cal_FF_RAW_spirou, cal_extract_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main(), cal_FF_RAW_spirou.main(), cal_extract_RAW_spirou.main(), SpirouDRS.spirouBACK.spirouBACK.measure_background_and_get_central_pixels(), SpirouDRS.spirouCore.spirouPlot.slit_sorder_plot(), SpirouDRS.spirouEXTOR.spirouEXTOR.extract_AB_order(), SpirouDRS.spirouLOCOR.spirouLOCOR.find_order_centers(), SpirouDRS.spirouLOCOR.spirouLOCOR.initial_order_fit()
  Level: Public

- **Localization window row size (ic_ext_window)**

Defines the size of the localization window in rows (y-axis). Value must be an integer larger than zero and less than the number of rows (y-axis dimension).

ic_ext_window = 12

Used in: cal_loc_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.find_order_centers
Level: Public

Note: Formally this was called 'ic_ccdcolc' in cal_loc_RAW_spirou

- **Localization window column step (ic_locstepc)**

For the initial localization procedure interval points along the order (x-axis) are defined and the centers are found, this is used as the first estimate of the order shape. This parameter defines that interval step in columns (x-axis). Value must be an integer larger than zero and less than the number of columns (x-axis dimension).

ic_locstepc = 12

Used in: cal_loc_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.find_order_centers
Level: Public

- **Image gap index (ic_image_gap)**

Defines the image gap index. The order is skipped if the top of the row (row number - ic_ext_window) or bottom of the row (row number + ic_ext_window) is inside this image gap index. i.e. a order is skipped if:

$$\text{(top of the row} < \text{ic\_image\_gap) OR (bottom of the row} > \text{ic\_image\_gap)} \tag{10.2}$$

Value must be an integer between zero and the number of rows (y-axis dimension).

ic_image_gap = 0

Used in: cal_loc_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.find_order_centers
Level: Public

Note: This is set to zero and never used in a meaningful way, should it be removed?

- **Minimum order row size (ic_widthmin)**

  Defines the minimum row width (width in y-axis) to accept an order as valid. If below this threshold order is not recorded. Value must be an integer between zero and the number of rows (y-axis dimension).

  ic_widthmin = 5

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.find_order_centers
  Level: Public

- **Center noise multiplier threshold (ic_noise_mult_thres)**

  Defines the noise multiplier threshold in order to accept an order center as usable such that:

  $$max(value) - min(value) > \text{ic\_noise\_mult\_thres} * sigdet \quad (10.3)$$

  where the value is row center pixel value and sigdet is the image's read out noise. Values must be a float.

  ic_noise_mult_thres = 100

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.find_order_centers
  Level: Public

- **Polynomial fit coefficients for bad region interp. (bad_region_fit)**

  Defines the polynomial fit parameters for interpolating over the bad regions (holes) before the localization is done. Must be a python list of floats. The coefficient list should be in the following order:

  $$p[0] * x^{(N-1)} + p[1] * x^{(N-2)} + ... + p[N-2] * x + p[N-1] \quad (10.4)$$

  bad_region_fit = [3.19884964e-05, -1.08289228e-01, 2.16643659e+03]

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.interp_bad_regions
  Level: Public

- **Median filter box size 1 for bad region interp. (bad_region_med_size)**

  Defines the median filter box size used in interpolating over the bad regions (holes) before the localization is done. Value must be a positive integer.

  bad_region_med_size = 101

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.interp_bad_regions
  Level: Public

- **Threshold for bad pixels for bad region interp. (bad_region_threshold)**

  Defines the threshold below which the image (normalised between 0 and 1) should be regarded as bad. Used in interpolating over the bad regions (holeS) before localization is done. Value must be a float between 0 and 1.

  bad_region_threshold = 0.2

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.interp_bad_regions
  Level: Public

- **Box size (kernel) for convolve in bad region interp. (bad_region_kernel_size)**

  Defines the box size (kernel) for the convolution. Used in interpolating over the bad regions (holes) before localization is done. Value must be a positive integer.

  bad_region_kernel_size = 51

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.interp_bad_regions
  Level: Public

- **Median filter box size 2 for bad region interp. (bad_region_med_size2)**

  Defines the median filter box size used (during the convolution) in interpolating over the bad regions (holes) before the localization is done. Value must be a positive integer.

  bad_region_med_size2 = 11

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.interp_bad_regions
  Level: Public

- **Threshold for good ratio for bad region interp. (bad_region_good_value)**

  Defines the threshold (of the ratio between original image and the interpolated image) where pixels are deemed "good". For use in interpolating over the bad regions (holes) before the localization is done.

  bad_region_good_value = 0.5

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.interp_bad_regions
  Level: Public

- **Threshold for bad ratio for bad region interp. (bad_region_bad_value)**

  Defines the threshold (of the ratio between original image and the interpolated image) where pixels are deemed "bad". For use in interpolating over the bad regions (holes) before the localization is done.

  bad_region_bad_value = 0.25

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.interp_bad_regions
  Level: Public

- **Min/Max smoothing box size (ic_locnbpix)**

  Defines the half-size of the rows to use when smoothing the image to work out the minimum and maximum pixel values. This defines the half-spacing between orders and is used to estimate background and the maximum signal. Value must be greater than zero and less than the number of rows (y-axis dimension).

  ic_locnbpix = 45

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouBACK.spirouBACK.measure_min_max()
  Level: Public

- **Minimum signal amplitude (ic_min_amplitude)**

  Defines a cut off (in e-) where below this point the central pixel values will be set to zero. Value must be a float greater than zero.

  ic_min_amplitude = 100.0

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouBACK.spirouBACK.measure_background_and_get_central_pixels()
  Level: Public

- **Normalized background amplitude threshold (ic_locseuil)**

  Defines the normalized amplitude threshold to accept pixels for background calculation (pixels below this normalized value will be used for the background calculation). Value must be a float between zero and one.

  ic_locseuil = 0.2

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouBACK.spirouBACK.measure_background_and_get_central_pixels()
  Level: Public

- **Saturation threshold on the order profile plot (ic_satseuil)**

  Defines the saturation threshold on the order profile plot, pixels above this value will be set this value (ic_satseuil). Value must be a float greater than zero.

  ic_satseuil = 64536

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Degree of the fitting polynomial for localization position (ic_locdfitc)**

Defines the degree of the fitting polynomial for locating the positions of each order i.e. if value is 1 is a linear fit, if the value is 2 is a quadratic fit. The value must be a positive integer equal to or greater than zero (zero would lead to a constant fit along the column direction (x-axis direction).

ic_locdfitc = 5

Used in: cal_loc_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.initial_order_fit(), SpirouDRS.spirouLOCOR.spirouLOCOR.sigmaclip_order_fit()
Level: Public

- **Degree of the fitting polynomial for localization width (ic_locdfitw)**

Defines the degree of the fitting polynomial for measuring the width of each order i.e. if value is 1 is a linear fit, if the value is 2 is a quadratic fit. The value must be a positive integer equal to or greater than zero (zero would lead to a constant fit along the row direction (y-axis direction).

ic_locdfitw = 5

Used in: cal_loc_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.initial_order_fit(), SpirouDRS.spirouLOCOR.spirouLOCOR.sigmaclip_order_fit()
Level: Public

- **Degree of the fitting polynomial for localization position error (ic_locdfitp)**

Defines the degree of the fitting polynomial for locating the positions error of each order i.e. if value is 1 is a linear fit, if the value is 2 is a quadratic fit. The value must be a positive integer equal to or greater than zero (zero would lead to a constant fit along the column direction (x-axis direction).

ic_locdfitp = 3

Used in: cal_loc_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouConfig.spirouKeywords, cal_loc_RAW_spirou.main(), SpirouDRS.spirouLOCOR.spirouLOCOR.sigmaclip_order_fit()
Level: Public

Note: This is only currently used to add the value to the localization file ('_loco_fiber.fits') but not used in any calculation. It could be removed?

- **Maximum RMS for sigma-clipping order fit (positions) (ic_max_rms_center)**

  Defines the maximum RMS allowed for an order, if RMS is above this value the position with the highest residual is removed and the fit is recalculated without that position (sigma-clipped). Value must be a positive float. i.e. position fit is recalculated if:

  $$max(RMS) > \text{ic\_max\_rms\_center} \tag{10.5}$$

  ic_max_rms_center = 0.2

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouLOCOR.spirouLOCOR .sigmaclip_order_fit()
  Level: Public

- **Maximum peak-to-peak for sigma-clipping order fit (positions) (ic_max_ptp_center)**

  Defines the maximum peak-to-peak value allowed for an order, if the peak to peak is above this value the position with the highest residual is removed and the fit is recalculated without that position (sigma-clipped). Value must be a positive float. i.e. position fit is recalculated if:

  $$max(|\text{residuals}|) > \text{ic\_max\_ptp\_center} \tag{10.6}$$

  ic_max_ptp_center = 0.2

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouLOCOR.spirouLOCOR .sigmaclip_order_fit()
  Level: Public

- **Maximum peak-to-peak-RMS ratio for sigma-clipping order fit(positions) (ic_ptporms_center)**

  Defines the maximum ratio of peak-to-peak residuals and RMS value allowed for an order, if the ratio is above this value the position with the highest residual is removed and the fit is recalculated without that position (sigma-clipped). Value must be a positive float. i.e. position

  fit is recalculated if:

  $$max(|\text{residuals}|)/\text{RMS} > \text{ic\_ptporms\_center} \quad (10.7)$$

  ic_ptporms_center = 8.0

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.sigmaclip_order_fit()
  Level: Public

- **Maximum RMS for sigma-clipping order fit (width) (ic_max_rms_fwhm)**

  Defines the maximum RMS allowed for an order, if RMS is above this value the width with the highest residual is removed and the fit is recalculated without that width (sigma-clipped). Value must be a positive float. i.e. width fit is recalculated if:

  $$max(RMS) > \text{ic\_max\_rms\_width} \quad (10.8)$$

  ic_max_rms_fwhm = 1.0

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.sigmaclip_order_fit()
  Level: Public

- **Maximum peak-to-peak for sigma-clipping order fit (widths) (ic_max_ptp_fracfwhm)**

  Defines the maximum peak-to-peak value allowed for an order, if the peak to peak is above this value the width with the highest residual is removed and the fit is recalculated without that width (sigma-clipped). Value must be a positive float. i.e. width fit is recalculated if:

  $$max(|\text{residuals/data}|) \times 100 > \text{ic\_max\_ptp\_fracfwhm} \quad (10.9)$$

  ic_max_ptp_fracfwhm = 1.0

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouLOCOR.spirouLOCOR.sigmaclip_order_fit()
  Level: Public

- **Delta width 3 convolve shape model (ic_loc_delta_width)**

  Defines the delta width in pixels for the 3 convolve shape model - currently not used. Value must be a positive float.

  ic_loc_delta_width = 1.85

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main(), SpirouDRS.spirouConfig.spirouKeywords
  Level: Public

  Note: This is currently not used (other than saving in the calibDB loco file. Can it be removed?).

- **Localization archiving option (ic_locopt1)**

  Whether we save the location image with the superposition of the fit (zeros). If this option is 1 or True it will save the file to '_with-order_fiber.fits' if 0 or False it will not save this file. Value must be 1, 0, True or False.

  ic_locopt1 = 1

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

10.10 Slit calibration variables

- **Tilt oversampling factor (ic_tilt_coi)**

  Defines the oversampling factor used to work out the tilt of the slit. Value must be an integer value larger than zero.

  ic_tilt_coi = 10

  Used in: cal_SLIT_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.get_tilt()
  Level: Public

  Note: Formally this was called 'coi' in cal_SLIT_spirou.

- **Slit fit order plot offset factor (ic_facdec)**

  Defines an offset of the position fit to show the edges of the illuminated area. (Final offset is ±× 2 of this offset away from the order fit. Value must be a positive float.)

  ic_facdec = 1.6

  Used in: cal_SLIT_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouCore.spirouPlot.slit_sorder_plot()
  Level: Public

- **Degree of the fitting polynomial for the tilt (ic_tilt_fit)**

  Defines the degree of the fitting polynomial for determining the tilt i.e. i.e. if value is 1 is a linear fit, if the value is 2 is a quadratic fit. The value must be a positive integer equal to or greater than zero (zero would lead to a constant fit).

  ic_tilt_fit = 4

  Used in: cal_SLIT_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.fit_tilt()
  Level: Public

- **Selected order in Slit fit order plot (ic_slit_order_plot)**

  Defines the selected order to plot the fit for in the Slit fir order plot. Value must be between zero and the maximum number of orders.

  ic_slit_order_plot = 10

  Used in: cal_SLIT_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouCore.spirouPlot.slit_sorder_plot()
  Level: Public

10.11 Flat fielding calibration variables

- **Measure background (ic_do_bkgr_subtraction)**

Define whether to measure the background and do a background subtraction. Value must be True or 1 to do the background measurement and subtraction or be False or 0 to not do the background measurement and subtraction.

ic_do_bkgr_subtraction = 0

Used in:	cal_FF_RAW_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_FF_RAW_spirou.main()
Level:	Public

Note: Currently even if True or 1 the background is not calculated as the INTERPOL function has not been converted to python.

- **Half-size of background window (ic_bkgr_window)**

Defines the half-size (in pixels) of the background window to create a sub-frame to find the minimum 2× ic_bkgr_window pixels for which to calculate the background from. Size is used in both row and column (y and x) direction. Value must be an integer between zero and the minimum(row number, column number) (minimum(x-axis dimension, y-axis dimension)).

ic_bkgr_window = 100

Used in:	cal_FF_RAW_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouBACK.spirouBACK.measure_background_flatfield()
Level:	Public

- **Number of orders in tilt measurement (ic_tilt_nbo)**

Defines the number of orders in the tilt measurement file (TILT key in the ic_calibDB_filename). This is the number of tilts that will be extracted. Value must be an integer larger than zero and smaller than or equal to the total number of orders present in the TILT file.

ic_tilt_nbo = 36

Used in:	cal_FF_RAW_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouImage.spirouFITS.read_tilt_file()
Level:	Public

Note: This can probably be removed and replaced with a check to the TILT file - to automatically determine how many orders there should be.

> **Note:** This was formally called 'nbo' and was hard coded in cal_FF_RAW_spirou.

- **Flat extraction start order (ff_start_order)**

  Start order of the extract for the flat finding recipe. If value is "None" uses start order = 0, else must be a positive order number less than the total number of orders.

  ff_start_order = None

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouFLAT.spirouFLAT.get_valid_orders()
  Level: Public

- **Flat extraction end order (ff_end_order)**

  End order of the extract for the flat finding recipe. If value is "None" uses last order, else must be a positive order number less than the total number of orders.

  ff_end_order = None

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouFLAT.spirouFLAT.get_valid_orders()
  Level: Public

- **The manually set sigdet for flat fielding. (ic_ff_sigdet)**

  This defines the sigdet to use in the weighted tilt extraction. Set to -1 to use from the input file ('fitsfilename') HEADER. Value must be either -1 or a positive float.

  ic_ff_sigdet = 100.0

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_FF_RAW_spirou.main()
  Level: Public

- **Half size blaze window (ic_extfblaz)**

  Defines the distance from the central column that should be used to measure the blaze for each order. Value must be an integer greater than zero and less than half the number of columns (x-axis dimension).

  ic_extfblaz = 50

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_FF_RAW_spirou.main()
  Level: Public

- **Fit degree for the blaze polynomial fit (ic_blaze_fitn)**

Defines the degree of the fitting polynomial for fitting the blaze function of each order i.e. if value is 1 is a linear fit, if the value is 2 is a quadratic fit. The value must be a positive integer equal to or greater than zero (zero would lead to a constant fit along the column direction (x-axis direction).

ic_blaze_fitn = 5

Used in: cal_FF_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_FF_RAW_spirou.main()
Level: Public

- **Selected order for flat fielding plot (ic_ff_order_plot)**

Defines the selected order to plot on the flat fielding image plot. Value must be a integer between zero and the number of orders.

ic_ff_order_plot = 5

Used in: cal_FF_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouCore.spirouPlot.ff_sorder_fit_edges
Level: Public

Note: This was formally called 'ic_plot_order' in cal_FF_RAW_spirou.

- **Plot all order fits for flat fielding plot (ic_ff_plot_all_orders)**

If True or 1, instead of plotting the selected order from ic_ff_order_plot will plot the order fits (and edges) for all orders. This is slower than just plotting one. Value must be True or 1 or False or 0.

ic_ff_plot_all_orders = 0

Used in: cal_FF_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_FF_RAW_spirou.main()
Level: Public

Note: This is a new plot, instead of plotting one selected order plots all orders - this is obviously slightly slower than just plotting one example order.

- **Flat-field plot skip orders (FF_RMS_PLOT_SKIP_ORDERS)**

  Define the orders not to plot on the RMS flat-field plot. Should be a valid python list of integers (with each value being an integer between 0 and the maximum number of orders)

  FF_RMS_PLOT_SKIP_ORDERS = `[22, 23, 24, 25, 48]`

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouCore.spirouPlot.`ff_rms_plot`
  Level: Public

- **Flat-field extraction type (IC_FF_EXTRACT_TYPE)**

  Defines the extraction type for the flat-fielding. Must be one of the following:

  - "0" - Simple extraction (function = SpirouDRS.spirouEXTOR.spirouEXTOR.`extract_const_range`)
  - "1" - Wieghted extraction (function = SpirouDRS.spirouEXTOR.spirouEXTOR.`extract_weight`)
  - "2" - tilt extraction (function = SpirouDRS.spirouEXTOR.spirouEXTOR.`extract_tilt`)
  - "3a" - tilt weight extraction - old (function = SpirouDRS.spirouEXTOR.spirouEXTOR.`extract_tilt_weight`)
  - "3b" - tilt weight extraction 2 - old (function = SpirouDRS.spirouEXTOR.spirouEXTOR.`extract_tilt_weight_old2`)
  - "3c" - tilt weight extraction 2 (function = SpirouDRS.spirouEXTOR.spirouEXTOR.`extract_tilt_weight2`)
  - "3d" - tilt weight extraction 2 - with cosmic correction (function = SpirouDRS.spirouEXTOR.spirouEXTOR.`extract_tilt_weight2cosm`)

  IC_FF_EXTRACT_TYPE = "3c"

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_FF_RAW_spirou.main, SpirouDRS.spirouEXTOR.spirouEXTOR.`extraction_wrapper`
  Level: Public

10.12 Extraction calibration variables

- **Extraction option - rough extraction (ic_extopt)**

  Extraction option for rough extraction:

  - if 0 extraction by summation over a constant range
  - if 1 extraction by summation over constants sigma (not currently available)
  - if 2 Horne extraction without cosmic elimination (not currently available)
  - if 3 Horne extraction with cosmic elimination (not currently available)

  Used for estimating the slit tilt and in calculating the blaze/flat fielding. Value must be a integer between 0 and 3.

  ic_extopt = 0

  Used in: cal_SLIT_spirou, cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouEXTOR.spirouEXTOR.extract_AB_order(), SpirouDRS.spirouEXTOR.spirouEXTOR.extract_order
  Level: Public

- **Extraction start order (ff_start_order)**

  Start order of the extract for the extraction. If value is "None" uses start order = 0, else must be a positive order number less than the total number of orders.

  ff_start_order = None

  Used in: cal_extract_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouEXTOR.spirouEXTOR.get_valid_orders()
  Level: Public

- **Extraction end order (ff_end_order)**

  End order of the extract for the extraction. If value is "None" uses last order, else must be a positive order number less than the total number of orders.

  ff_end_order = None

  Used in: cal_extract_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouEXTOR.spirouEXTOR.get_valid_orders()
  Level: Public

- **Extraction distance - rough extraction (ic_extnbsig)**

The pixels are extracted from the centre of the order out to the edges in the row direction (y-axis), i.e. defines the illuminated part of the order). Used for estimating the slit tilt and in calculating the blaze/flat fielding. Value must be a positive float between 0 and the total number of rows (y-axis dimension).

ic_extnbsig = 2.5

Used in: cal_SLIT_spirou , cal_FF_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouEXTOR.spirouEXTOR .extract_AB_order
Level: Public

- **Extraction type (ic_extract_type)**

Defines the extraction type for the cal_extract_RAW_spirou . Must be one of the following:

- "0" - Simple extraction (function = SpirouDRS.spirouEXTOR.spirouEXTOR .extract_const_range)
- "1" - Wieghted extraction (function = SpirouDRS.spirouEXTOR.spirouEXTOR .extract_weight)
- "2" - tilt extraction (function = SpirouDRS.spirouEXTOR.spirouEXTOR .extract_tilt)
- "3a" - tilt weight extraction - old (function = SpirouDRS.spirouEXTOR.spirouEXTOR .extract_tilt_weight)
- "3b" - tilt weight extraction 2 - old (function = SpirouDRS.spirouEXTOR.spirouEXTOR .extract_tilt_weight_old2)
- "3c" - tilt weight extraction 2 (function = SpirouDRS.spirouEXTOR.spirouEXTOR .extract_tilt_weight2)
- "3d" - tilt weight extraction 2 - with cosmic correction (function = SpirouDRS.spirouEXTOR .spirouEXTOR .extract_tilt_weight2cosm)

ic_extract_type = "3d"

Used in: cal_extract_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_extract_RAW_spirou .main()
Level: Public

- **Tilt border (ic_ext_tilt_bord)**

Set the number of pixels to set as the border (needed to allow for tilt to not go off edge of image). Value must be a positive integer, smaller than half the number of x pixels.

ic_ext_tilt_bord = 2

Used in: cal_extract_RAW_spirou, cal_FF_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouEXTOR.spirouEXTOR.extract_tilt_order(), SpirouDRS.spirouEXTOR.spirouEXTOR.extract_tilt_weight_order(), SpirouDRS.spirouEXTOR.spirouEXTOR.extract_tilt_weight_order2()
Level: Public

- **Manually set the extraction sigdet (ic_ext_sigdet)**

Set the sigdet used in the extraction process instead of using the sigdet in the FITS rec HEADER file. If the value is set to -1 the sigdet from the HEADER is used instead.

ic_ext_sigdet = 100

Used in: cal_extract_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_extract_RAW_spirou.main()
Level: Public

Note: Why is this value used and not the value in the header file?

- **Selected order in extract fit order plot (ic_ext_order_plot)**

Defines the selected order to plot the fit for in the extract fit order plot. Value must be between zero and the maximum number of orders.

ic_ext_order_plot = 20

Used in: cal_extract_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouCore.spirouPlot.ext_selected_order_plot()
Level: Public

- **Cosmic Sig Cut (IC_COSMIC_SIGCUT)**

  Defines the percentage of (normalized) flux above which we detect as cosmics.

  IC_COSMIC_SIGCUT = 0.25

  Used in: cal_FF_RAW_spirou , cal_extract_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouEXTOR.spirouEXTOR.extraction_wrapper, SpirouDRS.spirouEXTOR.spirouEXTOR.extract_tilt_weight2cosm
  Level: Public

- **Max Cosmic Threshold (IC_COSMIC_THRES)**

  Defines the maximum number of iterations to check for cosmics (for each pixel).

  IC_COSMIC_THRES = 5

  Used in: cal_FF_RAW_spirou , cal_extract_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouEXTOR.spirouEXTOR.extraction_wrapper, SpirouDRS.spirouEXTOR.spirouEXTOR.extract_tilt_weight2cosm
  Level: Public

10.13 Drift calibration variables

- **Noise value for SNR drift calculation (ic_drift_noise)**

Define the noise value for the signal to noise ratio in the drift calculation.

$$snr = flux/\sqrt{(\text{flux} + \text{noise}^2)} \tag{10.10}$$

Value must be a float larger than zero.

ic_drift_noise = 100.0

Used in: cal_DRIFT_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_DRIFT_RAW_spirou.main()
Level: Public

- **Do background correction (IC_DRIFT_BACK_CORR)**

Defines whether the background correction is done.

IC_DRIFT_BACK_CORR = 1

Used in: cal_DRIFT_E2DS_spirou, cal_DRIFTPEAK_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_DRIFT_E2DS_spirou.main(), cal_DRIFTPEAK_E2DS_spirou.main()
Level: Public

- **The maximum flux for a good (unsaturated) pixel (ic_drift_maxflux)**

Defines the maximum flux to define a good pixel. This pixels and those that surround it will not be used in determining the RV parameters. Value must be a float greater than zero.

ic_drift_maxflux = 1.e9

Used in: cal_DRIFT_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_DRIFT_RAW_spirou.main()
Level: Public

- **Saturated pixel flag size (ic_drift_boxsize)**

Defines the number of pixels around a saturated pixel to flag as unusable (and hence not used in determining the RV parameters). Value must be a integer larger than zero.

ic_drift_boxsize = 12

Used in: cal_DRIFT_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_DRIFT_RAW_spirou.main()
Level: Public

- **Large number of files for skip (drift_nlarge)**

Defines the number of files that is large enough to require the 'drift_file_skip' parameter (only uses one file in every 'drift_file_skip' files). This is done to speed up the code and avoid a bug. Value must be an integer larger than zero.

drift_nlarge = 300

Used in: cal_DRIFT_RAW_spirou, cal_DRIFT_E2DS_spirou, cal_DRIFTPEAK_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_DRIFT_RAW_spirou.main(), cal_DRIFT_E2DS_spirou.main(), cal_DRIFTPEAK_E2DS_spirou.main()
Level: Public

Note: Has this bug been fixed, do we need to skip for a large number of files?

- **Large number of files skip parameter (cal_DRIFT_RAW_spirou) (drift_file_skip)**

Defines how many files we skip. This is done by selecting one file every 'drift_file_skip' files. i.e. if skip is 3 the code uses every 3rd file to calculate the drift. Value must be an integer larger than zero. A value of 1 is equivalent to no skipping of files regardless of the file number.

drift_file_skip = 3

Used in: cal_DRIFT_RAW_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: cal_DRIFT_RAW_spirou.main()
Level: Public

- **Large number of files skip parameter (cal_DRIFT_E2DS_spirou) (drift_e2ds_file_skip)**

  Defines how many files we skip. This is done by selecting one file every 'drift_file_skip' files. i.e. if skip is 3 the code uses every 3rd file to calculate the drift. Value must be an integer larger than zero. A value of 1 is equivalent to no skipping of files regardless of the file number.

  drift_e2ds_file_skip = 1

  Used in: cal_DRIFT_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DRIFT_E2DS_spirou.main()
  Level: Public

- **Number of sigmas to cut in cosmic re-normalization (cal_DRIFT_RAW_spirou) (ic_drift_cut_raw)**

  Defines the number of standard deviations to remove fluxes at (and replace with the reference flux) for cal_DRIFT_RAW_spirou. Value must be a float larger than zero.

  ic_drift_cut_raw = 3

  Used in: cal_DRIFT_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DRIFT_RAW_spirou.main()
  Level: Public

- **Number of sigmas to cut in cosmic re-normalization (cal_DRIFT_E2DS_spirou) (ic_drift_cut_e2ds)**

  Defines the number of standard deviations to remove fluxes at (and replace with the reference flux) for cal_DRIFT_E2DS_spirou. Value must be a float larger than zero.

  ic_drift_cut_e2ds = 4.5

  Used in: cal_DRIFT_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DRIFT_E2DS_spirou.main()
  Level: Public

- **Number of orders to use in drift (ic_drift_n_order_max)**

  Defines the number of orders to use (starting from zero to maximum number). This is used to get the median drift. Value must be an integer between 0 and the maximum number of orders.

  ic_drift_n_order_max = 28

  Used in: cal_DRIFT_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DRIFT_RAW_spirou.main()
  Level: Public

- **Define the way to combine orders for drift (for cal_DRIFT_RAW_spirou) (drift_type_raw)**

  Defines the way to calculate the combine order drifts (to one drift per image) should either be 'weighted mean' (Equation 10.11) or 'median' (Equation 10.12) for cal_DRIFT_RAW_spirou.

  $$\text{drift} = \frac{\sum(\text{drift}_i * w_i)}{\sum w_i} \tag{10.11}$$

  where w_i is $1/\Delta v_{rms}$

  $$\text{drift} = \text{median}(\text{drift}_i) \tag{10.12}$$

  Value should be a valid python string either 'median' or 'weighted mean'.

  drift_type_raw = median

  Used in: cal_DRIFT_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DRIFT_RAW_spirou.main()
  Level: Public

- **Define the way to combine orders for drift cal_DRIFT_E2DS_spirou) (drift_type_e2ds)**

  Defines the way to calculate the combine order drifts (to one drift per image) should either be 'weighted mean' (Equation 10.13) or 'median' (Equation 10.14) for cal_DRIFT_E2DS_spirou.

  $$\text{drift} = \frac{\sum(\text{drift}_i * w_i)}{\sum w_i} \tag{10.13}$$

  where w_i is $1/\Delta v_{rms}$

  $$\text{drift} = \text{median}(\text{drift}_i) \tag{10.14}$$

  Value should be a valid python string either 'median' or 'weighted mean'.

  drift_type_e2ds = weighted mean

  Used in: cal_DRIFT_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DRIFT_E2DS_spirou.main()
  Level: Public

- **Selected order in drift fit order plot (ic_drift_order_plot)**

  Defines the selected order to plot the fit for in the drift fit order plot. Value must be between zero and the maximum number of orders.

  ic_drift_order_plot = 20

  Used in: cal_DRIFT_RAW_spirou, cal_DRIFT_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouCore.spirouPlot.drift_plot_selected_wave_ref()
  Level: Public

10.14 Drift-Peak calibration variables

- **First order to use in drift-peak (ic_drift_peak_n_order_min)**

  Defines the first order to use (from this to ic_drift_peak_n_order_max). This is used to get the median drift. Value must be an integer greater than or equal to 0 and less than ic_drift_peak_n_order_max.

  ic_drift_peak_n_order_min = 2

Used in:	cal_DRIFTPEAK_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_DRIFTPEAK_E2DS_spirou.main()
Level:	Public

- **Last order to use in drift-peak (ic_drift_peak_n_order_max)**

  Defines the last order to use (from ic_drift_peak_n_order_min to this). This is used to get the median drift. Value must be an integer greater than ic_drift_peak_n_order_min and less than or equal to the maximum number of orders.

  ic_drift_peak_n_order_max = 30

Used in:	cal_DRIFTPEAK_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_DRIFTPEAK_E2DS_spirou.main()
Level:	Public

- **Large number of files skip parameter (cal_DRIFT_E2DS_spirou) (drift_e2ds_file_skip)**

  Defines how many files we skip. This is done by selecting one file every 'drift_file_skip' files. i.e. if skip is 3 the code uses every 3rd file to calculate the drift. Value must be an integer larger than zero. A value of 1 is equivalent to no skipping of files regardless of the file number.

  drift_e2ds_file_skip = 1

Used in:	cal_DRIFTPEAK_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_DRIFTPEAK_E2DS_spirou.main()
Level:	Public

- **Minimum box size for min max smoothing (drift_peak_minmax_boxsize)**

  Defines the minimum size of the box used to get the minimum and maximum pixel values (specifically minimum pixel values). Each box (defined as the pixel position ±box size) is used to work out the background value for that pixel. Value must be an integer larger than zero and less than half the number of columns (x-dimension).

  drift_peak_minmax_boxsize = 6

  Used in: cal_DRIFTPEAK_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DRIFTPEAK_E2DS_spirou.main()
  Level: Public

- **Image column (x-dim) border size (drift_peak_border_size)**

  Defines the number of pixels on either side of an image that should not be used to find FP peaks. This size must be larger to or equal to drift_peak_fpbox_size, therefore the fit to an individual FP does not go off the edge of the image. Value must be an integer larger to or equal to drift_peak_fpbox_size and less than and less than half the number of columns (x-dimension).

  drift_peak_border_size = 3

  Used in: cal_DRIFTPEAK_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV.create_drift_file()
  Level: Public

- **Box size for fitting individual FP peak. (drift_peak_fpbox_size)**

  Defines the half-box size (i.e. central position ±box size) of the box used to fit an individual FP peak. This size must be large enough to fit a peak but not too large as to encompass multiple FP peaks. The value must be an integer larger than zero and smaller than or equal to drift_peak_border_size (to avoid fitting off the edges of the image).

  drift_peak_fpbox_size = 3

  Used in: cal_DRIFTPEAK_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV.create_drift_file(), SpirouDRS.spirouRV.spirouRV.get_drift()
  Level: Public

- **Minimum sigma above median for valid peak (drift_peak_peak_sig_lim)**

Defines the flux a valid peak must have in order to be recognized as a valid peak (before the peak fitting is done). If a peaks maximum is below this threshold it will not be used as a valid peak in finding the drifts. Value is a dictionary containing keys equivalent to the lamp types (currently this is 'fp' and 'hc'. The values of each must be a float greater than 1 for above the median and, between zero and 1 for below the median).

drift_peak_peak_sig_lim = `fp':1.0, 'hc':7.0'fp':1.0, 'hc':7.0`

Used in: cal_DRIFTPEAK_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouRV.spirouRV.create_drift_file()
Level: Public

- **Minimum spacing between valid peaks (drift_peak_inter_peak_spacing)**

Defines the minimum spacing peaks must have (between neighbouring peaks) in order to recognized as valid peaks (before the peak fitting is done). If peak is closer than this separation to a previous peak the peak will not be used as a valid peak in finding the drifts. Value must be an integer greater than zero.

drift_peak_inter_peak_spacing = 5

Used in: cal_DRIFTPEAK_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouRV.spirouRV.create_drift_file()
Level: Public

- **Expected width of FP peaks (drift_peak_exp_width)**

Defines the expected width of the FP peaks. Parameter is used to 'normalise' the peaks which are then subsequently removed if:

$$\text{normalized FP FWHM} > \text{drift\_peak\_norm\_width\_cut} \tag{10.15}$$

this is equivalent to:

$$\text{FP FWHM} > (\text{drift\_peak\_exp\_width} + \text{drift\_peak\_norm\_width\_cut}) \tag{10.16}$$

Value must be a float larger than zero and less than the number of columns (x-dimension).

drift_peak_exp_width = 0.8

Used in: cal_DRIFTPEAK_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouRV.spirouRV.remove_wide_peaks(), SpirouDRS.spirouRV.spirouRV.get_drift()
Level: Public

- **Normalized FP width threshold (drift_peak_norm_width_cut)**

  Defines the maximum 'normalized' width of FP peaks that is acceptable for a valid FP peak. i.e. widths above this threshold are rejected as valid FP peaks. This works as follows:

  $$\text{normalized FP FWHM} > \text{drift\_peak\_norm\_width\_cut} \tag{10.17}$$

  this is equivalent to:

  $$\text{FP FWHM} > (\text{drift\_peak\_exp\_width} + \text{drift\_peak\_norm\_width\_cut}) \tag{10.18}$$

  Value must be a float larger than zero and less than the number of columns (x-dimension) but if drift_peak_exp_width is defined sensibly then this number should be small.

  drift_peak_norm_width_cut = 0.2

Used in:	cal_DRIFTPEAK_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouRV.spirouRV.remove_wide_peaks()
Level:	Public

- **Get drift via a Gaussian fitting process (drift_peak_getdrift_gaussfit)**

  Defines whether the drift is calculated via a Gaussian fitting process (fitting the targeted order with a Gaussian) – $\sim \times 10$ slower, or adjusts a barycentre to get the drift. Value must be True or 1 to do the Gaussian fit, or False or 0 to use the barycentre adjustment.

  drift_peak_getdrift_gaussfit = False

Used in:	cal_DRIFTPEAK_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_DRIFTPEAK_E2DS_spirou.main()
Level:	Public

- **Pearson R coefficient (between reference and image) (drift_peak_pearsonr_cut)**

  Defines the threshold below which a image is deemed to dissimilar from the reference image to be used. A Pearson R test is performed between the reference image (E2DS file) and the current iteration image (E2DS file), the minimum of all usable orders is then tested. If any order does not pass the criteria:

  $$\text{coefficient}_{\text{order}} > \text{drift\_peak\_pearsonr\_cut} \tag{10.19}$$

  then the whole image (E2DS file) is rejected. Value must be a float larger than zero and less than 1.0, values should be close to unity for a good fit i.e. 0.97.

  drift_peak_pearsonr_cut = 0.9

Used in:	cal_DRIFTPEAK_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_DRIFTPEAK_E2DS_spirou.main()
Level:	Public

Note: This value is currently set below a recommended level and should be set back to 0.97 as soon as possible, even coefficients at 0.95 are from very bad orders, and orders should be removed. A plot currently is made when a bad file is found (i.e. when the above cut is not met).

- **Sigma clip for found FP peaks (drift_peak_sigmaclip)**

Defines the number of sigmas above the median that is used to remove bad FP peaks from the drift calculation process. Value must be a float larger than zero.

drift_peak_sigmaclip = 1.0

Used in:	cal_DRIFTPEAK_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_DRIFTPEAK_E2DS_spirou.main()
Level:	Public

- **Plot linelist vs log Amplitude (drift_peak_plot_line_log_amp)**

Defines whether we plot the line list against log amplitude. Value must be 1 or True to plot, or 0 or False to not plot

drift_peak_plot_line_log_amp = False

Used in:	cal_DRIFTPEAK_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouCore.spirouPlot.drift_peak_plot_ll-peak_amps()
Level:	Public

- **Selected order for line-list vs log Amplitude plot (drift_peak_selected_order)**

Defines the selected order to plot the wave vs extracted spectrum for over-plotting on the line list against log amplitude plot. Value must be an integer between 0 and the number of orders

drift_peak_selected_order = 30

Used in:	cal_DRIFTPEAK_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouCore.spirouPlot.drift_peak_plot_ll-peak_amps()
Level:	Public

10.15 Bad pixel calibration variables

- **Bad pixel median image box width (badpix_flat_med_wid)**

  A similar flat is produced by taking the running median of the flat in the column direction (x-dimension) over a boxcar width of badpix_flat_med_wid. This assumes that the flux level varies only by a small amount over badpix_flat_med_wid pixels and that the bad pixels are isolated enough that the median along that box will be representative of the flux they should have if they were not bad. Value should be an integer larger than zero and less than the number of columns (x-axis dimension).

  badpix_flat_med_wid = 7

  Used in: cal_BADPIX_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.normalise_median_flat(), SpirouDRS.spirouImage.spirouImage.locate_bad_pixels()
  Level: Public

  Note: Formally this was called wmed in cal_BADPIX_spirou

- **Bad pixel illumination cut parameter (badpix_illum_cut)**

  Threshold below which a pixel is considered unilluminated. As we cut the pixels that fractionally deviate by more than a certain amount (badpix_flat_cut_ratio) this would lead to lots of bad pixels in unilluminated regions of the array. This parameter stops this, as the pixels are normalised this value must be a float greater than zero and less than 1.

  badpix_illum_cut = 0.05

  Used in: cal_BADPIX_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.locate_bad_pixels()
  Level: Public

  Note: Formally this was called illum_cut in cal_BADPIX_spirou

- **Bad pixel maximum differential pixel cut ratio (badpix_flat_cut_ratio)**

  This sets the maximum differential pixel response relative to the expected value. Value must be a float larger than zero.

  badpix_flat_cut_ratio = 0.5

  Used in: cal_BADPIX_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouImage.spirouImage.locate_bad_pixels()
  Level: Public

Note: Formally this was called cut_ratio in cal_BADPIX_spirou

- **Bad pixel maximum flux to considered too hot (badpix_max_hotpix)**

Defines the maximum flux value to be considered too hot to user.

badpix_max_hotpix = 100.0

Used in: cal_BADPIX_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouImage.spirouImage.locate_bad_pixels()
Level: Public

Note: Formally this was called max_hotpix in cal_BADPIX_spirou

- **Bad pixel normalisation percentile (badpix_norm_percentile)**

Defines the percentile at which the bad pixels are normalised to in order to locate bad and dead pixels.

badpix_norm_percentile = 90.0

Used in: cal_BADPIX_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouImage.spirouImage.locate_bad_pixels()
Level: Public

- **Full detector flat file (badpix_full_flat)**

Defines the full detector flat file (located in the data folder)

badpix_full_flat = 'detector_flat_full.fits'

Used in: cal_BADPIX_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouImage.spirouImage.locate_bad_pixels_full()
Level: Public

- **Full detector good threshold (badpix_full_threshold)**

Defines the threshold on the full detector flat file to deem pixels as good

badpix_full_threshold = 0.3

Used in: cal_BADPIX_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouImage.spirouImage.locate_bad_pixels_full()
Level: Public

10.16 Wavelength solution variables

- **Lamp types (IC_LAMPS)**

  Define the lamp types. These must be present in IC_LL_LINE_FILE_ALL and IC_CAT_TYPE_ALL to be used. Each dictionary entry must be a list of strings (to look for in the header).

  IC_LAMPS = `dict(UNe=['hcone', 'hc1'], TH=['hctwo', 'hc2'])`

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.decide_on_lamp_type
  Level: Public

- **Catalogue line list files (IC_LL_LINE_FILE_ALL)**

  Define the lamp types. These must be present in IC_LAMPS and IC_CAT_TYPE_ALL to be used.

  IC_LL_LINE_FILE_ALL = `dict(UNe='catalogue_UNe.dat', TH='catalogue_ThAr.dat')`

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.get_lamp_parameters
  Level: Public

- **Type of line list catalogue (IC_CAT_TYPE_ALL)**

  Define the lamp types. These must be present in IC_LAMPS and IC_LL_LINE_FILE_ALL to be used.

  IC_CAT_TYPE_ALL = `dict(UNe='fullcat', TH='thcat')`

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.get_lamp_parameters
  Level: Public

- **Resolution of the detector (IC_RESOL)**

  Define the resolution of the detector

  IC_RESOL = 65000

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.`find_lines`
  Level: Public

- **Wavelength free-span (IC_LL_FREE_SPAN)**

  Define the wavelength free span parameter in find lines. Must be a list of line widths (in pixels). Ordered from largest to smallest.

  IC_LL_FREE_SPAN = `[6.0, 3.5]`

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.`first_guess_at_wave_solution`, SpirouDRS.spirouTHORCA.spirouTHORCA.`find_lines`
  Level: Public

- **Minimum wavelength - find lines (IC_LL_SP_MIN)**

  Define the minimum wavelength of the detector to use in find lines (in nm).

  IC_LL_SP_MIN = 900

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.`find_lines`
  Level: Public

- **Maximum wavelength - find lines (IC_LL_SP_MAX)**

  Define the maximum wavelength of the detector to use in find lines (in nm).

  IC_LL_SP_MAX = 2400

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.`find_lines`
  Level: Public

- **Readout noise - find lines (IC_HC_NOISE)**

  Define the read out noise to use in find lines

  IC_HC_NOISE = 60

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.find_lines
  Level: Public

- **Maximum sig-fit of guess lines (IC_MAX_SIGLL_CAL_LINES)**

  Define the maximum sigma fit of the guessed lines (FWHM/2.35 of the lines). Used to filter out bad lines after a guess on the lines.

  IC_MAX_SIGLL_CAL_LINES = 4

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.detect_bad_lines
  Level: Public

- **Maximum error on guess lines (IC_MAX_ERRW_ONFIT)**

  Define the maximum error on the guess of the lines. Used to filter out bad lines after a guess on the lines.

  IC_MAX_ERRW_ONFIT = 1

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.detect_bad_lines
  Level: Public

- **Maximum amplitude on guess lines (IC_MAX_AMPL_LINE)**

  Define the maximum amplitude on the guess of the lines. Used to filter out bad lines after a guess on the lines.

  IC_MAX_AMPL_LINE = 1.0e8

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.detect_bad_lines
  Level: Public

- **Wave solution first order (IC_HC_N_ORD_START)**

  Defines the first order to at which the wave solution is calculated.

  IC_HC_N_ORD_START = 13

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.first_guess_at_wave_solution, SpirouDRS.spirouTHORCA.spirouTHORCA.calculate_littrow_sol
  Level: Public

- **Wave solution last order (IC_HC_N_ORD_START)**

  Defines the last order to at which the wave solution is calculated.

  IC_HC_N_ORD_START = 40

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.first_guess_at_wave_solution, SpirouDRS.spirouTHORCA.spirouTHORCA.calculate_littrow_sol, SpirouDRS.spirouTHORCA.spirouWAVE.calculate_instrument_drift
  Level: Public

- **First echelle order number (IC_HC_T_ORDER_START)**

  Defines the first echelle order number extracted.

  IC_HC_T_ORDER_START = 79

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.first_guess_at_wave_solution, SpirouDRS.spirouTHORCA.spirouTHORCA.detect_bad_lines, SpirouDRS.spirouTHORCA.spirouTHORCA.extrapolate_littrow_sol, SpirouDRS.spirouTHORCA.spirouTHORCA.second_guess_at_wave_solution, SpirouDRS.spirouTHORCA.spirouTHORCA.find_lines, SpirouDRS.spirouTHORCA.spirouTHORCA.fit_1d_ll_solution
  Level: Public

- **Minimum x error in 1D fit (IC_ERRX_MIN)**

  Define the maximum weight given to a line via defining the minimum instrumental error, such that:

  $$\text{max\_weight} = 1.0/(\text{IC\_ERRX\_MIN})^2 \quad (10.20)$$

  IC_ERRX_MIN = 0.01

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.fit_1d_ll_solution
  Level: Public

- **Wavelength solution poly-fit degree (IC_LL_DEGR_FIT)**

  Define the wavelength fit polynomial order

  IC_LL_DEGR_FIT = 4

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.fit_1d_ll_solution, SpirouDRS.spirouTHORCA.spirouTHORCA.invert_1ds_ll_solution
  Level: Public

- **Maximum RMS for wave solution sigma-clip fit (IC_MAX_LLFIT_RMS)**

  Define the maximum RMS for the wavelength sigma-clip fitting process

  IC_MAX_LLFIT_RMS = 3.0

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.fit_1d_ll_solution
  Level: Public

- **Littrow poly-fit degree 1 (IC_LITTROW_FIT_DEG_1)**

Define the polynomial fit degree for the Littrow fit (iteration 1)

IC_LITTROW_FIT_DEG_1 = 5

Used in:	cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouTHORCA.`calculate_littrow_sol` with iteration=1
Level:	Public

- **Littrow poly-fit degree 2 (IC_LITTROW_FIT_DEG_2)**

Define the polynomial fit degree for the Littrow fit (iteration 2)

IC_LITTROW_FIT_DEG_2 = 7

Used in:	cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouTHORCA.`calculate_littrow_sol` with iteration=2
Level:	Public

- **Littrow cut steps (IC_LITTROW_CUT_STEP_1)**

Define the x-pixel interval between positions to measure the Littrow at (iteration 1) i.e.:

$$xcutpoints = [1 * step, 2 * step, 3 * step, ..., (N-1) * step] \quad (10.21)$$

where 'step' = IC_LITTROW_CUT_STEP_1 and 'N' = length of x-dimension divided by 'step'.

IC_LITTROW_CUT_STEP_1 = 250

Used in:	cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouTHORCA.`calculate_littrow_sol` with iteration=1
Level:	Public

- **Littrow cut steps (IC_LITTROW_CUT_STEP_2)**

  Define the x-pixel interval between positions to measure the Littrow at (iteration 2) i.e.:

  $$xcutpoints = [1 * step, 2 * step, 3 * step, ..., (N-1) * step] \quad (10.22)$$

  where 'step' = IC_LITTROW_CUT_STEP_2 and 'N' = length of x-dimension divided by 'step'.

  IC_LITTROW_CUT_STEP_2 = 500

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.calculate_littrow_sol with iteration=2
  Level: Public

- **Littrow fit - first order (IC_LITTROW_ORDER_INIT)**

  Defines the first order to start the LIttrow fit from

  IC_LITTROW_ORDER_INIT = 0

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.calculate_littrow_sol, SpirouDRS.spirouTHORCA.spirouTHORCA.extrapolate_littrow_sol
  Level: Public

- **Littrow fit - remove orders (IC_LITTROW_REMOVE_ORDERS)**

  Define the orders to ignore to ignore in the Littrow fit. Must be a valid python list of integers where each integer is a number between zero and the maximum number of orders.

  IC_LITTROW_REMOVE_ORDERS = []

  Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.calculate_littrow_sol
  Level: Public

- **Littrow-Wavelength solution poly-fit degree (IC_LITTROW_ORDER_FIT_DEG)**

Define the wavelength fit polynomial order

IC_LITTROW_ORDER_FIT_DEG = 4

Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.extrapolate_littrow_sol
Level: Public

Note: Same as IC_LL_DEGR_FIT?

- **Wavelength free-span 2 (IC_LL_FREE_SPAN_2)**

Define the wavelength free span parameter in find lines used after Littrow fit. Must be a list of line widths (in pixels). Ordered from largest to smallest.

IC_LL_FREE_SPAN_2 = `[4.25, 3.0]`

Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.second_guess_at_wave_solution, SpirouDRS.spirouTHORCA.spirouTHORCA.find_lines
Level: Public

- **Wave solution first order 2 (IC_HC_N_ORD_START_2)**

Defines the first order to at which the wave solution is calculated used after Littrow fit.

IC_HC_N_ORD_START_2 = 0

Used in: cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouTHORCA.spirouTHORCA.second_guess_at_wave_solution, SpirouDRS.spirouTHORCA.spirouTHORCA.join_orders, SpirouDRS.spirouTHORCA.spirouWAVE.insert_fp_lines
Level: Public

- **Wave solution last order 2 (IC_HC_N_ORD_START_2)**

Defines the last order to at which the wave solution is calculated used after Littrow fit.

IC_HC_N_ORD_START_2 = 46

Used in:	cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouTHORCA.`second_guess_at_wave_solution`, SpirouDRS.spirouTHORCA.spirouTHORCA.`join_orders`, SpirouDRS.spirouTHORCA.spirouWAVE.`insert_fp_lines`
Level:	Public

- **Find lines mode (HC_FIND_LINES_MODE)**

Defines the mode to find lines. Currently allowed modes are:

- 0: Fortran `fitgaus.f` routine
- 1: Python fit using `scipy.optimize.curve_fit`
- 2: Python fit using `lmfit.models` (Model, GaussianModel)
- 3: Python conversion of Fortran 'fitgaus' - direct translation
- 4: Python conversion of Fortran 'fitgaus' - gaussj improvement MH
- 5: Python conversion of Fortran 'fitgaus' - gaussj improvement NJC

where value must be an integer between 0 and 5.

HC_FIND_LINES_MODE = 0

Used in:	cal_HC_E2DS_spirou, cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_HC_E2DS_spirou.main(), cal_WAVE_E2DS_spirou.main(), but used in SpirouDRS.spirouTHORCA.spirouTHORCA.`fitgaus_wrapper` via SpirouDRS.spirouTHORCA.spirouTHORCA.`fit_emi_line` via SpirouDRS.spirouTHORCA.spirouTHORCA.`find_lines`
Level:	Public

Note: If mode = 0 fitgaus.f needs to be compiled with:

```
>> f2py -c -m fitgaus --noopt --quiet fitgaus.f
```

While located in the `.../INTROOT/SpirouDRS/fortan` directory.

Note: If mode = 2 lmfit must be installed with

```
>> pip install lmfit
```

- **FP solution - first order (IC_FP_N_ORD_START)**

Defines the first order to start the FP solution fitting at.

IC_FP_N_ORD_START = 0

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouWAVE.fp_wavelength_sol, SpirouDRS.spirouTHORCA.spirouWAVE.insert_fp_lines
Level:	Public

- **FP solution - last order (IC_FP_N_ORD_FINAL)**

Defines the last order to stop the FP solution fitting at.

IC_FP_N_ORD_FINAL = 24

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouWAVE.fp_wavelength_sol, SpirouDRS.spirouTHORCA.spirouWAVE.insert_fp_lines, SpirouDRS.spirouCore.spirouPlot.wave_plot_final_fp_order
Level:	Public

- **FP solution - region size (IC_FP_SIZE)**

Region size (in pixels) that each FP peak is fitted to

IC_FP_SIZE = 3

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouWAVE.fp_wavelength_sol
Level:	Public

- **FP solution - position threshold (IC_FP_THRESHOLD)**

Defines the threshold used in detecting the positions of the FP peaks.

IC_FP_THRESHOLD = 0.2

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouWAVE.fp_wavelength_sol
Level:	Public

- **FP solution - cavity width (IC_FP_DOPD0)**

Defines the initial value of the FP effective cavity width (in nm). For SPIRou this is:

$$2 \times d = 24.5mm = 24.5 \times 10^6 nm \tag{10.23}$$

IC_FP_DOPD0 = 2.45e7

Used in: cal_WAVE_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouTHORCA.spirouWAVE.fp_wavelength_sol, SpirouDRS.spirouTHORCA.spirouWAVE.find_fp_lines, SpirouDRS.spirouTHORCA.spirouWAVE.correct_for_large_fp_jumps
Level: Public

- **FP solution - poly-fit degree (IC_FP_FIT_DEGREE)**

Defines the polynomial fit degree between FP line numbers and the measured cavity width for each line.

IC_FP_FIT_DEGREE = 9

Used in: cal_WAVE_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouTHORCA.spirouWAVE.fp_wavelength_sol
Level: Public

- **FP solution - large jump size (IC_FP_LARGE_JUMP)**

Defines the "jump" value (in pixels) that is considered too large between the current and previous FP peaks.

IC_FP_LARGE_JUMP = 0.7

Used in: cal_WAVE_E2DS_spirou
Defined in: constants_SPIROU_H4RG.py
Called in: SpirouDRS.spirouTHORCA.spirouWAVE.correct_for_large_fp_jumps
Level: Public

- **Instrumental drift plot order (IC_WAVE_IDRIFT_PLOT_ORDER)**

  Defines the plot order for the comparison between the reference spectrum and this iterations spectrum during the instrumental drift calculation. Must be an integer between zero and the maximum number of orders.

  IC_WAVE_IDRIFT_PLOT_ORDER = 14

  Used in: cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouCore.spirouPlot.wave_plot_instrument_drift
  Level: Public

- **Instrumental drift noise (IC_WAVE_IDRIFT_NOISE)**

  Defines the noise to use in the instrumental drift calculation

  IC_WAVE_IDRIFT_NOISE = 50.0

  Used in: cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouWAVE.calculate_instrument_drift
  Level: Public

- **Instrumental drift maximum flux (IC_WAVE_IDRIFT_MAXFLUX)**

  Defines the maximum flux for a good (unsaturated) pixel to use in the instrumental drift calculation.

  IC_WAVE_IDRIFT_MAXFLUX = 350000

  Used in: cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouWAVE.calculate_instrument_drift
  Level: Public

- **Instrumental drift saturation box size (IC_WAVE_IDRIFT_BOXSIZE)**

  Defines the size around a saturated pixel to flag as unusable in the instrumental drift calculation.

  IC_WAVE_IDRIFT_BOXSIZE = 12

  Used in: cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouTHORCA.spirouWAVE.calculate_instrument_drift
  Level: Public

- **Instrumental drift cosmic sigma (IC_WAVE_IDRIFT_CUT_E2DS)**

  Defines the number of standard deviations to cut at in the cosmic re-normalization calculation during the instrumental drift calculation.

  IC_WAVE_IDRIFT_CUT_E2DS = 4.5

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouWAVE.calculate_instrument_drift
Level:	Public

- **Instrumental drift maximum uncertainty (IC_WAVE_IDRIFT_MAX_ERR)**

  Defines the maximum uncertainty allowed on the RV for a given order. Used during the instrumental drift calculation.

  IC_WAVE_IDRIFT_MAX_ERR = 3.0

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouWAVE.calculate_instrument_drift
Level:	Public

- **Instrumental drift RV cut (IC_WAVE_IDRIFT_RV_CUT)**

  Defines the RV cut above which the RV from orders are not used. Used during the instrumental drift calculation.

  IC_WAVE_IDRIFT_RV_CUT = 20.0

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouWAVE.calculate_instrument_drift
Level:	Public

10.17 CCF calibration variables

- **CCF Width (ic_ccf_width)**

  Define the width of the CCF range. Value must be a positive integer.

  ic_ccf_width = 30.0

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_CCF_E2DS_spirou.main()
  Level: Public

- **CCF Step (ic_ccf_step)**

  Define the computations steps of the CCF

  ic_ccf_step = 0.5

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_CCF_E2DS_spirou.main()
  Level: Public

- **CCF mask weight (ic_w_mask_min)**

  Define the weight of the CCF mask (if 1 force all weights equal)

  ic_w_mask_min = 0.0

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV.get_ccf_mask()
  Level: Public

- **Template line width (ic_mask_width)**

  Define the width of the template line (if 0 use natural)

  ic_mask_width = 1.7

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV.get_ccf_mask()
  Level: Public

- **Barycentric Earth RV (ccf_berv)**

  Define the barycentric Earth RV (berv)

  ccf_berv = 0.0

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV .coravelation()
  Level: Public

- **Maximum barycentric Earth RV (ccf_berv_max)**

  Define the maximum barycentric Earth RV

  ccf_berv_max = 0.0

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV .coravelation()
  Level: Public

- **Detector CCF noise (ccf_det_noise)**

  Define the detector noise to use in the CCF.

  ccf_det_noise = 100.0

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV .coravelation()
  Level: Public

- **CCF Fit type (ccf_fit_type)**

  Define the type of fit for the CCF fit.

  ccf_fit_type = 0

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouRV.spirouRV .coravelation()
  Level: Public

- **CCF end order number. (ccf_num_orders_max)**

  Define the number of orders (from zero to ccf_num_orders_max) to use to calculate the CCF and RV

  ccf_num_orders_max = 25

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_CCF_E2DS_spirou.main()
  Level: Public

- **Earth velocity calculation mode (CCF_BERVMODE)**

  Deine the mode to work out the Earth velocity correction calculation. Options are:

  - "off" - berv is set to zero
  - "old" - berv is calculated with FORTRAN newbervmain.f
  - "new" - berv is calculated using barycorrpy

  CCF_BERVMODE = "new"

  Used in: cal_CCF_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_CCF_E2DS_spirou.main(), SpirouDRS.spirouRV.spirouRV.EarthVelocityCorrection
  Level: Public

Note: If mode = "old" newbervmain.f needs to be compiled with:

```
>> f2py -c -m newbervmain --noopt --quiet newbervmain.f
```

While located in the \ldots/INTROOT/SpirouDRS/fortan directory.

Note: If mode = "new" barycorrpy must be installed with

```
>> pip install barycorrpy
```

10.18 Polarimetry variables

- **Stokes parameters (IC_POLAR_STOKES_PARAMS)**

  Define all possible stokes parameters

  IC_POLAR_STOKES_PARAMS = ['V', 'Q', 'U']

  Used in: pol_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouPOLAR.spirouPOLAR.load_data
  Level: Public

- **Polar fibers (IC_POLAR_FIBERS)**

  Define all possible fibers used for polarimetry

  IC_POLAR_FIBERS = ['A', 'B']

  Used in: pol_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouPOLAR.spirouPOLAR.load_data
  Level: Public

- **Polar method (IC_POLAR_METHOD)**

  Define the polarimetry method. Currently must be either: "Ratio" or "Difference".

  IC_POLAR_METHOD = 'Ratio'

  Used in: pol_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouPOLAR.spirouPOLAR.calculate_polarimetry_wrapper
  Level: Public

- **Polar continuum bin size (IC_POLAR_CONT_BINSIZE)**

  Define the polarimetry continuum bin size (for plotting)

  IC_POLAR_CONT_BINSIZE = 1000

  Used in: pol_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouPOLAR.spirouPOLAR.calculate_continuum
  Level: Public

- **Polar continuum overlap size (IC_POLAR_CONT_OVERLAP)**

  Define the polarimetry continuum overlap size (for plotting)

  IC_POLAR_CONT_OVERLAP = 0

  Used in: pol_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouPOLAR.spirouPOLAR.calculate_continuum
  Level: Public

- **Polar tell mask (IC_POLAR_CONT_TELLMASK)**

  Define the telluric mask for calculation of the continuum.

  IC_POLAR_CONT_TELLMASK = `[[930,967],[1109,1167],[1326,1491],[1782,1979],[1997,2027],[2047,2076]]`

  Used in: pol_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouPOLAR.spirouPOLAR.calculate_continuum
  Level: Public

10.19 Exposure-meter variables

- **Exposure-meter Fiber Type (EM_FIB_TYPE)**

  Define which fiber to extract. Must be either AB or A or B or C.

  EM_FIB_TYPE = "AB"

  Used in: cal_exposure_meter
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_exposure_meter.main()
  Level: Public

- **Telluric threshold (EM_TELL_THRESHOLD)**

  Define the telluric threshold (transmission) level to mask at.

  EM_TELL_THRESHOLD = 0.95

  Used in: cal_exposure_meter
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_exposure_meter.main(), SpirouDRS.spirouImage.spirouExposeMeter.create_mask
  Level: Public

- **Minimum wavelength for exposure-meter (EM_MIN_LAMBDA)**

  Define the minimum wavelength (in nm) to mask at.

  EM_MIN_LAMBDA = 1478.7

  Used in: cal_exposure_meter
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_exposure_meter.main(), SpirouDRS.spirouImage.spirouExposeMeter.create_mask
  Level: Public

- **Maximum wavelength for exposure-meter (EM_MAX_LAMBDA)**

  Define the maximum wavelength (in nm) to mask at.

  EM_MAX_LAMBDA = 1823.1

  Used in: cal_exposure_meter
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_exposure_meter.main(), SpirouDRS.spirouImage.spirouExposeMeter.create_mask
  Level: Public

- **Exposure-meter file output type (EM_OUTPUT_TYPE)**

Define what shape/properties we want the output mask to be. Options are:

- "raw" - shape and rotation as in the raw input files
- "drs" - shape and rotation as in the DRS files
- "preprocess" - shape and rotation as in the pre-processed files
- "all" - produces all of the above

EM_OUTPUT_TYPE = "all"

Used in:	cal_exposure_meter	
Defined in:	constants_SPIROU_H4RG.py	
Called in:	cal_exposure_meter.main(),	SpirouDRS.spirouConfig
	.spirouConst.EM_SPE_FILE,	SpirouDRS.spirouConfig
	.spirouConst.EM_WAVE_FILE,	SpirouDRS.spirouConfig
	.spirouConst.EM_MASK_FILE	
Level:	Public	

- **Exposure-meter - include bad pixel mask (EM_COMBINED_BADPIX)**

Define whether to combine exposure meter mask with the bad pixel mask. If True bad pixel mask is combined if False it is not.

EM_COMBINED_BADPIX = True

Used in:	cal_exposure_meter
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_exposure_meter.main()
Level:	Public

- **Exposure-meter - save wave map (EM_SAVE_WAVE_MAP)**

Define whether to save the 2D wavelength map. If True saves map if False does not save the map.

EM_SAVE_WAVE_MAP = True

Used in:	cal_exposure_meter
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_exposure_meter.main()
Level:	Public

- **Exposure-meter - save telluric spectrum (EM_SAVE_TELL_SPEC)**

  Define whether to save the telluric spectrum mapped on to the 2D wavelength map. If True saves map if False does not save the map.

  EM_SAVE_TELL_SPEC = True

Used in:	cal_exposure_meter
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_exposure_meter.main()
Level:	Public

- **Exposure-meter - save mask (EM_SAVE_MASK_MAP)**

  Define whether to save the mask mapped on to the 2D wavelength map. If True saves map if False does not save the map.

  EM_SAVE_MASK_MAP = True

Used in:	cal_exposure_meter
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_exposure_meter.main()
Level:	Public

10.20 Quality control variables

- **Maximum dark median level (qc_max_darklevel)**

  Defines the maximum dark median level in ADU/s. If this is greater than median flux it does not pass the quality control criteria:

  $$\text{Median Flux} < \text{qc\_max\_darklevel} \tag{10.24}$$

  Value must be a float equal to or larger than zero.

  qc_max_darklevel = 0.5

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DARK_spirou.main()
  Level: Public

- **Maximum percentage of dead pixels (qc_max_dead)**

  Defines the maximum allowed percentage of dead pixels in a dark image. If the number of dead pixels is greater than this it does not pass the quality control criteria:

  $$\text{dead pixels} = (\text{pixel value} > \text{dark\_cutlimit}) \text{ and } (\text{pixel value} \neq \text{NaN}) \tag{10.25}$$

  $$\text{Percentage of dead pixels} < \text{qc\_max\_dead} \tag{10.26}$$

  qc_max_dead = 20.0

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DARK_spirou.main()
  Level: Public

- **Maximum percentage of bad dark pixels (qc_max_dark)**

  Defines the maximum allowed percentage of bad dark pixels in a dark image. If the number of dead pixels is greater than this it does not pass the quality control criteria:

  $$\text{bad dark pixels} = \text{pixel value} > \text{dark\_cutlimit} \tag{10.27}$$

  $$\text{Percentage of bad dark pixels} < \text{qc\_max\_dead} \tag{10.28}$$

  qc_max_dark = 6.0

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DARK_spirou.main()
  Level: Public

- **Minimum dark exposure time (qc_dark_time)**

  Defines the minimum dark exposure time. If exposure time (from FITS rec HEADER) is below this the code will exit with 'Dark exposure time too short' message. Value must be a float greater than zero.

  qc_dark_time = 599.0

  Used in: cal_DARK_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_DARK_spirou.main()
  Level: Public

- **Maximum points removed in localization position fit (qc_loc_maxlocfit_removed_ctr)**

  Defines the maximum allowed number of points removed in the position fitting process (during localization). If number is more than this it does not pass the quality control criteria:

  $$\text{Number of rejected orders in center fit} > \text{qc\_loc\_maxlocfit\_removed\_ctr} \tag{10.29}$$

  Value must be a integer greater than zero.

  qc_loc_maxlocfit_removed_ctr = 1500

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Maximum points removed in localization width fit (qc_loc_maxlocfit_removed_wid)**

  Defines the maximum allowed number of points removed in the width fitting process (during localization). If number is more than this it does not pass the quality control criteria:

  $$\text{Number of rejected orders in width fit} > \text{qc\_loc\_maxlocfit\_removed\_width} \tag{10.30}$$

  Value must be a integer greater than zero.

  qc_loc_maxlocfit_removed_wid = 105

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Maximum allowed RMS in fitting in localization position fit (qc_loc_rmsmax_center)**

  Defines the maximum RMS allowed in the position fitting process (during localization). If the RMs is higher than this value it does not pass the quality control criteria:

  $$\text{Mean rms center fit} > \text{qc\_loc\_rmsmax\_center} \tag{10.31}$$

  Value must be a float greater than zero.

  qc_loc_rmsmax_center = 100

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Maximum allowed RMS in fitting in localization width fit (qc_loc_rmsmax_fwhm)**

  Defines the maximum RMS allowed in the width fitting process (during localization). If the RMs is higher than this value it does not pass the quality control criteria:

  $$\text{Mean rms width fit} > \text{qc\_loc\_rmsmax\_fwhm} \tag{10.32}$$

  Value must be a float greater than zero.

  qc_loc_rmsmax_fwhm = 500

  Used in: cal_loc_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_loc_RAW_spirou.main()
  Level: Public

- **Maximum allowed RMS (qc_ff_rms)**

  Defines the maximum RMS allowed to accept a flat-field for calibration. Value must be a float greater than zero.

  qc_ff_rms = 0.12

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_FF_RAW_spirou.main()
  Level: Public

- **Saturation level reached warning (qc_loc_flumax)**

  Defines the level above which a warning is generated in the form 'SATURATION LEVEL REACHED on Fiber fiber'. Value must be a float greater than zero.

  qc_loc_flumax = 64500

  Used in: cal_FF_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_FF_RAW_spirou.main()
  Level: Public

- **Maximum RMS allowed for slit TILT (qc_slit_rms)**

  Defines the maximum allowed RMS in the calculated TILT to add TILT profile to the calibration database. Value must be a float larger than zero.

  qc_slit_rms = 0.1

  Used in: cal_SLIT_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_SLIT_spirou.main()
  Level: Public

- **Minimum angle allowed for slit TILT (qc_slit_min)**

  Defines the minimum tilt angle allowed to add TILT profile to the calibration database. Value must be a float and must be less than qc_slit_max

  qc_slit_min = -8.0

  Used in: cal_SLIT_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_SLIT_spirou.main()
  Level: Public

- **Maximum angle allowed for slit TILT (qc_slit_max)**

  Defines the maximum tiult angle allowed to add TILT profile to the calibration database. Value must be a float and must be greater than qc_slit_min

  qc_slit_max = 0.0

  Used in: cal_SLIT_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_SLIT_spirou.main()
  Level: Public

- **Saturation point (qc_max_signal)**

  Defines the maximum signal allowed (when defining saturation limit). Currently this does not contribute to failing the quality test. Value must be a float greater than zero.

  qc_max_signal = 65500

  Used in: cal_extract_RAW_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_extract_RAW_spirou.main()
  Level: Public

  Note: Currently this does not stop the file from passing the quality control criteria, it either should fail or should be removed.

- **Maximum Littrow RMS (HC) (QC_HC_RMS_LITTROW_MAX)**

  Defines the maximum Littrow RMS value for cal_HC_E2DS_spirou.

  QC_HC_RMS_LITTROW_MAX = 0.1

  Used in: cal_HC_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_HC_E2DS_spirou.main()
  Level: Public

- **Maximum Littrow Deviation (HC) (QC_HC_DEV_LITTROW_MAX)**

  Defines the maximum Littrow deviation from the wave solution for cal_HC_E2DS_spirou.

  QC_HC_DEV_LITTROW_MAX = 0.4

  Used in: cal_HC_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_HC_E2DS_spirou.main()
  Level: Public

- **Maximum Littrow RMS (WAVE) (QC_WAVE_RMS_LITTROW_MAX)**

  Defines the maximum Littrow RMS value for cal_WAVE_E2DS_spirou

  QC_WAVE_RMS_LITTROW_MAX = 0.1

  Used in: cal_WAVE_E2DS_spirou
  Defined in: constants_SPIROU_H4RG.py
  Called in: cal_WAVE_E2DS_spirou.main()
  Level: Public

- **Maximum Littrow Deviation (WAVE) (QC_WAVE_DEV_LITTROW_MAX)**

  Defines the maximum Littrow deviation from the wave solution for cal_WAVE_E2DS_spirou.

  QC_WAVE_DEV_LITTROW_MAX = 0.3

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	cal_WAVE_E2DS_spirou.main()
Level:	Public

- **Instrumental drift, maximum No. order removed (QC_WAVE_IDRIFT_NBORDEROUT)**

  Defines the maximum number of orders allowed to be removed from the RV calculation in order to pass the quality control, for the instrumental drift calculation.

  QC_WAVE_IDRIFT_NBORDEROUT = 15

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouWAVE.`calculate_instrument_drift`
Level:	Public

- **Instrumental drift, maximum allowed RV (QC_WAVE_IDRIFT_RV_MAX)**

  Defines the maximum allowed RV drift (in m/s) allowed for the instrumental drift calculation.

  QC_WAVE_IDRIFT_RV_MAX = 150.0

Used in:	cal_WAVE_E2DS_spirou
Defined in:	constants_SPIROU_H4RG.py
Called in:	SpirouDRS.spirouTHORCA.spirouWAVE.`calculate_instrument_drift`
Level:	Public

10.21 Calibration database variables

- **The calibration database master filename (ic_calibDB_filename)**

  Defines the name of the master calibration database text file for use in all calibration database operation.

  ic_calibDB_filename = master_calib_SPIROU.txt

  Used in: All Recipes
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouConfig.spirouConst.CALIBDB_MASTERFILE()
  Level: Public

  Note: This should probably be private, unless we want the user to be able to change calibDB files.

- **Maximum wait time for locked calibration database (calib_max_wait)**

  Defines the maximum time the code waits for the calibration database when it is locked. A locked file is created every time the calibration database is open (and subsequently closed when reading of the database was successful). If a lock file is present the code will wait a maximum of this many seconds and keep checking whether the lock file has been removed. After which time the code will exit with an error. Value must be a positive float greater than zero. Measured in seconds.

  calib_max_wait = 3600

  Used in: All Recipes
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouCDB.spirouCDB.get_check_lock_file()
  Level: Public

- **Calibration database duplicate key handler (calib_db_match)**

  Defines the mechanism to use in deciding between duplicate keys in the calibration database file. Value must be a string and must be either 'older' or 'closest'. If 'older' the calibration database will only use keys that are older than the timestamp in the input fits file (first argument) using the key kw_ACQTIME_KEY

  calib_db_match = 'closest'

  Used in: All Recipes
  Defined in: constants_SPIROU_H4RG.py
  Called in: SpirouDRS.spirouCDB.spirouCDB.get_check_lock_file()
  Level: Public

10.22 Startup variables

- **Configuration Folder Path (config_folder)**

  Defines the location of the configuration directory relative to the module directory (defined in variable = 'package'). Value must be a string containing a valid directory location.

  config_folder = USER_DIR/config

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.CONFIGFOLDER()
  Called in: All Recipes
  Level: Private

- **Configuration file name (config_file)**

  Defines the main configuration (containing the data directories etc). Value must be a string containing a valid file name i.e. the main configuration file should be at TDATA/config_folder/config_file.

  config_file = USER_DIR/config/config.py

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.CONFIGFILE()
  Called in: All Recipes
  Level: Private

- **Constant data folder relative path (const_data_folder)**

  Defines the storage folder for data files that are used in the DRS. This included masks and lookup tables used by the DRS and not changed by the user. Value should be a string with a path that is relative to the DRS module folder (i.e. SpirouDRS) for example the path './data' would be located under the SpirouDRS folder.

  const_data_folder = './data'

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.CDATA_FOLDER
  Called in: All Recipes
  Level: Private

- **Filenames from run time arguments (arg_file_names)**

  Gets the filenames from run time arguments.

  arg_file_names

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.ARG_FILE_NAMES()
  Called in: SpirouDRS.spirouStartup.spirouStartup.run_time_args()

- **Night name from run time arguments (arg_night_name)**

Gets the night name from run time arguments.

arg_night_name

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouConst.ARG_NIGHT_NAME()
Called in: SpirouDRS.spirouStartup.spirouStartup.run_time_args()

- **Calibration database file path (masterfilepath)**

Gets the full calibration database file path

masterfilepath

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouConst.CALIBDB_MASTERFILE()
Called in: SpirouDRS.spirouCDB.spirouCDB.write_files_to_master(), SpirouDRS.spirouCDB.spirouCDB.read_master_file(), SpirouDRS.spirouImage.spirouImage.correct_for_dark()

- **Calibration database lock file path (lockfilepath)**

Gets the full calibration database lock file path

lockfilepath

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouConst.CALIBDB_LOCKFILE()
Called in: SpirouDRS.spirouCDB.spirouCDB.get_check_lock_file()

- **Calibration database file prefix (calib_prefix)**

Defines the prefix for calibration database files. Value must be a string.

calib_prefix

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouConst.CALIB_PREFIX(()
Called in: cal_DARK_spirou.main(), cal_loc_RAW_spirou.main(), cal_SLIT_spirou.main(), cal_FF_RAW_spirou.main()

- **Fits file name (fitsfilename)**

  Gets the full file path of the first file in 'arg_file_names'

  fitsfilename

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.FITSFILENAME()
  Called in: SpirouDRS.spirouStartup.spirouStartup.run_time_args()

- **Log program name (log_opt)**

  Chooses the display format for the program in the logging system.

  log_opt

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.LOG_OPT()
  Called in: SpirouDRS.spirouStartup.spirouStartup.run_time_args()

- **The Recipe name (recipe)**

  The definition of the recipe name (usually from __NAME__).

  recipe

  Used in: All Recipes
  Defined in: All Recipes
  Called in: All Recipes

- **Documentation info manual file path (manual_file)**

  Gets the full documentation info manual file path

  manual_file

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.MANUAL_FILE()
  Called in: SpirouDRS.spirouStartup.spirouStartup.display_help_file()

- **Number of frames (nbframes)**

  Gets the number of frames from the list of files ('arg_file_names').

  nbframes

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.NBFRAMES
  Called in: SpirouDRS.spirouStartup.spirouStartup.run_time_args()

- **Program name from run time (program)**

  Gets the run program name from run time.

  program

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .PROGRAM()
  Called in: SpirouDRS.spirouStartup.spirouStartup .run_time_args()

- **Full path of raw data directory (raw_dir)**

  Gets the full path of the raw data directory.

  raw_dir

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .RAW_DIR()
  Called in: SpirouDRS.spirouImage.spirouFITS .math_controller(),
  SpirouDRS.spirouImage.spirouImage .get_all_similar_files(),
  SpirouDRS.spirouStartup.spirouStartup .display_run_files()

- **Full path of reduced data directory (reduced_dir)**

  Gets the full path of the reduced data directory.

  reduced_dir

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .REDUCED_DIR()
  Called in: SpirouDRS.spirouStartup.spirouStartup .run_time_args()

10.23 Output file variables

- **The dark calibration filename (darkfile)**

  The full path of the processed dark calibration file

  darkfile

  Used in: cal_DARK_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst .DARK_FILE()
  Called in: cal_DARK_spirou

- **The dark bad pixel map calibration file (darkbadpixfile)**

  The full path of the processed dark bad pixel map calibration filename

  darkbadpixfile

  Used in: cal_DARK_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst .DARK_BADPIX_FILE()
  Called in: cal_DARK_spirou

- **The bad pixel map calibration filename (badpixfile)**

  The full path of the processed bad pixel map calibration file

  badpixfile

  Used in: cal_BADPIX_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst .BADPIX_FILE()
  Called in: cal_BADPIX_spirou

- **The order profile image filename (orderpfile)**

  The full path of the order profile image file.

  orderpfile

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst .LOC_ORDER_PROFILE_FILE()
  Called in: cal_loc_RAW_spirou

- **Localization filename 1 (locofits)**

  The full path of the processed localisation file containing the center fits for each order.

  locofits

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst .LOC_LOCO_FILE()
  Called in: cal_loc_RAW_spirou

- **Localization filename 2 (locofits2)**

  The full path of the processed localisation file containing the width fits for each order

  locofits2

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.LOC_LOCO_FILE2()
  Called in: cal_loc_RAW_spirou

- **Localization filename 3 (locofits3)**

  The full path of the fits super-imposed onto the original image file.

  locofits3

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.LOC_LOCO_FILE3()
  Called in: cal_loc_RAW_spirou

- **Tilt filename (tiltfits)**

  The full path of the processed tilt file.

  tiltfits

  Used in: cal_SLIT_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.SLIT_TILT_FILE()
  Called in: cal_SLIT_spirou

- **Blaze filename (blazefits)**

  The full path of the processed blaze file.

  blazefits

  Used in: cal_FF_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.FF_BLAZE_FILE()
  Called in: cal_FF_RAW_spirou

- **Flat filename (flatfits)**

  The full path of the processed flat file

  flatfits

  Used in: cal_FF_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.FF_FLAT_FILE()
  Called in: cal_FF_RAW_spirou

- **E2DS filename (e2dsfits)**

  The full path of the processed and extracted E2DS file

  e2dsfits

  Used in: cal_extract_RAW_spirou, cal_extract_RAW_spirouAB, cal_extract_RAW_spirouC()
  Defined in: SpirouDRS.spirouConfig.spirouConst.EXTRACT_E2DS_FILE
  Called in: cal_extract_RAW_spirou

- **The extraction filenames (exfitslist)**

  Defines the extraction names (and locations) for the extraction process for all types of E2DS file. i.e. the filenames for 'simple', 'tilt', 'tiltweight', 'tiltweight2' and 'weight'

  exfitslist

  Used in: cal_extract_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.EXTRACT_E2DS_ALL_FILES()
  Called in: cal_extract_RAW_spirou

- **The raw drift filename (driftfits_raw)**

  Defines the raw drift fits file name and location

  driftfits_raw

  Used in: cal_DRIFT_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.DRIFT_RAW_FILE()
  Called in: cal_DRIFT_RAW_spirou

- **The E2DS drift filename (driftfits_e2ds)**

  Defines the E2DS drift fits filename and location

  driftfits_e2ds

  Used in: cal_DRIFT_E2DS_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.DRIFT_E2DS_FITS_FILE()
  Called in: cal_DRIFT_E2DS_spirou

- **The E2DS drift table filename (drifttblfilename_e2ds)**

  Defines the E2DS drift table filename and location

  drifttblfilename_e2ds

  Used in: cal_DRIFT_E2DS_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.DRIFT_E2DS_TBL_FILE()
  Called in: cal_DRIFT_E2DS_spirou

- **The E2DS drift-peak fits filename (driftfits_peak_e2ds)**

  Defines the E2DS drift-peak fits filename and location

  driftfits_peak_e2ds

  Used in: cal_DRIFTPEAK_E2DS_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.DRIFTPEAK_E2DS_FITS_FILE
  Called in: cal_DRIFTPEAK_E2DS_spirou

- **The E2DS drift-peak table filename (drifttblfilename_peak_e2ds)**

  Defines the E2DS drift-peak table filename and location

  drifttblfilename_peak_e2ds

  Used in: cal_DRIFTPEAK_E2DS_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.DRIFTPEAK_E2DS_TBL_FILE
  Called in: cal_DRIFTPEAK_E2DS_spirou

- **CCF filename and location (corfile)**

  Defines the CCF fits filename and location

  corfile

  Used in: cal_CCF_E2DS_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.CCF_FITS_FILE
  Called in: cal_CCF_E2DS_spirou

- **CCF table filename and location (ccf_table_file)**

  Defines the CCF table file location and name

  ccf_table_file

  Used in: cal_CCF_E2DS_spirou
  Defined in: SpirouDRS.spirouConfig.spirouConst.CCF_TABLE_FILE
  Called in: cal_CCF_E2DS_spirou

10.24 Formatting variables

- **Header date format (date_fmt_header)**

  Defines the format of the date in the FITS rec header files

  date_fmt_header = %Y-%m-%d-%H:%M:%S.%f

  Used in: SpirouDRS.spirouCDB.spirouCDB
  Defined in: SpirouDRS.spirouConfig.spirouConst.DATE_FMT_HEADER()
  Called in: SpirouDRS.spirouCDB.spirouCDB.update_database(), SpirouDRS.spirouCDB.spirouCDB.get_database()
  Level: Private

- **Calibration database date format (date_fmt_calibdb)**

  Defines the format of the date in the calibration database file

  date_fmt_calibdb = %Y-%m-%d-%H:%M:%S.%f

  Used in: SpirouDRS.spirouCDB.spirouCDB
  Defined in: SpirouDRS.spirouConfig.spirouConst.DATE_FMT_CALIBDB()
  Called in: SpirouDRS.spirouCDB.spirouCDB.update_database(), SpirouDRS.spirouCDB.spirouCDB.get_database()
  Level: Private

10.25 FITS rec variables

- **Forbidden copy keys**

  Lists the keys that should not be copied when call to copy all FITS rec keys is made. Should be a list of python strings.

```
forbidden_keys = ['SIMPLE', 'BITPIX', 'NAXIS', 'NAXIS1', 'NAXIS2', 'EXTEND',
                  'COMMENT', 'CRVAL1', 'CRPIX1', 'CDELT1', 'CRVAL2', 'CRPIX2',
                  'CDELT2', 'BSCALE', 'BZERO', 'PHOT_IM', 'FRAC_OBJ', 'FRAC_SKY',
                  'FRAC_BB']
```

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .FORBIDDEN_COPY_KEYS()
  Called in: SpirouDRS.spirouImage.spirouFITS
  Level: Private

10.26 Logging and printing variables

- **Print message level (PRINT_LEVEL)**

The level of messages to print, values can be as follows:

- "all" – prints all events
- "info" – prints info, warning and error events
- "warning" – prints warning and error events
- "error" – print only error events

Value must be a valid string. See section 7.2.2 for more details.

PRINT_LEVEL = all

Used in:	All Recipes
Defined in:	USER_DIR/config/config.py
Called in:	SpirouDRS.spirouConfig.spirouConfig.check_params()
Level:	Public

- **Log message level (LOG_LEVEL)**

The level of messages to print, values can be as follows:

- "all" – prints all events
- "info" – prints info, warning and error events
- "warning" – prints warning and error events
- "error" – print only error events

Value must be a valid string. See section 7.2.2 for more details.

LOG_LEVEL = all

Used in:	All Recipes
Defined in:	USER_DIR/config/config.py
Called in:	SpirouDRS.spirouConfig.spirouConfig.writelog()
Level:	Public

- **Logging keys (trig_key)**

Defines the logging keys to use for each logging levels. Value should be a dictionary of key value pairs (where all keys and values are strings). When using the SpirouDRS.spirouCore .spirouLog .logger() (aliases to WLOG in recipes) the first argument must be one of these keys and the returned string is the corresponding value. The keys of write_level and trig_key must be identical. See section 7.2.2 for more details.

trig_key = `dict(all=' ', error='!', warning='@', info='*', graph='~')`

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouConst .LOG_TRIG_KEYS()
Called in: All Recipes
Level: Private

i.e.:

Python/Ipython

```
trig_key = dict(error='!')
WLOG('error', 'program', 'Message')
```

returns

```
HH:MM:SS.s - ! |program|Message
```

- **Write level (write_level)**

Defines the write levels to use for each write level. A write level is defined by a number. The higher the number to more exclusive the level i.e. if A and B are write levels and A > B and write level is set to A, any log or print messages at level B will not be logged/printed. Printing is controlled by variable PRINT_LEVEL and logging by variable LOG_LEVEL. The keys of write_level and trig_key must be identical. See section 7.2.2 for more details.

write_level = `dict(error=3, warning=2, info=1, graph=0, all=0)`

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouConst.LOG_TRIG_KEYS()
Called in: All Recipes
Level: Private

i.e.:

Python/Ipython

```
write_level = dict(error=3, warning=2, info=1)
trig_key = dict(all=' ', error='!', warning='@', info='*', graph='~')
PRINT_LEVEL = 'warning'

WLOG('error', 'program', 'Error message')
WLOG('warning', 'program', 'Warning message')
WLOG('info', 'program', 'Info message')
```

returns

```
HH:MM:SS.s - ! |program|Error message
HH:MM:SS.s - @ |program|Warning message
```

Note: Note the info message was not shown as info=1 and PRINT_LEVEL is set to warning=2.

- **Logger exit type (log_exit_type)**

What to do when a logging 'error' is raise. Options are: 'None', 'os' or 'sys'. If 'None' the code continues on an 'error', if 'os' then python executes a 'os._exit' command (a hard exit), if 'sys' then python executes a 'sys.exit' command (a soft exit).

log_exit_type = sys

Used in: All Recipes
Defined in: SpirouDRS.spirouConfig.spirouConst.LOG_EXIT_TYPE()
Called in: SpirouDRS.spirouConfig.spirouConst.EXIT() which is called in SpirouDRS.spirouCore.spirouLog
Level: Private

- **Exit controller (exit)**

  Controls the exit type from 'log_exit_type' and SpirouDRS.spirouConfig.spirouConst.LOG_EXIT_TYPE().

  exit

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.EXIT()
  Called in: SpirouDRS.spirouCore.spirouLog

- **Exit levels (exit_levels)**

  Controls which levels (defined in write_level and trig_key) will lead to the exit statement (given in exit and log_exit_type). Values must be a list of strings where each entry must be in write_level and trig_key.

  exit_levels

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.EXIT_LEVELS()
  Called in: SpirouDRS.spirouCore.spirouLog

- **Log caught warnings (log_caught_warnings)**

  If True or 1, then if warnings are passed to SpirouDRS.spirouCore.spirouLog.warninglogger() and there are warnings present, will attempt to log these warnings using the SpirouDRS.spirouCore.spirouLog.logger function. i.e. will print the warning to screen/log file depending on logging settings.

  log_caught_warnings = True

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.LOG_CAUGHT_WARNINGS()
  Called in: SpirouDRS.spirouCore.spirouLog
  Level: Private

- **Configuration key error message (cerrmsg)**

  Defines the message that is used when a configuration key is missing

  cerrmsg

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.CONFIG_KEY_ERROR
  Called in: SpirouDRS.spirouCore.spirouLog.get_logfilepath()

- **Colour of levels text (clevels)**

  The text colour for each level in trig_key and write_level. Value must be a dictionary with the keys identical to the keys in trig_key and write_level. One can use REDCOLOUR(), YELLOWCOLOUR(), GREENCOLOUR() to access the predefined values of red, yellow and green respectively. The default colour is given by NORMALCOLOUR().

  clevels = `dict(error=red, warning=yellow, info=green, graph=norm, all=green)`

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.COLOUREDLEVELS()
  Called in: SpirouDRS.spirouCore.spirouLog.debug_start(), SpirouDRS.spirouCore.spirouLog.printlog()
  Level:

- **Default text colour (norm)**

  Defines the string that describes the default text colour (retrieves colour from user). This in turn is turned into the colour defined by the python console/terminal that is default for that user. Value must be a string. This is used at the end of any colour change to set the text colour back to default (otherwise colour will remain until changed).

  norm = `"\033[0;37;40m"`

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.NORMALCOLOUR()
  Called in: SpirouDRS.spirouConfig.spirouConst.COLOUREDLEVELS(), SpirouDRS.spirouCore.spirouLog.debug_start(), SpirouDRS.spirouCore.spirouLog.printlog()
  Level:

- **Red text colour (red)**

  Defines the string that describes the colour "red".

  red = `"\033[0;31;48m"`

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.REDCOLOUR()
  Called in: SpirouDRS.spirouConfig.spirouConst.COLOUREDLEVELS()
  Level:

- **Yellow text colour (yellow)**

  Defines the string that describes the colour "yellow".

  yellow = `"\033[0;33;48m"`

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst.YELLOWCOLOUR()
  Called in: SpirouDRS.spirouConfig.spirouConst.COLOUREDLEVELS()
  Level:

- **Green text colour (green)**

  Defines the string that describes the colour "green".

  green = "\033[0;32;48m"

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .GREENCOLOUR()
  Called in: SpirouDRS.spirouConfig.spirouConst .COLOUREDLEVELS()
  Level:

- **Toggle coloured log (COLOURED_LOG)**

  Defines whether the log (printed to the standard output) is coloured according to clevels . Value must be True or 1 to colour the log or False or 0 to use the default console colour throughout.

  COLOURED_LOG = True

  Used in: All Recipes
  Defined in: USER_DIR /config /config.py
  Called in: SpirouDRS.spirouConfig.spirouConst .COLOURED_LOG()
  Level: Public

- **Coloured log controller (clog)**

  Contoller for coloured log (value is set from COLOURED_LOG)

  clog

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouConst .COLOURED_LOG()
  Called in: SpirouDRS.spirouCore.spirouLog .debug_start(), SpirouDRS .spirouCore.spirouLog .printlog()

Chapter 11

Output header keywords

Keywords are defined as a list of three strings, the first key is the HEADER key, the second is the HEADER value and the last is the HEADER comment i.e.

Python/Ipython

```
kw_KEYWORD = [key, value, comment]
```

and in a FITS rec would product the following:

```
key     = value              / comment
```

To better understand the keywords in the DRS we have laid out each keyword in the following way:

- **Keyword title (kw_variable)**

  Description of the keyword

  kw_variable = ["HEADER key", "Default HEADER value", "HEADER comment"]

  HEADER file entry:

  HEADER key = Default HEADER value \ HEADER comment

  Used in: The recipe the keyword is used in
  Defined in: The place where the keyword is defined
  Called in: The code where the keyword is used.

Note: All keywords are (and should be) loaded into the main parameter dictionary 'p' in all recipes and thus are accessed via:

Python/Ipython

```
variable = p["kw_variable"]
```

11.1 Global keywords

- **DRS Version (kw_version)**

  The current name and version of the DRS

  kw_version = ["VERSION", "DRS_NAME_DRS_VERSION", "DRS version"]

  HEADER file entry:

  VERSION = DRS_NAME_DRS_VERSION \ DRS version

  Used in: All Recipes
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords

- **Root for localization keywords (kw_root_drs_loc)**

  The root (prefix) for localization keywords

  kw_root_drs_loc = LO

  Used in: cal_extract_RAW_spirou, SpirouDRS.spirouConfig.spirouKeywords
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_extract_RAW_spirou.main(), SpirouDRS.spirouConfig.spirouKeywords
  Level: Private

- **Root for flat field keywords (kw_root_drs_flat)**

  The root (prefix) for flat field keywords

  kw_root_drs_flat = FF

  Used in: SpirouDRS.spirouConfig.spirouKeywords
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords
  Level: Private

- **Root for HC keywords (kw_root_drs_hc)**

  The root (prefix) for the HC keywords

  kw_root_drs_hc = LMP

  Used in: SpirouDRS.spirouConfig.spirouKeywords
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: SpirouDRS.spirouConfig.spirouKeywords
  Level: Private

> **Note:** Not currently used

- **File Type (kw_DPRTYPE)**

  The data fits file type

  kw_DPRTYPE = ["DPRTYPE", "0", "The type of file (from pre-process)"]

  HEADER file entry:

  ```
  DPRTYPE  =  0  \  The type of file (from pre-process)
  ```

  Used in: cal_DARK_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_DARK_spirou .main()

11.2 Dark calibration keywords

- **Fraction of dead pixels (kw_DARK_DEAD)**

  Percentage of dead pixels on image

  kw_DARK_DEAD = ["DADEAD", "0", "Fraction dead pixels [%]"]

  HEADER file entry:

  ```
  DADEAD  =  0  \  Fraction dead pixels [%]
  ```

  Used in: cal_DARK_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_DARK_spirou.main()

- **Median dark level (kw_DARK_MED)**

  Median dark level of the image in ADU/s

  kw_DARK_MED = ["DAMED", "0", "median dark level [ADU/s]"]

  HEADER file entry:

  ```
  DAMED  =  0  \  median dark level [ADU/s]
  ```

  Used in: cal_DARK_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_DARK_spirou.main()

- **Fraction of dead pixels (blue part) (kw_DARK_B_DEAD)**

  Percentage of dead pixels on image on the blue part of the image

  kw_DARK_B_DEAD = ["DABDEAD", "0", "Fraction dead pixels blue part [%]"]

  HEADER file entry:

  ```
  DABDEAD  =  0  \  Fraction dead pixels blue part [%]
  ```

  Used in: cal_DARK_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_DARK_spirou.main()

- **Median dark level (blue part) (kw_DARK_B_MED)**

Median dark level of the image in ADU/s on the blue part of the image

kw_DARK_B_MED = ["DABMED", "0", "median dark level blue part [ADU/s]"]

HEADER file entry:

```
DABMED  =  0  \  median dark level blue part [ADU/s]
```

Used in: cal_DARK_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_DARK_spirou.main()

- **Fraction of dead pixels (red part) (kw_DARK_R_DEAD)**

Percentage of dead pixels on image on the red part of the image

kw_DARK_R_DEAD = ["DARDEAD", "0", "Fraction dead pixels red part [%]"]

HEADER file entry:

```
DARDEAD  =  0  \  Fraction dead pixels red part [%]
```

Used in: cal_DARK_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_DARK_spirou.main()

- **Median dark level (red part) (kw_DARK_R_MED)**

Median dark level of the image in ADU/s on the red part of the image

kw_DARK_R_MED = ["DARMED", "0", "median dark level red part [ADU/s]"]

HEADER file entry:

```
DARMED  =  0  \  median dark level red part [ADU/s]
```

Used in: cal_DARK_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_DARK_spirou.main()

- **Dark level threshold (kw_DARK_CUT)**

  The dark level threshold in ADU/s

  kw_DARK_CUT = ["DACUT", "dark_cutlimit", "Threshold of dark level retain [ADU/s]"]

  HEADER file entry:

  ```
  DACUT    =    dark_cutlimit    \    Threshold of dark level retain [ADU/s]
  ```

  Used in: cal_DARK_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_DARK_spirou.main()

11.3 Localization calibration keywords

- **Mean packground (kw_LOCO_BCKGRD)**

  The mean background of an image (as a percentage).

  kw_LOCO_BCKGRD = ["kw_root_drs_loc BCKGRD", "0", "mean background [%]"]

  HEADER file entry:

  ```
  kw_root_drs_loc BCKGRD   =   0   \   mean background [%]
  ```

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_loc_RAW_spirou.main()

- **Image conversion factor (kw_CCD_CONAD)**

  Image conversion factor [e-/ADU]

  kw_CCD_CONAD = ["CONAD", "0", "CCD conv factor [e-/ADU]"]

  HEADER file entry:

  ```
  CONAD      =   0   \   CCD conv factor [e-/ADU]
  ```

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_loc_RAW_spirou.main()

  Note: Currently not set

- **CCD Readout Noise (kw_CCD_SIGDET)**

  The image readout noise in e-

  kw_CCD_SIGDET = ["SIGDET", "0", "CCD Readout Noise [e-]"]

  HEADER file entry:

  ```
  SIGDET     =   0   \   CCD Readout Noise [e-]
  ```

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_loc_RAW_spirou.main()

- **Coefficients position fits for orders (kw_LOCO_CTR_COEFF)**

The coefficients of the position fits

kw_LOCO_CTR_COEFF = ["kw_root_drs_loc CTR", "0", "'Coeff center'"]

HEADER file entry:

```
kw_root_drs_loc CTR    =    0    \    'Coeff center'
```

Used in: cal_loc_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_loc_RAW_spirou.main()

- **Degree of the fitting polynomial for localization position (kw_LOCO_DEG_C)**

The fit degree used in the position fit during localization

kw_LOCO_DEG_C = ["kw_root_drs_loc DEGCTR", "ic_locdfitc", "degree fit ctr ord"]

HEADER file entry:

```
kw_root_drs_loc DEGCTR    =    ic_locdfitc    \    degree fit ctr ord
```

Used in: cal_loc_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_loc_RAW_spirou.main()

- **Degree of the fitting polynomial for localization width (kw_LOCO_DEG_W)**

The fit degree used in the width fit during localization

kw_LOCO_DEG_W = ["kw_root_drs_loc DEGFWH", "0", "degree fit width ord"]

HEADER file entry:

```
kw_root_drs_loc DEGFWH    =    0    \    degree fit width ord
```

Used in: cal_loc_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_loc_RAW_spirou.main()

- **Degree of the fitting polynomial for localization position error (kw_LOCO_DEG_E)**

The fit degree used in the position error fit during localization

kw_LOCO_DEG_E = ["kw_root_drs_loc DEGERR", "0", "degree fit profile error"]

HEADER file entry:

```
kw_root_drs_loc DEGERR   =   0   \   degree fit profile error
```

Used in: cal_loc_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_loc_RAW_spirou.main()

Note: Currently not set

- **Delta width 3 convolve shape model (kw_LOCO_DELTA)**

The delta width used in pixels for the 3 convolve shape model

kw_LOCO_DELTA = ["kw_root_drs_loc PRODEL", "IC_LOC_DELTA_WIDTH", "param model 3gau"]

HEADER file entry:

```
kw_root_drs_loc PRODEL   =   IC_LOC_DELTA_WIDTH   \   param model 3gau
```

Used in: cal_loc_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_loc_RAW_spirou.main()

- **Coefficients width fits for orders (kw_LOCO_FWHM_COEFF)**

The coefficients of the width fits

kw_LOCO_FWHM_COEFF = ["kw_root_drs_loc FW", "0", "'Coeff fwhm'"]

HEADER file entry:

```
kw_root_drs_loc FW   =   0   \   'Coeff fwhm'
```

Used in: cal_loc_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_loc_RAW_spirou.main()

- **Number of orders localized (kw_LOCO_NBO)**

The number of orders obtained during localization

kw_LOCO_NBO = ["kw_root_drs_loc NBO", "0", "nb orders localized"]

HEADER file entry:

```
kw_root_drs_loc NBO    =    0    \    nb orders localized
```

Used in: cal_loc_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_loc_RAW_spirou.main()

- **Max image flux (kw_LOC_MAXFLX)**

The maximum flux in the image in ADU

kw_LOC_MAXFLX = ["kw_root_drs_loc FLXMAX", "0", "max flux in order [ADU]"]

HEADER file entry:

```
kw_root_drs_loc FLXMAX    =    0    \    max flux in order [ADU]
```

Used in: cal_loc_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_loc_RAW_spirou.main()

- **Max removed points - position fit (kw_LOC_SMAXPTS_CTR)**

Maximum number of removed points allowed for location fit

kw_LOC_SMAXPTS_CTR = ["kw_root_drs_loc CTRMAX", "0", "max rm pts ctr"]

HEADER file entry:

```
kw_root_drs_loc CTRMAX    =    0    \    max rm pts ctr
```

Used in: cal_loc_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_loc_RAW_spirou.main()

- **Max removed points - width fit (kw_LOC_SMAXPTS_WID)**

  Maximum number of removed points allowed for width fit

  kw_LOC_SMAXPTS_WID = ["kw_root_drs_loc WIDMAX", "0", "max rm pts width"]

  HEADER file entry:

  ```
  kw_root_drs_loc WIDMAX    =    0    \    max rm pts width
  ```

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_loc_RAW_spirou.main()

  Note: Formally this was called 'kw_LOC_Smaxpts_width'

- **Maximum RMS position fit (kw_LOC_RMS_CTR)**

  Maximum rms allowed for location fit

  kw_LOC_RMS_CTR = ["kw_root_drs_loc RMSCTR", "0", "max rms ctr"]

  HEADER file entry:

  ```
  kw_root_drs_loc RMSCTR    =    0    \    max rms ctr
  ```

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_loc_RAW_spirou.main()

- **Maximum RMS width fit (kw_LOC_RMS_WID)**

  Maximum rms allowed for width fit

  kw_LOC_RMS_WID = ["kw_root_drs_loc RMSWID", "0", "max rms width"]

  HEADER file entry:

  ```
  kw_root_drs_loc RMSWID    =    0    \    max rms width
  ```

  Used in: cal_loc_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_loc_RAW_spirou.main()

  Note: Formally this was called 'kw_LOC_rms_fwhm'

11.4 Slit calibration keywords

- **Tilt order prefix (kw_TILT)**

Tilt order keyword prefix

kw_TILT = ["kw_root_drs_loc TILT", "0", "Tilt order"]

HEADER file entry:

```
kw_root_drs_loc TILT    =    0    \    Tilt order
```

Used in: cal_SLIT_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_SLIT_spirou.main()

11.5 Flat fielding calibration keywords

- **SNR (kw_EXTRA_SN)**

Signal to noise ratio for order centre

kw_EXTRA_SN = ["EXTSN", "0", "S_N order center"]

HEADER file entry:

```
EXTSN       =    0    \    S_N order center
```

Used in: cal_FF_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_FF_RAW_spirou.main()

- **Flat field RMS (kw_FLAT_RMS)**

Flat field RMS for order

kw_FLAT_RMS = ["kw_root_drs_loc RMS", "0", "FF RMS order"]

HEADER file entry:

```
kw_root_drs_loc RMS    =    0    \    FF RMS order
```

Used in: cal_FF_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_FF_RAW_spirou.main()

11.6 Extraction calibration keywords

- **Localization filename (kw_LOCO_FILE)**

  localization file used in extraction process

  kw_LOCO_FILE = ["kw_root_drs_loc FILE", "0", "Localization file used"]

  HEADER file entry:

  ```
  kw_root_drs_loc FILE  =  0  \  Localization file used
  ```

  Used in: cal_extract_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_extract_RAW_spirou.main()

- **Extraction Method (kw_E2DS_EXTM)**

  Defines the extraction method used.

  kw_E2DS_EXTM = ["EXTMETH", "0", "Extraction method"]

  HEADER file entry:

  ```
  EXTMETH  =  0  \  Extraction method
  ```

  Used in: cal_extract_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_extract_RAW_spirou.main()

- **Extraction function (kw_E2DS_FUNC)**

  Defines the extraction function (in SpirouDRS.spirouEXTOR.spirouEXTOR)

  kw_E2DS_FUNC = ["EXTFUNC", "0", "Extraction function"]

  HEADER file entry:

  ```
  EXTFUNC  =  0  \  Extraction function
  ```

  Used in: cal_extract_RAW_spirou
  Defined in: SpirouDRS.spirouConfig.spirouKeywords
  Called in: cal_extract_RAW_spirou.main()

- **Extraction SNR (kw_E2DS_SNR)**

The extraction signal to noise ratio

kw_E2DS_SNR = ["SNR", "0", "Singal to Noise Ratio"]

HEADER file entry:

```
SNR     =    0    \    Singal to Noise Ratio
```

Used in: cal_extract_RAW_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_extract_RAW_spirou.main()

11.7 Bad pixel calibration keywords

- **Fraction of hot pixels (kw_BHOT)**

The Fraction of hot pixels on dark image (as a percentage)

kw_BHOT = ["BHOT", "0", "Frac of hot px [%]"]

HEADER file entry:

```
BHOT    =    0    \    Frac of hot px [%]
```

Used in: cal_BADPIX_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_BADPIX_spirou.main()

- **Fraction of bad pixels from flat (kw_BBFLAT)**

The Fraction of bad pixels from flat image (as a percentage)

kw_BBFLAT = ["BBFLAT", "0", "Frac of bad px from flat [%]"]

HEADER file entry:

```
BBFLAT  =    0    \    Frac of bad px from flat [%]
```

Used in: cal_BADPIX_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_BADPIX_spirou.main()

- **Fraction of non-finite pixels from dark (kw_BNDARK)**

The Fraction of non-finite pixels from dark image (as a percentage)

kw_BNDARK = ["BNDARK", "0", "Frac of non-finite px in dark [%]"]

HEADER file entry:

```
BNDARK  =  0  \  Frac of non-finite px in dark [%]
```

Used in: cal_BADPIX_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_BADPIX_spirou.main()

- **Fraction of non-finite pixels from flat (kw_BNFLAT)**

The Fraction of non-finite pixels from flat image (as a percentage)

kw_BNFLAT = ["BNFLAT", "0", "Frac of non-finite px in flat [%]"]

HEADER file entry:

```
BNFLAT  =  0  \  Frac of non-finite px in flat [%]
```

Used in: cal_BADPIX_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_BADPIX_spirou.main()

- **Fraction of bad pixels (kw_BBAD)**

The Fraction of bad pixels conforming to all criteria (as a percentage)

kw_BBAD = ["BBAD", "0", "Frac of bad px with all criteria [%]"]

HEADER file entry:

```
BBAD  =  0  \  Frac of bad px with all criteria [%]
```

Used in: cal_BADPIX_spirou
Defined in: SpirouDRS.spirouConfig.spirouKeywords
Called in: cal_BADPIX_spirou.main()

Chapter 12

The Recipes

12.1 General

The recipes are designed to have a layout that minimizes repetition and looks familiar between recipes. Much of the functionality in the recipes is used in multiple recipes and thus appears in functional form as opposed to be redefined in each and every recipe.

Some of this functionality was explained in section 7.2, explicitly the following:

- the logging functionality – logging to both screen and file (Section 7.2.2).
- the parameter dictionary – specialized dictionary object to store key value pairs with a source attached to each, i.e. to keep track of where key value pairs are defined and changed (Section 7.2.4).
- the configuration error and exception class – a special error and exception handling class for dealing with the configuration files and parameters associated with them (Section 7.2.5).

In addition to these each recipe is defined with a function itself (the main() function), to enable calling of said recipe from inside other python scripts (i.e. for batch runs).

The rest of this section details the different parts of the recipes.

12.1.1 The setup procedures

The first functionality of any recipe main() function is to set-up the recipe for running. This is done in three main steps (recipes may or may not use all three steps).

1. SpirouDRS.spirouStartup.spirouStartup .Begin() – Loads the initial parameters from the main configuration file.
2. SpirouDRS.spirouStartup.spirouStartup .LoadArguments() – Loads parameters from run time arguments (in default configuration or custom argument configuration, see sections 12.1.1.1 and 12.1.1.2)

> **Note:** Required prefixes (such as 'dark_dark', 'fp_fp', 'flat_dark') can be set here to cause an exception if the file names provided to not have one or more of these prefixes (useful in controlling which files are allowed to be used in the recipe).

As mentioned above there are two ways to load arguments, the 'default' way or the 'custom' way. These are described in sections 12.1.1.1 and 12.1.1.2 below.

12.1.1.1 Standard recipes

The standard way of getting arguments from the user is as follows:

```
>> RECIPE_NAME.py FOLDER FILENAME1
```

with more files defined the following way:

```
>> RECIPE_NAME.py FOLDER FILENAME1 FILENAME2
```

These (using SpirouDRS.spirouStartup.spirouStartup.LoadArguments()) are loaded in to parameters. 'FOLDER' becomes arg_night_name, 'FILENAME1' or 'FILENAME1 FILENAME2' become a python list accessed via arg_file_names with the first file name also being defined as fitsfilename. The 'RECIPE_NAME' is loaded into log_opt for use in the log.

An example standard load up can be seen in cal_DARK_spirou.

```
>> cal_DARK_spirou.py 20170710 dark_dark02d406.fits
```

Python/Ipython

```
def main(night_name=None, files=None):
    # ----------------------------------------------------------------------
    # Set up
    # ----------------------------------------------------------------------
    # get parameters from config files/run time args/load paths + calibdb
    p = spirouStartup.Begin()
    p = spirouStartup.LoadArguments(p, night_name, files)
    p = spirouStartup.InitialFileSetup(p, kind='dark', prefixes=['dark_dark'])
```

Note: Here 'night_name' and 'files' come from the main() definition (i.e. if called from python as a function we must have a way to get the arguments as they will not be defined at run time). As this is for cal_DARK_spirou the 'kind' of file is 'dark' (used in logging) and the prefixes allowed for dark files are 'dark_dark' only.

Note that after we have loaded the arguments we use SpirouDRS.spirouStartup.spirouStartup.InitialFileSetup(). This pushes values such as 'log_opt', 'fitsfilename', 'arg_night_name' into the main constant parameter dictionary 'p' and loads the calibration database, if present (using SpirouDRS.spirouStartup.spirouStartup.LoadCalibDB).

12.1.1.2 Recipes with custom arguments

For custom argument recipes the way of getting arguments from the user is as follows:

```
>> RECIPE_NAME.py FOLDER ARG1 ARG2 ARG3
```

In some cases the standard arguments are not sufficient for user input (i.e. for a certain recipe we may need more than just a list of file names). In this case the function SpirouDRS.spirouStartup.spirouStartup.LoadArguments() is used with keyword 'customargs' with a valid python dictionary of key names and their respective values. The helper function SpirouDRS.spirouStartup.spirouStartup.GetCustomFromRuntime() can be used to construct this dictionary accessing variables from the run time.

An example can be seen in cal_CCF_E2DS_spirou.

```
>> cal_CCF_E2DS_spirou.py 20170710 fp_fp02a203_e2ds_AB.fits UrNe.mas 0 10 0.1
```

Python/Ipython

```
def main(night_name=None, reffile=None, mask=None, rv=None, width=None,
         step=None):
    # ----------------------------------------------------------------------
    # Set up
    # ----------------------------------------------------------------------
    # get parameters from config files/run time args/load paths + calibdb
    p = spirouStartup.Begin()

    # deal with arguments being None (i.e. get from sys.argv)
    pos = [0, 1, 2, 3, 4]
    fmt = [str, str, float, float, float]
    name = ['reffile', 'ccf_mask', 'target_rv', 'ccf_width', 'ccf_step']
    lname = ['input_file', 'CCF_mask', 'RV', 'CCF_width', 'CCF_step']
    req = [True, True, True, False, False]
    call = [reffile, mask, rv, width, step]
    call_priority = [True, True, True, True, True]
    # now get custom arguments
    customargs = spirouStartup.GetCustomFromRuntime(pos, fmt, name, req, call,
                                                    call_priority, lname)

    # get parameters from configuration files and run time arguments
    p = spirouStartup.LoadArguments(p, night_name, customargs=customargs)
    # as we have custom arguments need to load the calibration database
    p = spirouStartup.LoadCalibDB(p)
```

Note: Here cal_CCF_E2DS_spirou requires the custom arguments 'reffile', 'ccf_mask', 'target_rv', 'ccf_width' and 'ccf_step'. These must be defined in main() and must be defined in the list 'call'.
The other parameters required by SpirouDRS.spirouStartup.spirouStartup.GetCustomFromRuntime() are:

- 'pos' – the position expected in the run time arguments (after the folder name)
- 'fmt' – the format expected from an argument (i.e. string or float, or integer)
- 'name' – the name in the parameter dictionary for each argument
- 'lname' – the log name (the name the user will see in the log if there is an error)
- 'req' – whether the argument is required (True) or optional (False)
- 'call' – the name from main()
- 'call_priority' – whether arguments from main() overrides values from run time (most the time this will be True for use from python functions).

This is then fed into SpirouDRS.spirouStartup.spirouStartup.`LoadArguments` with the output of SpirouDRS.spirouStartup.spirouStartup.GetCustomFromRuntime() (i.e. customargs) passed to SpirouDRS.spirouStartup.spirouStartup.`LoadArguments`. As we do not use SpirouDRS.spirouStartup

.spirouStartup .InitialFileSetup() with custom arguments the calibration database has to be loaded separately, this is done using SpirouDRS.spirouStartup.spirouStartup .LoadCalibDB and is the last step in the set-up for recipes with custom arguments.

12.1.1.3 Summary of set-up

There are two ways to set-up a recipe, the first is with standard arguments, i.e.:

1. SpirouDRS.spirouStartup.spirouStartup .Begin()
2. SpirouDRS.spirouStartup.spirouStartup .LoadArguments()
3. SpirouDRS.spirouStartup.spirouStartup .InitialFileSetup()

Python/Ipython

```
def main(night_name=None, files=None):
    # ----------------------------------------------------------------------
    # Set up
    # ----------------------------------------------------------------------
    # get parameters from config files/run time args/load paths + calibdb
    p = spirouStartup.Begin()
    # load standard arguments from run time
    p = spirouStartup.LoadArguments(p, night_name, files)
    # set files
    p = spirouStartup.InitialFileSetup(p, kind='dark', prefixes=['dark_dark'])
    # ----------------------------------------------------------------------
```

the second is with custom arguments, i.e.:

1. SpirouDRS.spirouStartup.spirouStartup .Begin()
2. SpirouDRS.spirouStartup.spirouStartup .GetCustomFromRuntime
3. SpirouDRS.spirouStartup.spirouStartup .LoadArguments()
4. SpirouDRS.spirouStartup.spirouStartup .LoadCalibDB()

Python/Ipython

```
def main(night_name=None, arg1=None, arg2=None):
    # ------------------------------------------------------------------
    # Set up
    # ------------------------------------------------------------------
    # get parameters from config files/run time args/load paths + calibdb
    p = spirouStartup.Begin()
    # get custom arguments (from run time and function call)
    if hcfiles is None or fpfile is None:
        names, types = ['arg1', 'arg2'], [str, str]
        customargs = spirouStartup.GetCustomFromRuntime([0, 1], types, names,
                                                        last_multi=True)
    else:
        customargs = dict(arg1=arg1, arg2=arg2)
    # get parameters from configuration files and run time arguments
    p = spirouStartup.LoadArguments(p, night_name, customargs=customargs)

    # as we have custom arguments need to load the calibration database
    p = spirouStartup.LoadCalibDB(p)
    # ------------------------------------------------------------------
```

12.1.2 Main recipe code

After the setup proceedure the main code is run. Most heavy lifting should be done in functions and for ease of the reader/developer the main code should be kept to one line codes calling functions from the DRS python module. Many codes are reused throughout the drs a few of them are listed below:

- SpirouDRS.spirouImage.spirouImage.ReadImageAndCombine – Loads fitsfilename image and header (and if framemath define combines with all other files in arg_file_names).
- SpirouDRS.spirouImage.spirouImage.GetSigdet – gets the read noise value from the fitsfilename header
- SpirouDRS.spirouImage.spirouImage.GetExpTime – gets the exposure time from the fitsfilename header
- SpirouDRS.spirouImage.spirouImage.GetGain – gets the gain from the fitsfilename header.
- SpirouDRS.spirouImage.spirouImage.CorrectForDark – Loads the dark from the calibration database and applies it to the 'data' keyword
- SpirouDRS.spirouImage.spirouImage.ConvertToE – Converts image from ADU/s into e- using the exposure time and the gain
- SpirouDRS.spirouImage.spirouImage.FlipImage – flips the image in one or both of the dimensions (using the 'flipx' and 'flipy' keywords)
- SpirouDRS.spirouImage.spirouImage.ResizeImage – Resizes the image based on 'xlow', 'xhigh', 'ylow' and 'yhigh' keywords

12.1.3 Writing to file

Files are written to disk using the SpirouDRS.spirouImage.spirouImage.WriteImage() function. This requires a file name (python string), a image file (the data), and a header dictionary. Most filenames are defined in SpirouDRS.spirouConfig.spirouConfig.Constants (see Section 10.23). The header dictionary can be taken straight from the raw fitsfilename header (the output of SpirouDRS.spirouImage.spirouImage.ReadImageAndCombine for example), but key can be added using the following commands:

- SpirouDRS.spirouImage.spirouImage.CopyOriginalKeys – copies the original keys from the fitsfilename except those keys in forbidden keys.
- SpirouDRS.spirouImage.spirouImage.AddKey – adds a single key to the header (using the header keyword list parameters, see Section 11) and a value defined by the user using the 'value' keyword, if not defined the default value will be used.
- SpirouDRS.spirouImage.spirouImage.AddKey2DList – adds a 2D list to the header (using the header keyword list parameters, see Section 11)

An example is shown below

Python/Ipython

```
# ------------------------------------------------------------------
# Save and record of image of localization with order centre and keywords
# ------------------------------------------------------------------
# log that we are saving localization file
wmsg = 'Saving FWHM information in file: {0}'
WLOG('', p['log_opt'], wmsg.format(locofits2name))

# add keys from original header file
hdict = spirouImage.CopyOriginalKeys(hdr, cdr)

# define new keys to add
hdict = spirouImage.AddKey(hdict, p['kw_version'])
hdict = spirouImage.AddKey(hdict, p['kw_CCD_SIGDET'])
hdict = spirouImage.AddKey(hdict, p['kw_LOCO_NBO'], value=rorder_num)

# write 2D list of position fit coefficients
hdict = spirouImage.AddKey2DList(hdict, p['kw_LOCO_CTR_COEFF'],
                                 values=loc['acc'][0:rorder_num])

# add quality control
hdict = spirouImage.AddKey(hdict, p['kw_drs_QC'], value=p['QC'])

# write image and add header keys (via hdict)
spirouImage.WriteImage(locofits2, width_fits, hdict)
```

Note: Here we add the original keys not in forbidden keys to the header dictionary 'hdict' and then add version, sigdet and the number of orders localized as single keys, add the localization centers as a 2D list and add the flag for whether the quality control was passed.

12.1.4 Quality control

Quality control parameters decide whether a file is written to the calibration database. They consist of a standard python if statement where the variable 'passed' must be set to False if a quality control criteria fails the processed file (i.e. this is done inside the if or an else statement). As well as this a message may be passed to the log (standard output/screen and the log file), this is done by appending to 'fail_msg' which is subsequently printed for all quality control criteria that fail the test.

An example is shown below

Python/Ipython

```
# ------------------------------------------------------------------------
# Quality control
# ------------------------------------------------------------------------
passed, fail_msg = True, []
# check that max number of points rejected in centre fit is below threshold
if np.sum(loc['max_rmpts_pos']) > p['QC_LOC_MAXLOCFIT_REMOVED_CTR']:
    fmsg = 'abnormal points rejection during ctr fit ({0} > {1})'
    fail_msg.append(fmsg.format(np.sum(loc['max_rmpts_pos']),
                                p['QC_LOC_MAXLOCFIT_REMOVED_CTR']))
    passed = False
# check that max number of points rejected in width fit is below threshold
if np.sum(loc['max_rmpts_wid']) > p['QC_LOC_MAXLOCFIT_REMOVED_WID']:
    fmsg = 'abnormal points rejection during width fit ({0} > {1})'
    fail_msg.append(fmsg.format(np.sum(loc['max_rmpts_wid']),
                                p['QC_LOC_MAXLOCFIT_REMOVED_WID']))
    passed = False
# finally log the failed messages and set QC = 1 if we pass the
# quality control QC = 0 if we fail quality control
if passed:
    WLOG('info', p['log_opt'], 'QUALITY CONTROL SUCCESSFUL - Well Done -')
    p['QC'] = 1
    p.set_source('QC', __NAME__ + '/main()')
else:
    for farg in fail_msg:
        wmsg = 'QUALITY CONTROL FAILED: {0}'
        WLOG('info', p['log_opt'], wmsg.format(farg))
    p['QC'] = 0
    p.set_source('QC', __NAME__ + '/main()')
```

Note: Here we check that the maximum number of points rejected in centre fit is below a threshold and check that the maximum number of points rejected in the width fit is below a threshold if either of these fail then their 'fail_msg' is logged and printed, else a message saying 'quality control successful' is displayed.

12.1.5 Writing to the calibration database

The calibration database is automatically opened at the start of the recipes (see Section 12.1.1). Two commands are used to interface with the calibration database. The first SpirouDRS.spirouCDB .spirouCDB .PutFile() adds the file to the calibration database folder. The second (SpirouDRS.spirouCDB .spirouCDB .UpdateMaster) updates the ic_calibDB_filename with the correct key (set using the 'keys' keyword, e.g. 'DARK' or 'LOC_AB').

An example is shown below

Python/Ipython

```
# ------------------------------------------------------------------
# Update the calibration database
# ------------------------------------------------------------------
if p['QC'] == 1:
    keydb = 'LOC_' + p['fiber']
    # copy localisation file to the calibDB folder
    spirouCDB.PutFile(p, locofits)
    # update the master calib DB file with new key
    spirouCDB.UpdateMaster(p, keydb, locofitsname, hdr)
```

Note: Here we add, for example, key 'LOC_AB' or 'LOC_C' to the calibration database. The file is first put in the calibration database folder and then the key, file name and date/time are added to the ic_calibDB_filename. The date/time that is used is that of the fitsfilename.

12.1.6 End of code

After all the main section is completed, the code should end with the final log statement. This is followed by a returning of the local-scope variables (via the 'locals()' command), this allows the developer to have access to the local-scope of the functions on calling the function from another python script (this is used extensively in the unit test functions). For consistency this finishing message should not change and be present at the end of each recipe, thus on seeing this message the user and developer know that the recipe is finished.

If the user is running the recipe externally i.e.:

```
>> python recipe.py arg1 arg2 arg3
>> ipython recipe.py arg1 arg2 arg3
>> recipe.py arg1 arg2 arg3
```

then we must deal with plotting interactivity (for example plot should stay open until the user has had chance to interactive with them and any user may need/want use of interactiveness). This is all taken care of with the SpirouDRS.spirouStartup.spirouStartup `.Exit()` function, which should be part of any recipes `__main__` section.

An example is shown below

Python/Ipython

```
# -----------------------------------------------------------------------
# End Message
# -----------------------------------------------------------------------
wmsg = 'Recipe {0} has been successfully completed'
WLOG('info', p['log_opt'], wmsg.format(p['program']))
    # return a copy of locally defined variables in the memory
    return dict(locals())

# ==========================================================================
# Start of code
# ==========================================================================
if __name__ == "__main__":
    # run main with no arguments (get from command line - sys.argv)
    ll = main()
    # exit message
    spirouStartup.Exit(ll)

# ==========================================================================
# End of code
# ==========================================================================
```

12.2 The cal_DARK recipe

Dark with short exposure time (5min, to be defined during AT-4) to check if read-out noise, dark current and hot pixel mask are consistent with the ones obtained during technical night. Quality control is done automatically by the pipeline

12.2.1 The inputs

The input of cal_DARK_spirou is as follows:

```
>> cal_DARK_spirou.py night_repository filenames
```

for example:

example

```
>> cal_DARK_spirou.py 20170710 dark_dark02d406.fits
```

or

Python/Ipython

```
import cal_DARK_spirou
night_repository = '20170710'
filenames = ['dark_dark02d406.fits']
cal_DARK_spirou.main(night_repository, files=filenames)
```

where 'night_repository' defines arg_night_name and 'filenames' define the list of files in arg_file_names. All files in filenames must be valid python strings separated by a space (command line) or in a line (python).

Filename prefixes allowed are:

- dark_dark

12.2.2 The outputs

The outputs of cal_DARK_spirou are as follows:

- darkfile in form:
  {reduced_dir}/{date prefix}_{file}.fits
- darkbadpixfile in form:
  {reduced_dir}/{date prefix}_{file}_badpixel.fits

where 'date prefix' is constructed from arg_night_name and the file name is the first file in arg_file_names.

For example for reduced_dir='/drs/data/reduced/20170710' and arg_file_names=['dark_dark02d406.fits'] the output files would be:

- /drs/data/reduced/20170710/20170710_dark_dark02d406.fits
- /drs/data/reduced/20170710/20170710_dark_dark02d406_badpixel.fits

12.2.3 Summary of procedure

1. adds defined 'dark_dark' files together
2. resizes the image
3. calculates the fraction of dead pixels [full, blue part, red part]
4. calculates median dark level [full, blue part, red part]
5. calculates threshold of dark level to retain
6. removes dead pixels by setting them to 0
7. does some quality control
8. updates calibDB with key "DARK"

12.2.4 Quality Control

There are currently three quality control checks for cal_DARK_spirou

- Unexpected median dark level if:

$$\text{Median Flux} > \text{qc\_max\_darklevel} \tag{12.1}$$

- Unexpected fraction of dead pixels if:

$$\text{Number of dead pixels} > \text{qc\_max\_dead} \tag{12.2}$$

- Unexpected fraction of dark pixels if:

$$\text{Number of bad dark pixels} > \text{qc\_max\_dark} \tag{12.3}$$

If none of these quality control criteria are valid then the output file is passed into the calibration database with key 'DARK' for the 'darkfile' and 'BADPIX' for the 'darkbadpixfile'.

For example the following lines are added to the calibration database for arg_night_name = "20170710" and arg_file_names = "dark_dark02d406.fits" .

In calibration database file

```
DARK 20170710 20170710_dark_dark02d406.fits 2017-07-10-12:37:48.260000 1499690268.26
BADPIX 20170710 20170710_dark_dark02d406_badpixel.fits 2017-07-10-12:37:48.260000 1499690268.26
```

12.2.5 Example working run

An example run where everything worked is below:

example

```
>> cal_DARK_spirou.py 20170710 dark_dark02d406.fits
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (VERSION)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp/
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |ipython:2d406|Now running : ipython on file(s): dark_dark02d406.fits
HH:MM:SS.S -   |ipython:2d406|On directory /drs/data/raw/20170710
HH:MM:SS.S -   |ipython:2d406|ICDP_NAME loaded from: /drs/INTROOT/config/constants_SPIROU.py
HH:MM:SS.S - * |ipython:2d406|Correct type of image for dark (dark_dark)
HH:MM:SS.S - * |ipython:2d406|Now processing Image TYPE UNKNOWN with ipython recipe
HH:MM:SS.S -   |ipython:2d406|Reading Image /drs/data/raw/20170710/dark_dark02d406.fits
HH:MM:SS.S -   |ipython:2d406|Image 2048 x 2048 loaded
HH:MM:SS.S - * |ipython:2d406|Dark Time = 597.489 s
HH:MM:SS.S -   |ipython:2d406|Doing Dark measurement
HH:MM:SS.S - * |ipython:2d406|In Whole det: Frac dead pixels= 14.7 % - Median= 0.35 ADU/s - Percent
     [5:95]= 0.08-99.57 ADU/s
HH:MM:SS.S - * |ipython:2d406|In Blue part: Frac dead pixels= 1.0 % - Median= 0.15 ADU/s - Percent
     [5:95]= 0.09-0.53 ADU/s
HH:MM:SS.S - * |ipython:2d406|In Red part : Frac dead pixels= 20.5 % - Median= 2.11 ADU/s - Percent
     [5:95]= 0.18-232.09 ADU/s
HH:MM:SS.S - * |ipython:2d406|Frac pixels with DARK > 100.0 ADU/s = 4.3 %
HH:MM:SS.S - @ |python warning|Line 138 warning reads: invalid value encountered in greater
HH:MM:SS.S - * |ipython:2d406|Total Frac dead pixels (N.A.N) + DARK > 100.0 ADU/s = 18.9 %
HH:MM:SS.S - * |ipython:2d406|QUALITY CONTROL SUCCESSFUL - Well Done -
HH:MM:SS.S -   |ipython:2d406|Saving Dark frame in 20170710_dark_dark02d406.fits
HH:MM:SS.S - @ |python warning|Line 980 warning reads: Card is too long, comment will be truncated.
HH:MM:SS.S -   |ipython:2d406|Saving Bad Pixel Map in 20170710_dark_dark02d406_badpixel.fits
HH:MM:SS.S - @ |python warning|Line 980 warning reads: Card is too long, comment will be truncated.
HH:MM:SS.S - * |ipython:2d406|Updating Calib Data Base with DARK
HH:MM:SS.S - * |ipython:2d406|Updating Calib Data Base with BADPIX
HH:MM:SS.S - * |ipython:2d406|Recipe ipython has been succesfully completed
```

12.2.6 Interactive mode

In interactive mode (DRS_PLOT = 1) three figures will also appear (see Figure 12.1).

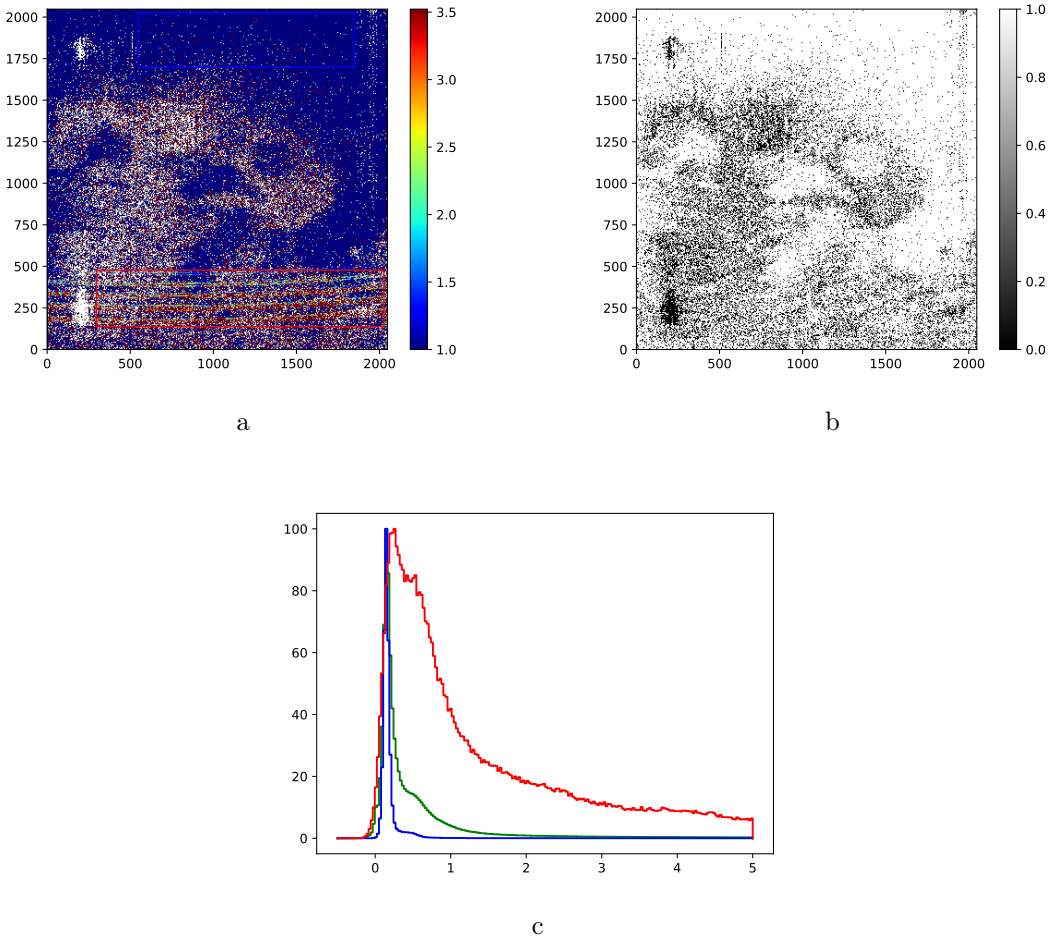


Figure 12.1 (a) The image with over-plot red and blue regions (red/blue rectangles). **(b)** The bad pixel mask, bad pixels have a value=1 (in black) and good pixels have a value=0 (in white). **(c)** Histograms of the image regions, the full image (in green), the blue section (in blue) and the red section (in red).

12.3 The cal_BADPIX recipe

Recipe to generate the bad pixel map.

12.3.1 The inputs

The input of cal_BADPIX_spirou is as follows:

```
>> cal_BADPIX_spirou.py night_repository flatfile darkfile
```

for example:

example

```
>> cal_BADPIX_spirou.py 20170710 flat_flat02f10.fits dark_dark02d406.fits
```

or

Python/Ipython

```
import cal_DARK_spirou
night_reposityory = '20170710'
darkfile = 'dark_dark02d406.fits'
flatfile = 'flat_flat02f10.fits'
cal_DARK_spirou.main(night_repository, flatfile=flatfile, darkfile=darkfile)
```

where 'night_repository' defines arg_night_name and 'filenames' define the list of files in arg_file_names. All files in filenames must be valid python strings separated by a space (command line) or in a line (python) and must have the folowing prefixes: File prefixes allowed:

- flat_flat (flatfile)
- dark_dark (darkfile)

12.3.2 The outputs

The outputs of badpixelfits are as follows:

- badpixelfits in form:

  {reduced_dir}/{date prefix}_{file}_badpixelfits.fits

where 'date prefix' is constructed from arg_night_name and the file name is the flatfile name.
for example for reduced_dir ='/drs/data/reduced/20170710' and flatfile='flat_flat02f10.fits' the output file would be:

- /drs/data/reduced/20170710/20170710_flat_flat02f10_badpixelfits.fits

12.3.3 Summary of procedure

1. Normalise the flats
2. Look for isolated hot pixels
3. Calculate how much pixels deviate compared to expected values

4. Select hot pixels compared to neighbours
5. Combine bad pixel map
6. Save bad pixel mask to file

12.3.4 Quality Control

There are no quality control parameters for cal_BADPIX_spirou.

The output file is passed into the calibration database with key 'BADPIX' for the 'badpixfile'.

For example the following lines are added to the calibration database for arg_night_name = "20170710", flatfile = "flat_flat02f10.fits" and darkfile = "dark_dark02d406.fits".

In calibration database file

```
BADPIX 20170710 20170710_flat_flat02f10_badpixel.fits 2017-07-10-13:07:49.470000 1499692069.47
```

12.3.5 Example working run

An example run where everything worked is below:

example

```
>> cal_BADPIX_spirou.py 20170710 flat_flat02f10.fits dark_dark02d406.fits
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (VERSION)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |cal_BADPIX_spirou|Now running : cal_BADPIX_spirou with:
HH:MM:SS.S -   |cal_BADPIX_spirou|       -- flatfile=flat_flat02f10.fits
HH:MM:SS.S -   |cal_BADPIX_spirou|       -- darkfile=dark_dark02d406.fits
HH:MM:SS.S -   |cal_BADPIX_spirou|Config Error: ICDP_NAME loaded from: /drs/spirou_py3/INTROOT/config/
    constants_SPIROU.py
HH:MM:SS.S - * |cal_BADPIX_spirou|Now processing Image TYPE FLAT with cal_BADPIX_spirou recipe
HH:MM:SS.S - * |cal_BADPIX_spirou|Now processing Image TYPE DARK with cal_BADPIX_spirou recipe
HH:MM:SS.S -   |cal_BADPIX_spirou|Reading FLAT Image /drs/data/raw/20170710/flat_flat02f10.fits
HH:MM:SS.S -   |cal_BADPIX_spirou|FLAT Image 2048 x 2048 loaded
HH:MM:SS.S -   |cal_BADPIX_spirou|Reading DARK Image /drs/data/raw/20170710/dark_dark02d406.fits
HH:MM:SS.S -   |cal_BADPIX_spirou|DARK Image 2048 x 2048 loaded
HH:MM:SS.S -   |cal_BADPIX_spirou|Normalising the flat
HH:MM:SS.S -   |cal_BADPIX_spirou|Looking for bad pixels
HH:MM:SS.S -   |cal_BADPIX_spirou|Fraction of hot pixels from dark: 3.01 %
HH:MM:SS.S -   |cal_BADPIX_spirou|Fraction of bad pixels from flat: 1.66 %
HH:MM:SS.S -   |cal_BADPIX_spirou|Fraction of non-finite pixels in dark: 20.76 %
HH:MM:SS.S -   |cal_BADPIX_spirou|Fraction of non-finite pixels in flat: 14.66 %
HH:MM:SS.S -   |cal_BADPIX_spirou|Fraction of bad pixels with all criteria: 24.87 %
HH:MM:SS.S - * |cal_BADPIX_spirou|QUALITY CONTROL SUCCESSFUL - Well Done -
HH:MM:SS.S -   |cal_BADPIX_spirou|Saving Bad Pixel Map in 20170710_flat_flat02f10_badpixel.fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S - * |cal_BADPIX_spirou|Updating Calib Data Base with BADPIX
HH:MM:SS.S - * |cal_BADPIX_spirou|Recipe cal_BADPIX_spirou has been successfully completed
```

12.4 The cal_loc recipe

Locates the orders on the 'dark_flat' or 'flat_dark' images.

12.4.1 The inputs

The input of cal_loc_RAW_spirou is as follows:

```
>> cal_loc_RAW_spirou.py night_repository filenames
```

for example:

example

```
>> cal_loc_RAW_spirou.py 20170710 flat_dark02f10.fits flat_dark03f10.fits flat_dark04f10.fits
    flat_dark05f10.fits flat_dark06f10.fits
```

or

Python/Ipython

```
import cal_loc_RAW_spirou
night_repository = '20170710'
filenames = ['flat_dark02f10.fits', 'flat_dark03f10.fits', 'flat_dark04f10.fits',
             'flat_dark05f10.fits', 'flat_dark06f10.fits']
cal_loc_RAW_spirou.main(night_repository, files=filenames)
```

where 'night_repository' defines arg_night_name and 'filenames' define the list of files in arg_file_names. All files in filenames must be valid python strings separated by a space (command line) or in a line (python) and must have the following prefixes: File prefixes allowed:

- dark_flat
- flat_dark

12.4.2 The outputs

The outputs of cal_loc_RAW_spirou are as follows:

- order_profile in form:
  > {reduced_dir}/{date prefix}_{file}_order_profile_{fiber}.fits
- locofitsfile in form:
  > {reduced_dir}/{date prefix}_{file}_loco_{fiber}.fits
- locofitsfile2 in form:
  > {reduced_dir}/{date prefix}_{file}_fwhm-order_{fiber}.fits
- locofitsfile3 in form:
  > {reduced_dir}/{date prefix}_{file}_with-order_{fiber}.fits

where 'date prefix' is constructed from arg_night_name and the file name is the first file in arg_file_names.

For example for reduced_dir='/drs/data/reduced/20170710' and arg_file_names=['flat_dark02f10.fits'] the output files would be:

- /drs/data/reduced/20170710/20170710_flat_dark02f10_order_profile_{fiber}.fits
- /drs/data/reduced/20170710/20170710_flat_dark02f10_loco_{fiber}.fits
- /drs/data/reduced/20170710/20170710_flat_dark02f10_fwhm-order_{fiber}.fits
- /drs/data/reduced/20170710/20170710_flat_dark02f10_with-order_{fiber}.fits

12.4.3 Summary of procedure

1. adds all defined 'dark_flat' or 'flat_dark' files together
2. corrects for darks
3. resizes the image
4. constructs 'order_profile' image
5. locates the central pixel of each order
6. steps out in large steps along the order (toward beginning and end)
7. fits the position of each order (using a small 2D box around each fit point)
   - includes a rejection of bad points (while loop)
8. fits the width of each order (using a small 2D box around each fit point)
   - includes a rejection of bad points (while loop)
9. saves the 'order_profile' image (with a superposition of the fit orders as zero values)
10. does some quality control
11. updates calibDB with keys "ORDER_PROFILE_{fiber}" "LOC_{fiber}" where {fiber} = [AB, C] etc

12.4.4 Quality Control

There are currently five quality control checks for cal_loc_RAW_spirou

- Too many rejected orders in centre position fit:

$$\text{Number of rejected orders in centre fit} > \texttt{qc\_loc\_maxlocfit\_removed\_ctr} \quad (12.4)$$

- Too many rejected orders in width fit:

$$\text{Number of rejected orders in width fit} > \texttt{qc\_loc\_maxlocfit\_removed\_wid} \quad (12.5)$$

- RMS on centre fit too high:

$$\text{Mean rms centre fit} > \texttt{qc\_loc\_rmsmax\_center} \quad (12.6)$$

- RMS on width fit too high:

$$\text{Mean rms width fit} > \texttt{qc\_loc\_rmsmax\_fwhm} \tag{12.7}$$

- Abnormal number of identified orders:

$$\text{Number of orders found} \neq \texttt{qc\_loc\_nbo} \tag{12.8}$$

If none of these quality control criteria are valid then the output file is passed into the calibration database with keys 'ORDER_PROFILE_{fiber}' for the 'order_profile' file and 'LOC_{fiber}' for the 'locofitsname' file.

For example the following lines are added to the calibration database for arg_night_name = "20170710" and arg_file_names = ["flat_dark02f10.fits"] .

In calibration database file

```
ORDER_PROFILE_AB 20170710 20170710_flat_dark02f10_order_profile_AB.fits 2017-07-10-13:04:34.440000
    1499691874.44
LOC_AB 20170710 20170710_flat_dark02f10_loco_AB.fits 2017-07-10-13:04:34.440000 1499691874.44
```

12.4.5 Example working run

An example run where everything worked is below:

example

```
>> cal_loc_RAW_spirou.py 20170710 flat_dark02f10.fits flat_dark03f10.fits flat_dark04f10.fits
    flat_dark05f10.fits flat_dark06f10.fits
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (VERSION)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |cal_loc_RAW_spirou|Now running : ipython on file(s):
HH:MM:SS.S -   |cal_loc_RAW_spirou|On directory /drs/data/raw/20170710
HH:MM:SS.S -   |cal_loc_RAW_spirou|ICDP_NAME loaded from: /drs/spirou_py3/INTROOT/config/
    constants_SPIROU.py
HH:MM:SS.S - * |cal_loc_RAW_spirou|Correct type of image for localisation (dark_flat or flat_dark)
HH:MM:SS.S -   |cal_loc_RAW_spirou|Calibration file: 20170710_flat_flat02f10_badpixel.fits already
    exists - not copied
HH:MM:SS.S -   |cal_loc_RAW_spirou|Calibration file: 20170710_flat_dark02f10_blaze_AB.fits already
    exists - not copied
```

```
...
HH:MM:SS.S -   |cal_loc_RAW_spirou|Calibration file: spirou_wave_ini3.fits already exists - not copied
HH:MM:SS.S -   |cal_loc_RAW_spirou|Calibration file: 2017-10-11_21-32-17_hcone_hcone02c406_wave_C.fits
      already exists - not copied
HH:MM:SS.S - * |cal_loc_RAW_spirou|Now processing Image TYPE UNKNOWN with ipython recipe
HH:MM:SS.S -   |cal_loc_RAW_spirou|Reading Image /drs/data/raw/20170710/flat_dark02f10.fits
HH:MM:SS.S -   |cal_loc_RAW_spirou|Image 2048 x 2048 loaded
HH:MM:SS.S - * |cal_loc_RAW_spirou|Adding 4 frame(s)
HH:MM:SS.S -   |cal_loc_RAW_spirou|Reading File: /drs/data/raw/20170710/flat_dark03f10.fits
HH:MM:SS.S -   |cal_loc_RAW_spirou|Reading File: /drs/data/raw/20170710/flat_dark04f10.fits
HH:MM:SS.S -   |cal_loc_RAW_spirou|Reading File: /drs/data/raw/20170710/flat_dark05f10.fits
HH:MM:SS.S -   |cal_loc_RAW_spirou|Reading File: /drs/data/raw/20170710/flat_dark06f10.fits
HH:MM:SS.S -   |cal_loc_RAW_spirou|Doing Dark Correction using /drs/data/calibDB/20170710
     _dark_dark02d406.fits
HH:MM:SS.S -   |cal_loc_RAW_spirou|Image format changed to 1930x2035
HH:MM:SS.S -   |cal_loc_RAW_spirou|Saving processed raw frame in 20170710
     _flat_dark02f10_order_profile_AB.fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S - * |cal_loc_RAW_spirou|Updating Calib Data Base with ORDER_PROFILE_AB
HH:MM:SS.S - * |cal_loc_RAW_spirou|Maximum flux/pixel in the spectrum: 412475.3 [e-]
HH:MM:SS.S - * |cal_loc_RAW_spirou|Average background level: 1.36 [%]
HH:MM:SS.S -   |cal_loc_RAW_spirou|Searching order center on central column
HH:MM:SS.S - * |cal_loc_RAW_spirou|On fiber AB 36 orders have been detected on 2 fiber(s)
HH:MM:SS.S -   |cal_loc_RAW_spirou|ORDER: 0 center at pixel 102.5 width 11.6 rms 0.047
HH:MM:SS.S -   |cal_loc_RAW_spirou| - center fit rms/ptp/sigrms: 0.047/0.108/2.305 with 0 rejected
     points
HH:MM:SS.S -   |cal_loc_RAW_spirou| - width  fit rms/ptp/ptp%: 0.442/0.814/6.951 with 0 rejected
     points
HH:MM:SS.S -   |cal_loc_RAW_spirou|ORDER: 1 center at pixel 116.9 width 11.4 rms 0.072
HH:MM:SS.S -   |cal_loc_RAW_spirou| - center fit rms/ptp/sigrms: 0.072/0.167/2.331 with 0 rejected
     points
HH:MM:SS.S -   |cal_loc_RAW_spirou| - width  fit rms/ptp/ptp%: 0.457/0.875/7.295 with 0 rejected
     points
...
HH:MM:SS.S -   |cal_loc_RAW_spirou|ORDER: 71 center at pixel 1881.0 width 10.8 rms 0.414
HH:MM:SS.S -   |cal_loc_RAW_spirou|      center fit converging with rms/ptp/sigrms: 0.414/2.989/7.215
...
HH:MM:SS.S -   |cal_loc_RAW_spirou|      center fit converging with rms/ptp/sigrms: 0.100/0.203/2.026
HH:MM:SS.S -   |cal_loc_RAW_spirou| - center fit rms/ptp/sigrms: 0.098/0.193/1.973 with 21 rejected
     points
HH:MM:SS.S -   |cal_loc_RAW_spirou|      fwhm fit converging with rms/ptp/ptp%: 0.970/5.272/87.869
...
HH:MM:SS.S -   |cal_loc_RAW_spirou|      fwhm fit converging with rms/ptp/ptp%: 0.478/1.258/10.480
HH:MM:SS.S -   |cal_loc_RAW_spirou| - width  fit rms/ptp/ptp%: 0.459/1.199/9.993 with 12 rejected
     points
HH:MM:SS.S - * |cal_loc_RAW_spirou|On fiber AB 72 orders geometry have been measured
HH:MM:SS.S - * |cal_loc_RAW_spirou|Average uncertainty on position: 65.96 [mpix]
HH:MM:SS.S - * |cal_loc_RAW_spirou|Average uncertainty on width: 388.67 [mpix]
HH:MM:SS.S - * |cal_loc_RAW_spirou|QUALITY CONTROL SUCCESSFUL - Well Done -
HH:MM:SS.S -   |cal_loc_RAW_spirou|Saving localization information in file: 20170710
     _flat_dark02f10_loco_AB.fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S -   |cal_loc_RAW_spirou|Saving FWHM information in file: 20170710_flat_dark02f10_fwhm-
     order_AB.fits
```

```
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S -   |cal_loc_RAW_spirou|Saving localization image with superposition of orders in
HH:MM:SS.S -   |cal_loc_RAW_spirou|file: 20170710_flat_dark02f10_with-order_AB.fits
HH:MM:SS.S - * |cal_loc_RAW_spirou|Updating Calib Data Base with LOC_AB
HH:MM:SS.S - * |cal_loc_RAW_spirou|Recipe ipython has been successfully completed
```

12.4.6 Interactive mode

In interactive mode three figures will also appear (see Figure 12.2).

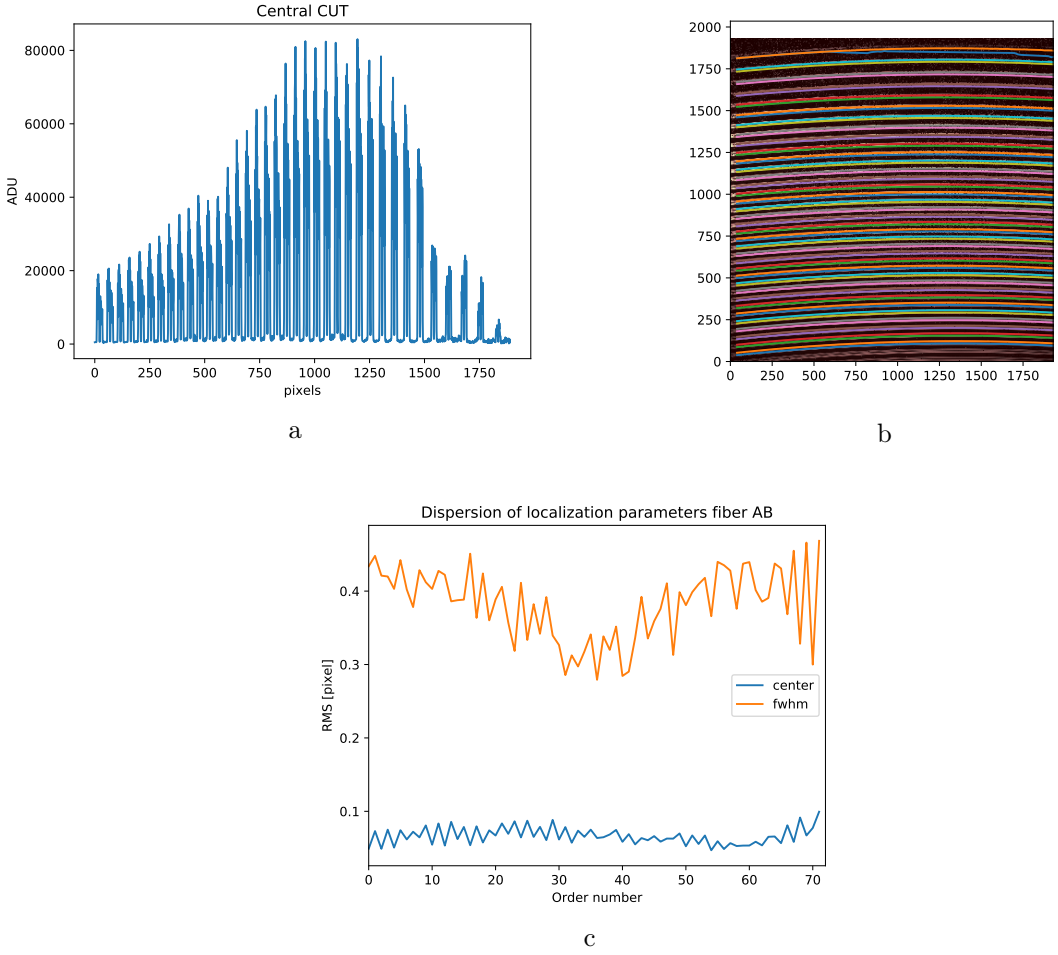


Figure 12.2 (a) Pixel number (across order) against flux value of central pixel. **(b)** Image with fits to each order. **(c)** The dispersion of localization parameters.

12.5 The cal_SLIT recipe

Fabry-Perot exposures in which the three fibres are simultaneously fed by light from the Fabry-Perot filter. Each exposure is used to build the slit orientation. Finds the tilt of the orders.

12.5.1 The inputs

The input of cal_SLIT_spirou is as follows:

```
>> cal_SLIT_spirou.py night_repository filenames
```

for example:

```
>> cal_SLIT_spirou.py 20170710 fp_fp02a203.fits fp_fp03a203.fits fp_fp04a203.fits
```

or

Python/Ipython

```
import cal_SLIT_spirou
night_repository = '20170710'
filenames = ['fp_fp02a203.fits', 'fp_fp03a203.fits', 'fp_fp04a203.fits']
cal_SLIT_spirou.main(night_repository, files=filenames)
```

where 'night_repository' defines arg_night_name and 'filenames' define the list of files in arg_file_names. All files in filenames must be valid python strings separated by a space (command line) or in a line (python) and must have the following prefixes:

- fp_fp

12.5.2 The outputs

The outputs of cal_SLIT_spirou are as follows:

- tiltfits in form:

  {reduced_dir}/{date prefix}_{file}_tilt.fits

where 'date prefix' is constructed from arg_night_name and the file name is the first file in arg_file_names. for example for reduced_dir ='/drs/data/reduced/20170710' and arg_file_names =['fp_fp02a203.fits', 'fp_fp03a203.fits', 'fp_fp04a203.fits'] the output files would be:

- /drs/data/reduced/20170710/20170710_fp_fp02a203_tilt.fits

12.5.3 Summary of procedure

1. adds all fp_fp files together
2. corrects for dark
3. resizes the image
4. extracts the orders (no weight no tilt)
5. works out the tilt for each order using the location and width

6. saves the tilt to file
7. should do some quality control
8. updates calibDB with key "TILT"

12.5.4 Quality Control

There are currently two quality control checks for cal_SLIT_spirou

- Abnormal RMS of SLIT angle if:

$$\mathrm{RMS}_{\mathrm{tilt}} > \mathrm{qc\_slit\_rms} \tag{12.9}$$

- Abnormal SLIT angle if:

$$\max(\mathrm{tilt}) > \mathrm{qc\_slit\_max} \tag{12.10}$$

or

$$\min(\mathrm{tilt}) < \mathrm{qc\_slit\_min} \tag{12.11}$$

If none of these quality control criteria are valid then the output file is passed into the calibration database with key 'TILT'.

For example the following lines are added to the calibration database for arg_night_name = "20170710" and arg_file_names = ['fp_fp02a203.fits', 'fp_fp03a203.fits', 'fp_fp04a203.fits'] .

In calibration database file

```
TILT 20170710 20170710_fp_fp02a203_tilt.fits 2017-07-10-13:25:15.590000 1499693115.59
```

12.5.5 Example working run

An example run where everything worked is below:

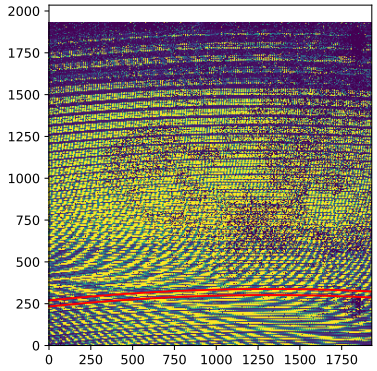
```
>> cal_SLIT_spirou.py 20170710 fp_fp02a203.fits fp_fp03a203.fits fp_fp04a203.fits
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (VERSION)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Now running : cal_SLIT_spirou on file(s): fp_fp02a203.fits
    , fp_fp03a203.fits, fp_fp04a203.fits
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|On directory /drs/data/raw/20170710
```

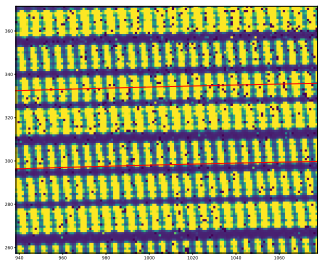
```
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|ICDP_NAME loaded from: /scratch/Projects/spirou_py3/
    spirou_py3/INTROOT/config/constants_SPIROU.py
HH:MM:SS.S - * |cal_SLIT_spirou:2a203+[...]|Correct type of image for slit (f or p or _ or f or p)
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Calibration file: 20170710_flat_flat02f10_badpixel.fits
    already exists - not copied
...
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Calibration file: spirou_wave_ini3.fits already exists -
    not copied
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Calibration file: 2017-10-11_21-32-17
    _hcone_hcone02c406_wave_AB.fits already exists - not copied
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Calibration file: spirou_wave_ini3.fits already exists -
    not copied
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Calibration file: 2017-10-11_21-32-17
    _hcone_hcone02c406_wave_C.fits already exists - not copied
HH:MM:SS.S - * |cal_SLIT_spirou:2a203+[...]|Now processing Image TYPE UNKNOWN with cal_SLIT_spirou
    recipe
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Reading Image /drs/data/raw/20170710/fp_fp02a203.fits
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Image 2048 x 2048 loaded
HH:MM:SS.S - * |cal_SLIT_spirou:2a203+[...]|Adding 2 frame(s)
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Reading File: /drs/data/raw/20170710/fp_fp03a203.fits
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Reading File: /drs/data/raw/20170710/fp_fp04a203.fits
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Doing Dark Correction using /drs/data/calibDB/20170710
    _dark_dark02d406.fits
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Image format changed to 1930x2035
HH:MM:SS.S - * |cal_SLIT_spirou:2a203+[...]|Nb dead pixels = 611716 / 15.58 %
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Reading localization parameters of Fiber AB
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Order 0.0: Tilt = 4.70 on pixel 37.0 = -7.23 deg
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Order 1.0: Tilt = 4.60 on pixel 37.4 = -7.02 deg
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Order 2.0: Tilt = 4.50 on pixel 36.8 = -6.97 deg
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Order 3.0: Tilt = 4.30 on pixel 36.3 = -6.75 deg
...
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Order 32.0: Tilt = 1.40 on pixel 33.3 = -2.41 deg
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Order 33.0: Tilt = 1.10 on pixel 32.3 = -1.95 deg
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Order 34.0: Tilt = 1.00 on pixel 32.1 = -1.79 deg
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Order 35.0: Tilt = 0.30 on pixel 17.1 = -1.01 deg
HH:MM:SS.S - * |cal_SLIT_spirou:2a203+[...]AB|Tilt dispersion = 0.091 deg
HH:MM:SS.S -   |cal_SLIT_spirou:2a203+[...]|Saving tilt  information in file: 20170710
    _fp_fp02a203_tilt.fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S - * |cal_SLIT_spirou:2a203+[...]|QUALITY CONTROL SUCCESSFUL - Well Done -
HH:MM:SS.S - * |cal_SLIT_spirou:2a203+[...]|Updating Calib Data Base with TILT
HH:MM:SS.S - * |cal_SLIT_spirou:2a203+[...]|Recipe cal_SLIT_spirou has been successfully completed
    HHMSSS
```

12.5.6 Interactive mode

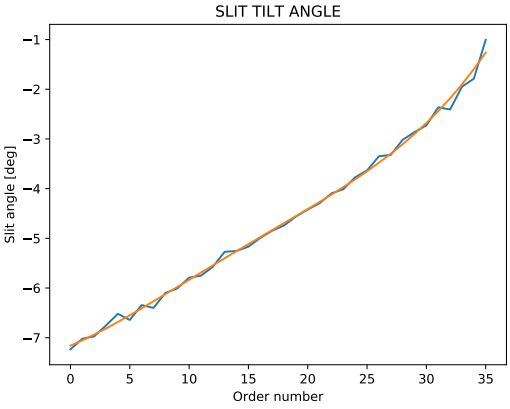
In interactive mode three figures will also appear (see Figure 12.3).



a



b



c

Figure 12.3 (a) The full 'fp_fp' image with one orders fit plotted. **(b)** Zoom in on a section of the 'fp_fp' image showing the tilt. **(c)** Slit angle as a function of order number with the fit to the tilt also show.

12.6 The cal_FF recipe

Creates the flat fields.

12.6.1 The inputs

The input of cal_FF_RAW_spirou is as follows:

```
>> cal_FF_RAW_spirou.py night_repository filenames
```

for example

example

```
>> cal_FF_RAW_spirou.py 20170710 dark_flat02f10.fits dark_flat03f10.fits dark_flat04f10.fits
    dark_flat05f10.fits dark_flat06f10.fits
```

or

Python/Ipython

```
import cal_FF_RAW_spirou
night_repository = '20170710'
filenames = ['dark_flat02f10.fits', 'dark_flat03f10.fits', 'dark_flat04f10.fits',
             'dark_flat05f10.fits', 'dark_flat06f10.fits']
cal_FF_RAW_spirou.main()
```

where 'night_repository' defines arg_night_name and 'filenames' define the list of files in arg_file_names. All files in filenames must be valid python strings separated by a space (command line) or in a line (python) and must have the following prefixes:

- dark_flat
- flat_dark

12.6.2 The outputs

The outputs of cal_FF_RAW_spirou are as follows:

- blazefits in form:
  {reduced_dir}/{date prefix}_{file}_blaze_{fiber}.fits
- flatfits in form:
  {reduced_dir}/{date prefix}_{file}_flat_{fiber}.fits

where 'date prefix' is constructed from arg_night_name and the file name is the first file in arg_file_names. for example for reduced_dir ='/drs/data/reduced/20170710' and arg_file_names =['dark_flat02f10.fits', 'dark_flat03f10.fits', 'dark_flat04f10.fits', 'dark_flat05f10.fits', 'dark_flat06f10.fits'] the output files would be:

- /drs/data/reduced/20170710/20170710_flat_dark02f10_blaze_AB.fits
- /drs/data/reduced/20170710/20170710_flat_dark02f10_flat_AB.fits

12.6.3 Summary of procedure

1. adds all 'dark_flat' or 'flat_dark' files together
2. corrects for darks
3. resizes the image
4. possible background subtraction?
5. extracts the orders using tilt and weight
6. calculates the blaze
7. calculates the flat field, (flat = extraction / blaze)
8. stores the flat fields
9. does some quality control
10. updates calibDB with key "FLAT_{fiber}" where {fiber} = [AB, C] etc.

12.6.4 Quality Control

There is currently one quality control check for cal_FF_RAW_spirou

- Too much flux in the image:

$$\text{maximum signal} > \text{qc\_max\_signal} * \text{nbframes} \tag{12.12}$$

Note: This check does not currently lead to a failed run and all files are processed as passing quality checks

The output file is passed into the calibration database with key 'FLAT_{fiber}' for the 'flatfits' file.

For example the following lines are added to the calibration database for arg_night_name = "20170710" and arg_file_names =['dark_flat02f10.fits', 'dark_flat03f10.fits', 'dark_flat04f10.fits', 'dark_flat05f10.fits', 'dark_flat06f10.fits'].

In calibration database file

```
FLAT_C 20170710 20170710_dark_flat02f10_flat_C.fits 2017-07-10-13:03:50.440000 1499691830.44
BLAZE_C 20170710 20170710_dark_flat02f10_blaze_C.fits 2017-07-10-13:03:50.440000 1499691830.44
```

12.6.5 Example working run

An example run where everything worked is below:

example

```
>> cal_FF_RAW_spirou.py 20170710 dark_flat02f10.fits dark_flat03f10.fits dark_flat04f10.fits
    dark_flat05f10.fits dark_flat06f10.fits
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (VERSION)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Now running : cal_FF_RAW_spirou on file(s):
    dark_flat02f10.fits, dark_flat03f10.fits, dark_flat04f10.fits, dark_flat05f10.fits,
    dark_flat06f10.fits
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|On directory /drs/data/raw/20170710
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|ICDP_NAME loaded from: /scratch/Projects/spirou_py3/
    spirou_py3/INTROOT/config/constants_SPIROU.py
HH:MM:SS.S - * |cal_FF_RAW_spirou:02f10+[...]|Correct type of image for Flat-field (dark_flat or
    flat_dark)
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Calibration file: 20170710_flat_flat02f10_badpixel.fits
    already exists - not copied
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Calibration file: 20170710_dark_dark02d406.fits already
    exists - not copied
...
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Calibration file: 20170710
    _flat_dark02f10_order_profile_AB.fits already exists - not copied
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Calibration file: 20170710
    _dark_flat02f10_order_profile_C.fits already exists - not copied
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Calibration file: 20170710_fp_fp02a203_tilt.fits already
     exists - not copied
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Calibration file: spirou_wave_ini3.fits already exists -
     not copied
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Calibration file: 2017-10-11_21-32-17
    _hcone_hcone02c406_wave_AB.fits already exists - not copied
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Calibration file: spirou_wave_ini3.fits already exists -
     not copied
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Calibration file: 2017-10-11_21-32-17
    _hcone_hcone02c406_wave_C.fits already exists - not copied
HH:MM:SS.S - * |cal_FF_RAW_spirou:02f10+[...]|Now processing Image TYPE UNKNOWN with cal_FF_RAW_spirou
     recipe
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Reading Image /drs/data/raw/20170710/dark_flat02f10.fits
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Image 2048 x 2048 loaded
HH:MM:SS.S - * |cal_FF_RAW_spirou:02f10+[...]|Adding 4 frame(s)
```

```
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Reading File: /drs/data/raw/20170710/dark_flat03f10.fits
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Reading File: /drs/data/raw/20170710/dark_flat04f10.fits
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Reading File: /drs/data/raw/20170710/dark_flat05f10.fits
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Reading File: /drs/data/raw/20170710/dark_flat06f10.fits
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Doing Dark Correction using /drs/data/calibDB/20170710
    _dark_dark02d406.fits
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Image format changed to 2035x1930
HH:MM:SS.S - * |cal_FF_RAW_spirou:02f10+[...]|Nb dead pixels = 568541 / 14.48 %
HH:MM:SS.S - * |cal_FF_RAW_spirou:02f10+[...]|Maximum average flux/pixel in the spectrum: 73636.3 [ADU
    ]
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|Reading localization parameters of Fiber C
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]C|Reading order profile of Fiber C
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|On fiber C order 0: S/N= 1158.4  - FF rms=4.68 %
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|On fiber C order 1: S/N= 1193.9  - FF rms=4.80 %
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|On fiber C order 2: S/N= 1232.6  - FF rms=4.76 %
...
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|On fiber C order 33: S/N= 1686.9  - FF rms=5.67 %
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|On fiber C order 34: S/N= 1574.5  - FF rms=8.17 %
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]|On fiber C order 35: S/N= 1260.6  - FF rms=8.10 %
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]C|Saving blaze spectrum for fiber: C in 20170710
    _dark_flat02f10_blaze_C.fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S -   |cal_FF_RAW_spirou:02f10+[...]C|Saving FF spectrum for fiber: C in 20170710
    _dark_flat02f10_flat_C.fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S - * |cal_FF_RAW_spirou:02f10+[...]|QUALITY CONTROL SUCCESSFUL - Well Done -
HH:MM:SS.S - * |cal_FF_RAW_spirou:02f10+[...]|Updating Calib Data Base with FLAT_C
HH:MM:SS.S - * |cal_FF_RAW_spirou:02f10+[...]|Updating Calib Data Base with BLAZE_C
HH:MM:SS.S - * |cal_FF_RAW_spirou:02f10+[...]|Recipe cal_FF_RAW_spirou has been successfully completed
```

12.6.6 Interactive mode

In interactive mode three figures will also appear (see Figure 12.4).

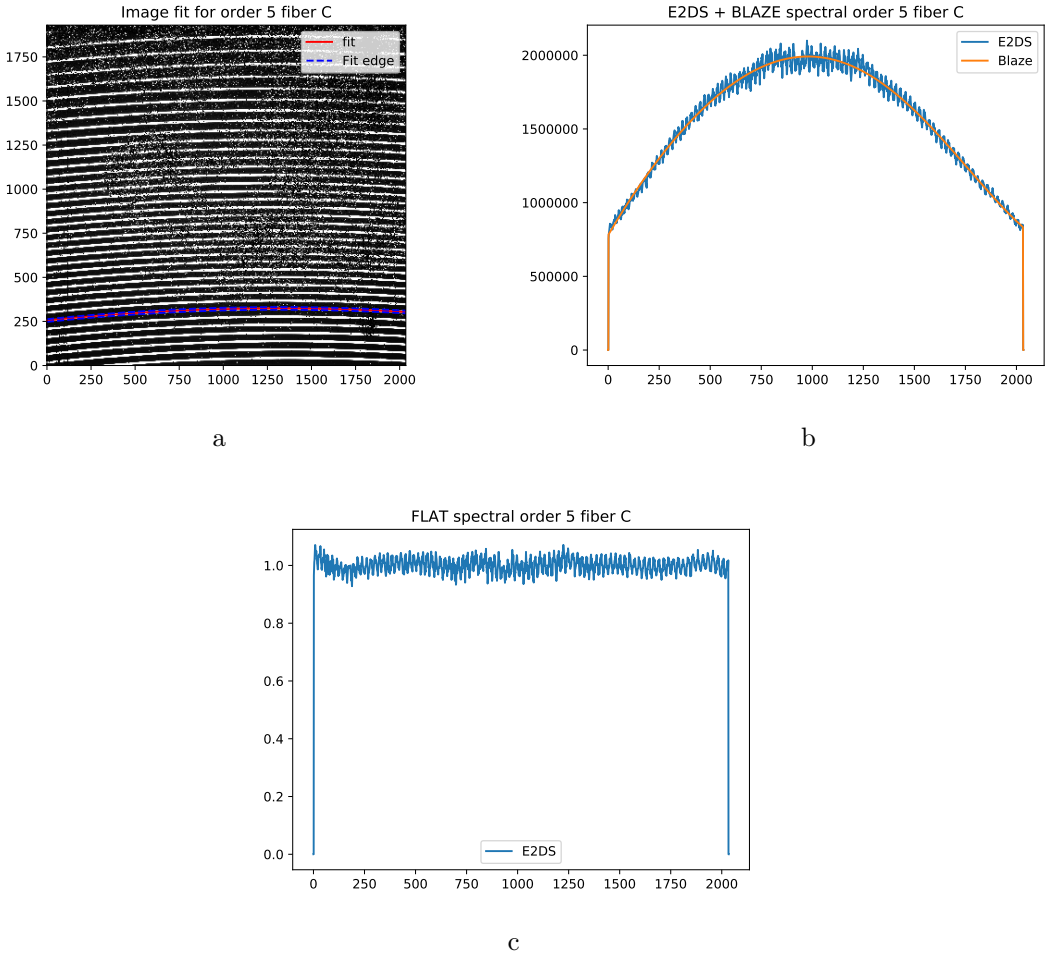


Figure 12.4 (a) the full processed image with one order fit highlighted. **(b)** An extracted over-plotted with the blaze fit. **(c)** A flattened order.

12.7 The cal_extract recipes

Extracts orders for specific fibers and files. There are currently three extraction recipes. There is the main extraction recipe (cal_extract_RAW_spirou) and two wrapper recipes (cal_extract_RAW_spirouAB and cal_extract_RAW_spirouC, which push certain options into cal_extract_RAW_spirou).

12.7.1 The inputs

The input of cal_extract_RAW_spirou is as follows:

```
>> cal_extract_RAW_spirou.py night_repository filenames
```

for example

example

```
>> cal_extract_RAW_spirou.py 20170710 fp_fp02a203.fits
```

or

Python/Ipython

```
import cal_extract_RAW_spirou
night_repository = '20170710'
filenames = ['fp_fp02a203.fits']
cal_extract_RAW_spirou.main(night_repository, files=filenames)
```

where 'night_repository' defines arg_night_name and 'filenames' define the list of files in arg_file_names. In addition to this one can add optional arguments (and this is the case for the wrapper recipes of cal_extract_RAW_spirouAB and cal_extract_RAW_spirouC).

All files in filenames must be valid python strings separated by a space (command line) or in a line (python) and should have the following prefixes:

- fp_fp
- hcone_dark
- dark_hcone
- hcone_hcone
- dark_dark_AHC1
- dark_hctwo
- hctwo_hctwo
- dark_dark_AHC2

12.7.1.1 Optional arguments

By default cal_extract_RAW_spirou will extract all fibers defined in fiber_types (i.e. 'AB', 'A', 'B' and 'C'). It may be the case that one wishes to extract only one fiber, this can be done by setting the 'fiber_type' option (when run from python). For example:

Python/Ipython

```
import cal_extract_RAW_spirou
night_repository = '20170710'
filenames = ['fp_fp02a203.fits']
cal_DARK_spirou.main(night_repository, filenames, fiber_type='A')
```

where 'fiber_type' must be a valid python string and defined in fiber_types. On can also overwrite any other variable that is (or even is not) defined in any of the constant files (or run time) by simply adding it as an optional argument in the python function call:

Python/Ipython

```
import cal_extract_RAW_spirou
night_repository = '20170710'
filenames = ['fp_fp02a203.fits', 'fp_fp03a203.fits', 'fp_fp04a203.fits']
kwargs = dict(parameter1='This', parameter2='That')
cal_DARK_spirou.main(night_repository, filenames, fiber_type='A', **kwargs)
```

Note: This will extract fiber 'A' and also set the values `parameter1='This'` and `parameter2='That'` in the main constant parameter dictionary.

This is the case for the two wrapper recipes (cal_extract_RAW_spirouAB and cal_extract_RAW_spirouC) which add some specific keywords that are used to individually extract fibers 'AB' and 'C' respectively. The two calls to the main extraction code are shown below but can be called from the console or python as with all other recipes.

```
>> cal_extract_RAW_spirouAB.py night_repository filenames
>> cal_extract_RAW_spirouC.py night_repository filenames
```

for example

example

```
>> cal_extract_RAW_spirouAB.py 20170710 fp_fp02a203.fits fp_fp03a203.fits fp_fp04a203.fits
>> cal_extract_RAW_spirouC.py 20170710 fp_fp02a203.fits fp_fp03a203.fits fp_fp04a203.fits
```

or

Python/Ipython

```
import cal_extract_RAW_spirouAB
import cal_extract_RAW_spirouC
night_repository = '20170710'
filenames = ['fp_fp02a203.fits']
# extract fiber AB
cal_extract_RAW_spirouAB.main(night_repository, files=filenames)
# extract fiber C
cal_extract_RAW_spirouC.main(night_repository, files=filenames)
```

as mentioned above this can also be done in python just by adding additional arguments to cal_extract_RAW_spirou:

Python/Ipython

```
import cal_extract_RAW_spirou
night_repository = '20170710'
filenames = ['fp_fp02a203.fits', 'fp_fp03a203.fits', 'fp_fp04a203.fits']
# extract fiber AB (for all extraction types)
cal_extract_RAW_spirou.main(night_repository, files=filenames,fiber_type='AB',
                            ic_extract_type='all', ic_ext_sigdet=-1)
# extract fiber C (for all extraction types)
cal_extract_RAW_spirou.main(night_repository, files=filenames,fiber_type='C',
                            ic_extract_type='all', ic_ext_sigdet=-1)
```

Note: Here we set the optional arguments `ic_extract_type='all', ic_ext_sigdet=-1`. ic_extract_type defines which type of extraction should be used (simple, tilt, weigh, tiltweight, all) and ic_ext_sigdet manually sets the extraction sigdet (-1 sets it to the value from the HEADER else it is defined in constants_SPIROU_H4RG.py).

The above python code is exactly what cal_extract_RAW_spirouAB and cal_extract_RAW_spirouC do by default.

12.7.2 The outputs

The outputs of cal_extract_RAW_spirou depend on the extraction type (ic_extract_type).

- e2ds in form:

  {reduced_dir}/{date prefix}_{file}_e2ds_{fiber}.fits

- simple in form:

  {reduced_dir}/{date prefix}_{file}_e2ds_{fiber}_simple.fits

  Note: Only if ic_extract_type is 'all'

- tilt in form:

  {reduced_dir}/{date prefix}_{file}_e2ds_{fiber}_tilt.fits

  Note: Only if ic_extract_type is 'all'

- tiltweight in form:

  {reduced_dir}/{date prefix}_{file}_e2ds_{fiber}_tiltweight.fits

  Note: Only if ic_extract_type is 'all'

- tiltweight2 in form:

{reduced_dir}/{date prefix}_{file}_e2ds_{fiber}_tiltweight2.fits

Note: Only if ic_extract_type is 'all'

- weight in form:

{reduced_dir}/{date prefix}_{file}_e2ds_{fiber}_weight.fits

Note: Only if ic_extract_type is 'all'

where 'date prefix' is constructed from arg_night_name and the file name is the first file in arg_file_names. for example for reduced_dir `='/drs/data/reduced/20170710'` and arg_file_names `=['fp_fp02a203.fits']` and, `fiber_type ='all'` the output files would be:

- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_A.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_B.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_C.fits`

or if `ic_extract_type='all'` and `fiber_type='AB'` the output files would be:

- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_simple.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_tilt.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_tiltweight.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_tiltweight2.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_weight.fits`

12.7.3 Summary of procedure

1. adds all files together (if more than one)
2. corrects for darks
3. resizes the image
4. checks for saturation
5. possible background subtraction?
6. extracts orders (depending on ic_extract_type)
   - without tilt/weight Fortran
   - without tilt/weight python
   - with tilt (no weight)

- with tilt and weight
- with weight (no tilt)

7. saves extraction to e2ds file(s)

12.7.4 Quality Control

There is currently one quality control check for cal_extract_RAW_spirou

- Too much flux in the image:

$$\text{maximum signal} > \text{qc\_max\_signal} * \text{nbframes} \tag{12.13}$$

Note: This check does not currently lead to a failed run and all files are processed as passing quality checks

12.7.5 Example working run

An example run where everything worked is below:

example

```
>> cal_extract_RAW_spirou.py 20170710 fp_fp02a203.fits
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (VERSION)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Now running : cal_extract_RAW_spirou on file(s):
    fp_fp02a203.fits
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On directory /drs/data/raw/20170710
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|ICDP_NAME loaded from: /scratch/Projects/spirou_py3/
    spirou_py3/INTROOT/config/constants_SPIROU.py
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Calibration file: 20170710_flat_flat02f10_badpixel.fits
    already exists - not copied
...
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Calibration file: 2017-10-11_21-32-17
    _hcone_hcone02c406_wave_C.fits already exists - not copied
HH:MM:SS.S - * |cal_extract_RAW_spirou:2a203|Now processing Image TYPE UNKNOWN with
    cal_extract_RAW_spirou recipe
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Reading Image /drs/data/raw/20170710/fp_fp02a203.fits
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Image 2048 x 2048 loaded
```

```
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Doing Dark Correction using /drs/data/calibDB/20170710
    _dark_dark02d406.fits
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Image format changed to 2035x1930
HH:MM:SS.S - * |cal_extract_RAW_spirou:2a203|Nb dead pixels = 568485 / 14.47 %
HH:MM:SS.S - * |cal_extract_RAW_spirou:2a203|Maximum average flux/pixel in the spectrum: 109726.7 [ADU
    ]
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Reading localization parameters of Fiber AB
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203AB|Reading order profile of Fiber AB
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber AB order 0: S/N= 352.4
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber AB order 1: S/N= 398.4
...
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber AB order 34: S/N= 138.8
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber AB order 35: S/N= 47.8
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Saving E2DS spectrum of Fiber AB in fp_fp02a203_e2ds_AB.
    fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Reading localization parameters of Fiber A
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203A|Reading order profile of Fiber A
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber A order 0: S/N= 210.2
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber A order 1: S/N= 237.4
...
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber A order 34: S/N= 114.3
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber A order 35: S/N= 33.8
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Saving E2DS spectrum of Fiber A in fp_fp02a203_e2ds_A.
    fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Reading localization parameters of Fiber B
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203B|Reading order profile of Fiber B
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber B order 0: S/N= 286.8
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber B order 1: S/N= 324.4
...
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber B order 34: S/N= 87.2
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber B order 35: S/N= 36.1
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Saving E2DS spectrum of Fiber B in fp_fp02a203_e2ds_B.
    fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Reading localization parameters of Fiber C
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203C|Reading order profile of Fiber C
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber C order 0: S/N= 376.6
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber C order 1: S/N= 424.3
...
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber C order 34: S/N= 252.7
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|On fiber C order 35: S/N= 130.0
HH:MM:SS.S -   |cal_extract_RAW_spirou:2a203|Saving E2DS spectrum of Fiber C in fp_fp02a203_e2ds_C.
    fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S - * |cal_extract_RAW_spirou:2a203|Too much flux in the image (max authorized=65500)
HH:MM:SS.S - * |cal_extract_RAW_spirou:2a203|QUALITY CONTROL SUCCESSFUL - Well Done -
HH:MM:SS.S - * |cal_extract_RAW_spirou:2a203|Recipe cal_extract_RAW_spirou has been successfully
    complted
```

12.8 The cal_DRIFT recipes

There are currently three different drift recipes: cal_DRIFT_RAW_spirou , cal_DRIFT_E2DS_spirou and cal_DRIFTPEAK_E2DS_spirou . The cal_DRIFT_E2DS_spirou and cal_DRIFTPEAK_E2DS_spirou recipes are the primary drift code recipes and cal_DRIFT_RAW_spirou is an older version where spectra are re-extracted.

12.8.1 The inputs

12.8.1.1 cal_DRIFT_E2DS_spirou and cal_DRIFTPEAK_E2DS_spirou

The input of cal_DRIFT_E2DS_spirou and cal_DRIFTPEAK_E2DS_spirou are as follows:

```
>> cal_DRIFT_E2DS_spirou.py night_repository referenece_file
>> cal_DRIFTPEAK_E2DS_spirou.py night_repository referenece_file
```

for example

example

```
>> cal_DRIFT_E2DS_spirou.py 20170710 fp_fp02a203_e2ds_AB.fits
>> cal_DRIFTPEAK_E2DS_spirou.py 20170710 fp_fp02a203_e2ds_AB.fits
```

or

Python/Ipython

```
import cal_DRIFT_E2DS_spirou
import cal_DRIFTPEAK_E2DS_Spirou
night_repository = '20170710'
reffilename = 'fp_fp02a203_e2ds_AB.fits'
cal_DRIFT_E2DS_spirou.main(night_repository, reffile=reffilename)
cal_DRIFTPEAK_E2DS_spirou.main(night_repository, reffile=reffilename)
```

where 'night_repository' defines arg_night_name and 'refilename' define the file to use as the reference spectrum. The reference file must be a valid python string and must have the following prefixes:

- fp_fp

and must contain the fiber type (i.e. 'AB' or 'A' or 'C').

12.8.1.2 cal_DRIFT_RAW_spirou

The input of cal_DRIFT_RAW_spirou is as follows:

```
>> cal_DRIFT_RAW_spirou.py night_repository files
```

for example

example

```
>> cal_DRIFT_RAW_spirou.py 20170710 fp_fp02a203.fits
```

or

Python/Ipython

```
import cal_DRIFT_RAW_spirou
night_repository = '20170710'
filenames = ['fp_fp02a203.fits']
cal_DRIFT_RAW_spirou.main(night_repository, files=filenames)
```

where 'night_repository' defines arg_night_name and 'filenames' define the list of files in arg_file_names. All files in filenames must be valid python strings separated by a space (command line) or in a line (python) and must have the following prefixes:

- fp_fp

Note: cal_DRIFT_RAW_spirou can also take an addition argument. By default it will extract fiber 'AB', but this can be changed if using python by specifying the 'fiber' keyword, for example:

Python/Ipython

```
import cal_DRIFT_RAW_spirou
night_repository = '20170710'
filenames = ['fp_fp02a203.fits']
cal_DRIFT_RAW_spirou.main(night_repository, files=filenames, fiber='A')
```

12.8.2 The outputs

12.8.2.1 cal_DRIFT_E2DS_spirou

The outputs of cal_DRIFT_E2DS_spirou is as follows:

- driftfits_e2ds in form:

  {reduced_dir}/{date prefix}_{file}_drift_{fiber}.fits

- drifttblfilename_e2ds in form:

  {reduced_dir}/{date prefix}_{file}_drift_{fiber}.tbl

where 'date prefix' is constructed from arg_night_name and the file name is the 'reference filename'. for example for reduced_dir `='/drs/data/reduced/20170710'`, `reffile = 'fp_fp02a203_e2ds_AB.fits'` and `fiber='AB'` the output files would be:

- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_drift_AB.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_drift_AB.tbl`

12.8.2.2 cal_DRIFTPEAK_E2DS_spirou

The outputs of cal_DRIFTPEAK_E2DS_spirou is as follows:

- driftfits_peak_e2ds in form:

  {reduced_dir}/{date prefix}_{file}_driftnew_{fiber}.fits

- drifttblfilename_peak_e2ds in form:

  {reduced_dir}/{date prefix}_{file}_driftnew_{fiber}.tbl

where 'date prefix' is constructed from arg_night_name and the file name is the 'reference filename'. for example for reduced_dir `='/drs/data/reduced/20170710'`, `reffile = 'fp_fp02a203_e2ds_AB.fits'` and `fiber='AB'` the output files would be:

- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_driftnew_AB.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_driftnew_AB.tbl`

12.8.2.3 cal_DRIFT_RAW_spirou

The outputs of cal_DRIFT_RAW_spirou are as follows:

- driftfits_raw in form:

  {reduced_dir}/{date prefix}_{file}_drift_{fiber}.fits

where 'date prefix' is constructed from arg_night_name and the file name is the first file in arg_file_names. for example for reduced_dir `='/drs/data/reduced/20170710'`, arg_file_names `=['fp_fp02a203.fits']` and `fiber='AB'` the output files would be:

- `/drs/data/reduced/20170710/fp_fp02a203_e2ds_AB_drift_AB.fits`

12.8.3 Summary of procedure

12.8.3.1 cal_DRIFT_E2DS_spirou

1. first file is reference image (must be an E2DS file) extracted using one of the cal_extract_RAW recipes
2. loops around all other '*_e2ds_{fiber}.fits' files in directory
3. calculates photon noise uncertainty and estimated RV uncertainty on spectrum
   - uses wave file
4. calculates RV drift and mean RV drift between reference (mean of files) and other 'fp_fp' files
5. saves drift values to file

12.8.3.2 cal_DRIFTPEAK_E2DS_spirou

1. first file is reference image
2. resizes the image
3. background correction
4. Identifies FP peaks in reference file
5. Creates a reference ASCII file that contains the positions of the FP peaks
6. Removes lines with suspicious widths
7. loops around all other '*_e2ds_{fiber}.fits' files

8. Gets the centroid of all peaks using Gaussian fitting
9. Performs a Pearson R test to look for issues with extraction and/or illumination
10. Performs sigma clipping on measured drift
11. Saves drifts to file

12.8.3.3 cal_DRIFT_RAW_spirou

1. first file is reference image
2. resizes the image
3. extracts with weight (no tilt)
4. loops around all other 'fp_fp' files in directory
5. calculates photon noise uncertainty and estimated RV uncertainty on spectrum
   - uses wave file
6. calculates RV drift and mean RV drift between reference (mean of files) and other 'fp_fp' files
7. saves drift values to file

12.8.4 Example working run

12.8.4.1 cal_DRIFT_E2DS_spirou

An example run where everything worked is below:

example

```
>> cal_DRIFT_E2DS_spirou.py 20170710 fp_fp02a203_e2ds_AB.fits
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU /@(#) Geneva Observatory (0.1.022)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Now running : cal_DRIFT_E2DS_spirou with:
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|       -- reffile=fp_fp02a203_e2ds_AB.fits
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|ICDP_NAME loaded from: /scratch/Projects/spirou_py3/spirou_py3/
    INTROOT/config/constants_SPIROU.py
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_flat_flat02f10_badpixel.fits already
    exists - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_flat_dark02f10_blaze_AB.fits already
    exists - not copied
```

```
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_dark_flat02f10_blaze_C.fits already
    exists - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_dark_dark02d406.fits already exists -
     not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_flat_dark02f10_flat_AB.fits already
    exists - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_dark_flat02f10_flat_C.fits already
    exists - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_flat_dark02f10_loco_AB.fits already
    exists - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_dark_flat02f10_loco_C.fits already
    exists - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_flat_dark02f10_order_profile_AB.fits
    already exists - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_dark_flat02f10_order_profile_C.fits
    already exists - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 20170710_fp_fp02a203_tilt.fits already exists
    - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: spirou_wave_ini3.fits already exists - not
    copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 2017-10-11_21-32-17_hcone_hcone02c406_wave_AB.
    fits already exists - not copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: spirou_wave_ini3.fits already exists - not
    copied
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Calibration file: 2017-10-11_21-32-17_hcone_hcone02c406_wave_C.
    fits already exists - not copied
HH:MM:SS.S - * |cal_DRIFT_E2DS_spirou|Now processing Image TYPE DRIFT with cal_DRIFT_E2DS_spirou
    recipe
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Reading File: /drs/data/reduced/20170710/fp_fp02a203_e2ds_AB.
    fits
HH:MM:SS.S - * |cal_DRIFT_E2DS_spirou|On fiber AB estimated RV uncertainty on spectrum is 0.025 m/s
HH:MM:SS.S - * |cal_DRIFT_E2DS_spirou|Number of fp_fp files found on directory = 2
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Reading file fp_fp03a203_e2ds_AB.fits
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Time from ref=0.09 h  - Drift mean= -2.92 +- 0.036 m/s - Flux
    ratio= 0.99 - Nb Comsic= 391
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Reading file fp_fp04a203_e2ds_AB.fits
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Time from ref=0.18 h  - Drift mean= 10.22 +- 0.036 m/s - Flux
    ratio= 0.98 - Nb Comsic= 346
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Total drift Peak-to-Peak=13.142 m/s RMS=6.571 m/s in 0.18 hour
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirou|Saving drift values of Fiber AB in fp_fp02a203_e2ds_AB_drift_AB.
    fits
HH:MM:SS.S -   |cal_DRIFT_E2DS_spirouAB|Average Drift saved in fp_fp02a203_e2ds_AB_drift_AB.tbl Saved
HH:MM:SS.S - * |cal_DRIFT_E2DS_spirou|Recipe cal_DRIFT_E2DS_spirou has been successfully completed
```

12.8.4.2 cal_DRIFTPEAK_E2DS_spirou

An example run where everything worked is below:

example

```
>> cal_DRIFTPEAK_E2DS_spirou.py 20170710 fp_fp02a203_e2ds_AB.fits
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU 0(#) Geneva Observatory (VERSION)
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Now running : cal_DRIFTPEAK_E2DS_spirou with:
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|       -- reffile=fp_fp02a203_e2ds_AB.fits
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|ICDP_NAME loaded from: /scratch/Projects/spirou_py3/
    spirou_py3/INTROOT/config/constants_SPIROU.py
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Calibration file: 20170710_flat_flat02f10_badpixel.fits
    already exists - not copied
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Calibration file: 20170710_flat_dark02f10_blaze_AB.fits
    already exists - not copied
...
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Calibration file: spirou_wave_ini3.fits already exists - not
     copied
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Calibration file: 2017-10-11_21-32-17
    _hcone_hcone02c406_wave_C.fits already exists - not copied
HH:MM:SS.S - * |cal_DRIFTPEAK_E2DS_spirou|Now processing Image TYPE DRIFT with
    cal_DRIFTPEAK_E2DS_spirou recipe
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Reading File: /drs/data/reduced/20170710/fp_fp02a203_e2ds_AB
    .fits
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Perform background correction
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Identification of lines in reference file:
    fp_fp02a203_e2ds_AB.fits
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Order 0 : 389 peaks found, 0 peaks rejected
...
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Order 28 : 235 peaks found, 0 peaks rejected
HH:MM:SS.S - @ |python warning Line 779  warning reads: Covariance of the parameters could not be
    estimated|
HH:MM:SS.S - @ |python warning Line 779  warning reads: Covariance of the parameters could not be
    estimated|
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Order 29 : 231 peaks found, 1 peaks rejected
...
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Order 33 : 311 peaks found, 28 peaks rejected
HH:MM:SS.S - @ |python warning Line 779  warning reads: Covariance of the parameters could not be
    estimated|
...
HH:MM:SS.S - @ |python warning Line 779  warning reads: Covariance of the parameters could not be
    estimated|
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Order 34 : 348 peaks found, 24 peaks rejected
HH:MM:SS.S - @ |python warning Line 779  warning reads: Covariance of the parameters could not be
    estimated|
...
HH:MM:SS.S - @ |python warning Line 779  warning reads: Covariance of the parameters could not be
    estimated|
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Order 35 : 483 peaks found, 77 peaks rejected
```

```
HH:MM:SS.S - * |cal_DRIFTPEAK_E2DS_spirou|Total Nb of FP lines found = 11054
HH:MM:SS.S - * |cal_DRIFTPEAK_E2DS_spirou|Nb of lines removed due to suspicious width = 1245
HH:MM:SS.S - * |cal_DRIFTPEAK_E2DS_spirou|Nb of lines removed with no width measurement = 0
HH:MM:SS.S - * |cal_DRIFTPEAK_E2DS_spirou|Number of files found on directory = 2
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Reading file fp_fp03a203_e2ds_AB.fits
HH:MM:SS.S - * |cal_DRIFTPEAK_E2DS_spirou|Time from ref= 0.09 h - Flux Ratio= 1.01 - Drift mean= -2.31
      +- 0.24 m/s
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Reading file fp_fp04a203_e2ds_AB.fits
HH:MM:SS.S - * |cal_DRIFTPEAK_E2DS_spirou|Time from ref= 0.18 h - Flux Ratio= 1.01 - Drift mean= 9.41
     +- 0.26 m/s
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Total drift Peak-to-Peak=11.718 m/s RMS=5.859 m/s in 0.18
     hour
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirou|Saving drift values of Fiber AB in
     fp_fp02a203_e2ds_AB_driftnew_AB.fits
HH:MM:SS.S -   |cal_DRIFTPEAK_E2DS_spirouAB|Average Drift saved in fp_fp02a203_e2ds_AB_driftnew_AB.tbl
      Saved
HH:MM:SS.S - * |cal_DRIFTPEAK_E2DS_spirou|Recipe cal_DRIFTPEAK_E2DS_spirou has been successfully
     completed
```

12.8.4.3 cal_DRIFT_RAW_spirou

example

```
>> cal_DRIFT_RAW_spirou.py 20170710 fp_fp02a203.fits
```

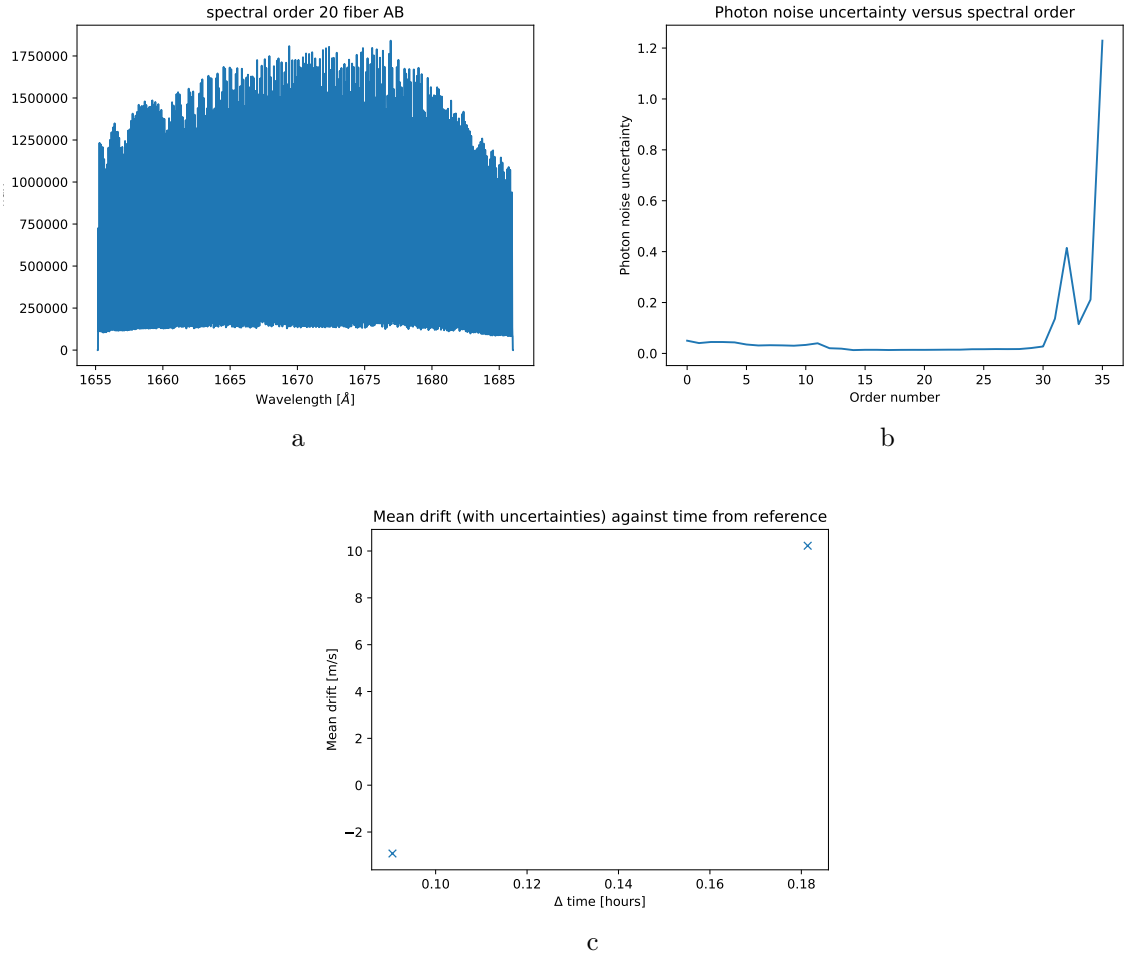
```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (VERSION)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Now running : cal_DRIFT_RAW_spirou on file(s): fp_fp02a203.
     fits
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|On directory /drs/data/raw/20170710
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|ICDP_NAME loaded from: /scratch/Projects/spirou_py3/
     spirou_py3/INTROOT/config/constants_SPIROU.py
HH:MM:SS.S - * |cal_DRIFT_RAW_spirou:2a203|Correct type of image for Drift (f or p or _ or f or p)
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Calibration file: 20170710_flat_flat02f10_badpixel.fits
     already exists - not copied
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Calibration file: 20170710_flat_dark02f10_blaze_AB.fits
     already exists - not copied
...
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Calibration file: 2017-10-11_21-32-17
     _hcone_hcone02c406_wave_C.fits already exists - not copied
```

```
HH:MM:SS.S - * |cal_DRIFT_RAW_spirou:2a203|Now processing Image TYPE UNKNOWN with cal_DRIFT_RAW_spirou
     recipe
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Reading Image /drs/data/raw/20170710/fp_fp02a203.fits
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Image 2048 x 2048 loaded
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Doing Dark Correction using /drs/data/calibDB/20170710
    _dark_dark02d406.fits
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Image format changed to 2035x1930
HH:MM:SS.S - * |cal_DRIFT_RAW_spirou:2a203|Nb dead pixels = 568485 / 14.47 %
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Reading localization parameters of Fiber AB
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203AB|Reading order profile of Fiber AB
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Extraction Reference file /drs/data/raw/20170710/
    fp_fp02a203.fits
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|On fiber AB order 0: S/N= 513.5
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|On fiber AB order 1: S/N= 561.6
...
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|On fiber AB order 34: S/N= 283.3
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|On fiber AB order 35: S/N= 149.9
HH:MM:SS.S - * |cal_DRIFT_RAW_spirou:2a203|On fiber AB estimated RV uncertainty on spectrum is 0.028 m
    /s
HH:MM:SS.S - * |cal_DRIFT_RAW_spirou:2a203|Number of fp_fp files found on directory = 2
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Reading file fp_fp03a203.fits
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Time from ref=0.09 h  - Drift mean=2.53 m/s - Flux ratio
    =0.99 = Nb Comsic=1218
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Reading file fp_fp04a203.fits
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Time from ref=0.18 h  - Drift mean=-10.13 m/s - Flux ratio
    =0.98 = Nb Comsic=1246
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Total drift Peak-to-Peak=12.596 m/s RMS=6.298 m/s in 0.18
    hour
HH:MM:SS.S -   |cal_DRIFT_RAW_spirou:2a203|Saving drift values of Fiber AB in fp_fp02a203_drift_AB.
    fits
HH:MM:SS.S - @ |python warning Line 980  warning reads: Card is too long, comment will be truncated.|
HH:MM:SS.S - * |cal_DRIFT_RAW_spirou:2a203|Recipe cal_DRIFT_RAW_spirou has been successfully completed
```

12.8.5 Interactive mode

12.8.5.1 cal_DRIFT_E2DS_spirou

In interactive mode three figures will also appear (see Figure 12.5).

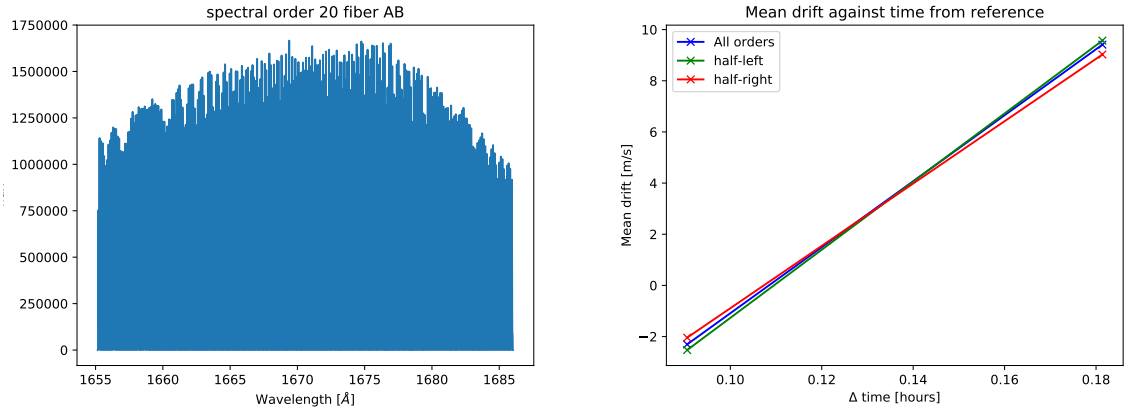


a b c

Figure 12.5 (a) The extract FP orders for spectral order 20, x-axis is wavelength and y-axis is extract flux. **(b)** Photon noise uncertainty versus spectral order. **(c)** Mean drift in m/s against the time from the reference extraction.

12.8.5.2 cal_DRIFTPEAK_E2DS_spirou

In interactive mode three figures will also appear (see Figure 12.6).



a b

Figure 12.6 (a) The extract FP orders for spectral order 20, x-axis is wavelength and y-axis is extract flux. **(b)** Mean drift in m/s against the time from the reference extraction for the left side of the orders, the right side of the orders and the whole image.

12.8.5.3 cal_DRIFT_RAW_spirou

In interactive mode three figures will also appear (see Figure 12.7).

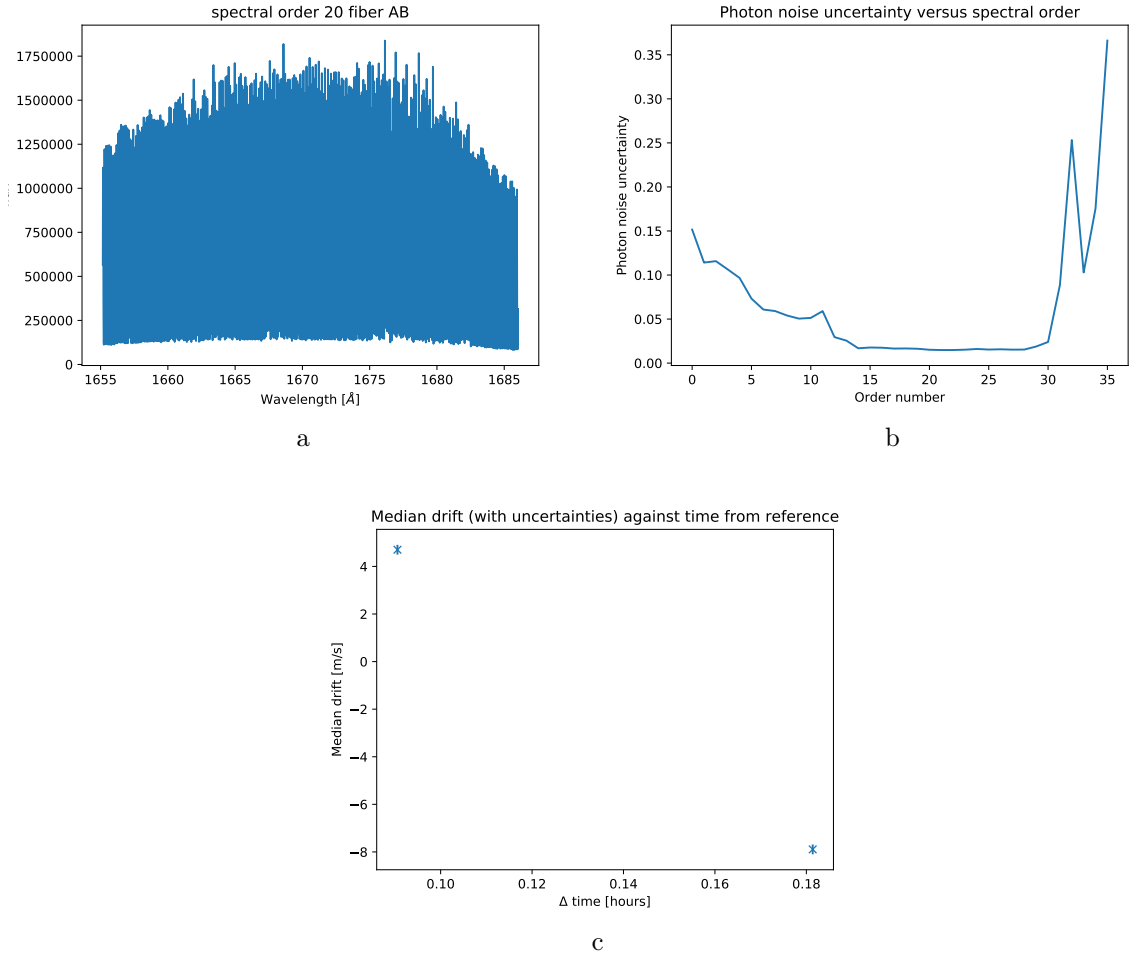


Figure 12.7 **(a)** The extract FP orders for spectral order 20, x-axis is wavelength and y-axis is extract flux. **(b)** Photon noise uncertainty versus spectral order. **(c)** Mean drift in m/s against the time from the reference extraction.

12.9 The cal_CCF recipe

Computes the CCF for a specific file using a CCF mask, target RV, CCF width and CCF step.

12.9.1 The inputs

The input of cal_CCF_E2DS_spirou is as follows:

```
>> cal_CCF_E2DS_spirou.py night_repository e2dsfile mask rv width step
```

for example

example

```
>> cal_CCF_E2DS_spirou.py 20170710 fp_fp02a203_e2ds_AB.fits UrNe.mas 0 10 0.1
```

or

Python/Ipython

```
import cal_CCF_E2DS_spirou
filename = 'fp_fp02a203_e2ds_AB.fits'
ccf_mask = 'UrNe.mas'
target_rv = 0.0
ccf_width = 10
ccf_step = 0.1
cal_CCF_E2DS_spirou.main(night_repository, filename=e2dsfile, mask=ccfmask,
                         rv=target_rv, width=ccf_width, step=ccf_step)
```

where:

- 'night_repository' defines arg_night_name
- 'e2dsfile' is the E2DS file to calculate the CCF for
- 'mask' is the name (or full path) of the CCF mask file.

> **Note:** If the CCF mask is not defined as a 'full system path' and cannot be found in the current working directory the recipe will search for it in the const_data_folder (defined in SpirouDRS.spirouConfig.spirouConst). If it is still not located an error will be raised and the recipe will exit. Therefore in most cases the CCF masks should be located in the const_data_folder directory.

- 'rv' is the target radial velocity (central velocity) in km/s
- 'width' is the CCF width (half range in velocity) in km/s
- 'step' is the CCF step in km/s

Filename suffixes allowed are:

- *e2ds_{fiber}.fits

where {fiber} is the fiber type (i.e. 'AB', 'A', 'B' or 'C').

12.9.2 The outputs

The outputs of cal_CCF_E2DS_spirou are as follows:

- corfile in form:

  {reduced_dir}/{date prefix}_{file}_ccf_{ccf_mask}.fits

- ccf_table_file in form:

  {reduced_dir}/{date prefix}_{file}_ccf_{ccf_mask}.tbl

where 'date prefix' is constructed from arg_night_name, the file name the 'e2dsfile', and {ccf_mask} is 'mask' from the inputs.
for example for reduced_dir `='/drs/data/reduced/20170710'`, `e2dsfile=fp_fp02a203_e2ds_AB.fits`, and `"UrNe.mas"` the output files would be:

- `/drs/data/reduced/20170710/fp_fp02a203_ccf_UrNe_AB.fits`
- `/drs/data/reduced/20170710/fp_fp02a203_ccf_UrNe_AB.tbl`

12.9.3 Summary of procedure

1. reads the E2DS file
2. reads the wavelength solution and the flat file
3. corrects for the flat
4. computes photon noise uncertainty
5. gets the CCF mask
6. calculates the CCF for each order
7. fits the CCF for each each
8. averages the CCFs over all orders
9. fits the average CCF over all orders
10. archives the ccf to .fits and .tbl format

Note: Blaze and Flat are currently set to arrays of one. There for no flat or blaze correction is currently done. Blaze file is currently not read and will need to be read to correct for it.

12.9.4 Example working run

An example run where everything worked is below:

example

```
>> cal_CCF_E2DS_spirou.py 20170710 fp_fp02a203_e2ds_AB.fits UrNe.mas 0 10 0.1
```

```
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (VERSION)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Now running : cal_CCF_E2DS_spirou with:
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|       -- e2dsfile=fp_fp02a203_e2ds_AB.fits
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|       -- ccf_mask=UrNe.mas
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|       -- target_rv=0.0
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|       -- ccf_width=10.0
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|       -- ccf_step=0.1
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|ICDP_NAME loaded from: /drs/INTROOT/config/constants_SPIROU.py
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Calibration file: 20170710_flat_flat02f10_badpixel.fits already
    exists - not copied
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Calibration file: 20170710_flat_dark02f10_blaze_AB.fits already
    exists - not copied
...
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Calibration file: spirou_wave_ini3.fits already exists - not
    copied
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Calibration file: 2017-10-11_21-32-17_hcone_hcone02c406_wave_C.
    fits already exists - not copied
HH:MM:SS.S - * |cal_CCF_E2DS_spirou|Now processing Image TYPE DRIFT with cal_CCF_E2DS_spirou recipe
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Reading File: /drs/data/reduced/20170710/fp_fp02a203_e2ds_AB.fits
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Reading wavelength solution
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Reading Flat-Field
HH:MM:SS.S - * |cal_CCF_E2DS_spirou|On fiber AB estimated RV uncertainty on spectrum is 0.025 m/s
HH:MM:SS.S - * |cal_CCF_E2DS_spirou|Template used for CCF computation: /drs/INTROOT/SpirouDRS/data/
    UrNe.mas
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Using RV template: UrNe.mas (2052 rows)
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Computing CCF at RV=    0.0 [km/s]
HH:MM:SS.S - * |cal_CCF_E2DS_spirou|Correlation: C=2.8[%] RV=-0.55305[km/s] FWHM=4.2613[km/s] maxcpp
    =422345.0
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Archiving CCF on file /drs/data/reduced/20170710/
    fp_fp02a203_ccf_UrNe_AB.tbl
HH:MM:SS.S -   |cal_CCF_E2DS_spirou|Archiving CCF on file fp_fp02a203_ccf_UrNe_AB.fits
HH:MM:SS.S - * |cal_CCF_E2DS_spirou|Recipe cal_CCF_E2DS_spirou has been successfully completed
```

12.9.5 Interactive mode

In interactive mode three figures will also appear (see Figure 12.8).

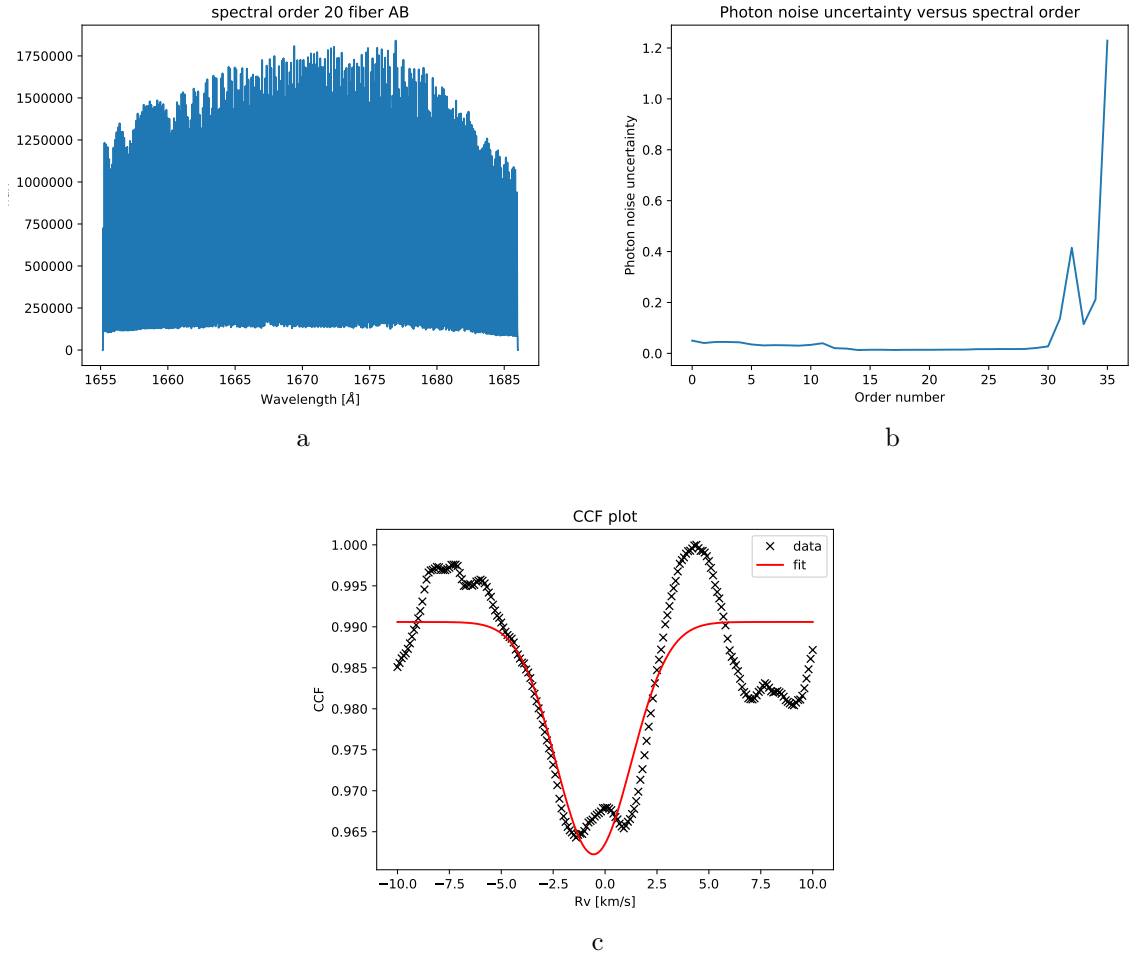


Figure 12.8 (a) The extract FP orders for spectral order 20, x-axis is wavelength and y-axis is extract flux. **(b)** Photon noise uncertainty versus spectral order. **(c)** The normalised CCF averaged across all orders, in red the fit to this CCF.

12.10 The validation recipe

This recipe validates that the DRS has be installed correctly and insures that all recipes should run. This is mentioned in Section 3.6.

> **Note:** The validation recipe does not protect against incorrect or missing constants and keywords. If all constants are defined as they were when installed and if all paths were set up correctly, then running the validation recipe should be enough to confirm that the DRS installed correctly. Again this does not protect against invalid files and other user inputs.

12.10.1 The inputs

The cal_validate_spirou recipe requires no input. Thus run the validation as follows:

```
>> cal_validate_spirou.py
```

or

Python/Ipython

```
import cal_validate_spirou
cal_validate_spirou.main()
```

One can also define an optional argument to put the validation recipe into debug mode with:

Python/Ipython

```
import cal_validate_spirou
cal_validate_spirou.main(DEBUG=1)
```

where a value of True or 1 runs the validation script in debugging mode. This lists all sub-module tests that are performed during the validation and thus allows problems to be identified more easily.

12.10.2 The outputs

There are no outputs to cal_validate_spirou other than printing to screen and the log file (see Section 12.10.4).

12.10.3 Summary of procedure

1. checks core module imports:
   - SpirouDRS
   - SpirouDRS.spirouConfig.spirouConfig
   - SpirouDRS.spirouCore.spirouCore
2. checks that configuration files can be read
3. checks recipe modules
   - SpirouDRS.spirouBACK.spirouBACK
   - SpirouDRS.spirouCDB.spirouCDB
   - SpirouDRS.spirouEXTOR.spirouEXTOR
   - SpirouDRS.spirouFLAT.spirouFLAT

- SpirouDRS.spirouImage.spirouImage
- SpirouDRS.spirouLOCOR.spirouLOCOR
- SpirouDRS.spirouRV.spirouRV
- SpirouDRS.spirouStartup.spirouStartup
- SpirouDRS.spirouTHORCA.spirouTHORCA

4. displays and prints the configuration file paths
5. confirms validation of the DRS installation

12.10.4 Example working run

An example run where everything worked is below:

```
>> cal_validate_spirou.py
```

```
 ****************************************
 *        VALIDATING DRS
 ****************************************

1) Running core module tests

2) Running config test

         Testing /drs/INTROOT/config/config.py

                Congratulations all paths in /drs/INTROOT/config/config.py set up correctly.

2) Running sub-module tests

4) Running recipe test

HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   || * SPIROU @(#) Geneva Observatory (VERSION)
HH:MM:SS.S -   || ****************************************
HH:MM:SS.S -   ||(dir_data_raw)      DRS_DATA_RAW=/drs/data/raw
HH:MM:SS.S -   ||(dir_data_reduc)    DRS_DATA_REDUC=/drs/data/reduced
HH:MM:SS.S -   ||(dir_calib_db)      DRS_CALIB_DB=/drs/data/calibDB
HH:MM:SS.S -   ||(dir_data_msg)      DRS_DATA_MSG=/drs/data/msg
HH:MM:SS.S -   ||(print_level)       PRINT_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(log_level)         LOG_LEVEL=all         %(error/warning/info/all)
HH:MM:SS.S -   ||(plot_graph)        DRS_PLOT=1            %(def/undef/trigger)
HH:MM:SS.S -   ||(used_date)         DRS_USED_DATE=undefined
HH:MM:SS.S -   ||(working_dir)       DRS_DATA_WORKING=/drs/data/tmp
HH:MM:SS.S -   ||                    DRS_INTERACTIVE is not set, running on-line mode
HH:MM:SS.S -   ||                    DRS_DEBUG is set, debug mode level:1
HH:MM:SS.S -   ||
HH:MM:SS.S -   ||Validation successful. DRS installed corrected.
```

12.11 The cal_HC recipe

Recipe not yet updated.

12.12 The cal_WAVE recipe

Recipe not yet updated.

12.13 The pol_spirou recipe

Recipe not yet updated.

Chapter 13

The DRS Module

13.1 Introduction

In the below sections the DRS sub modules are defined. They are called from the DRS in the following manner.

Python/Ipython

```
from SpirouDRS import spirouX
spirouX.MyFunction(arg1, arg2, kwarg1='value')   # Alias to internal function
spirouX.spirouY.my_function(arg1, arg2, kwarg1='value')   # internal function
```

Described in each section are the functions used in the recipes (defined in each sub-modules `__init__.py` file) and as such are in CamelCase (capitalized). The internal links are provided to locate the code for each function but only the CamelCased aliases should be used in recipes.

Functions used with in these sub-modules that are used internally but not called from recipes are not defined here but all functions are described as with those below and can be read in any interactive python or ipython terminal in the following way:

Python/Ipython

```
# in python
from SpirouDRS import spirouConfig
print(spirouConfig.CheckConfig.__doc__)
```

```
Check whether we have certain keys in dictionary
raises a Config Error if keys are not in params
:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        the keys defined in "keys" (else ConfigError raised)
:param keys: string or list of strings containing the keys to look for
:return None:
```

Python/Ipython

```
# in ipython
from SpirouDRS import spirouConfig
spirouConfig.CheckConfig?
```

```
Signature: spirouConfig.CheckConfig(params, keys)
Docstring:
Check whether we have certain keys in dictionary
raises a Config Error if keys are not in params
:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        the keys defined in "keys" (else ConfigError raised)
:param keys: string or list of strings containing the keys to look for
:return None:
File:      /drs/INTROOT/SpirouDRS/spirouConfig/spirouConfig.py
Type:      function
```

13.2 The spirouBACK module

13.2.1 BoxSmoothedMinMax

Defined in SpirouDRS.spirouBACK.spirouBACK .measure_box_min_max

Python/Ipython

```
from SpirouDRS import spirouBACK
spirouBACK.BoxSmoothedMinMax(y, size)
spirouBACK.spirouBACK.measure_box_min_max(y, size)
```

```
Measure the minimum and maximum pixel value for each pixel using a box which
surrounds that pixel by:  pixel-size to pixel+size.

Edge pixels (0-->size and (len(y)-size)-->len(y) are
set to the values for pixel=size and pixel=(len(y)-size)

:param y: numpy array (1D), the image
:param size: int, the half size of the box to use (half height)
             so box is defined from  pixel-size to pixel+size

:return min_image: numpy array (1D length = len(y)), the values
                   for minimum pixel defined by a box of pixel-size to
                   pixel+size for all columns
:return max_image: numpy array (1D length = len(y)), the values
                   for maximum pixel defined by a box of pixel-size to
                   pixel+size for all columns
```

13.2.2 MeasureBackgroundFF

Defined in SpirouDRS.spirouBACK.spirouBACK .measure_background_flatfield

Python/Ipython

```
from SpirouDRS import spirouBACK
spirouBACK.MeasureBackgroundFF(p, image)
spirouBACK.spirouBACK.measure_background_flatfield(p, image)
```

```
Measures the background of a flat field image - currently does not work
as need an interpolation function (see code)

:param p: parameter dictionary, ParamDict containing constants

        Must contain at least:
            IC_BKGR_WINDOW: int, Half-size of window for background
                            measurements
            GAIN: float, the gain of the image (from HEADER)
            SIGDET: float, the read noise of the image (from HEADER)
            log_opt: string, log option, normally the program name

:param image: numpy array (2D), the image to measure the background of

:return background: numpy array (2D), the background image (currently all
                    zeros) as background not implemented
:return xc: numpy array (1D), the box centres (x positions) used to create
            the background image
:return yc: numpy array (1D), the box centres (y positions) used to create
            the background image
:return minlevel: numpy array (2D), the 2 * size -th minimum pixel value
                  of each box for each pixel in the image
```

13.2.3 MeasureMinMax

Defined in SpirouDRS.spirouBACK.spirouBACK `measure_box_min_max`

Python/Ipython

```
from SpirouDRS import spirouBACK
spirouBACK.MeasureMinMax(y, size)
spirouBACK.spirouBACK.measure_box_min_max(y, size)
```

```
Measure the minimum and maximum pixel value for each pixel using a box which
surrounds that pixel by:  pixel-size to pixel+size.

Edge pixels (0-->size and (len(y)-size)-->len(y) are
set to the values for pixel=size and pixel=(len(y)-size)

:param y: numpy array (1D), the image
:param size: int, the half size of the box to use (half height)
             so box is defined from  pixel-size to pixel+size

:return min_image: numpy array (1D length = len(y)), the values
                   for minimum pixel defined by a box of pixel-size to
                   pixel+size for all columns
:return max_image: numpy array (1D length = len(y)), the values
                   for maximum pixel defined by a box of pixel-size to
                   pixel+size for all columns
```

13.2.4 MeasureMinMaxSignal

Defined in SpirouDRS.spirouBACK.spirouBACK `measure_min_max`

Python/Ipython

```
from SpirouDRS import spirouBACK
spirouBACK.MeasureMinMaxSignal(pp, y)
spirouBACK.spirouBACK.measure_min_max(pp, y)
```

```
Measure the minimum, maximum peak to peak values in y, the third biggest
pixel in y and the peak-to-peak difference between the minimum and
maximum values in y

:param pp: parameter dictionary, ParamDict containing constants
           Must contain at least:
               IC_LOCNBPIX: int, Half spacing between orders

:param y: numpy array (1D), the central column pixel values

:return miny: numpy array (1D length = len(y)), the values
              for minimum pixel defined by a box of pixel-size to
              pixel+size for all columns
:return maxy: numpy array (1D length = len(y)), the values
              for maximum pixel defined by a box of pixel-size to
              pixel+size for all columns
:return max_signal: float, the pixel value of the third biggest value
                    in y
:return diff_maxmin: float, the difference between maxy and miny
```

13.2.5 MeasureBkgrdGetCentPixs

Defined in SpirouDRS.spirouBACK.spirouBACK `measure_background_and_get_central_pixels`

Python/Ipython

```
from SpirouDRS import spirouBACK
spirouBACK.MeasureBkgrdGetCentPixs(pp, loc, image)
spirouBACK.spirouBACK.measure_background_and_get_central_pixels(pp, loc, image)
```

```
Takes the image and measure the background

:param pp: parameter dictionary, ParamDict containing constants
        Must contain at least:
            IC_OFFSET: int, row number of image to start processing at
            IC_CENT_COL: int, Definition of the central column
            IC_MIN_AMPLITUDE: int, Minimum amplitude to accept (in e-)
            IC_LOCSEUIL: float, Normalised amplitude threshold to accept
                         pixels for background calculation
            log_opt: string, log option, normally the program name
            DRS_DEBUG: int, Whether to run in debug mode
                           0: no debug
                           1: basic debugging on errors
                           2: recipes specific (plots and some code runs)
            DRS_PLOT: bool, Whether to plot (True to plot)

:param loc: parameter dictionary, ParamDict containing data

:param image: numpy array (2D), the image

:return ycc: the normalised values the central pixels

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            ycc: numpy array (1D), normalized central column of pixels
            mean_backgrd: float, 100 times the mean of the good background
                          pixels
            max_signal: float, the maximum value of the central column of
                        pixels
```

13.3 The spirouCDB module

13.3.1 CopyCDBfiles

Defined in SpirouDRS.spirouCDB.spirouCDB `.copy_files`

Python/Ipython

```
from SpirouDRS import spirouCDB
spirouCDB.CopyCDBfiles(p, header=None)
spirouCDB.spirouCDB.copy_files(p, header=None)
```

```
Copy the files from calibDB to the reduced folder
   p['DRS_DATA_REDUC']/p['arg_night_name']
based on the latest calibDB files from header, if there is not header file
use the parameter dictionary "p" to open the header in 'arg_file_names[0]'

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            calibDB: dictionary, the calibration database dictionary
            reduced_dir: string, the reduced data directory
                         (i.e. p['DRS_DATA_REDUC']/p['arg_night_name'])
            DRS_CALIB_DB: string, the directory that the calibration
                          files should be saved to/read from
            log_opt: string, log option, normally the program name
:param header: dictionary, the header dictionary created by
               spirouFITS.ReadImage

:return None:
```

13.3.2 GetAcqTime

Defined in SpirouDRS.spirouCDB.spirouCDB .get_acquisition_time

Python/Ipython

```
from SpirouDRS import spirouCDB
spirouCDB.GetAcqTime(p, header=None, kind='human', filename=None)
spirouCDB.spirouCDB.get_acquisition_time(p, header=None, kind='human', filename=None)
```

```
Get the acquision time from the header file, if there is not header file
use the parameter dictionary "p" to open the header in 'arg_file_names[0]'

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            arg_file_names: list, list of files taken from the command line
                            (or call to recipe function) must have at least
                            one string filename in the list
            log_opt: string, log option, normally the program name
            kw_ACQTIME_KEY: list, the keyword store for acquisition time
                            (string time-stamp)
                        [name, value, comment] = [string, object, string]
            kw_ACQTIME_KEY_UNIX: list, the keyword store fore acquisition
                                 time (float unixtime)
                        [name, value, comment] = [string, object, string]
:param header: dictionary or None, the header dictionary created by
               spirouFITS.ReadImage, if header is None code tries to get
               header from p['arg_file_names'][0]
:param kind: string, 'human' for 'YYYY-mm-dd-HH-MM-SS.ss' or 'unix'
             for time since 1970-01-01
:param filename: string or None, location of the file if header is None

:return acqtime: string, the human or unix time from header file
```

13.3.3 GetDatabase

Defined in SpirouDRS.spirouCDB.spirouCDB .get_database

Python/Ipython

```
from SpirouDRS import spirouCDB
spirouCDB.GetDatabase(p, max_time=None, update=False)
spirouCDB.spirouCDB.get_database(p, max_time=None, update=False)
```

```
Gets all entries from calibDB where unix time <= max_time.
If update is False then will first search for and use 'calibDB' in p
(if it exists)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            calibDB: dictionary, the calibration database dictionary
            log_opt: string, log option, normally the program name
:param max_time: str, maximum time allowed for all calibDB entries
                 format = (YYYY-MM-DD HH:MM:SS.MS)
:param update: bool, if False looks for "calibDB' in p, and if found does
               not load new database

:return c_database: dictionary, the calibDB database in form:
                c_database[key] = [dirname, filename]
    lines in calibDB must be in form:
        {key} {dirname} {filename} {human_time} {unix_time}

:return p: parameter dictionary, the updated parameter dictionary
        Adds the following:
            max_time_human: string, maximum time from "max_time"
            max_time_unix: float, maximum time from "max_time"
```

13.3.4 GetFile

Defined in SpirouDRS.spirouCDB.spirouCDB .get_file_name

Python/Ipython

```
from SpirouDRS import spirouCDB
spirouCDB.GetFile(p, key, hdr=None, filename=None)
spirouCDB.spirouCDB.get_file_name(p, key, hdr=None, filename=None)
```

```
Get the filename for "key" in the calibration database (for use when
the calibration database is not needed for more than one use and does
not exist already (i.e. called via spirouCDB.GetDatabase() )

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            arg_file_names: list, list of files taken from the command line
                            (or call to recipe function) must have at least
                            one string filename in the list
            calibDB: dictionary, the calibration database dictionary
            max_time_human: string, maximum time from "max_time"
            log_opt: string, log option, normally the program name
            reduced_dir: string, the reduced data directory
                         (i.e. p['DRS_DATA_REDUC']/p['arg_night_name'])
:param key: string, the key to look for in the calibration database
:param hdr: dict or None, the header dictionary to use to get the
            acquisition time, if hdr is None code tries to get
            header from p['arg_file_names'][0]
:param filename: string or None, if defined this is the filename returned
                 (means calibration database is not used)

:return read_file: string, the filename in calibration database for
                   "key" (selected via unix_time in calibDB)
```

13.3.5 PutFile

Defined in SpirouDRS.spirouCDB.spirouCDB `.put_file`

Python/Ipython

```
from SpirouDRS import spirouCDB
spirouCDB.PutFile(p, inputfile)
spirouCDB.spirouCDB.put_file(p, inputfile)
```

```
Copies the "inputfile" to the calibration database folder

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            DRS_CALIB_DB: string, the directory that the calibration
                          files should be saved to/read from
            log_opt: string, log option, normally the program name
:param inputfile: string, the input file path and file name

:return None:
```

13.3.6 UpdateMaster

Defined in SpirouDRS.spirouCDB.spirouCDB .update_datebase

Python/Ipython

```
from SpirouDRS import spirouCDB
spirouCDB.UpdateMaster(p, keys, filenames, hdrs, timekey=None)
spirouCDB.spirouCDB.update_datebase(p, keys, filenames, hdrs, timekey=None)
```

```
Updates (or creates) the calibDB with an entry or entries in the form:

    {key} {arg_night_name} {filename} {human_time} {unix_time}

where arg_night_name comes from p["arg_night_name']
where "human_time" and "unix_time" come from the filename headers (hdrs)
    using HEADER_KEY = timekey (or "ACQTIME1" if timekey=None)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            arg_night_name: string, the folder within data raw directory
                            containing files (also reduced directory) i.e.
                            /data/raw/20170710 would be "20170710"
            log_opt: string, log option, normally the program name
            kw_ACQTIME_KEY: list, the keyword store for acquisition time
                            (string time-stamp)
                        [name, value, comment] = [string, object, string]
            kw_ACQTIME_KEY_UNIX: list, the keyword store fore acquisition
                                 time (float unixtime)
:param keys: string or list of strings, keys to add to the calibDB
:param filenames: string or list of strings, filenames to add to the
                  calibDB, if keys is a list must be a list of same length
                  as "keys"
:param hdrs: dictionary or list of dictionaries, header dictionary/
             dictionaries to find 'timekey' in - the acquisition time,
             if keys is a list must be a list of same length  as "keys"
:param timekey: string, key to find acquisition time in header "hdr" if
                None defaults to the program default ('ACQTIME1')

:return None:
```

13.4 The spirouConfig module

13.4.1 ConfigError

Defined in SpirouDRS.spirouConfig.spirouConfig `.spirouConfig.ConfigError`. See Section 7.2.5 for details on use.

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.ConfigError(message='', level='error')
spirouConfig.spirouConfig.ConfigError(message='', level='error')
```

```
Custom Config Error class for passing errors and exceptions to the log.

Inherits:

  spirouConfig.spirouConfigFile.ConfigException

Methods:

  __init__(self, message=None, level=None)

    Constructor for ConfigError sets message to self.message and level to
    self.level

    if key is not None defined self.message reads "key [key] must be
    defined in config file (located at [config_file]

    if config_file is None then default config file is used in its place

    :param message: list or string, the message to print in the error
    :param level: string, level (for logging) must be key in TRIG key above
                  default = all, error, warning, info or graph

  __repr__(self)
      String representation of ConfigError

      :return message: string, the message assigned in constructor

  __str__(self)
      String printing of ConfigError

      :return message: string, the message assigned in constructor
```

13.4.2 CheckCparams

Defined in SpirouDRS.spirouConfig.spirouConfig .spirouConfig.check_params

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.CheckCparams(p)
spirouConfig.spirouConfig.check_params(p)
```

```
Check the parameter dictionary has certain required values, p must contain
at the very least keys 'DRS_ROOT' and 'TDATA'

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        DRS_ROOT: string, the installation root directory
        TDATA: string, the data root directory

:return p: parameter dictionary, the updated parameter dictionary
        Adds the following (if not already in "p"):
            DRS_DATA_RAW: string, the directory that the raw data should
                          be saved to/read from
                            should be saved to/read from
            DRS_DATA_MSG: string, the directory that the log messages
                          should be saved to
            DRS_CALIB_DB: string, the directory that the calibration
                          files should be saved to/read from
            DRS_CONFIG: string, the directory that contains the config files
            DRS_MAN: string, the directory the manual files are stored in
            DRS_PLOT: bool, whether to plot or not
            DRS_DATA_WORKING: string, the working data directory (temporary
                              storage folder)
      DRS_UCONFIG: string, the directory that contains the user config
                   files
            DRS_USED_DATE: string, ???
            DRS_DEBUG: int, sets the debug level
                           0: no debug
                           1: basic debugging on errors
                           2 : recipes specific (plots and some code runs)
            DRS_INTERACTIVE: bool, sets whether plots are interactive or
                             static
            PRINT_LEVEL: string, sets the print level
                                 'all' - to print all events
                                 'info' - to print info/warning/error events
                                 'warning' - to print warning/error events
                                 'error' - to print only error events
            LOG_LEVEL: string, sets the logging level
                                 'all' - to print all events
                                 'info' - to print info/warning/error events
                                 'warning' - to print warning/error events
                                 'error' - to print only error events
    Only updated if not already defined in primary config file
    (i.e. in "p")
```

13.4.3 CheckConfig

Defined in SpirouDRS.spirouConfig.spirouConfig `.spirouConfig.check_config`

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.CheckConfig(params, keys)
spirouConfig.check_config(params, keys)
```

```
Check whether we have certain keys in dictionary
raises a Config Error if keys are not in params

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        the keys defined in "keys" (else ConfigError raised)

:param keys: string or list of strings containing the keys to look for

:return None:
```

13.4.4 ExtractDictParams

Defined in SpirouDRS.spirouConfig.spirouConfig .spirouConfig.extract_dict_params

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.ExtractDictParams(pp, suffix, fiber, merge=False)
spirouConfig.spirouConfig.extract_dict_params(pp, suffix, fiber, merge=False)
```

```
Extract parameters from parameter dictionary "pp" with a certain suffix
"suffix" (whose value must be a dictionary containing fibers) add them
to a new parameter dictionary (if merge=False) if merge is True then
add them back to the "pp" parameter dictionary

:param pp: parameter dictionary, ParamDict containing constants
           If pp has keys with "suffix" they are extracted and used
           if there are no keys with "suffix" then this function does
           nothing other than add "fiber" to "p"

:param suffix: string, the suffix string to look for in "pp", all keys
               must have values that are dictionaries containing (at least)
               the key "fiber"

               i.e. in the constants file:
               param1_suffix = {'AB'=1, 'B'=2, 'C'=3}
               param2_suffix = {'AB'='yes', 'B'='no', 'C'='no'}
               param3_suffix = {'AB'=True, 'B'=False, 'C'=True}

:param fiber: string, the key within the value dictionary to look for
              (i.e. in the above example 'AB' or 'B' or 'C' are valid
:param merge: bool, if True merges new keys with "pp" else provides
              a new parameter dictionary with all parameters that had the
              suffix in (with the suffix removed)

:return p: parameter dictionary, the updated parameter dictionary

           if merge is True "pp" is returned with the new constants
           added, else a new parameter dictionary is returned

            i.e. for the above example return is the following:

                "fiber" = "AB"

           ParamDict(param1=1, param2='yes', param3=True)
```

13.4.5 GetKeywordArguments

Defined in SpirouDRS.spirouConfig.spirouConfig .spirouKeywords.get_keywords

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.GetKeywordArguments(pp=None)
spirouConfig.spirouKeywords.get_keywords(pp=None)
```

```
Get keywords defined in spirouKeywords.USE_KEYS
(must be named exactly as in USE_KEYS list)

:param pp: parameter dictionary or None, if not None then keywords are added
           to the specified ParamDict else a new ParamDict is created

:return pp: if pp is None returns a new dictionary of keywords
            else adds USE_KEYS as keys with value = eval(key)
```

13.4.6 GetKeywordValues

Defined in SpirouDRS.spirouConfig.spirouConfig .spirouKeywords.get_keyword_values_from_header

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.GetKeywordValues(pp, hdict, keys, filename=None)
spirouConfig.spirouKeywords.get_keyword_values_from_header(pp, hdict, keys, filename=None)
```

```
Gets a keyword or keywords from a header or dictionary

:param pp: parameter dictionary, ParamDict containing constants
            if "key" (element in "keys") is in pp and it is a
            keyword list then this is used as the key instead of "key"

:param hdict: dictionary, raw dictionary or FITS rec header file containing
              all the keys in "keys" (spirouConfig.ConfigError raised if
              any key does not exist)
:param keys: list of strings or list of lists, the keys to find in "hdict"
             OR a list of keyword lists ([key, value, comment])
:param filename: string or None, if defined when an error is caught the
                 filename is logged, this filename should be where the
                 fits rec header is from (or where the dictionary was
                 compiled from) - if not from a file this should be left
                 as None

:return values: list, the values in the header for the keys
                (size = len(keys))
```

13.4.7 GetAbsFolderPath

Defined in SpirouDRS.spirouConfig.spirouConfig .spirouConfigFile.get_relative_folder

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.GetAbsFolderPath(package, folder)
spirouConfig.spirouConfigFile.get_relative_folder(package, folder)
```

```
Get the absolute path of folder defined at relative path
folder from package

:param package: string, the python package name
:param folder: string, the relative path of the configuration folder

:return data: string, the absolute path and filename of the default config
              file
```

13.4.8 GetDefaultConfigFile

Defined in SpirouDRS.spirouConfig.spirouConfig .spirouConfigFile.get_default_config_file

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.GetDefaultConfigFile(package, configfolder, configfile)
spirouConfig.spirouConfigFile.get_default_config_file(package, configfolder, configfile)
```

```
Get the absolute path for the  default config file defined in
configfile at relative path configfolder from package

:param package: string, the python package name
:param configfolder: string, the relative path of the configuration folder
:param configfile: string, the name of the configuration file

:return config_file: string, the absolute path and filename of the
                     default config file
```

13.4.9 LoadConfigFromFile

Defined in SpirouDRS.spirouConfig.spirouConfig .spirouConfig.load_config_from_file

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.LoadConfigFromFile(p, key, required=False, logthis=False)
spirouConfig.spirouConfig.load_config_from_file(p, key, required=False, logthis=False)
```

```
Load a secondary level configuration file filename = "key", this requires
the primary config file to already be loaded into "p"
(i.e. p['DRS_CONFIG'] and p[key] to be set)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            DRS_CONFIG: string, the directory that contains the config files
            "key": string, the key to access the config file name
                   (key variable defined by "key")

:param key: string, the key to access the config file name for (in "p")
:param required: bool, if required is True then the secondary config file
                 is required for the DRS to run and a ConfigError is raised
                 (program exit)
:param logthis: bool, if True loading of this config file is logged to
                screen/log file

:return p: parameter, dictionary, the updated parameter dictionary with
           the secondary configuration files loaded into it as key/value
           pairs
```

13.4.10 ParamDict

Defined in SpirouDRS.spirouConfig.spirouConfig .spirouConfig.ParamDict
See Section 7.2.4 for details on use.

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.ParamDict(self, *arg, **kw)
spirouConfig.spirouConfig.ParamDict(self, *arg, **kw)
```

```
Custom dictionary class to retain source of a parameter (added via setSource,
retreived via getSource). String keys are case insensitive.

Interits:

  dict

Methods:

  __init__(self, *arg, **kw)
          Constructor for parameter dictionary, calls dict.__init__
          i.e. the same as running dict(*arg, *kw)

          :param arg: arguments passed to dict
          :param kw: keyword arguments passed to dict

  __getitem__(self, key)
          Method used to get the value of an item using "key"
          used as x.__getitem__(y) <==> x[y]
          where key is case insensitive

          :param key: string, the key for the value returned (case insensitive)

          :return value: object, the value stored at position "key"

  __setitem__(self, key, value, source=None)
          Sets an item wrapper for self[key] = value
          :param key: string, the key to set for the parameter
          :param value: object, the object to set (as in dictionary) for the
                        parameter
          :param source: string, the source for the parameter
          :return:

  __contains__(self, key)
          Method to find whether ParamDict instance has key="key"
          used with the "in" operator
          if key exists in ParamDict True is returned else False is returned

          :param key: string, "key" to look for in ParamDict instance

          :return bool: True if ParamDict instance has a key "key", else False
```

```
ParamDict Methods (continued I):

  __delitem__(self, key)
          Deletes the "key" from ParamDict instance, case insensitive

          :param key: string, the key to delete from ParamDict instance,
                      case insensitive

          :return None:

  get(self, key, default=None)
          Overrides the dictionary get function
          If "key" is in ParamDict instance then returns this value, else
          returns "default" (if default returned source is set to None)
          key is case insensitive

          :param key: string, the key to search for in ParamDict instance
                      case insensitive
          :param default: object or None, if key not in ParamDict instance this
                          object is returned

          :return value: if key in ParamDict instance this value is returned else
                         the default value is returned (None if undefined)

  set_source(self, key, source)
          Set a key to have sources[key] = source

          raises a ConfigError if key not found

          :param key: string, the main dictionary string
          :param source: string, the source to set

          :return None:

  append_source(self, key, source)
          Adds source to the source of key (appends if exists)
          i.e. sources[key] = oldsource + source

          :param key: string, the main dictionary string
          :param source: string, the source to set

          :return None:

  set_sources(self, keys, sources)
          Set a list of keys sources

          raises a ConfigError if key not found

          :param keys: list of strings, the list of keys to add sources for
          :param sources: string or list of strings or dictionary of strings,
                          the source or sources to add,
                          if a dictionary source = sources[key] for key = keys[i]
                          if list source = sources[i]  for keys[i]
                          if string all sources with these keys will = source

          :return None:
```

```
ParamDict Methods (continued II):

  append_sources(self, keys, sources)
          Adds list of keys sources (appends if exists)

          raises a ConfigError if key not found

          :param keys: list of strings, the list of keys to add sources for
          :param sources: string or list of strings or dictionary of strings,
                          the source or sources to add,
                          if a dictionary source = sources[key] for key = keys[i]
                          if list source = sources[i]  for keys[i]
                          if string all sources with these keys will = source

          :return None:

  set_all_sources(self, source)
          Set all keys in dictionary to this source

          :param source: string, all keys will be set to this source

          :return None:

  append_all_sources(self, source)
          Sets all sources to this "source" value

          :param source: string, the source to set

          :return None:

  get_source(self, key)
          Get a source from the parameter dictionary (must be set)

          raises a ConfigError if key not found

          :param key: string, the key to find (must be set)

          :return source: string, the source of the parameter

  source_keys(self)
          Get a dict_keys for the sources for this parameter dictionary
          order the same as self.keys()

          :return sources: values of sources dictionary

  source_values(self)
          Get a dict_values for the sources for this parameter dictionary
          order the same as self.keys()

          :return sources: values of sources dictionary
```

```
ParamDict Methods (continued III):

  startswith(self, substring)
          Return all keys that start with this sub-string

          :param substring: string, the prefix that the keys start with

          :return keys: list of strings, the keys with this sub-string at the
    start

  __capitalise_keys__(self)
          Capitalizes all keys in ParamDict (used to make ParamDict case
          insensitive), only if keys entered are strings

          :return None:

  __capitalise_key__(self, key)
        Capitalizes "key" (used to make ParamDict case insensitive), only if
        key is a string

        :param key: string or object, if string then key is capitalized else
                    nothing is done

        :return key: capitalized string (or unchanged object)
```

13.4.11 ReadConfigFile

Defined in SpirouDRS.spirouConfig.spirouConfig .spirouConfig.read_config_file

Python/Ipython

```
from SpirouDRS import spirouConfig
spirouConfig.ReadConfigFile(config_file=None)
spirouConfig.spirouConfig.read_config_file(config_file=None)
```

```
Read config file wrapper (push into ParamDict)

:param config_file: string or None, the config_file name, if none uses
                    PACKAGE/CONFIGFOLDER and CONFIG_FILE to get config
                    file name
:return params: parameter dictionary with key value pairs from config file
:return warning_messages: list, a list of warning messages to pipe to the logger
```

13.5 The spirouCore module

13.5.1 wlog

Defined in SpirouDRS.spirouCore.spirouCore `.spirouLog.logger`, also aliased in code to 'WLOG'. See Section 7.2.2 for usage details.

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.wlog(key='', option='', message='', printonly=False, logonly=False)
spirouCore.spirouLog.logger(key='', option='', message='', printonly=False, logonly=False)
```

```
Parses a key (error/warning/info/graph), an option and a message to the
stdout and the log file.

keys are controlled by "spirouConfig.Constants.LOG_TRIG_KEYS()"
printing to screen is controlled by "PRINT_LEVEL" constant (config.py)
printing to log file is controlled by "LOG_LEVEL" constant (config.py)
based on the levels described in "spirouConfig.Constants.WRITE_LEVEL"

:param key: string, either "error" or "warning" or "info" or graph, this
            gives a character code in output
:param option: string, option code
:param message: string or list of strings, message to display or messages
                to display (1 line for each message in list)
:param printonly: bool, print only do not save to log (default = False)
:param logonly: bool, log only do not save to log (default = False)

output to stdout/log is as follows:

    HH:MM:SS.S - CODE |option|message

time is output in UTC to nearest .1 seconds

:return None:
```

13.5.2 WarnLog

Defined in SpirouDRS.spirouCore.spirouCore .spirouLog.warninglogger

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.warnlog(w, funcname=None)
spirouCore.spirouLog.warninglogger(w, funcname=None)
```

```
Warning logger - takes "w" - a list of caught warnings and pipes them on
to the log functions.
If "funcname" is not None then "funcname" is printed with the line reference
(intended to be used to identify the code/function/module warning was
generated in)

to catch warnings use the following:

>>> import warnings
>>> with warnings.catch_warnings(record=True) as w:
>>>     code_to_generate_warnings()
>>> warninglogger(w, 'some name for logging')

:param w: list of warnings, the list of warnings from
          warnings.catch_warnings
:param funcname: string or None, if string then also pipes "funcname" to the
                 warning message (intended to be used to identify the code/
                 function/module warning was generated in)
:return:
```

13.5.3 GaussFunction

Defined in SpirouDRS.spirouCore.spirouCore .spirouMath.gauss_function

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.GaussFunction(x, a, x0, sigma, dc)
spirouCore.spirouMath.gauss_function(x, a, x0, sigma, dc)
```

```
A standard 1D Gaussian function (for fitting against)]=

:param x: numpy array (1D), the x data points
:param a: float, the amplitude
:param x0: float, the mean of the Gaussian
:param sigma: float, the standard deviation (FWHM) of the Gaussian
:param dc: float, the constant level below the Gaussian

:return gauss: numpy array (1D), size = len(x), the output Gaussian
```

13.5.4 GetTimeNowUnix

Defined in SpirouDRS.spirouCore.spirouCore .spirouMath.get_time_now_unix

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.GetTimeNowUnix(zone='UTC')
spirouCore.spirouMath.get_time_now_unix(zone='UTC')
```

```
Get the unix_time now.

Default is to return unix_time in UTC/GMT time

:param zone: string, if UTC displays the time in UTC else displays local
             time

:return unix_time: float, the unix_time
```

13.5.5 GetTimeNowString

Defined in SpirouDRS.spirouCore.spirouCore .spirouMath.get_time_now_string

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.GetTimeNowString(fmt=TIME_FMT, zone='UTC')
spirouCore.spirouMath.get_time_now_string(fmt=TIME_FMT, zone='UTC')
```

```
Get the time now (in string format = "fmt")

Default is to return string time in UTC/GMT time

    Commonly used format codes:

    %Y  Year with century as a decimal number.
    %m  Month as a decimal number [01,12].
    %d  Day of the month as a decimal number [01,31].
    %H  Hour (24-hour clock) as a decimal number [00,23].
    %M  Minute as a decimal number [00,59].
    %S  Second as a decimal number [00,61].
    %z  Time zone offset from UTC.
    %a  Locale's abbreviated weekday name.
    %A  Locale's full weekday name.
    %b  Locale's abbreviated month name.
    %B  Locale's full month name.
    %c  Locale's appropriate date and time representation.
    %I  Hour (12-hour clock) as a decimal number [01,12].
    %p  Locale's equivalent of either AM or PM.


:param fmt: string, the format code for the returned time
:param zone: string, if UTC displays the time in UTC else displays local
             time

:return stringtime: string, the time in a string in format = "fmt"
```

13.5.6 PrintLog

Defined in SpirouDRS.spirouCore.spirouCore .spirouMath.printlog

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.PrintLog(message, key='all')
spirouCore.spirouCore.printlog(message, key='all')
```

```
print message to stdout (if level is correct - set by PRINT_LEVEL)
is coloured unless spirouConfig.Constants.COLOURED_LOG() is False

:param message: string, the formatted log line to write to stdout
:param key: string, either "error" or "warning" or "info" or "graph" or
            "all", this gives a character code in output

:return None:
```

13.5.7 PrintColours

Defined in SpirouDRS.spirouCore.spirouCore .spirouMath.printlog

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.PrintColour(key='all', func_name=None)
spirouCore.spirouCore.printcolour(key='all', func_name=None)
```

```
Get the print colour (start and end) based on "key".
This should be used as follows:
    >> c1, c2 = printcolour(key='all')
    >> print(c1 + message + c2)

:param key: string, either "error" or "warning" or "info" or "graph" or
            "all", this gives a character code in output
:param func_name: string or None, if not None then defines the function to
                  report in the error
:return colour1: string or None, if key is found and we are using coloured
                 log returns the starting colour, if not returns empty
                 string if key is not accepted does not print
:return colour2: string or None, if key is found and we are using coloured
                 log returns the ending colour, if not returns empty
                 string if key is not accepted does not print
```

13.5.8 Unix2stringTime

Defined in SpirouDRS.spirouCore.spirouCore .spirouMath.unixtime2stringtime

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.Unix2stringTime(ts, fmt=DATE_FMT, zone='UTC')
spirouCore.spirouMath.unixtime2stringtime(ts, fmt=DATE_FMT, zone='UTC')
```

```
Convert a unix time (seconds since  1970-01-01 00:00:00 GMT) into a
string in format "fmt". Currently supported time-zones are UTC and local
(i.e. your current time zone).

Default is to return string time in UTC/GMT time

Commonly used format codes:

    %Y  Year with century as a decimal number.
    %m  Month as a decimal number [01,12].
    %d  Day of the month as a decimal number [01,31].
    %H  Hour (24-hour clock) as a decimal number [00,23].
    %M  Minute as a decimal number [00,59].
    %S  Second as a decimal number [00,61].
    %z  Time zone offset from UTC.
    %a  Locale's abbreviated weekday name.
    %A  Locale's full weekday name.
    %b  Locale's abbreviated month name.
    %B  Locale's full month name.
    %c  Locale's appropriate date and time representation.
    %I  Hour (12-hour clock) as a decimal number [01,12].
    %p  Locale's equivalent of either AM or PM.

:param ts: float or int, the unix time (seconds since 1970-01-01 00:00:00
           GMT)
:param fmt: string, the format of the string to convert
:param zone: string, the time zone for the input string
                     (currently supported =  "UTC" or "local")

:return stringtime: string, the time in format "fmt"
```

13.5.9 String2unixTime

Defined in SpirouDRS.spirouCore.spirouCore .spirouMath.stringtime2unixtime

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.String2unixTime(string, fmt=DATE_FMT, zone='UTC')
spirouCore.spirouMath.stringtime2unixtime(string, fmt=DATE_FMT, zone='UTC')
```

```
Convert a string in format "fmt" into a float Unix time (seconds since
1970-01-01 00:00:00 GMT). Currently supported time-zones are UTC and local
(i.e. your current time zone).

Default is to assume string is in UTC/GMT time

Commonly used format codes:

    %Y  Year with century as a decimal number.
    %m  Month as a decimal number [01,12].
    %d  Day of the month as a decimal number [01,31].
    %H  Hour (24-hour clock) as a decimal number [00,23].
    %M  Minute as a decimal number [00,59].
    %S  Second as a decimal number [00,61].
    %z  Time zone offset from UTC.
    %a  Locale's abbreviated weekday name.
    %A  Locale's full weekday name.
    %b  Locale's abbreviated month name.
    %B  Locale's full month name.
    %c  Locale's appropriate date and time representation.
    %I  Hour (12-hour clock) as a decimal number [01,12].
    %p  Locale's equivalent of either AM or PM.

:param string: string, the time string to convert
:param fmt: string, the format of the string to convert
:param zone: string, the time zone for the input string
                      (currently supported =  "UTC" or "local")

:return unix_time: float, Unix time (seconds since 1970-01-01 00:00:00 GMT)
```

13.5.10 sPlt

Defined in SpirouDRS.spirouCore.spirouCore `.spirouPlot` (alias to the plotting module).

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.sPlt
spirouCore.spirouPlot
```

```
Spirou Plotting functions available:

start_interactive_session(interactive=False)
    Start interactive plot session, if required and if
    spirouConfig.Constants.INTERACITVE_PLOTS_ENABLED() is True

    :param interactive: bool, if True start interactive session

    :return None:

end_interactive_session(interactive=False)
    End interactive plot session, if required and if
    spirouConfig.Constants.INTERACITVE_PLOTS_ENABLED() is True

    :param interactive: bool, if True end interactive session

    :return None:


define_figure
    Define a figure number (mostly for use in interactive mode)

    :param num: int, a figure number

    :return figure: plt.figure instance

closeall()
    Close all matplotlib plots currently open

    :return None:

And all plotting functions from specific recipes.
```

13.5.11 Getll

Defined in SpirouDRS.spirouCore.spirouCore .spirouMath.get_ll_from_coefficients

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.Getll(params, nx, nbo)
spirouCore.spirouMath.get_ll_from_coefficients(params, nx, nbo)
```

```
Use the coefficient matrix "params" to construct fit values for each order
(dimension 0 of coefficient matrix) for values of x from 0 to nx
(interger steps)

:param params: numpy array (2D), the coefficient matrix
               size = (number of orders x number of fit coefficients)

:param nx: int, the number of values and the maximum value of x to use
           the coefficients for, where x is such that

            yfit = p[0]*x**(N-1) + p[1]*x**(N-2) + ... + p[N-2]*x + p[N-1]

            N = number of fit coefficients
            and p is the coefficients for one order
            (i.e. params = [ p_1, p_2, p_3, p_4, p_5, ... p_nbo]

:param nbo: int, the number of orders to use

:return ll: numpy array (2D): the yfit values for each order
            (i.e. ll = [yfit_1, yfit_2, yfit_3, ..., yfit_nbo] )
```

13.5.12 Getdll

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.get_dll_from_coefficients

Python/Ipython

```
from SpirouDRS import spirouCore
spirouCore.Getdll(params, nx, nbo)
spirouCore.spirouMath.get_dll_from_coefficients(params, nx, nbo)
```

```
Derivative of the coefficients, using the coefficient matrix "params"
to construct the derivative of the fit values for each order
(dimension 0 of coefficient matrix) for values of x from 0 to nx
(interger steps)

:param params: numpy array (2D), the coefficient matrix
               size = (number of orders x number of fit coefficients)

:param nx: int, the number of values and the maximum value of x to use
           the coefficients for, where x is such that

            yfit = p[0]*x**(N-1) + p[1]*x**(N-2) + ... + p[N-2]*x + p[N-1]

            dyfit = p[0]*(N-1)*x**(N-2) + p[1]*(N-2)*x**(N-3) + ... +
                    p[N-3]*x + p[N-2]

            N = number of fit coefficients
            and p is the coefficients for one order
            (i.e. params = [ p_1, p_2, p_3, p_4, p_5, ... p_nbo]

:param nbo: int, the number of orders to use

:return ll: numpy array (2D): the yfit values for each order
            (i.e. ll = [dyfit_1, dyfit_2, dyfit_3, ..., dyfit_nbo] )
```

13.6 The spirouEXTOR module

13.6.1 Extraction

Defined in SpirouDRS.spirouEXTOR.spirouEXTOR .spirouEXTOR.extraction_wrapper

Python/Ipython

```
from SpirouDRS import spirouEXTOR
spirouEXTOR.Extraction(p, loc, image, rnum, mode=0, order_profile=None, **kwargs)
spirouEXTOR.spirouEXTOR.extract_wrapper(p, loc, image, rnum, mode=0, order_profile=None, **kwargs)
```

```
Extraction wrapper - takes in p, loc, image, rnum (order number) and mode
and decides which extract method to run (based on mode), checks all
required inputs are present and valid else displays a helpful error.

:param p: parameter dictionary, ParamDict containing constants
        Required parameters depends on mode selected
:param loc: parameter dictionary, ParamDict containing data
        Required parameters depends on mode selected

:param image: numpy array (2D), the image
:param rnum: int, the order number for this iteration
:param mode: string, the extraction mode, currently supported modes are:

        0 - Simple extraction
                (function = spirouEXTOR.extract_const_range)

        1 - weighted extraction
                (function = spirouEXTOR.extract_weight)

        2 - tilt extraction
                (function = spirouEXTOR.extract_tilt)

        3a - tilt weight extraction (old 1)
                (function = spirouEXTOR.extract_tilt_weight)

        3b - tilt weight extraction 2 (old)
                (function = spirouEXTOR.extract_tilt_weight_old2)

        3c - tilt weight extraction 2
                (function = spirouEXTOR.extract_tilt_weight2)

        3d - tilt weight extraction 2 (cosmic correction)
                (function = spirouEXTOR.extract_tilt_weight2cosm)

:param order_profile: numpy array (2D) or None, the order profile image.
                      Can be none if not used by extraction method.

:param kwargs: keyword arguments, any value used from "p" or "loc" can be
               overwritten with this parameter (which parameters depends on
               which mode selected)
:return: the outputs of extraction method function (see mode functions)
```

13.6.2 ExtractABOrderOffset

Defined in SpirouDRS.spirouEXTOR.spirouEXTOR .spirouEXTOR.extract_AB_order

Python/Ipython

```
from SpirouDRS import spirouEXTOR
spirouEXTOR.ExtractABOrderOffset(pp, loc, image, rnum)
spirouEXTOR.spirouEXTOR.extract_AB_order(pp, loc, image, rnum)
```

```
Perform the extraction on the AB fibers separately using the summation
over constant range

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            IC_CENT_COL: int, the column number (x-axis) of the central
                         column
            IC_FACDEC: float, the offset multiplicative factor for width
            IC_EXTOPT: int, the extraction option
            gain: float, the gain of the image
            IC_EXTNBSIG: float, distance away from centre to extract
                         out to +/- (in rows or y-axis direction)

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            ass: numpy array (2D), the fit coefficients array for
                  the widths fit
                  shape = (number of orders x number of fit coefficients)
            acc: numpy array (2D), the fit coefficients array for
                  the centres fit
                  shape = (number of orders x number of fit coefficients)


:param image: numpy array (2D), the image
:param rnum: int, the order number for this iteration

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            offset: numpy array (1D), the centre values with the
                    offset in 'IC_CENT_COL' added
            cent1: numpy array (2D), the extraction for A, updated is
                   the order "rnum"
            nbcos: int, 0 (constant)
            cent2: numpy array (2D), the extraction for B, updated is
                   the order "rnum"
```

13.6.3 GetValidOrders

Defined in SpirouDRS.spirouEXTOR.spirouEXTOR .spirouEXTOR.get_valid_orders

Python/Ipython

```
from SpirouDRS import spirouEXTOR
spirouEXTOR.GetValidOrders(p, loc)
spirouEXTOR.spirouEXTOR.get_valid_orders(p, loc)
```

```
Get valid order range (from min to max) from constants

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        EXT_START_ORDER: int or None, the order number to start with, if
                         None this is set to zero
        EXT_END_ORDER: int or None, the order number to end with, if None
                       this is set to the last order number
:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            number_orders: int, the number of orders in reference spectrum
:return valid_ordesr: list, all integer values between the start order and
                      end order
```

13.6.4 GetExtMethod

Defined in SpirouDRS.spirouEXTOR.spirouEXTOR .spirouEXTOR.get_extraction_method

Python/Ipython

```
from SpirouDRS import spirouEXTOR
spirouEXTOR.GetExtMethod(p, mode)
spirouEXTOR.spirouEXTOR.get_extraction_method(p, mode)
```

```
Get the extraction method key and function

:param mode: string, the mode

    0 - Simple extraction
            (function = spirouEXTOR.extract_const_range)

    1 - weighted extraction
            (function = spirouEXTOR.extract_weight)

    2 - tilt extraction
            (function = spirouEXTOR.extract_tilt)

    3a - tilt weight extraction (old 1)
            (function = spirouEXTOR.extract_tilt_weight)

    3b - tilt weight extraction 2 (old)
            (function = spirouEXTOR.extract_tilt_weight_old2)

    3c - tilt weight extraction 2
            (function = spirouEXTOR.extract_tilt_weight2)

    3d - tilt weight extraction 2 (cosmic correction)
            (function = spirouEXTOR.extract_tilt_weight2cosm)

:return: string the mode and function
```

13.7 The spirouFLAT module

13.7.1 MeasureBlazeForOrder

Defined in SpirouDRS.spirouFLAT.spirouFLAT .spirouFLAT.measure_blaze_for_order

Python/Ipython

```
from SpirouDRS import spirouFLAT
spirouFLAT.MeasureBlazeForOrder(p, y)
spirouFLAT.spirouFLAT.measure_blaze_for_order(p, y)
```

```
Measure the blaze function (for good pixels this is a polynomial fit of
order = fitdegree, for bad pixels = 1.0).

bad pixels are defined as less than or equal to zero

:param y: numpy array (1D), the extracted pixels for this order
:param fitdegree: int, the polynomial degree

:return blaze: numpy array (1D), size = len(y), the blaze function: for
               good pixels this is the value of the fit, for bad pixels the
               value = 1.0
```

13.7.2 GetFlat

Defined in SpirouDRS.spirouFLAT.spirouFLAT .spirouFLAT.get_flat

Python/Ipython

```
from SpirouDRS import spirouFLAT
spirouFLAT.GetFlat(p=None, loc=None, hdr=None, filename=None)
spirouFLAT.spirouFLAT.get_flat(p=None, loc=None, hdr=None, filename=None)
```

```
    Attempts to read the flat and if it fails uses a constant flat

    :param p or None: parameter dictionary, ParamDict containing constants
        If defined must contain at least:
            fitsfilename: string, the full path of for the main raw fits
                          file for a recipe
                          i.e. /data/raw/20170710/filename.fits
            fiber: string, the fiber used for this recipe (eg. AB or A or C)
            log_opt: string, log option, normally the program name
    :param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            data: numpy array (2D), the image (used for shape)
    :param hdr or None: dictionary or None, if defined is the header file used
                        to choose the date used in teh calibDB (else uses
                        FITSFILENAME to get date for calibDB)
    :param filename or None: string, the flat file name to read, if None
                             uses FLAT_{FIBER} and gets the flat filename from
                             calibDB

    :return lloc: parameter dictionary, the updated parameter dictionary
            Adds/updates the following:
                FLAT: numpy array (2D), the flat image should be the same
                      shape as data
```

13.7.3 GetValidOrders

Defined in SpirouDRS.spirouFLAT.spirouFLAT .spirouFLAT.get_valid_orders

Python/Ipython

```
from SpirouDRS import spirouFLAT
spirouFLAT.GetValidOrders(p, loc)
spirouFLAT.spirouFLAT.get_valid_orders(p, loc)
```

```
Get valid order range (from min to max) from constants

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        FF_START_ORDER: int or None, the order number to start with, if
                         None this is set to zero
        FF_END_ORDER: int or None, the order number to end with, if None
                       this is set to the last order number
:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            number_orders: int, the number of orders in reference spectrum
:return valid_ordesr: list, all integer values between the start order and
                      end order
```

13.8 The spirouImage module

13.8.1 AddKey

Defined in SpirouDRS.spirouImage.spirouImage .

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.AddKey(hdict=None, keywordstore=None, value=None)
spirouImage.spirouFITS.add_new_key(hdict=None, keywordstore=None, value=None)
```

```
Add a new key to hdict from keywordstore, if value is not None then the
keywordstore value is updated. Each keywordstore is in form:
        [key, value, comment]    where key and comment are strings
if hdict is None creates a new dictionary

:param hdict: dictionary or None, storage for adding to FITS rec
:param keywordstore: list, keyword list (defined in spirouKeywords.py)
                     must be in form [string, value, string]
:param value: object or None, if any python object (other than None) will
              replace the value in keywordstore (i.e. keywordstore[1]) with
              value, if None uses the value = keywordstore[1]

:return hdict: dictionary, storage for adding to FITS rec
```

13.8.2 AddKey1DList

Defined in SpirouDRS.spirouImage.spirouImage .pirouFITS.add_key_1d_list

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.AddKey1DList(hdict, keywordstore, values=None, dim1name='order')
spirouImage.pirouFITS.add_key_1d_list(hdict, keywordstore, values=None, dim1name='order')
```

```
Add a new 1d list to key using the keywordstorage[0] as prefix in form
keyword = kewordstoreage + row number

:param hdict: dictionary, storage for adding to FITS rec
:param keywordstore: list, keyword list (defined in spirouKeywords.py)
                     must be in form [string, value, string]
:param values: numpy array or 1D list of keys or None

               if numpy array or 1D list will create a set of keys in form
               keyword = kewordstoreage + row number
               where row number is the position in values
               with value = values[row number][column number]

               if None uses the value = keywordstore[1]
:param dim1name: string, the name for dimension 1 (rows), used in FITS rec
                 HEADER comments in form:
      COMMENT = keywordstore[2] dim1name={row number}

:return hdict: dictionary, storage for adding to FITS rec
```

13.8.3 AddKey2DList

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.add_key_2d_list

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.AddKey2DList
spirouImage.spirouFITS.add_key_2d_list
```

```
Add a new 2d list to key using the keywordstorage[0] as prefix in form
keyword = kewordstoreage + number

where number = (row number * number of columns) + column number

:param hdict: dictionary, storage for adding to FITS rec
:param keywordstore: list, keyword list (defined in spirouKeywords.py)
                     must be in form [string, value, string]
:param values: numpy array or 2D list of keys or None

               if numpy array or 2D list will create a set of keys in form
               keyword = kewordstoreage + number
               where number = (row number*number of columns)+column number
               with value = values[row number][column number]

               if None uses the value = keywordstore[1]
:param dim1name: string, the name for dimension 1 (rows), used in FITS rec
                 HEADER comments in form:
      COMMENT = keywordstore[2] dim1name={row number} dim2name={col number}
:param dim2name: string, the name for dimension 2 (cols), used in FITS rec
                 HEADER comments in form:
      COMMENT = keywordstore[2] dim1name={row number} dim2name={col number}

:return hdict: dictionary, storage for adding to FITS rec
```

13.8.4 CheckFile

Defined in SpirouDRS.spirouImage.spirouFile `.check_file_id`

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.CheckFile(p, filename, recipe, hdr=None, **kwargs)
spirouImage.spirouFile.check_file_id(p, filename, recipe, hdr=None, **kwargs)
```

```
Checks the "filename" against the "recipe" (using recipe control)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_FORCE_PREPROCESS: bool, if True only allows pre-processed files
        PROCESSED_SUFFIX: string, the processed suffix
        DRS_DATA_RAW: string, the directory that the raw data should
                      be saved to/read from
            DRS_DATA_REDUC: string, the directory that the reduced data
                            should be saved to/read from
        ARG_NIGHT_NAME: string, the folder within data raw directory
                        containing files (also reduced directory) i.e.
                        /data/raw/20170710 would be "20170710"

:param filename: string, the filename to check
:param recipe: string, the recipe name to check
:param hdr: dictionary or None, if defined must be a HEADER dictionary if
            none loaded from filename
:param kwargs: keyword arguments. Current allowed keys are:
                - return_path: bool, if True returns the path as well as p

:return p: parameter dictionary, the updated parameter dictionary
        Adds the following:
            PREPROCESSED: bool, flag whether file is detected as
                          pre-processed
            DPRTYPE: string, the type from the recipe control file
            FIBER: string, if recipe control entry for this file has a
                   fiber defined, fiber is set to this
```

13.8.5 CheckFiles

Defined in SpirouDRS.spirouImage.spirouFile `.check_files_id`

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.CheckFiles(p, files, recipe, hdr=None, **kwargs)
spirouImage.spirouFile.check_files_id(p, files, recipe, hdr=None, **kwargs)
```

```
Wrapper for spirouFile.check_file_id

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_FORCE_PREPROCESS: bool, if True only allows pre-processed files
        PROCESSED_SUFFIX: string, the processed suffix
        DRS_DATA_RAW: string, the directory that the raw data should
                      be saved to/read from
            DRS_DATA_REDUC: string, the directory that the reduced data
                            should be saved to/read from
        ARG_NIGHT_NAME: string, the folder within data raw directory
                        containing files (also reduced directory) i.e.
                        /data/raw/20170710 would be "20170710"
:param files: list of strings, the files to check with check_file_id
:param recipe: string, the recipe name to check
:param hdr: dictionary or None, if defined must be a HEADER dictionary if
            none loaded from filename
:param kwargs: keyword arguments. Current allowed keys are:
                - return_path: bool, if True returns the path as well as p
:return p: parameter dictionary, the updated parameter dictionary
    Adds the following:
        PREPROCESSED: bool, flag whether file is detected as
                      pre-processed
        DPRTYPE: string, the type from the recipe control file
        DPRTYPES: list of stings, the DPRTYPE for each file in "files"
        FIBER: string, if recipe control entry for this file has a
               fiber defined, fiber is set to this
```

13.8.6 ConvertToE

Defined in SpirouDRS.spirouImage.spirouImage .convert_to_e

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ConvertToE(image, p=None, gain=None, exptime=None)
spirouImage.spirouImage.convert_to_e(image, p=None, gain=None, exptime=None)
```

```
Converts image from ADU/s into e-

:param image:
:param p: parameter dictionary, ParamDict containing constants
        Must contain at least: (if exptime is None)
            exptime: float, the exposure time of the image
            gain: float, the gain of the image

:param gain: float, if p is None, used as the gain to multiple the image by
:param exptime: float, if p is None, used as the exposure time the image
                is multiplied by

:return newimage: numpy array (2D), the image in e-
```

13.8.7 ConvertToADU

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.convert_to_adu

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ConvertToADU(image, p=None, exptime=None)
spirouImage.spirouImage.convert_to_adu(image, p=None, exptime=None)
```

```
Converts image from ADU/s into ADU

:param image:

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least: (if exptime is None)
        exptime: float, the exposure time of the image

:param exptime: float, if p is None, used as the exposure time the image
                is multiplied by

:return newimage: numpy array (2D), the image in e-
```

13.8.8 CopyOriginalKeys

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.copy_original_keys

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.CopyOriginalKeys(header, comments, hdict=None, forbid_keys=True)
spirouImage.spirouFITS.copy_original_keys(header, comments, hdict=None, forbid_keys=True)
```

```
Copies keys from hdr dictionary to hdict, if forbid_keys is True some
keys will not be copies (defined in python code)

:param header: header dictionary from readimage (ReadImage) function

:param comments: comment dictionary from readimage (ReadImage) function

:param hdict: dictionary or None, header dictionary to write to fits file
              if None hdict is created

            Must be in form:

                    hdict[key] = (value, comment)
            or
                    hdict[key] = value     (comment will be equal to
                                            "UNKNOWN"
:param forbid_keys: bool, if True uses the forbidden copy keys (defined in
                    spirouConfig.Constants.FORBIDDEN_COPY_KEYS() to remove
                    certain keys from those being copied, if False copies
                    all keys from input header

:return hdict: dictionary, (updated or new) header dictionary containing
               key/value pairs from the header (that are NOT in
               spirouConfig.spirouConst.FORBIDDEN_COPY_KEY)
```

13.8.9 CopyRootKeys

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.copy_root_keys

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.CopyRootKeys(hdict=None, filename=None, root=None, ext=0)
spirouImage.spirouFITS.copy_root_keys(hdict=None, filename=None, root=None, ext=0)
```

```
Copy keys from a filename to hdict

:param hdict: dictionary or None, header dictionary to write to fits file
              if None hdict is created
:param filename: string, location and filename of the FITS rec to open

:param ext: int, the extension of the FITS rec to open header from
            (defaults to 0)
:return:
```

13.8.10 CorrectForDark

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.correct_for_dark

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.CorrectForDark(p, image, header, nfiles=None, return_dark=False)
spirouImage.spirouImage.correct_for_dark(p, image, header, nfiles=None, return_dark=False)
```

```
Corrects "image" for "dark" using calibDB file (header must contain
value of p['ACQTIME_KEY'] as a keyword)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            nbframes: int, the number of frames/files (usually the length
                      of "arg_file_names")
            calibDB: dictionary, the calibration database dictionary
                     (if not in "p" we construct it and need "max_time_unix"
            max_time_unix: float, the unix time to use as the time of
                           reference (used only if calibDB is not defined)
            log_opt: string, log option, normally the program name
            DRS_CALIB_DB: string, the directory that the calibration
                          files should be saved to/read from

:param image: numpy array (2D), the image
:param header: dictionary, the header dictionary created by
               spirouFITS.ReadImage
:param nfiles: int or None, number of files that created image (need to
               multiply by this to get the total dark) if None uses
               p['nbframes']
:param return_dark: bool, if True returns corrected_image and dark
                    if False (default) returns corrected_image

:return corrected_image: numpy array (2D), the dark corrected image
                         only returned if return_dark = True:
:return darkimage: numpy array (2D), the dark
```

13.8.11 CorrectForBadPix

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.correct_for_badpix

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.CorrectForBadPix(p, image, header)
spirouImage.spirouImage.correct_for_badpix(p, image, header)
```

```
Corrects "image" for "BADPIX" using calibDB file (header must contain
value of p['ACQTIME_KEY'] as a keyword) - sets all bad pixels to zeros

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            calibDB: dictionary, the calibration database dictionary
                     (if not in "p" we construct it and need "max_time_unix"
            max_time_unix: float, the unix time to use as the time of
                           reference (used only if calibDB is not defined)
            log_opt: string, log option, normally the program name
            DRS_CALIB_DB: string, the directory that the calibration
                          files should be saved to/read from

:param image: numpy array (2D), the image
:param header: dictionary, the header dictionary created by
               spirouFITS.ReadImage

:return corrected_image: numpy array (2D), the corrected image where all
                         bad pixels are set to zeros
```

13.8.12 FiberParams

Defined in SpirouDRS.spirouImage.spirouImage .spirouFile.fiber_params

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouImage.FiberParams(pp, fiber, merge=False)
spirouImage.spirouFile.fiber_params(pp, fiber, merge=False)
```

```
Takes the parameters defined in FIBER_PARAMS from parameter dictionary
(i.e. from config files) and adds the correct parameter to a fiber
parameter dictionary

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name

:param fiber: string, the fiber type (and suffix used in configuration file)
              i.e. for fiber AB fiber="AB" and nbfib_AB should be present
              in config if "nbfib" is in FIBER_PARAMS
:param merge: bool, if True merges with pp and returns

:return fparam: dictionary, the fiber parameter dictionary (if merge False)
:treun pp: dictionary, parameter dictionary (if merge True)
```

13.8.13 FitTilt

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.fit_tilt

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.FitTilt(pp, lloc)
spirouImage.spirouImage.fit_tilt(pp, lloc)
```

```
Fit the tilt (lloc['tilt'] with a polynomial of size = p['ic_tilt_filt']
return the coefficients, fit and residual rms in lloc dictionary

:param pp: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_TILT_FIT: int, Order of polynomial to fit for tilt

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            number_orders: int, the number of orders in reference spectrum
            tilt: numpy array (1D), the tilt angle of each order

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            xfit_tilt: numpy array (1D), the order numbers
            yfit_tilt: numpy array (1D), the fit for the tilt angle of each
                       order
            a_tilt: numpy array (1D), the fit coefficients (generated by
                    numpy.polyfit but IN REVERSE ORDER)
            rms_tilt: float, the RMS (np.std) of the residuals of the
                      tilt - tilt fit values
```

13.8.14 FixNonPreProcess

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.fix_non_preprocessed

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.FixNonPreProcess
spirouImage.spirouImage.fix_non_preprocessed
```

```
Fit the tilt (lloc['tilt'] with a polynomial of size = p['ic_tilt_filt']
return the coefficients, fit and residual rms in lloc dictionary

:param pp: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_TILT_FIT: int, Order of polynomial to fit for tilt

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            number_orders: int, the number of orders in reference spectrum
            tilt: numpy array (1D), the tilt angle of each order

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            xfit_tilt: numpy array (1D), the order numbers
            yfit_tilt: numpy array (1D), the fit for the tilt angle of each
                       order
            a_tilt: numpy array (1D), the fit coefficients (generated by
                    numpy.polyfit but IN REVERSE ORDER)
            rms_tilt: float, the RMS (np.std) of the residuals of the
                      tilt - tilt fit values
```

13.8.15 FlipImage

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.flip_image

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.FlipImage(image, fliprows=True, flipcols=True)
spirouImage.spirouImage.flip_image(image, fliprows=True, flipcols=True)
```

```
Flips the image in the x and/or the y direction

:param image: numpy array (2D), the image
:param fliprows: bool, if True reverses row order (axis = 0)
:param flipcols: bool, if True reverses column order (axis = 1)

:return newimage: numpy array (2D), the flipped image
```

13.8.16 FixNonPreProcess

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.fix_non_preprocessed

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.FixNonPreProcess(p, image, filename=None)
spirouImage.spirouImage.fix_non_preprocessed(p, image, filename=None)
```

```
If a raw file is not preprocessed, then fix it (i.e. rotate it) so
it conforms to DRS standards

:param pp: parameter dictionary, ParamDict containing constants
    Must contain at least:
        PROCESSED_SUFFIX: string, the processed suffix
        PREPROCESSED: bool, flag whether file is detected as
                      pre-processed
        IC_IMAGE_TYPE: string, the detector type
        RAW_TO_PP_ROTATION: int, rotation angle in degrees (in degrees,
                            counter-clockwise direction) must be a multiple
                            of 90 degrees
:param image: numpy array (2D), the image to manipulate
:param filename: string, if p['PREPROCESSED'] is not defined the file
                 is checked (can be done if PREPROCESSED in p

:return newimage: numpy array (2D), the new image that emulates a pre-
                  processed file
```

13.8.17 GetBadPixMap

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.get_badpixel_map

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetBadPixMap(p, header=None)
spirouImage.spirouImage.get_badpixel_map(p, header=None)
```

```
Get the bad pixel map from the calibDB

    Must contain at least:
            calibDB: dictionary, the calibration database dictionary
                     (if not in "p" we construct it and need "max_time_unix"
            max_time_unix: float, the unix time to use as the time of
                           reference (used only if calibDB is not defined)
            log_opt: string, log option, normally the program name
            DRS_CALIB_DB: string, the directory that the calibration
                          files should be saved to/read from

:param header: dictionary, the header dictionary created by
               spirouFITS.ReadImage

:return: badpixmask: numpy array (2D), the bad pixel mask
```

13.8.18 GetAllSimilarFiles

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.get_all_similar_files

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetAllSimilarFiles(p, directory, prefix=None, suffix=None)
spirouImage.spirouImage.get_all_similar_files(p, directory, prefix=None, suffix=None)
```

```
Get all similar files in a directory with matching prefix and suffix defined
either by "prefix" and "suffix" or by p["ARG_FILE_NAMES"][0]

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            arg_file_names: list, list of files taken from the command line
                            (or call to recipe function) must have at least
                            one string filename in the list
            log_opt: string, log option, normally the program name

:param directory: string, the directory to search for files
:param prefix: string or None, if not None the prefix to search for, if
               None defines the prefix from the first 5 characters of
               p["ARG_FILE_NAMES"][0]
:param suffix: string  or None, if not None the suffix to search for, if
               None defines the prefix from the last 8 characters of
               p["ARG_FILE_NAMES"][0]

:return filelist: list of strings, the full paths of all files that are in
                  "directory" with the matching prefix and suffix defined
                  either by "prefix" and "suffix" or by
                  p["ARG_FILE_NAMES"][0]
```

13.8.19 GetSigdet

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.get_sigdet

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetSigdet(p, hdr, name=None, return_value=False)
spirouImage.spirouImage.get_sigdet(p, hdr, name=None, return_value=False)
```

```
Get sigdet from HEADER. Wrapper for spirouImage.get_param

:param p: parameter dictionary, ParamDict of constants
:param hdr: dictionary, header dictionary to extract
:param name: string or None, if not None the name for the parameter
             logged if there is an error in getting parameter, if name is
             None the name is taken as "keyword"
:param return_value: bool, if True returns parameter, if False adds
                     parameter to "p" parameter dictionary (and sets source)

:return value: if return_value is True value of parameter is returned
:return p: if return_value is False, updated parameter dictionary p with
           key = name is returned
```

13.8.20 GetExpTime

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.get_exptime

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetExpTime(p, hdr, name=None, return_value=False)
spirouImage.spirouImage.get_exptime(p, hdr, name=None, return_value=False)
```

```
Get Exposure time from HEADER. Wrapper for spirouImage.get_param

:param p: parameter dictionary, ParamDict of constants
:param hdr: dictionary, header dictionary to extract
:param name: string or None, if not None the name for the parameter
             logged if there is an error in getting parameter, if name is
             None the name is taken as "keyword"
:param return_value: bool, if True returns parameter, if False adds
                     parameter to "p" parameter dictionary (and sets source)

:return value: if return_value is True value of parameter is returned
:return p: if return_value is False, updated parameter dictionary p with
           key = name is returned
```

13.8.21 GetGain

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.get_gain

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetGain(p, hdr, name=None, return_value=False)
spirouImage.spirouImage.get_gain(p, hdr, name=None, return_value=False)
```

```
Get Gain from HEADER. Wrapper for spirouImage.get_param

:param p: parameter dictionary, ParamDict of constants
:param hdr: dictionary, header dictionary to extract
:param name: string or None, if not None the name for the parameter
             logged if there is an error in getting parameter, if name is
             None the name is taken as "keyword"
:param return_value: bool, if True returns parameter, if False adds
                     parameter to "p" parameter dictionary (and sets source)

:return value: if return_value is True value of parameter is returned
:return p: if return_value is False, updated parameter dictionary p with
           key = name is returned
```

13.8.22 GetAcqTime

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.get_acqtime

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetAcqTime(p, hdr, name=None, kind='human', return_value=False)
spirouImage.spirouImage.get_acqtime(p, hdr, name=None, kind='human', return_value=False)
```

```
Get the acquision time from the header file, if there is not header file
use the parameter dictionary "p" to open the header in 'arg_file_names[0]'

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        "name" defined in call
        parameter dictionary to give to value

:param hdr: dictionary, the header dictionary created by
            spirouFITS.ReadImage
:param name: string, the name in parameter dictionary to give to value
             if return_value is False (i.e. p[name] = value)
:param kind: string, 'human' for 'YYYY-mm-dd-HH-MM-SS.ss' or 'Unix'
             for time since 1970-01-01
:param return_value: bool, if False value is returned in p as p[name]
                     if True value is returned

:return p or value: dictionary or string or float, if return_value is False
                    parameter dictionary is returned, if return_value is
                    True and kind=='human' returns a string, if return_value
                    is True and kind=='Unix' returns a float
```

13.8.23 IdentifyUnProFile

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.identify_unprocessed_file

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.IdentifyUnProFile(p, filename, hdr=None, cdr=None)
spirouImage.spirouImage.identify_unprocessed_file(p, filename, hdr=None, cdr=None)
```

```
Identify a unprocessed raw file (from recipe control file), adds suffix
and header key (DPRTYPE) accordingly

:param p: parameter dictionary, ParamDict containing constants
        Must contain at least:
            DRS_DEBUG: int, if 0 not in debug mode else in various levels of
                       debug
            KW_DPRTYPE: list, keyword list [key, value, comment]

:param filename: string, the filename to check
:param hdr: dictionary or None, the HEADER dictionary to check for keys and
            associated values, if None attempt to open HEADER from filename
:param cdr: dictionary or None, the HEADER dictionary to check for keys and
            associated comments, if None attempt to open HEADER from
            filename

:return newfn: string, the new filename with added suffix
:return hdr: dictionary, the HEADER with added HEADER key/value pair
             (DPRTYPE)
:return cdr: dictionary, the HEADER with added HEADER key/comment pair
             (DPRTYPE)
```

13.8.24 InterpolateBadRegions

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.identify_unprocessed_file

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.InterpolateBadRegions
spirouImage.spirouImage.interp_bad_regions
```

```
Interpolate over the bad regions to fill in holes on image (only to be used
for image localization)

:param p: parameter dictionary, ParamDict containing constants
        Must contain at least:
            IC_IMAGE_TYPE: string, the type of detector (H2RG or H4RG)
            BAD_REGION_FIT: list of floats, the fit to the curvature
            BAD_REGION_MED_SIZE: int, the median filter box size
            BAD_REGION_THRESHOLD: float, the threshold below which the
                                  image (normalised) should be regarded as
                                  bad (and the pixels in the image that
                                  should be set to the norm value)
            BAD_REGION_KERNAL_SIZE: int, the box size used to do the
                                    convolution
            BAD_REGION_MED_SIZE2: int, the median filter box size used
                                  during the convolution
            BAD_REGION_GOOD_VALUE: float, the final good ratio value
                                   (ratio between the original image and
                                    the interpolated image) to accept
                                    pixels as good pixels

            BAD_REGION_BAD_VALUE: float, the final bad ratio value
                                  (ratio between the original image and
                                   the interpolated image) to reject
                                   pixels as bad pixels

:param image: numpy array (2D), the image

:return image3: numpy array (2D), the corrected image
```

13.8.25 GetKey

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.keylookup

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetKey(p, d=None, key=None, has_default=False, default=None)
spirouImage.spirouFITS.keylookup(p, d=None, key=None, has_default=False, default=None)
```

```
Looks for a key in dictionary "p" or "d", if has_default is True sets
value of key to 'default' if not found else logs an error

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            "key": if d is None must contain key="key" or error is raised
:param d: dictionary, any dictionary, if None uses parameter dictionary
          if "d" is not None then must contain key="key" or error is raised
:param key: string, key in the dictionary to find
:param has_default: bool, if True uses "default" as the value if key
                    not found
:param default: object, value of the key if not found and
                has_default is True

:return value: object, value of p[key] or default (if has_default=True)
```

13.8.26 GetKeys

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.keyslookup

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetKeys(p, d=None, keys=None, has_default=False, defaults=None)
spirouImage.spirouFITS.keyslookup(p, d=None, keys=None, has_default=False, defaults=None)
```

```
Looks for keys in dictionary "p" or "d", if has_default is True sets
value of key to 'default' if not found else logs an error

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            "key": if d is None must contain key="key" or error is raised
:param d: dictionary, any dictionary, if None uses parameter dictionary
          if "d" is not None then must contain key="key" or error is raised
:param keys: list of strings, keys in the dictionary to find
:param has_default: bool, if True uses "default" as the value if key
                    not found
:param defaults: list of objects or None, values of the keys if not
                 found and has_default is True

:return values: list of objects, values of p[key] for key in keys
                or default value for each key (if has_default=True)
```

13.8.27 GetTilt

Defined in SpirouDRS.spirouImage.spirouImage .

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetTilt(pp, lloc, image)
spirouImage.spirouImage.get_tiltspirouImage.get_tilt(pp, lloc, image)
```

```
Get the tilt by correlating the extracted fibers

:param pp: parameter dictionary, ParamDict containing constants
    Must contain at least:
            ic_tilt_coi: int, oversampling factor
            log_opt: string, log option, normally the program name

:param lloc: parameter dictionary, ParamDict containing data
        Must contain at least:
            number_orders: int, the number of orders in reference spectrum
            cent1: numpy array (2D), the extraction for A, updated is
                   the order "rnum"
            cent2: numpy array (2D), the extraction for B, updated is
                   the order "rnum"
            offset: numpy array (1D), the centre values with the
                    offset in 'IC_CENT_COL' added

:param image: numpy array (2D), the image

:return lloc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            nbcos: numpy array, zero array  (length of "number_orderes")
            tilt: numpy array (1D), the tilt angle of each order
```

13.8.28 GetTypeFromHeader

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.get_type_from_header

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.GetTypeFromHeader(p, keywordstore, hdict=None, filename=None)
spirouImage.spirouFITS.get_type_from_header(p, keywordstore, hdict=None, filename=None)
```

```
Special FITS HEADER keyword - get the type of file from a FITS file HEADER
using "keywordstore"

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            fitsfilename: string, the full path of for the main raw fits
                          file for a recipe
                          i.e. /data/raw/20170710/filename.fits

:param keywordstore: list, a keyword store in the form
                     [name, value, comment] where the format is
                     [string, object, string]
:param hdict: dictionary or None, the HEADER dictionary containing
              key/value pairs from a FITS HEADER, if None uses the
              header from "FITSFILENAME" in "p", unless filename is not None
              This hdict is used to get the type of file
:param filename: string or None, if not None and hdict is None, this is the
                 file which is used to extract the HEADER from to get
                 the type of file

:return ftype: string, the type of file (extracted from a HEADER dictionary/
               file) if undefined set to 'UNKNOWN'
```

13.8.29 LocateBadPixels

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.locate_bad_pixels

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.LocateBadPixels(p, fimage, fmed, dimage, wmed=None)
spirouImage.spirouImage.locate_bad_pixels(p, fimage, fmed, dimage, wmed=None)
```

```
Locate the bad pixels in the flat image and the dark image

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            BADPIX_FLAT_MED_WID: float, the median image in the x
                                 dimension over a boxcar of this width
            BADPIX_FLAT_CUT_RATIO: float, the maximum differential pixel
                                   cut ratio
            BADPIX_ILLUM_CUT: float, the illumination cut parameter
            BADPIX_MAX_HOTPIX: float, the maximum flux in ADU/s to be
                               considered too hot to be used

:param fimage: numpy array (2D), the flat normalised image
:param fmed: numpy array (2D), the flat median normalised image
:param dimage: numpy array (2D), the dark image
:param wmed: float or None, if not None defines the median filter width
             if None uses p["BADPIX_MED_WID", see
             scipy.ndimage.filters.median_filter "size" for more details

:return bad_pix_mask: numpy array (2D), the bad pixel mask image
:return badpix_stats: list of floats, the statistics array:
                      Fraction of hot pixels from dark [%]
                      Fraction of bad pixels from flat [%]
                      Fraction of NaN pixels in dark [%]
                      Fraction of NaN pixels in flat [%]
                      Fraction of bad pixels with all criteria [%]
```

13.8.30 LocateFullBadPixels

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.locate_bad_pixels_full

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.LocateFullBadPixels(p, image)
spirouImage.spirouImage.locate_bad_pixels_full(p, image)
```

```
Locate the bad pixels identified from the full engineering flat image
(location defined from p['BADPIX_FULL_FLAT'])

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_IMAGE_TYPE: string, the detector type (this step is only for
                       H4RG)
        LOG_OPT: string, log option, normally the program name
        BADPIX_FULL_FLAT: string, the full engineering flat filename
        BADPIX_FULL_THRESHOLD: float, the threshold on the engineering
                               above which the data is good
:param image: numpy array (2D), the image to correct (for size only)

:return newimage: numpy array (2D), the mask of the bad pixels
:return stats: float, the fraction of un-illuminated pixels (percentage)
```

13.8.31 MakeTable

Defined in SpirouDRS.spirouImage.spirouImage .spirouTable.make_table

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.MakeTable
spirouImage.spirouTable.make_table
```

```
Construct an astropy table from columns and values

:param columns: list of strings, the list of column names
:param values: list of lists or numpy array (2D), the list of lists/array
               of values, first dimension must have same length as number
               of columns, there must be the same number of values in each
               column
:param formats: list of strings, the astropy formats for each column
                i.e. 0.2f  for a float with two decimal places, must have
                same length as number of columns
:param units: list of strings, the units for each column, must have
              same length as number of columns

:return table: astropy.table.Table instance, the astropy table containing
               all columns and data
```

13.8.32 MeasureDark

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.measure_dark

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.MeasureDark(pp, image, image_name, short_name)
spirouImage.spirouImage.measure_dark(pp, image, image_name, short_name)
```

```
Measure the dark pixels in "image"

:param pp: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            DARK_QMIN: int, The lower percentile (0 - 100)
            DARK_QMAX: int, The upper percentile (0 - 100)
            HISTO_BINS: int,  The number of bins in dark histogram
            HISTO_RANGE_LOW: float, the lower extent of the histogram
                             in ADU/s
            HISTO_RANGE_HIGH: float, the upper extent of the histogram
                              in ADU/s

:param image: numpy array (2D), the image
:param image_name: string, the name of the image (for logging)
:param short_name: string, suffix (for parameter naming -
                   parameters added to pp with suffix i)

:return pp: parameter dictionary, the updated parameter dictionary
        Adds the following: (based on "short_name")
            histo_full: numpy.histogram tuple (hist, bin_edges) for
                        the full image
            histo_blue: numpy.histogram tuple (hist, bin_edges) for
                        the blue part of the image
            histo_red: numpy.histogram tuple (hist, bin_edges) for
                        the red part of the image
            med_full: float, the median value of the non-Nan image values
                      for the full image
            med_blue: float, the median value of the non-Nan image values
                      for the blue part of the image
            med_red: float, the median value of the non-Nan image values
                     for the red part of the image
            dadead_full: float, the fraction of dead pixels as a percentage
                         for the full image
            dadead_blue: float, the fraction of dead pixels as a percentage
                         for the blue part of the image
            dadead_red: float, the fraction of dead pixels as a percentage
                        for the red part of the image

      where:
          hist : numpy array (1D) The values of the histogram.
          bin_edges : numpy array (1D) of floats, the bin edges
```

13.8.33 MergeTable

Defined in SpirouDRS.spirouImage.spirouImage .spirouTable.merge_table

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.MergeTable(table, filename, fmt='fits')
spirouImage.spirouTable.merge_table(table, filename, fmt='fits')
```

```
If a file already exists for "filename" try to merge this new table with
the old one (requires all columns/formats to be the same).
If filename does not exist writes "table" as if new table

:param table:  astropy table, the new table to be merged to existing file
:param filename: string, the filename and location of the table
                 to written to
:param fmt: string, the format of the table to read from (must be valid
            for astropy.table to read)

:return None:
```

13.8.34 NormMedianFlat

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.normalise_median_flat

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.NormMedianFlat(p, image, method='new', wmed=None, percentile=None)
spirouImage.spirouImage.normalise_median_flat(p, image, method='new', wmed=None, percentile=None)
```

```
Applies a median filter and normalises. Median filter is applied with width
"wmed" or p["BADPIX_FLAT_MED_WID"] if wmed is None) and then normalising by
the 90th percentile

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            BADPIX_FLAT_MED_WID: float, the median image in the x
                                 dimension over a boxcar of this width
            BADPIX_NORM_PERCENTILE: float, the percentile to normalise
                                    to when normalising and median
                                    filtering image
            log_opt: string, log option, normally the program name

:param image: numpy array (2D), the image to median filter and normalise
:param method: string, "new" or "old" if "new" uses np.percentile else
               sorts the flattened image and takes the "percentile" (i.e.
               90th) pixel value to normalise
:param wmed: float or None, if not None defines the median filter width
             if None uses p["BADPIX_MED_WID", see
             scipy.ndimage.filters.median_filter "size" for more details
:param percentile: float or None, if not None defines the percentile to
                   normalise the image at, if None used from
                   p["BADPIX_NORM_PERCENTILE"]

:return norm_med_image: numpy array (2D), the median filtered and normalised
                        image
:return norm_image: numpy array (2D), the normalised image
```

13.8.35 PPCorrectTopBottom

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.ref_top_bottom

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.PPCorrectTopBottom(p, image)
spirouImage.spirouImage.ref_top_bottom(p, image)
```

```
Correction for the top and bottom reference pixels

:param p: parameter dictionary, ParamDict containing constants
        Must contain at least:
            TOTAL_AMP_NUM: int, the total number of amplifiers on the
                           detector
            NUMBER_REF_TOP: int, the number of reference pixels at the top
                            of the image (highest y pixel values)
            NUMBER_REF_BOTTOM: int, the number of reference pixels at the
                               bottom of the image (lowest y pixel values)
:param image: numpy array (2D), the image

:return image: numpy array (2D), the corrected image
```

13.8.36 PPMedianFilterDarkAmps

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.median_filter_dark_amp

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.PPMedianFilterDarkAmps(p, image)
spirouImage.spirouImage.median_filter_dark_amp(p, image)
```

```
Use the dark amplifiers to produce a median pattern and apply this to the
image

:param p: parameter dictionary, ParamDict containing constants
        Must contain at least:
            TOTAL_AMP_NUM: int, the total number of amplifiers on the
                           detector
            NUMBER_DARK_AMP: int, the number of unilluminated (dark)
                             amplifiers on the detector
            DARK_MED_BINNUM: int, the number of bins to use in the median
                             filter binning process (higher number = finer
                             bins, lower number = bigger bins)
:param image: numpy array (2D), the image

:return image: numpy array (2D), the corrected image
```

13.8.37 PPMedianOneOverfNoise

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.median_one_over_f_noise

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.PPMedianOneOverfNoise(p, image)
spirouImage.spirouImage.median_one_over_f_noise(p, image)
```

```
Use the top and bottom reference pixels to create a map of the 1/f noise
and apply it to the image

:param p: parameter dictionary, ParamDict containing constants
        Must contain at least:
            NUMBER_REF_TOP: int, the number of reference pixels at the top
                            of the image (highest y pixel values)
            NUMBER_REF_BOTTOM: int, the number of reference pixels at the
                               bottom of the image (lowest y pixel values)
            NUMBER_DARK_AMP: int, the number of unilluminated (dark)
                             amplifiers on the detector
            DARK_MED_BINNUM: int, the number of bins to use in the median
                             filter binning process (higher number = finer
                             bins, lower number = bigger bins)
:param image: numpy array (2D), the image

:return image: numpy array (2D), the corrected image
```

13.8.38 ReadData

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.readdata

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadData(p, filename, log=True, return_header=True, return_shape=True)
spirouImage.spirouFITS.readdata(p, filename, log=True, return_header=True, return_shape=True)
```

```
Reads the image 'fitsfilename' defined in p and adds files defined in
'arg_file_names' if add is True

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name

:param filename: string, filename of the image to read
:param log: bool, if True logs opening and size
:param return_header: bool, if True returns header
:param return_shape: bool, if True returns shape

:return image: numpy array (2D), the image

if return_header also returns:
    :return header: dictionary, the header file of the image
    :return comments: dictionary, the header comment file

if return_shape also returns:
    if len(data.shape)==2
        :return nx: int, the shape in the first dimension,
                    i.e. data.shape[0]
        :return ny: int, the shape in the second dimension,
                    i.e. data.shape[1]
    if len(data.shape)!=2
        :return shape: tuple, data.shape
        :return empty: None, blank entry
```

13.8.39 ReadLineList

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.read_line_list

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadLineList
spirouImage.spirouImage.read_line_list
```

```
Read the line list file (if filename is None construct file from
p['IC_LL_LINE_FILE']

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        log_opt: string, log option, normally the program name

    May contain
        IC_LL_LINE_FILE: string, the file name of the line list to use
                         (required if filename is None)

:param filename: string or None, if defined the filename

:return ll: numpy array (1D), the wavelengths of the lines from line list
:return amp: numpy array (1D), the amplitudes of the lines from line list
```

13.8.40 ReadParam

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.get_param

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadParam(p, hdr, name=None, kind='human', return_value=False)
spirouImage.spirouImage.get_param(p, hdr, name=None, kind='human', return_value=False)
```

```
Get the acquisition time from the header file, if there is not header file
use the parameter dictionary "p" to open the header in 'arg_file_names[0]'

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        "name" defined in call
        parameter dictionary to give to value

:param hdr: dictionary, the header dictionary created by
            spirouFITS.ReadImage
:param name: string, the name in parameter dictionary to give to value
             if return_value is False (i.e. p[name] = value)
:param kind: string, 'human' for 'YYYY-mm-dd-HH-MM-SS.ss' or 'Unix'
             for time since 1970-01-01
:param return_value: bool, if False value is returned in p as p[name]
                     if True value is returned

:return p or value: dictionary or string or float, if return_value is False
                    parameter dictionary is returned, if return_value is
                    True and kind=='human' returns a string, if return_value
                    is True and kind=='Unix' returns a float
```

13.8.41 ReadImage

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.readimage

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadImage(p, filename=None, log=True, kind=None)
spirouImage.spirouFITS.readimage(p, filename=None, log=True, kind=None)
```

```
Reads the image 'fitsfilename' defined in p and adds files defined in
'arg_file_names' if add is True

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            fitsfilename: string, the full path of for the main raw fits
                  file for a recipe
                  i.e. /data/raw/20170710/filename.fits
            log_opt: string, log option, normally the program name
            arg_file_names: list, list of files taken from the command line
                            (or call to recipe function) must have at least
                            one string filename in the list

:param filename: string or None, filename of the image to read, if None
                 then p['fitsfilename'] is used
:param log: bool, if True logs opening and size
:param kind: string or None, if defined names the image else just image,
             used in logging (if log = True)

:return image: numpy array (2D), the image
:return header: dictionary, the header file of the image
:return nx: int, the shape in the first dimension, i.e. data.shape[0]
:return ny: int, the shape in the second dimension, i.e. data.shape[1]
```

13.8.42 ReadTable

Defined in SpirouDRS.spirouImage.spirouImage .spirouTable.read_table

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadTable(filename, fmt, colnames=None)
spirouImage.spirouTable.read_table(filename, fmt, colnames=None)
```

```
Reads a table from file "filename" in format "fmt", if colnames are defined
renames the columns to these name

:param filename: string, the filename and location of the table to read
:param fmt: string, the format of the table to read from (must be valid
            for astropy.table to read - see below)
:param colnames: list of strings or None, if not None renames all columns
                 to these strings, must be the same length as columns
                 in file that is read

:return None:

astropy.table readable formats are as follows:
```

13.8.43 ReadImageAndCombine

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.readimage_and_combine

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadImageAndCombine(p, framemath='+', filename=None, filenames=None, log=True)
spirouImage.spirouFITS.readimage_and_combine(p, framemath='+', filename=None, filenames=None, log=True
    )
```

```
Reads the image 'fitsfilename' defined in p and adds files defined in
'arg_file_names' if add is True

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name

            optional:
            fitsfilename: string, the full path of for the main raw fits
                  file for a recipe i.e. /data/raw/20170710/filename.fits
                  (if filename is None this is required)

            arg_file_names: list, list of files taken from the command line
                            (or call to recipe function) must have at least
                            one string filename in the list
                  (if filenames is None this is required)

:param framemath: string, controls how files should be added

            currently supported are:
            'add' or '+'           - adds the frames
            'sub' or '-'           - subtracts the frames
            'average' or 'mean'    - averages the frames
            'multiply' or '*'      - multiplies the frames
            'divide' or '/'        - divides the frames
            'none'                 - does not add

:param filename: string or None, filename of the image to read, if None
                 then p['fitsfilename'] is used

:param filenames: list of strings or None, filenames to combine with
                  "filename", if None then p['arg_file_names'] is used

:param log: bool, if True logs opening and size

:return image: numpy array (2D), the image
:return header: dictionary, the header file of the image
:return nx: int, the shape in the first dimension, i.e. data.shape[0]
:return ny: int, the shape in the second dimension, i.e. data.shape[1]
```

13.8.44 ReadFlatFile

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.read_flat_file

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadFlatFile(p, hdr=None, filename=None, key=None)
spirouImage.spirouFITS.read_flat_file(p, hdr=None, filename=None, key=None)
```

```
Reads the flat file (from calib database or filename)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            fitsfilename: string, the full path of for the main raw fits
                          file for a recipe
                          i.e. /data/raw/20170710/filename.fits
            fiber: string, the fiber used for this recipe (eg. AB or A or C)
            log_opt: string, log option, normally the program name

:param hdr: dictionary or None, the header dictionary to look for the
                 acquisition time in, if None loads the header from
                 p['fitsfilename']
:param filename: string or None, the filename and path of the tilt file,
                 if None gets the FLAT file from the calib database
                 keyword "FLAT_{fiber}"
:param key: string or None, if None key='FLAT_{fiber}' else uses string
            as key from calibDB (first entry) to get wave file

:return wave: numpy array (2D), the flat image
```

13.8.45 ReadBlazeFile

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.read_blaze_file

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadBlazeFile(p, hdr=None, filename=None, key=None)
spirouImage.spirouFITS.read_blaze_file(p, hdr=None, filename=None, key=None)
```

```
Reads the blaze file (from calib database or filename)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            fitsfilename: string, the full path of for the main raw fits
                          file for a recipe
                          i.e. /data/raw/20170710/filename.fits
            fiber: string, the fiber used for this recipe (eg. AB or A or C)

:param hdr: dictionary or None, the header dictionary to look for the
                acquisition time in, if None loads the header from
                p['fitsfilename']
:param filename: string or None, the filename and path of the tilt file,
                if None gets the WAVE file from the calib database
                keyword "BLAZE_{fiber}"
:param key: string or None, if None key='BLAZE_{fiber}' else uses string
            as key from calibDB (first entry) to get wave file

:param return_header: bool, if True returns header file else just returns
                      wave file
:return blaze: numpy array (2D), the blaze function (along x-direction)
              for each order
```

13.8.46 ReadHeader

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.read_header

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadHeader(p=None, filepath=None, ext=0)
spirouImage.spirouFITS.read_header(p=None, filepath=None, ext=0)
```

```
Read the header from a file at "filepath" with extention "ext" (default=0)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name

:param filepath: string, filename and path of FITS file to open
:param ext: int, extension in FITS rec to open (default = 0)

:return hdict: dictionary, the dictionary with key value pairs
```

13.8.47 ReadKey

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.read_key

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadKey(p, hdict=None, key=None)
spirouImage.spirouFITS.read_key(p, hdict=None, key=None)
```

```
Read a key from hdict (or p if hdict is not defined) and return it's
value.

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name

:param hdict: dictionary or None, the dictionary to add the key to once
              found, if None creates a new dictionary
:param key: string, key in the dictionary to find

:return value: object, the value of the key from hdict
               (or p if hdict is None)
```

13.8.48 Read2Dkey

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.read_key_2d_list

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.Read2Dkey(p, hdict, key, dim1, dim2)
spirouImage.spirouFITS.read_key_2d_list(p, hdict, key, dim1, dim2)
```

```
Read a set of header keys that were created from a 2D list

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name

:param hdict: dictionary, HEADER dictionary to extract key/value pairs from
:param key: string, prefix of HEADER key to construct 2D list from
             key[number]

             where number = (row number * number of columns) + column number
             where column number = dim2 and row number = range(0, dim1)
:param dim1: int, the number of elements in dimension 1 (number of rows)
:param dim2: int, the number of columns in dimension 2 (number of columns)

:return value: numpy array (2D), the reconstructed 2D list of variables
               from the HEADER dictionary keys
```

13.8.49 ReadTiltFile

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.read_tilt_file

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadTiltFile(p, hdr=None, filename=None, key=None, return_filename=False)
spirouImage.spirouFITS.read_tilt_file(p, hdr=None, filename=None, key=None, return_filename=False)
```

```
Reads the tilt file (from calib database or filename) and using the
'kw_TILT' keyword-store extracts the tilts for each order

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            fitsfilename: string, the full path of for the main raw fits
                          file for a recipe
                          i.e. /data/raw/20170710/filename.fits
            kw_TILT: list, the keyword list for kw_TILT (defined in
                     spirouKeywords.py)
            IC_TILT_NBO: int, Number of orders in tilt file

:param hdr: dictionary or None, the header dictionary to look for the
                acquisition time in, if None loads the header from
                p['fitsfilename']
:param filename: string or None, the filename and path of the tilt file,
                if None gets the TILT file from the calib database
                keyword "TILT"
:param key: string or None, if None key='TILT' else uses string as key
            from calibDB (first entry) to get tilt file
:param return_filename: bool, if true return the filename only

if return_filename is False
    :return tilt: numpy array (1D), the tilts for each order
else
    :return read_file: string, name of tilt file
```

13.8.50 ReadWaveFile

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.read_wave_file

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadWaveFile(p, hdr=None, filename=None, key=None, return_header=False,
                         return_filename=False)
spirouImage.spirouFITS.read_wave_file(p, hdr=None, filename=None, key=None, return_header=False,
                                      return_filename=False)
```

```
Reads the wave file (from calib database or filename)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            fitsfilename: string, the full path of for the main raw fits
                          file for a recipe
                          i.e. /data/raw/20170710/filename.fits
            fiber: string, the fiber used for this recipe (eg. AB or A or C)

:param hdr: dictionary or None, the header dictionary to look for the
                acquisition time in, if None loads the header from
                p['fitsfilename']
:param filename: string or None, the filename and path of the tilt file,
                if None gets the WAVE file from the calib database
                keyword "WAVE_{fiber}"
:param key: string or None, if None key='WAVE' else uses string as key
            from calibDB (first entry) to get wave file
:param return_header: bool, if True returns header file else just returns
                      wave file
:param return_filename: bool, if true return the filename only


if return_filename is False and return header is False

    :return wave: numpy array (2D), the wavelengths for each pixel
                  (x-direction) for each order
elif return_filename is False:
    :return wave: numpy array (2D), the wavelengths for each pixel
                  (x-direction) for each order
    :return hdict: dictionary, the header file of the wavelength solution
else:
    :return read_file: string, the file name associated with the wavelength
                       solution
```

13.8.51 ReadOrderProfile

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.read_order_profile_superposition

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ReadOrderProfile(p, hdr=None, filename=None)
spirouImage.spirouFITS.read_order_profile_superposition(p, hdr=None, filename=None)
```

```
Read the order profile superposition image from either "filename" (if not
None) or get filename from the calibration database using "p"

"ORDER_PROFILE_{X}" must be in calibration database if filename is None
where X is either p["ORDERP_FILE"] or p["FIBER"] (precedence in that order)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            ORDERP_FILE: string, the suffix for the order profile
                         calibration database key (usually the fiber type)
                         - read from "orderp_file_fpall"
            fiber: string, the fiber used for this recipe (eg. AB or A or C)
            log_opt: string, log option, normally the program name

:param hdr: dictionary or None, header dictionary (used to get the
            acquisition time if trying to get "ORDER_PROFILE_{X}" from
            the calibration database, if None uses the header from the
            first file in "ARG_FILE_NAMES" i.e. "FITSFILENAME"
:param filename: string or None, if defined no need for "hdr" or keys from
                 "p" the order profile is read straight from "filename"

:return orderp: numpy array (2D), the order profile image read from file
```

13.8.52 ResizeImage

Defined in SpirouDRS.spirouImage.spirouImage .spirouImage.resize

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.ResizeImage(image, x=None, y=None, xlow=0, xhigh=None, ylow=0, yhigh=None, getshape=True)
spirouImage.spirouImage.resize(image, x=None, y=None, xlow=0, xhigh=None, ylow=0, yhigh=None, getshape
    =True)
```

```
Resize an image based on a pixel values

:param image: numpy array (2D), the image
:param x: None or numpy array (1D), the list of x pixels
:param y: None or numpy array (1D), the list of y pixels
:param xlow: int, x pixel value (x, y) in the bottom left corner,
             default = 0
:param xhigh:  int, x pixel value (x, y) in the top right corner,
             if None default is image.shape(1)
:param ylow: int, y pixel value (x, y) in the bottom left corner,
             default = 0
:param yhigh: int, y pixel value (x, y) in the top right corner,
             if None default is image.shape(0)
:param getshape: bool, if True returns shape of newimage with newimage

if getshape = True
:return newimage: numpy array (2D), the new resized image
:return nx: int, the shape in the first dimension, i.e. data.shape[0]
:return ny: int, the shape in the second dimension, i.e. data.shape[1]

if getshape = False
:return newimage: numpy array (2D), the new resized image
```

13.8.53 WriteImage

Defined in SpirouDRS.spirouImage.spirouImage .spirouFITS.writeimage

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.WriteImage(filename, image, hdict=None, dtype=None)
spirouImage.spirouFITS.writeimage(filename, image, hdict=None, dtype=None)
```

```
Writes an image and its header to file

:param filename: string, filename to save the fits file to
:param image: numpy array (2D), the image
:param hdict: dictionary or None, header dictionary to write to fits file

            Must be in form:

                    hdict[key] = (value, comment)
            or
                    hdict[key] = value     (comment will be equal to
                                            "UNKNOWN"
            if None does not write header to fits file

:param dtype: None or hdu format type, forces the image to be in the
              format type specified (if not None)

              valid formats are for example: 'int32', 'float64'

:return None:
```

13.8.54 WriteTable

Defined in SpirouDRS.spirouImage.spirouImage .spirouTable.write_table

Python/Ipython

```
from SpirouDRS import spirouImage
spirouImage.WriteTable(table, filename, fmt='fits')
spirouImage.spirouTable.write_table(table, filename, fmt='fits')
```

```
Writes a table to file "filename" with format "fmt"

:param filename: string, the filename and location of the table to read
:param fmt: string, the format of the table to read from (must be valid
            for astropy.table to read - see below)

:return None:
```

13.9 The spirouLOCOR module

13.9.1 BoxSmoothedImage

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.smoothed_boxmean_image

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.BoxSmoothedImage(image, size, weighted=True, mode='convolve')
spirouLOCOR.spirouLOCOR.smoothed_boxmean_image(image, size, weighted=True, mode='convolve')
```

```
Produce a (box) smoothed image, smoothed by the mean of a box of
    size=2*"size" pixels.

    if mode='convolve' (default) then this is done
    by convolving a top-hat function with the image (FAST)
    - note produces small inconsistencies due to FT of top-hat function

    if mode='manual' then this is done by working out the mean in each
    box manually (SLOW)

:param image: numpy array (2D), the image
:param size: int, the number of pixels to mask before and after pixel
             (for every row)
             i.e. box runs from  "pixel-size" to "pixel+size" unless
             near an edge
:param weighted: bool, if True pixel values less than zero are weighted to
                 a value of 1e-6 and values above 0 are weighted to a value
                 of 1
:param mode: string, if 'convolve' convolves with a top-hat function of the
                     size "box" for each column (FAST) - note produces small
                     inconsistencies due to FT of top-hat function

                     if 'manual' calculates every box individually (SLOW)

:return newimage: numpy array (2D), the smoothed image
```

13.9.2 CalcLocoFits

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.calculate_location_fits

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.CalcLocoFits(coeffs, dim)
spirouLOCOR.spirouLOCOR.calculate_location_fits(coeffs, dim)
```

```
Calculates all fits in coeffs array across pixels of size=dim

:param coeffs: coefficient array,
               size = (number of orders x number of coefficients in fit)
               output array will be size = (number of orders x dim)
:param dim: int, number of pixels to calculate fit for
            fit will be done over x = 0 to dim in steps of 1
:return yfits: array,
               size = (number of orders x dim)
               the fit for each order at each pixel values from 0 to dim
```

13.9.3 FindPosCentCol

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.find_position_of_cent_col

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.FindPosCentCol(values, threshold)
spirouLOCOR.spirouLOCOR.find_position_of_cent_col(values, threshold)
```

```
Finds the central positions based on the central column values

:param values: numpy array (1D) size = number of rows,
                the central column values
:param threshold: float, the threshold above which to find pixels as being
                  part of an order

:return position: numpy array (1D), size= number of rows,
                  the pixel positions in cvalues where the centres of each
                  order should be
```

13.9.4 FindOrderCtrs

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.find_order_centers

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.FindOrderCtrs(pp, image, loc, order_num)
spirouLOCOR.spirouLOCOR.find_order_centers(pp, image, loc, order_num)
```

```
Find the centre pixels and widths of this order at specific points
along this order="order_num"

specific points are defined by steps (ic_locstepc) away from the
central pixel (ic_cent_col)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            IC_LOCSTEPC: int, the column separation for fitting orders
            IC_CENT_COL: int, the column number (x-axis) of the central
                         column
            IC_EXT_WINDOW: int, extraction window size (half size)
            IC_IMAGE_GAP: int, the gap index in the selected area
            sigdet: float, the read noise of the image
            IC_LOCSEUIL: float, Normalised amplitude threshold to accept
                         pixels for background calculation
            IC_WIDTHMIN: int, minimum width of order to be accepted
            DRS_DEBUG: int, Whether to run in debug mode
                            0: no debug
                            1: basic debugging on errors
                            2: recipes specific (plots and some code runs)
            DRS_PLOT: bool, Whether to plot (True to plot)

:param image: numpy array (2D), the image

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            ctro: numpy array (2D), storage for the centre positions
                  shape = (number of orders x number of columns (x-axis)

:param order_num: int, the current order to process

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            ctro: numpy array (2D), storage for the centre positions
                  shape = (number of orders x number of columns (x-axis)
                  updated the values for "order_num"
            sigo: numpy array (2D), storage for the width positions
                  shape = (number of orders x number of columns (x-axis)
                  updated the values for "order_num"
```

13.9.5 GetCoeffs

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.get_loc_coefficients

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.GetCoeffs(p, hdr=None, loc=None)
spirouLOCOR.spirouLOCOR.get_loc_coefficients(p, hdr=None, loc=None)
```

```
Extracts loco coefficients from parameters keys (uses header="hdr" provided
to get acquisition time or uses p['fitsfilename'] to get acquisition time if
"hdr" is None

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            fitsfilename: string, the full path of for the main raw fits
                          file for a recipe
                          i.e. /data/raw/20170710/filename.fits
            kw_LOCO_NBO: list, keyword store for the number of orders
                         located
            kw_LOCO_DEG_C: list, keyword store for the fit degree for
                           order centers
            kw_LOCO_DEG_W: list, keyword store for the fit degree for
                           order widths
            kw_LOCO_CTR_COEFF: list, keyword store for the coeff center
                               order
            kw_LOCO_FWHM_COEFF: list, keyword store for the coeff width
                                order
            LOC_FILE: string, the suffix for the location calibration
                      database key (usually the fiber type)
                         - read from "loc_file_fpall", if not defined
                           uses p["fiber"]
            fiber: string, the fiber used for this recipe (eg. AB or A or C)
            calibDB: dictionary, the calibration database dictionary
            reduced_dir: string, the reduced data directory
                         (i.e. p['DRS_DATA_REDUC']/p['arg_night_name'])
            log_opt: string, log option, normally the program name

:param hdr: dictionary, header file from FITS rec (opened by spirouFITS)
:param loc: parameter dictionary, ParamDict containing data

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            number_orders: int, the number of orders in reference spectrum
            nbcoeff_ctr: int, number of coefficients for the centre fit
            nbcoeff_wid: int, number of coefficients for the width fit
            acc: numpy array (2D), the fit coefficients array for
                  the centres fit
                  shape = (number of orders x number of fit coefficients)
            ass: numpy array (2D), the fit coefficients array for
                  the widths fit
```

13.9.6 ImageLocSuperimp

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.image_localization_superposition

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.ImageLocSuperimp(image, coeffs)
spirouLOCOR.spirouLOCOR.image_localization_superposition(image, coeffs)
```

```
Take an image and superimpose zeros over the positions in the image where
the central fits where found to be

:param image: numpy array (2D), the image
:param coeffs: coefficient array,
               size = (number of orders x number of coefficients in fit)
               output array will be size = (number of orders x dim)
:return newimage: numpy array (2D), the image with super-imposed zero filled
                  fits
```

13.9.7 InitialOrderFit

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.initial_order_fit

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.InitialOrderFit(pp, loc, mask, onum, rnum, kind, fig=None, frame=None)
spirouLOCOR.spirouLOCOR.initial_order_fit(pp, loc, mask, onum, rnum, kind, fig=None, frame=None)
```

```
Performs a crude initial fit for this order, uses the ctro positions or sigo
width values found in "FindOrderCtrs" or "find_order_centers" to do the fit

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            IC_LOCDFITC: int, order of polynomial to fit for positions
            IC_LOCDFITW: int, order of polynomial to fit for widths
            DRS_PLOT: bool, Whether to plot (True to plot)
            IC_CENT_COL: int, Definition of the central column

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            x: numpy array (1D), the order numbers
            ctro: numpy array (2D), storage for the center positions
                  shape = (number of orders x number of columns (x-axis)
            sigo: numpy array (2D), storage for the width positions
                  shape = (number of orders x number of columns (x-axis)

:param mask: numpy array (1D) of booleans, True where we have non-zero
             widths
:param onum: int, order iteration number (running number over all
             iterations)
:param rnum: int, order number (running number of successful order
             iterations only)
:param kind: string, 'center' or 'fwhm', if 'center' then this fit is for
             the central positions, if 'fwhm' this fit is for the width of
             the orders
:param fig: plt.figure, the figure to plot initial fit on
:param frame: matplotlib axis i.e. plt.subplot(), the axis on which to plot
              the initial fit on (carries the plt.imshow(image))
:return fitdata: dictionary, contains the fit data key value pairs for this
                 initial fit. keys are as follows:

        a = coefficients of the fit from key
        size = 'ic_locdfitc' [for kind='center'] or
             = 'ic_locdftiw' [for kind='fwhm']
        fit = the fity values for the fit (for x = loc['x'])
            where fity = Sum(a[i] * x^i)
        res = the residuals from y - fity
             where y = ctro [kind='center'] or
                     = sigo [kind='fwhm'])
        abs_res = abs(res)
        rms = the standard deviation of the residuals
        max_ptp = maximum residual value max(res)
        max_ptp_frac = max_ptp / rms  [kind='center']
                     = max(abs_res/y) * 100   [kind='fwhm']
```

13.9.8 LocCentralOrderPos

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.locate_order_center

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.LocCentralOrderPos(values, threshold, min_width=None)
spirouLOCOR.spirouLOCOR.locate_order_center(values, threshold, min_width=None)
```

```
Takes the values across the oder and finds the order centre by looking for
the start and end of the order (and thus the centre) above threshold

:param values: numpy array (1D) size = number of rows, the pixels in an
                order

:param threshold: float, the threshold above which to find pixels as being
                  part of an order

:param min_width: float, the minimum width for an order to be accepted

:return positions: numpy array (1D), size= number of rows,
                   the pixel positions in cvalues where the centres of each
                   order should be

:return widths:    numpy array (1D), size= number of rows,
                   the pixel positions in cvalues where the centres of each
                   order should be
```

13.9.9 MergeCoefficients

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.merge_coefficients

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.MergeCoefficients(loc, coeffs, step)
spirouLOCOR.spirouLOCOR.merge_coefficients(loc, coeffs, step)
```

```
Takes a list of coefficients "coeffs" and merges them based on "step"
using the mean of "step" blocks

i.e. shrinks a list of N coefficients to N/2 (if step = 2) where
     indices 0 and 1 are averaged, indices 2 and 3 are averaged etc

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            number_orders: int, the number of orders in reference spectrum

:param coeffs: numpy array (2D), the list of coefficients
               shape = (number of orders x number of fit parameters)

:param step: int, the step between merges
             i.e. total size before = "number_orders"
                  total size after = "number_orders"/step

:return newcoeffs: numpy array (2D), the new list of coefficients
            shape = (number of orders/step x number of fit parameters)
```

13.9.10 SigClipOrderFit

Defined in SpirouDRS.spirouLOCOR.spirouLOCOR .spirouLOCOR.sigmaclip_order_fit

Python/Ipython

```
from SpirouDRS import spirouLOCOR
spirouLOCOR.SigClipOrderFit(pp, loc, fitdata, mask, onum, rnum, kind)
spirouLOCOR.spirouLOCOR.sigmaclip_order_fit(pp, loc, fitdata, mask, onum, rnum, kind)
```

```
Performs a sigma clip fit for this order, uses the ctro positions or
sigo width values found in "FindOrderCtrs" or "find_order_centers" to do
the fit. Removes the largest residual from the initial fit (or subsequent
sigmaclips) value in x and y and recalculates the fit.

Does this until all the following conditions are NOT met:
       rms > 'ic_max_rms'   [kind='center' or kind='fwhm']
    or max_ptp > 'ic_max_ptp [kind='center']
    or max_ptp_frac > 'ic_ptporms_center'   [kind='center']
    or max_ptp_frac > 'ic_max_ptp_frac'     [kind='fwhm'

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            IC_MAX_RMS_CENTER: required when kind="center", float, Maximum
                               rms for sigma-clip order fit (center
                               positions)
            IC_MAX_RMS_FWHM: required when kind="fwhm", float, Maximum
                             rms for sigma-clip order fit (width)
            IC_LOCDFITC: int, order of polynomial to fit for positions
            IC_MAX_PTP_CENTER: required when kind="center", float, Maximum
                               peak-to-peak for sigma-clip order fit
                               (center positions)
            IC_PTPORMS_CENTER: required when kind="center", float, Maximum
                               frac ptp/rms for sigma-clip order fit
                               (center positions)
            IC_LOCDFITW: int, order of polynomial to fit for widths
            IC_MAX_PTP_FRAC_FWHM: required when kind="fwhm", float, Maximum
                                  fractional peak-to-peak for sigma-clip
                                  order fit (width)
            DRS_DEBUG: int, Whether to run in debug mode
                            0: no debug
                            1: basic debugging on errors
                            2: recipes specific (plots and some code runs)
            DRS_PLOT: bool, Whether to plot (True to plot)

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            ctro: numpy array (2D), storage for the center positions
                  shape = (number of orders x number of columns (x-axis)
            sigo: numpy array (2D), storage for the width positions
                  shape = (number of orders x number of columns (x-axis)
            max_rmpts_pos: int, maximum number of removed points in sigma
                           clipping process, for center fits
            max_rmpts_wid: int, maximum number of removed points in sigma
                           clipping process, for width fits
```

```
sigmaclip_order_fit (continued)

:param fitdata: dictionary, contains the fit data key value pairs for this
                initial fit. keys are as follows:

        a = coefficients of the fit from key
        size = 'ic_locdfitc' [for kind='centre'] or
             = 'ic_locdftiw' [for kind='fwhm']
        fit = the fity values for the fit (for x = loc['x'])
            where fity = Sum(a[i] * x^i)
        res = the residuals from y - fity
             where y = ctro [kind='center'] or
                     = sigo [kind='fwhm'])
        abs_res = abs(res)
        rms = the standard deviation of the residuals
        max_ptp = maximum residual value max(res)
        max_ptp_frac = max_ptp / rms  [kind='center']
                     = max(abs_res/y) * 100   [kind='fwhm']

:param mask: numpy array (1D) of booleans, True where we have non-zero
             widths
:param onum: int, order iteration number (running number over all
             iterations)
:param rnum: int, order number (running number of successful order
             iterations only)
:param kind: string, 'center' or 'fwhm', if 'center' then this fit is for
             the central p

:return fitdata: dictionary, contains the fit data key value pairs for this
                 initial fit. keys are as follows:

        a = coefficients of the fit from key
        size = 'ic_locdfitc' [for kind='center'] or
             = 'ic_locdftiw' [for kind='fwhm']
        fit = the fity values for the fit (for x = loc['x'])
            where fity = Sum(a[i] * x^i)
        res = the residuals from y - fity
             where y = ctro [kind='center'] or
                     = sigo [kind='fwhm'])
        abs_res = abs(res)
        rms = the standard deviation of the residuals
        max_ptp = maximum residual value max(res)
        max_ptp_frac = max_ptp / rms  [kind='center']
                     = max(abs_res/y) * 100   [kind='fwhm']
```

13.10 The spirouPOLAR module

13.10.1 SortPolarFiles

Defined in SpirouDRS.spirouPOLAR.spirouPOLAR .spirouPOLAR.sort_polar_files

Python/Ipython

```
from SpirouDRS import spirou
spirouPOLAR.SortPolarFiles(p, polardict)
spirouPOLAR.spirouPOLAR.sort_polar_files(p, polardict)
```

```
Function to sort input data for polarimetry.

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, option for logging
        REDUCED_DIR: string, directory path where reduced data are stored
        ARG_FILE_NAMES: list, list of input filenames
        KW_CMMTSEQ: string, FITS keyword where to find polarimetry
                    information

:return polardict: dictionary, ParamDict containing information on the
                   input data
```

13.10.2 LoadPolarData

Defined in SpirouDRS.spirouPOLAR.spirouPOLAR .spirouPOLAR.load_data

Python/Ipython

```
from SpirouDRS import spirou
spirouPOLAR.LoadPolarData(p, polardict, loc)
spirouPOLAR.spirouPOLAR.load_data(p, polardict, loc)
```

```
Function to load input E2DS data for polarimetry.

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, option for logging
        IC_POLAR_STOKES_PARAMS: list, list of stokes parameters
        IC_POLAR_FIBERS: list, list of fiber types used in polarimetry

:param polardict: dictionary, ParamDict containing information on the
                  input data

:param loc: parameter dictionary, ParamDict to store data

:return p, loc: parameter dictionaries,
    The updated parameter dictionary adds/updates the following:
        FIBER: saves reference fiber used for base file in polar sequence
               The updated data dictionary adds/updates the following:
        DATA: array of numpy arrays (2D), E2DS data from all fibers in
              all input exposures.
        BASENAME, string, basename for base FITS file
        HDR: dictionary, header from base FITS file
        CDR: dictionary, header comments from base FITS file
        STOKES: string, stokes parameter detected in sequence
        NEXPOSURES: int, number of exposures in polar sequence
```

13.10.3 CalculatePolarimetry

Defined in SpirouDRS.spirouPOLAR.spirouPOLAR .spirouPOLAR.calculate_polarimetry_wrapper

Python/Ipython

```
from SpirouDRS import spirou
spirouPOLAR.CalculatePolarimetry(p, loc)
spirouPOLAR.spirouPOLAR.calculate_polarimetry_wrapper(p, loc)
```

```
Function to call functions to calculate polarimetry either using
the Ratio or Difference methods.

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, option for logging
        IC_POLAR_METHOD: string, to define polar method "Ratio" or
                         "Difference"

:param loc: parameter dictionary, ParamDict containing data

:return polarfunc: function, either polarimetry_diff_method(p, loc)
                   or polarimetry_ratio_method(p, loc)
```

13.10.4 CalculateContinuum

Defined in SpirouDRS.spirouPOLAR.spirouPOLAR .spirouPOLAR.calculate_continuum

Python/Ipython

```
from SpirouDRS import spirou
spirouPOLAR.CalculateContinuum(p, loc, in_wavelength=True)
spirouPOLAR.spirouPOLAR.calculate_continuum(p, loc, in_wavelength=True)
```

```
Function to calculate the continuum polarization

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, option for logging
        IC_POLAR_CONT_BINSIZE: int, number of points in each sample bin
        IC_POLAR_CONT_OVERLAP: int, number of points to overlap before and
                               after each sample bin
        IC_POLAR_CONT_TELLMASK: list of float pairs, list of telluric bands,
                                i.e, a list of wavelength ranges ([wl0,wlf])
                                for telluric absorption

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        POL: numpy array (2D), e2ds degree of polarization data
        POLERR: numpy array (2D), e2ds errors of degree of polarization
        NULL1: numpy array (2D), e2ds 1st null polarization
        NULL2: numpy array (2D), e2ds 2nd null polarization
        STOKESI: numpy array (2D), e2ds Stokes I data
        STOKESIERR: numpy array (2D), e2ds errors of Stokes I

:param in_wavelength: bool, to indicate whether or not there is wave cal

:return loc: parameter dictionary, the updated parameter dictionary
    Adds/updates the following:
        FLAT_X: numpy array (1D), flatten polarimetric x data
        FLAT_POL: numpy array (1D), flatten polarimetric pol data
        FLAT_POLERR: numpy array (1D), flatten polarimetric pol error data
        FLAT_STOKESI: numpy array (1D), flatten polarimetric stokes I data
        FLAT_STOKESIERR: numpy array (1D), flatten polarimetric stokes I
                         error data
        FLAT_NULL1: numpy array (1D), flatten polarimetric null1 data
        FLAT_NULL2: numpy array (1D), flatten polarimetric null2 data
        CONT_POL: numpy array (1D), e2ds continuum polarization data
                  interpolated from xbin, ybin points, same shape as
                  FLAT_POL
        CONT_XBIN: numpy array (1D), continuum in x polarization samples
        CONT_YBIN: numpy array (1D), continuum in y polarization samples
```

13.10.5 CalculateStokesI

Defined in SpirouDRS.spirouPOLAR.spirouPOLAR .spirouPOLAR.calculate_stokes_I

Python/Ipython

```
from SpirouDRS import spirou
spirouPOLAR.CalculateStokesI(p, loc)
spirouPOLAR.spirouPOLAR.calculate_stokes_I(p, loc)
```

```
Function to calculate the Stokes I polarization

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, option for logging

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        DATA: array of numpy arrays (2D), E2DS data from all fibers in
              all input exposures.
        NEXPOSURES: int, number of exposures in polar sequence

:return loc: parameter dictionary, the updated parameter dictionary
    Adds/updates the following:
        STOKESI: numpy array (2D), the Stokes I parameters, same shape as
                 DATA
        STOKESIERR: numpy array (2D), the Stokes I error parameters, same
                    shape as DATA
```

13.11 The spirouRV module

13.11.1 CalcRVdrift2D

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.calculate_rv_drifts_2d

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.CalcRVdrift2D(speref, spe, wave, sigdet, threshold, size)
spirouRV.spirouRV.calculate_rv_drifts_2d(speref, spe, wave, sigdet, threshold, size)
```

```
Calculate the RV drift between the REFERENCE (speref) and COMPARISON (spe)
extracted spectra.

:param speref: numpy array (2D), the REFERENCE extracted spectrum
               size = (number of orders by number of columns (x-axis))
:param spe:  numpy array (2D), the COMPARISON extracted spectrum
             size = (number of orders by number of columns (x-axis))
:param wave: numpy array (2D), the wave solution for each pixel
:param sigdet: float, the read noise (sigdet) for calculating the
               noise array
:param threshold: float, upper limit for pixel values, above this limit
                  pixels are regarded as saturated
:param size: int, size (in pixels) around saturated pixels to also regard
             as bad pixels

:return rvdrift: numpy array (1D), the RV drift between REFERENCE and
                 COMPARISON spectrum for each order
```

13.11.2 Coravelation

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.coravelation

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.Coravelation(p, loc)
spirouRV.spirouRV.coravelation(p, loc)
```

```
Calculate the CCF and fit it with a Gaussian profile

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            ccf_berv: float, the barycentric Earth RV (berv)
            ccf_berv_max: float, the maximum barycentric Earth RV
            target_rv: float, the target RV
            ccf_width: float, the CCF width
            ccf_step: float, the CCF step
            ccf_det_noise: float, the detector noise to use in the ccf
            ccf_fit_type: int, the type of fit for the CCF fit
            log_opt: string, log option, normally the program name
            DRS_DEBUG: int, Whether to run in debug mode
                            0: no debug
                            1: basic debugging on errors
                            2: recipes specific (plots and some code runs)
            DRS_PLOT: bool, Whether to plot (True to plot)

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            wave_ll: numpy array (1D), the line list values
            param_ll: numpy array (1d), the line list fit coefficients
                      (used to generate line list - read from file defined)
            ll_mask_d: numpy array (1D), the size of each line
                       (in wavelengths)
            ll_mask_ctr: numpy array (1D), the central point of each line
                         (in wavelengths)
            w_mask: numpy array (1D), the weight mask
            e2dsff: numpy array (2D), the flat fielded E2DS spectrum
                    shape = (number of orders x number of columns in image
                                                    (x-axis dimension) )
            blaze: numpy array (2D), the blaze function
                    shape = (number of orders x number of columns in image
                                                    (x-axis dimension) )

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            rv_ccf: numpy array (1D), the radial velocities for the CCF
            ccf: numpy array (2D), the CCF for each order and each RV
                 shape = (number of orders x number of RV points)
            ccf_max: float, numpy array (1D), the max value of the CCF for
                     each order
            pix_passed_all: numpy array (1D), the weighted line list
                            position for each order?
            tot_line: numpy array (1D), the total number of lines for each
                      order
            ll_range_all: numpy array (1D), the weighted line list width for
                      each order
            ccf_noise: numpy array (2D), the CCF noise for each order and
                       each RV
                       shape = (number of orders x number of RV points)
```

13.11.3 CreateDriftFile

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.create_drift_file

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.CreateDriftFile(p, loc)
spirouRV.spirouRV.create_drift_file(p, loc)
```

```
Creates a reference ascii file that contains the positions of the FP peaks
Returns the pixels positions and Nth order of each FP peak

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            drift_peak_border_size: int, the border size (edges in
                                    x-direction) for the FP fitting
                                    algorithm
            drift_peak_fpbox_size: int, the box half-size (in pixels) to
                                   fit an individual FP peak to - a
                                   Gaussian will be fit to +/- this size
                                   from the centre of the FP peak
            drift_peak_peak_sig_lim: dictionary, the sigma above the median
                                     that a peak must have to be recognised
                                     as a valid peak (before fitting a
                                     Gaussian) dictionary must have keys
                                     equal to the lamp types (hc, fp)
            drift_peak_inter_peak_spacing: int, the minimum spacing between
                                           peaks in order to be recognised
                                           as a valid peak (before fitting
                                           a Gaussian)
            log_opt: string, log option, normally the program name

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            speref: numpy array (2D), the reference spectrum
            wave: numpy array (2D), the wave solution image
            lamp: string, the lamp type (either 'hc' or 'fp')

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            ordpeak: numpy array (1D), the order number for each valid FP
                     peak
            xpeak: numpy array (1D), the central position each Gaussian fit
                   to valid FP peak
            ewpeak: numpy array (1D), the FWHM of each Gaussian fit
                    to valid FP peak
            vrpeak: numpy array (1D), the radial velocity drift for each
                    valid FP peak
            llpeak: numpy array (1D), the delta wavelength for each valid
                    FP peak
            amppeak: numpy array (1D), the amplitude for each valid FP peak
```

13.11.4 DeltaVrms2D

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.delta_v_rms_2d

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.DeltaVrms2D(spe, wave, sigdet, threshold, size)
spirouRV.spirouRV.delta_v_rms_2d(spe, wave, sigdet, threshold, size)
```

```
Compute the photon noise uncertainty for all orders (for the 2D image)

:param spe: numpy array (2D), the extracted spectrum
            size = (number of orders by number of columns (x-axis))
:param wave: numpy array (2D), the wave solution for each pixel
:param sigdet: float, the read noise (sigdet) for calculating the
               noise array
:param threshold: float, upper limit for pixel values, above this limit
                  pixels are regarded as saturated
:param size: int, size (in pixels) around saturated pixels to also regard
             as bad pixels

:return dvrms2: numpy array (1D), the photon noise for each pixel (squared)
:return weightedmean: float, weighted mean photon noise across all orders
```

13.11.5 DriftPerOrder

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.drift_per_order

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.DriftPerOrder(loc, fileno)
spirouRV.spirouRV.drift_per_order(loc, fileno)
```

13.11.6 DriftAllOrders

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.drift_all_orders

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.DriftAllOrders(loc, fileno, nomin, nomax)
spirouRV.spirouRV.drift_all_orders(loc, fileno, nomin, nomax)
```

```
Work out the weighted mean drift across all orders

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            drift: numpy array (2D), the median drift values for each
                   file and each order
                   shape = (number of files x number of orders)
            drift_left: numpy array (2D), the median drift values for the
                        left half of each order (for each file and each
                        order)
                        shape = (number of files x number of orders)
            drift_right: numpy array (2D), the median drift values for the
                         right half of each order (for each file and each
                         order)
                         shape = (number of files x number of orders)
            errdrift: numpy array (2D), the error in the drift for each
                      file and each order
                      shape = (number of files x number of orders)

:param fileno: int, the file number (iterator number)
:param nomin: int, the first order to use (i.e. from nomin to nomax)
:param nomax: int, the last order to use (i.e. from nomin to nomax)

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            meanrv: numpy array (1D), the weighted mean drift, for each file
                    shape = (number of files)
            meanrv_left: numpy array (1D), the weighted mean drift for the
                         left half of each order, for each file
                         shape = (number of files)
            meanrv_right: numpy array (1D), the weighted mean drift for the
                          right half of each order, for each file
                          shape = (number of files)
            merrdrift: numpy array (1D), the error in weighted mean for
                       each file
                       shape = (number of files)
```

13.11.7 FitCCF

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.fit_ccf

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.FitCCF(rv, ccf, fit_type)
spirouRV.spirouRV.fit_ccf(rv, ccf, fit_type)
```

```
Fit the CCF to a guassian function

:param rv: numpy array (1D), the radial velocities for the line
:param ccf: numpy array (1D), the CCF values for the line
:param fit_type: int, if "0" then we have an absorption line
                      if "1" then we have an emission line

:return result: numpy array (1D), the fit parameters in the
                following order:

            [amplitude, center, fwhm, offset from 0 (in y-direction)]

:return ccf_fit: numpy array (1D), the fit values, i.e. the Gaussian values
                 for the fit parameters in "result"
```

13.11.8 GetDrift

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.get_drift

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.GetDrift(p, sp, ordpeak, xpeak0, gaussfit=False)
spirouRV.spirouRV.get_drift(p, sp, ordpeak, xpeak0, gaussfit=False)
```

```
Get the centroid of all peaks provided an input peak position

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            drift_peak_fpbox_size: int, the box half-size (in pixels) to
                                   fit an individual FP peak to - a
                                   Gaussian will be fit to +/- this size
                                   from the centre of the FP peak
            drift_peak_exp_width: float, the expected width of FP peaks -
                                  used to "normalise" peaks (which are then
                                  subsequently removed if >
                                  drift_peak_norm_width_cut
            log_opt: string, log option, normally the program name

:param sp: numpy array (2D), E2DS fits file with FP peaks
           size = (number of orders x number of pixels in x-dim of image)
:param ordpeak: numpy array (1D), order of each peak
:param xpeak0: numpy array (1D), position in the x dimension of all peaks
:param gaussfit: bool, if True uses a Gaussian fit to get each centroid
                 (slow) or adjusts a barycentre (gaussfit=False)

:return xpeak: numpy array (1D), the central positions of the peaks
```

13.11.9 GetCCFMask

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.get_ccf_mask

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.GetCCFMask(p, loc, filename=None)
spirouRV.spirouRV.get_ccf_mask(p, loc, filename=None)
```

```
Get the CCF mask

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            ccf_mask: string, the name (and or location) of the CCF
                      mask file
            ic_w_mask_min: float, the weight of the CCF mask (if 1 force
                           all weights equal)
            ic_mask_width: float, the width of the template line
                           (if 0 use natural
            log_opt: string, log option, normally the program name

:param loc: parameter dictionary, ParamDict containing data

:param filename: string or None, the filename and location of the ccf mask
                 file, if None then file names is gotten from p["ccf_mask"]

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            ll_mask_d: numpy array (1D), the size of each pixel
                       (in wavelengths)
            ll_mask_ctr: numpy array (1D), the central point of each pixel
                         (in wavelengths)
            w_mask: numpy array (1D), the weight mask
```

13.11.10 PearsonRtest

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.pearson_rtest

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.PearsonRtest(nbo, spe, speref)
spirouRV.spirouRV.spirouRV.pearson_rtest(nbo, spe, speref)
```

```
Perform a Pearson R test on each order in spe against speref

:param nbo: int, the number of orders
:param spe: numpy array (2D), the extracted array for this iteration
            size = (number of orders x number of pixels in x-dim)
:param speref: numpy array (2D), the extracted array for the reference
               image, size = (number of orders x number of pixels in x-dim)

:return cc_orders: numpy array (1D), the Pearson correlation coefficients
                   for each order, size = (number of orders)
```

13.11.11 RemoveWidePeaks

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.remove_wide_peaks

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.RemoveWidePeaks(p, loc, expwidth=None, cutwidth=None)
spirouRV.spirouRV.remove_wide_peaks(p, loc, expwidth=None, cutwidth=None)
```

```
Remove peaks that are too wide

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            drift_peak_exp_width: float, the expected width of FP peaks -
                                  used to "normalise" peaks (which are then
                                  subsequently removed if >
                                  drift_peak_norm_width_cut
            drift_peak_norm_width_cut: float, the "normalised" width of
                                       FP peaks that is too large
                                       normalised width = FP FWHM -
                                       drift_peak_exp_width cut is
                                       essentially:=
                                       FP FWHM < (drift_peak_exp_width +
                                       drift_peak_norm_width_cut)
            log_opt: string, log option, normally the program name

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            ordpeak: numpy array (1D), the order number for each valid FP
                     peak
            xpeak: numpy array (1D), the central position each Gaussian fit
                   to valid FP peak
            ewpeak: numpy array (1D), the FWHM of each Gaussian fit
                    to valid FP peak
            vrpeak: numpy array (1D), the radial velocity drift for each
                    valid FP peak
            llpeak: numpy array (1D), the delta wavelength for each valid
                    FP peak
            amppeak: numpy array (1D), the amplitude for each valid FP peak

:param expwidth: float or None, the expected width of FP peaks - used to
                 "normalise" peaks (which are then subsequently removed
                 if > "cutwidth") if expwidth is None taken from
                 p["drift_peak_exp_width"]
:param cutwidth: float or None, the normalised width of FP peaks that is too
                 large normalised width FP FWHM - expwidth
                 cut is essentially: FP FWHM < (expwidth + cutwidth), if
                 cutwidth is None taken from p["drift_peak_norm_width_cut"]

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            ordpeak: numpy array (1D), the order number for each valid FP
                     peak (masked to remove wide peaks)
            xpeak: numpy array (1D), the central position each Gaussian fit
                   to valid FP peak (masked to remove wide peaks)
            ewpeak: numpy array (1D), the FWHM of each Gaussian fit
                    to valid FP peak (masked to remove wide peaks)
            vrpeak: numpy array (1D), the radial velocity drift for each
                    valid FP peak (masked to remove wide peaks)
            llpeak: numpy array (1D), the delta wavelength for each valid
                    FP peak (masked to remove wide peaks)
```

13.11.12 RemoveZeroPeaks

Defined in SpirouDRS.spirouRV.spirouRV `.remove_zero_peaks`

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.RemoveZeroPeaks(p, loc)
spirouRV.spirouRV.remove_zero_peaks(p, loc)
```

```
Remove peaks that have a value of zero

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            xref: numpy array (1D), the central positions of the peaks
            ordpeak: numpy array (1D), the order number for each valid FP
                     peak
            xpeak: numpy array (1D), the central position each Gaussian fit
                   to valid FP peak
            ewpeak: numpy array (1D), the FWHM of each Gaussian fit
                    to valid FP peak
            vrpeak: numpy array (1D), the radial velocity drift for each
                    valid FP peak
            llpeak: numpy array (1D), the delta wavelength for each valid
                    FP peak
            amppeak: numpy array (1D), the amplitude for each valid FP peak

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            xref: numpy array (1D), the central positions of the peaks
                  (masked with zero peaks removed)
            ordpeak: numpy array (1D), the order number for each valid FP
                     peak (masked with zero peaks removed)
            xpeak: numpy array (1D), the central position each Gaussian fit
                   to valid FP peak (masked with zero peaks removed)
            ewpeak: numpy array (1D), the FWHM of each Gaussian fit
                    to valid FP peak (masked with zero peaks removed)
            vrpeak: numpy array (1D), the radial velocity drift for each
                    valid FP peak (masked with zero peaks removed)
            llpeak: numpy array (1D), the delta wavelength for each valid
                    FP peak (masked with zero peaks removed)
            amppeak: numpy array (1D), the amplitude for each valid FP peak
                     (masked with zero peaks removed)
```

13.11.13 ReNormCosmic2D

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.remove_zero_peaks

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.ReNormCosmic2D(p, loc)
spirouRV.spirouRV.remove_zero_peaks(p, loc)
```

```
Remove peaks that have a value of zero

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            xref: numpy array (1D), the central positions of the peaks
            ordpeak: numpy array (1D), the order number for each valid FP
                     peak
            xpeak: numpy array (1D), the central position each Gaussian fit
                   to valid FP peak
            ewpeak: numpy array (1D), the FWHM of each Gaussian fit
                    to valid FP peak
            vrpeak: numpy array (1D), the radial velocity drift for each
                    valid FP peak
            llpeak: numpy array (1D), the delta wavelength for each valid
                    FP peak
            amppeak: numpy array (1D), the amplitude for each valid FP peak

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            xref: numpy array (1D), the central positions of the peaks
                  (masked with zero peaks removed)
            ordpeak: numpy array (1D), the order number for each valid FP
                     peak (masked with zero peaks removed)
            xpeak: numpy array (1D), the central position each Gaussian fit
                   to valid FP peak (masked with zero peaks removed)
            ewpeak: numpy array (1D), the FWHM of each Gaussian fit
                    to valid FP peak (masked with zero peaks removed)
            vrpeak: numpy array (1D), the radial velocity drift for each
                    valid FP peak (masked with zero peaks removed)
            llpeak: numpy array (1D), the delta wavelength for each valid
                    FP peak (masked with zero peaks removed)
            amppeak: numpy array (1D), the amplitude for each valid FP peak
                     (masked with zero peaks removed)
```

13.11.14 ReNormCosmic2D

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.renormalise_cosmic2d

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.ReNormCosmic2D(speref, spe, threshold, size, cut)
spirouRV.spirouRV.renormalise_cosmic2d(speref, spe, threshold, size, cut)
```

```
Correction of the cosmics and re-normalisation by comparison with
reference spectrum (for the 2D image)

:param speref: numpy array (2D), the REFERENCE extracted spectrum
               size = (number of orders by number of columns (x-axis))
:param spe:  numpy array (2D), the COMPARISON extracted spectrum
             size = (number of orders by number of columns (x-axis))
:param threshold: float, upper limit for pixel values, above this limit
                  pixels are regarded as saturated
:param size: int, size (in pixels) around saturated pixels to also regard
             as bad pixels
:param cut: float, define the number of standard deviations cut at in
                    cosmic renormalisation

:return spen: numpy array (2D), the corrected normalised COMPARISON
              extracted spectrum
:return cnormspe: numpy array (1D), the flux ratio for each order between
                  corrected normalised COMPARISON extracted spectrum and
                  REFERENCE extracted spectrum
:return cpt: float, the total flux above the "cut" parameter
             (cut * standard deviations above median)
```

13.11.15 SigmaClip

Defined in SpirouDRS.spirouRV.spirouRV .spirouRV.sigma_clip

Python/Ipython

```
from SpirouDRS import spirouRV
spirouRV.SigmaClip(loc, sigma=1.0)
spirouRV.spirouRV.sigma_clip(loc, sigma=1.0)
```

```
Perform a sigma clip on dv

:param loc: parameter dictionary, ParamDict containing data
        Must contain at least:
            dv: numpy array (1D), the drift values
            ordpeak: numpy array (1D), the order number for each drift
                     value

:param sigma: float, the sigma of the clip (away from the median)

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            dvc: numpy array (1D), the sigma clipped drift values
            orderpeakc: numpy array (1D), the order numbers for the sigma
                        clipped drift values
```

13.12 The spirouStartup module

13.12.1 Begin

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.run_begin

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.Begin(quiet=False)
spirouStartup.spirouStartup.run_begin(quiet=False)
```

```
Begin DRS - Must be run at start of every recipe
- loads the parameters from the primary configuration file, displays
  title, checks priamry constants and displays initial parameterization

:param recipe: string, the recipe name
:param quiet: bool, if True no messages are displayed

:return cparams: parameter dictionary, ParamDict constants from primary
                 configuration file
        Adds the following:
            all constants in primary configuration file
            DRS_NAME: string, the name of the DRS
            DRS_VERSION: string, the version of the DRS
```

13.12.2 DisplayTitle

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.display_title

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.DisplayTitle(title)
spirouStartup.spirouStartup.display_title(title)
```

```
Display any title between HEADER bars via the WLOG command

:param title: string, title string

:return None:
```

13.12.3 DisplaySysInfo

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.display_system_info

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.DisplaySysInfo()
spirouStartup.spirouStartup.display_system_info()
```

```
Display system information via the WLOG command

:param logonly: bool, if True will only display in the log (not to screen)
                default=True, if False prints to both log and screen

:return None:
```

13.12.4 GetCustomFromRuntime

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.get_custom_from_run_time_args

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.GetCustomFromRuntime(positions=None, types=None, names=None,
                                   required=None, calls=None, cprior=None,
                                   lognames=None, last_multi=False):
spirouStartup.spirouStartup.get_custom_from_run_time_args(positions=None, types=None, names=None,
                                   required=None, calls=None, cprior=None,
                                   lognames=None, last_multi=False):
```

```
Extract custom arguments from defined positions in sys.argv (defined at
run time)

:param positions: list of integers or None, the positions of the arguments
                  (i.e. first argument is 0)

:param types: list of python types or None, the type (i.e. int, float) for
              each argument. Note if last_multi = True, the type of the
              last defined parameter should be the type of each argument
              (but the output parameter will be a list of this type of
              arguments)

:param names: list of strings, the names of each argument (to access in
              parameter dictionary once extracted)

:param required: list of bools or None, states whether the program
                 should exit if runtime argument not found

:param calls: list of objects or None, if define these are the values that
              come from a function call (overwrite command line arguments)

:param cprior: list of bools, if True the call takes priority over arguments
               defined at run time, if False run time arguments take
               priority

:param lognames: list of strings, the names displayed in the log (on error)
                 theses should be similar to "names" but in a form the
                 user can easily understand for each variable

:param last_multi: bool, if True then last argument in positions/types/
                   names adds all additional arguments into a list

:return values: dictionary, if run time arguments are correct python type
                the name-value pairs are returned
```

13.12.5 GetFile

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.get_file

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.GetFile(p, path, name=None, prefixes=None, kind=None)
spirouStartup.spirouStartup.get_file(p, path, name=None, prefixes=None, kind=None)
```

```
Get full file path and check the path and file exist

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            program: string, the recipe/way the script was called
                     i.e. from sys.argv[0]

:param path: string, either the directory to the folder (if name is None) or
             the full path to the file
:param name: string or None, the name of the file, if None name is assumed
             to be in path
:param prefixes: string, list of strings or None, if not None this
                 substring must be in the filename
:param kind: string or None, the type of file (for logging)

:return location: string, the full file path of the file
```

13.12.6 GetFiles

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.get_files

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.GetFiles(p, path, names, prefixes=None, kind=None)
spirouStartup.spirouStartup.get_files(p, path, names, prefixes=None, kind=None)
```

```
Get a set of full file path and check the path and file exist
(wrapper around get_files)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            program: string, the recipe/way the script was called
                     i.e. from sys.argv[0]

:param path: string, either the directory to the folder (if name is None) or
             the full path to the files
:param names: list of strings, the names of the files
:param prefixes: string, list of strings or None, if not None this
                 substring must be in the filenames
:param kind: string or None, the type of files (for logging)

:return locations: list of strings, the full file paths of the files
```

13.12.7 GetFiberType

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.get_fiber_type

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.GetFiberType(p, filename, fibertypes=None)
spirouStartup.spirouStartup.get_fiber_type(p, filename, fibertypes=None)
```

```
Get fiber types and search for a valid fiber type in filename

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            FIBER_TYPES: list of strings, the types of fiber available
                         (i.e. ['AB', 'A', 'B', 'C'])
            log_opt: string, log option, normally the program name

:param filename: string, the filename to search for fiber types in
:param fibertypes: list of strings, the fiber types to search for

:return fiber: string, the fiber found (exits via WLOG if no fiber found)
```

13.12.8 LoadArguments

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.load_arguments

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.LoadArguments(cparams, night_name=None, files=None, customargs=None,
                            mainfitsfile=None, mainfitsdir=None, quiet=False)
spirouStartup.spirouStartup.load_arguments(cparams, night_name=None, files=None, customargs=None,
                                           mainfitsfile=None, mainfitsdir=None, quiet=False)
```

```
Deal with loading run time arguments:

1) display help file (if requested and exists)
2) loads run time arguments (and custom arguments, see below)
3) loads other config files

:param cparams: parameter dictionary, ParamDict containing constants

:param night_name: string or None, the name of the directory in DRS_DATA_RAW
                   to find the files in

                   if None (undefined) uses the first argument in command
                   line (i.e. sys.argv[1])

                   if defined overwrites call from
                   command line (i.e. overwrites sys.argv)

                   stored in p['ARG_NIGHT_NAME']

:param files: list of strings or None, the files to use for this program

              if None (undefined) uses the second and all other arguments in
              the command line (i.e. sys.argv[2:])

              if defined overwrites call from command line

              stored in p['ARG_FILE_NAMES']

:param customargs: None or list of strings, if list of strings then instead
                   of getting the standard runtime arguments

       i.e. in form:

            program.py night_dir arg_file_names[0] arg_file_names[1]...

       loads all arguments into customargs

       i.e. if customargs = ['night_dir', 'filename', 'a', 'b', 'c']
       expects command line arguments to be:

            program.py night_dir filename a b c

:param mainfitsfile: string or None, if "customargs" is not None (i.e. if we
                     are using custom arguments) we must define one
                     of the parameters in "customargs" to be the main fits
                     file (fitsfilename and arg_file_names[0]).
                     The parameter MUST be a string, a fits file,
                     and have HEADER key defining the acquisition time
                     as defined in kw_ACQTIME_KEY in spirouKeywords.py
                     if not using custom arguments (i.e. customargs=None
```

13.12.9 InitialFileSetup

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.initial_file_setup

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.InitialFileSetup(p, files=None, calibdb=False, no_night_name=False,
                               no_files=False, skipcheck=False)
spirouStartup.spirouStartup.initial_file_setup(p, files=None, calibdb=False,
                                               no_night_name=False, no_files=False, skipcheck=False)
```

```
Sets up the initial files (obtained with load_arguments) for a standard run
 (i.e. requiring night_name and files) and performs checks based on the
 recipe name p['RECIPE'] to see if files are correct for this recipe. If
 calibdb is True loads the calibration database, if no_night_name is True
 no night name is required. If no files is True no files are required. If
 skipcheck is True file is not checked.

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, log option, normally the program name
        RECIPE: string, the recipe name (must be defined in recipe control
                file)
        arg_night_name: string, the folder within data raw directory
                        containing files (also reduced directory) i.e.
                        /data/raw/20170710 would be "20170710"
        arg_file_names: list, list of files taken from the command line
                        (or call to recipe function) must have at least
                        one string filename in the list
        fitsfilename: string, the full path of for the main raw fits
                      file for a recipe
                      i.e. /data/raw/20170710/filename.fits

:param files: list or None, the files passed in as arguments
              (If None comes from ARG_FILE_NAMES)
:param calibdb: bool, if True calibration database loaded
:param no_night_name: bool, if True no night name expected
:param no_files: bool, if True no files expected
:param skipcheck: bool, if True does not check files


:return p: parameter dictionary, the updated parameter dictionary
        Adds the following:
            DPRTYPE: string, the datatype header key
            PREPROCESSED: bool, if True file is processed (or unchecked)

returns SystemExit if file is not valid for recipe
```

13.12.10 LoadCalibDB

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.load_calibdb

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.LoadCalibDB(p, calibdb=True)
spirouStartup.spirouStartup.load_calibdb(p, calibdb=True)
```

```
Load calibration (on start-up) this is loaded by default when
spirouStartup.spirouStartup.initial_file_setup
(spirouStartup.InitialFileSetup) so this is only needed to be run when
InitialFileSetup is not used (i.e. when custom arguments are used)


:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            DRS_CALIB_DB: string, the directory that the calibration
                          files should be saved to/read from
:param calibdb: bool, whether to load the calibration database (if False
                just makes sure DRS_CALIB_DB exists (and creates it if it
                doesn't)

:return p: parameter dictionary, the updated parameter dictionary
        Adds the following:
            if calibdb is True:
                calibDB: dictionary, the calibration database dictionary
```

13.12.11 LoadMinimum

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.load_minimum

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.LoadMinimum(p, customargs=None)
spirouStartup.spirouStartup.load_minimum(p, customargs=None)
```

```
Load minimal settings (without fitsfilename, arg_file_names, arg_file_dir
etc)

:param p: parameter dictionary, ParamDict containing constants
:param customargs: None or list of strings, if list of strings then instead
                   of getting the standard runtime arguments

       i.e. in form:

            program.py night_dir arg_file_names[0] arg_file_names[1]...

       loads all arguments into customargs

       i.e. if customargs = ['night_dir', 'filename', 'a', 'b', 'c']
       expects command line arguments to be:

            program.py night_dir filename a b c

:return p: dictionary, parameter dictionary
        Adds the following:
            program: string, the recipe/way the script was called
                     i.e. from sys.argv[0]
            log_opt: string, log option, normally the program name
            arg_night_name: string, empty (i.e. '')
            reduced_dir: string, the reduced data directory
                         (i.e. p['DRS_DATA_REDUC']/p['ARG_NIGHT_NAME'])
            raw_dir: string, the raw data directory
                     (i.e. p['DRS_DATA_RAW']/p['ARG_NIGHT_NAME'])
```

13.12.12 LoadOtherConfig

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.load_other_config_file

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.LoadOtherConfig(p, key, logthis=True, required=False)
spirouStartup.spirouStartup.load_other_config_file(p, key, logthis=True, required=False)
```

```
Load a secondary configuration file from p[key] with wrapper to deal
with ConfigErrors (pushed to WLOG)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            key: "key" defined in call

:param key: string, the key in "p" storing the location of the secondary
            configuration file
:param logthis: bool, if True loading of this config file is logged to
                screen/log file
:param required: bool, if required is True then the secondary config file
                 is required for the DRS to run and a ConfigError is raised
                 (program exit)

:return p: parameter, dictionary, the updated parameter dictionary with
           the secondary configuration files loaded into it as key/value
           pairs
```

13.12.13 SingleFileSetup

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.SingleFileSetup(p, filename, log=True, skipcheck=False)
spirouStartup.spirouStartup.single_file_setup(p, filename, log=True, skipcheck=False)
```

```
Check "filename" is valid for recipe p["RECIPE"] and get the correct path,
logs the output if log is True, and skips the check if skipcheck is True

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, log option, normally the program name
        RECIPE: string, the recipe name (must be defined in recipe control
                file)
:param filename: string, the filename to check
:param log: bool, whether to log "Now Processing" message
:param skipcheck: bool, whether to skip the check

:return p: parameter dictionary, the updated parameter dictionary
        Adds the following:
            DPRTYPE: string, the datatype header key
            PREPROCESSED: bool, if True file is processed (or unchecked)
:return location: string, the path of the filename (i.e. the raw or
                  reduced directory)

returns SystemExit if file is not valid for recipe
```

13.12.14 MultiFileSetup

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.multi_file_setup

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.MultiFileSetup(p, files=None, log=True, skipcheck=False)
spirouStartup.spirouStartup.multi_file_setup(p, files=None, log=True, skipcheck=False)
```

```
Wrapper for single_file_setup, checks that "files" are valid for recipe
p["RECIPE"] and get the correct path, logs the output if log is True,
and skips the check if skipcheck is True

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, log option, normally the program name
        RECIPE: string, the recipe name (must be defined in recipe control
                file)
:param files: list of string, the filenames to check
:param log: bool, whether to log "Now Processing" message
:param skipcheck: bool, whether to skip the check

:return p: parameter dictionary, the updated parameter dictionary
        Adds the following:
            DPRTYPE: string, the datatype header key
            PREPROCESSED: bool, if True file is processed (or unchecked)
:return locations: list of string, the paths of the filename
                   (i.e. the raw or reduced directory)

returns SystemExit if any file is not valid for recipe
```

13.12.15 Exit

Defined in SpirouDRS.spirouStartup.spirouStartup .spirouStartup.exit_script

Python/Ipython

```
from SpirouDRS import spirouStartup
spirouStartup.Exit(ll)
spirouStartup.spirouStartup.exit_script(ll)
```

```
Exit script for handling interactive endings to sessions (if DRS_PLOT is
active)

:param ll: dict, the local variables

:return None:
```

13.13 The spirouTHORCA module

13.13.1 CalcInstrumentDrift

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA `.spirouWAVE.calculate_instrument_drift`

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.CalcInstrumentDrift
spirouTHORCA.spirouWAVE.calculate_instrument_drift
```

```
Calculate the instrumental drift between the reference file and the current
file

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            DRS_PLOT: bool, if True do plots else do not
            FIBER: string, the fiber type (i.e. AB or A or B or C)
            IC_HC_N_ORD_FINAL: int, the order to which the solution is
                               calculated
            IC_WAVE_IDRIFT_NOISE: float, the noise used in drift calculation
            IC_WAVE_IDRIFT_BOXSIZE: int, the size around a saturated pixel
                                    to count as bad
            IC_WAVE_IDRIFT_MAXFLUX: int, the maximum flux for a good
                                    (unsaturated) pixel
            IC_WAVE_IDRIFT_RV_CUT: float, the rv cut above which rv from
                                   orders are not used
            QC_WAVE_IDRIFT_NBORDEROUT: int, the maximum number of orders to
                                       remove from RV calculation
            QC_WAVE_IDRIFT_RV_MAX: float, the maximum allowed drift (in m/s)

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        HCDATA: numpy array (2D), the image data
        HCHDR: dictionary, the header dictionary for HCDATA

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            DRIFT_REF: string, the filename of the reference data
            DRIFT_RV: float, the mean instrumental RV
            DRIFT_NBCOS: int, the number of cosmic pixels found
            DRIFT_RFLUX: float, the mean flux ratio
            DRIFT_NBORDKILL: int, the number of orders removed
            DRIFT_NOISE: flat, the weighted mean uncertainty on the RV
```

13.13.2 CalcLittrowSolution

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.calculate_littrow_sol

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.CalcLittrowSolution
spirouTHORCA.spirouTHORCA.calculate_littrow_sol
```

```
Calculate the Littrow solution for this iteration for a set of cut points

Uses ALL_LINES_i  where i = iteration to calculate the littrow solutions
for defiend cut points (given a cut_step and fit_deg of
IC_LITTROW_CUT_STEP_i and IC_LITTROW_FIT_DEG_i where i = iteration)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, log option, normally the program name
        FIBER: string, the fiber type (i.e. AB or A or B or C)
        IC_LITTROW_REMOVE_ORDERS: list of ints, if not empty removes these
                                  orders from influencing the fit
        IC_LITTROW_ORDER_INIT: int, defines the first order to for the fit
                               solution
        IC_HC_N_ORD_START: int, defines first order HC solution was
                           calculated from
        IC_HC_N_ORD_FINAL: int, defines last order HC solution was
                           calculated to
        IC_LITTROW_CUT_STEP_i: int, defines the step to use between
                               cut points
        IC_LITTROW_FIT_DEG_i: int, defines the polynomial fit degree

        where i = iteration

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        ECHELLE_ORDERS: numpy array (1D), the echelle order numbers
        HCDATA: numpy array (2D), the image data (used for shape)
        ALL_LINES_i: list of numpy arrays, length = number of orders
                     each numpy array contains gaussian parameters
                     for each found line in that order

        where i = iteration

:param ll: numpy array (1D), the initial guess wavelengths for each line
:param iteration: int, the iteration number (used so we can store multiple
                  calculations in loc, defines "i" in input and outputs
                  from p and loc
:param log: bool, if True will print a final log message on completion with
            some stats

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            X_CUT_POINTS_i: numpy array (1D), the x pixel cut points
            LITTROW_MEAN_i: list, the mean position of each cut point
            LITTROW_SIG_i: list, the mean FWHM of each cut point
            LITTROW_MINDEV_i: list, the minimum deviation of each cut point
            LITTROW_MAXDEV_i: list, the maximum deviation of each cut point
            LITTROW_PARAM_i: list of numpy arrays, the gaussian fit
                             coefficients of each cut point
            LITTROW_XX_i: list, the order positions of each cut point
            LITTROW_YY_i: list, the residual fit of each cut point
```

13.13.3 DetectBadLines

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.detect_bad_lines

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.DetectBadLines(p, loc, key=None, iteration=0)
spirouTHORCA.spirouTHORCA.detect_bad_lines(p, loc, key=None, iteration=0)
```

```
Detect and filter out the bad lines for this iteration

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_MAX_ERRW_ONFIT: float, the maximum error on the first guess
                           lines
        IC_MAX_SIGLL_CAL_LINES: float, the maximum sigma fit of the guessed
                                lines
        IC_MAX_AMPL_LINE: float, the maximum amplitude the guessed lines
                          can have
        IC_HC_T_ORDER_START: int, the echelle number of the first
                             extracted order

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        ECHELLE_ORDERS: numpy array (1D), the echelle order numbers
        ALL_LINES_i or "key": if key is None use ALL_LINES_i
                              where i = iteration
                       list of numpy arrays, length = number of orders
                       each numpy array contains gaussian parameters
                       for each found line in that order

:param key: string or None, if string should exist in "loc" and replaces
            "ALL_LINES_i" where i = iteration, and should point to a list
            of numpy arrays, length = number of orders, each numpy array
            contains gaussian parameters for each found line in that order
:param iteration: int, the iteration number (used so we can store multiple
                  calculations in loc, defines "i" in input and outputs
                  from p and loc

:return loc: parameter dictionary, the updated parameter dictionary
    Adds/updates the following:
        ALL_LINES_i or "key": list of numpy arrays, ALL_LINES_i or key input
                              with the bad lines filtered out

ALL_LINES_i definition:
    ALL_LINES_i[row] = [gparams1, gparams2, ..., gparamsN]

                where:
                    gparams[0] = output wavelengths
                    gparams[1] = output sigma (gauss fit width)
                    gparams[2] = output amplitude (gauss fit)
                    gparams[3] = difference in input/output wavelength
                    gparams[4] = input amplitudes
                    gparams[5] = output pixel positions
                    gparams[6] = output pixel sigma width
                                        (gauss fit width in pixels)
                    gparams[7] = output weights for the pixel position
```

13.13.4 ExtrapolateLittrowSolution

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.extrapolate_littrow_sol

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.ExtrapolateLittrowSolution(p, loc, ll, iteration=0)
spirouTHORCA.spirouTHORCA.extrapolate_littrow_sol(p, loc, ll, iteration=0)
```

```
Extrapolate and fit the Littrow solution at defined points and return
the wavelengths, solutions, and cofficients of the littorw fits

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_LITTROW_ORDER_FIT_DEG: int, defines the polynomial fit degree
        IC_HC_T_ORDER_START
        IC_LITTROW_ORDER_INIT int, defines the first order to for the fit
                               solution
:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        HCDATA: numpy array (2D), the image data (used for shape)
        LITTROW_PARAM_i: list of numpy arrays, the gaussian fit
                         coefficients of each cut point
        X_CUT_POINTS_i: numpy array (1D), the x pixel cut points

        where i = iteration

:param ll: numpy array (1D), the initial guess wavelengths for each line
:param iteration: int, the iteration number (used so we can store multiple
                  calculations in loc, defines "i" in input and outputs
                  from p and loc

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            LITTROW_EXTRAP_i: numpy array (2D),
                              size=([no. orders] by [no. cut points])
                              the wavelength values at each cut point for
                              each order
            LITTROW_EXTRAP_SOL_i: numpy array (2D),
                              size=([no. orders] by [no. cut points])
                              the wavelength solution at each cut point for
                              each order
            LITTROW_EXTRAP_PARAM_i: numy array (2D),
                              size=([no. orders] by [the fit degree +1])
                              the coefficients of the fits for each cut
                              point for each order

            where i = iteration
```

13.13.5 FirstGuessSolution

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.first_guess_at_wave_solution

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.FirstGuessSolution(p, loc, mode=0)
spirouTHORCA.spirouTHORCA.first_guess_at_wave_solution(p, loc, mode=0)
```

```
First guess at wave solution, consistency check, using the wavelength
solutions line list

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_HC_N_ORD_START: int, defines first order solution is calculated
        IC_HC_N_ORD_FINAL: int, defines last order solution is calculated
                           from
        IC_HC_T_ORDER_START: int, defines the echelle order of
                           the first e2ds order
        log_opt: string, log option, normally the program name
        fiber: string, the fiber number

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        ECHELLE_ORDERS: numpy array (1D), the echelle order numbers

:param mode: string, if mode="new" uses python to work out gaussian fit
                     if mode="old" uses FORTRAN (testing only) requires
                     compiling of FORTRAN fitgaus into spirouTHORCA dir

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            ECHELLE_ORDERS: numpy array, the echelle orders to fit
            LL_INIT: numpy array, the initial guess at the line list
            LL_LINE: numpy array, the line list wavelengths from file
            AMPL_LINE: numpy array, the line list amplitudes from file
            ALL_LINES: list of numpy arrays, length = number of orders
                        each numpy array contains gaussian parameters
                        for each found line in that order

ALL_LINES_i definition:
    ALL_LINES_i[row] = [gparams1, gparams2, ..., gparamsN]

                where:
                    gparams[0] = output wavelengths
                    gparams[1] = output sigma (gauss fit width)
                    gparams[2] = output amplitude (gauss fit)
                    gparams[3] = difference in input/output wavelength
                    gparams[4] = input amplitudes
                    gparams[5] = output pixel positions
                    gparams[6] = output pixel sigma width
                                 (gauss fit width in pixels)
                    gparams[7] = output weights for the pixel position
```

13.13.6 Fit1DSolution

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.fit_1d_solution

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.Fit1DSolution(p, loc, ll, iteration=0)
spirouTHORCA.spirouTHORCA.fit_1d_solution(p, loc, ll, iteration=0)
```

```
Fits the 1D solution between wavelength and pixel postion and inverts it
into a fit between pixel position and wavelength

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        LOG_OPT: string, log option, normally the program name
        FIBER: string, the fiber type (i.e. AB or A or B or C)

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        ECHELLE_ORDERS: numpy array (1D), the echelle order numbers
        HCDATA: numpy array (2D), the image data (used for shape)
        ALL_LINES_i: list of numpy arrays, length = number of orders
                     each numpy array contains gaussian parameters
                     for each found line in that order
        IC_ERRX_MIN: float, the minimum instrumental error
        IC_LL_DEGR_FIT: int, the wavelength fit polynomial order
        IC_MAX_LLFIT_RMS: float, the max rms for the wavelength
                          sigma-clip fit
        IC_HC_T_ORDER_START: int, defines the echelle order of
                             the first e2ds order

:param ll: numpy array (1D), the initial guess wavelengths for each line
:param iteration: int, the iteration number (used so we can store multiple
                  calculations in loc, defines "i" in input and outputs
                  from p and loc

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            X_MEAN_i: float, the mean residual from the fit [km/s]
            X_VAR_i: float, the rms residual from the fit [km/s]
            X_ITER_i: numpy array(1D), the last line central position, FWHM
            and the number of lines in each order
            X_PARAM_i: numpy array (1D), the coefficients of the fit to x
                       pixel position as a function of wavelength
            X_DETAILS_i: list, [lines, xfit, cfit, weight] where
                         lines= original wavelength-centers used for the fit
                         xfit= original pixel-centers used for the fit
                         cfit= fitted pixel-centers using fit coefficients
                         weight=the line weights used
            SCALE_i: list, the convertion for each line into wavelength
            LL_MEAN_i: float, the mean residual after inversion [km/s]
            LL_VAR_i: float, the rms residual after inversion [km/s]
            LL_PARAM_i: numpy array (1D), the cofficients of the inverted
                        fit (i.e. wavelength as a function of x pixel
                        position)
            LL_DETAILS_i: numpy array (1D), the [nres, wei] where
                          nres = normalised residuals in km/s
                          wei = the line weights
            LL_OUT_i: numpy array (2D), the output wavelengths for each
                      pixel and each order (in the shape of original image)
            DLL_OUT_i: numpy array (2D), the output delta wavelengths for
```

13.13.7 FPWaveSolution

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouWAVE.fp_wavelength_sol

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.FPWaveSolution(p, loc, mode='new')
spirouTHORCA.spirouWAVE.fp_wavelength_sol(p, loc, mode='new')
```

```
Derives the FP line wavelengths from the first solution
Follows the Bauer et al 2015 procedure

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_FP_N_ORD_START: int, defines first order FP solution is
                           calculated from

        IC_FP_N_ORD_FINAL: int, defines last order FP solution is
                           calculated to

        IC_HC_N_ORD_START_2: int, defines first order HC solution was
                             calculated from

        IC_HC_N_ORD_FINAL_2: int, defines last order HC solution was
                             calculated to

        IC_FP_THRESHOLD: float, defines threshold for detecting FP lines

        IC_FP_SIZE: int, defines size of region where each line is fitted

        IC_FP_DOPD0: float, initial value of FP effective cavity width

        IC_FP_FIT_DEGREE: int, degree of polynomial fit

        log_opt: string, log option, normally the program name

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        FP_DATA: the FP e2ds data
        LITTROW_EXTRAP_SOL_1: the wavelength solution derived from the HC
                              and Littrow-constrained
        ALL_LINES: list of numpy arrays, length = number of orders
                   each numpy array contains gaussian parameters
                   for each found line in that order
        BLAZE: numpy array (2D), the blaze data

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            FP_LL_POS: numpy array, the initial wavelengths of the FP lines
            FP_XX_POS: numpy array, the pixel positions of the FP lines
            FP_M: numpy array, the FP line numbers
            FP_DOPD_T: numpy array, the measured cavity width for each line
            FP_AMPL: numpy array, the FP line amplitudes
            FP_LL_POS_NEW: numpy array, the corrected wavelengths of the
                           FP lines
            ALL_LINES_2: list of numpy arrays, length = number of orders
                         each numpy array contains gaussian parameters
                         for each found line in that order
```

13.13.8 GetE2DSll

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.get_e2ds_ll

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.GetE2DSll(p, hdr=None, filename=None, key=None)
spirouTHORCA.spirouTHORCA.get_e2ds_ll(p, hdr=None, filename=None, key=None)
```

```
Get the line list for the e2ds file from "filename" or from calibration
database using hdr (aqctime) and key. Line list is constructed from
fit coefficients stored in keywords:
    'kw_TH_ORD_N', 'kw_TH_LL_D', 'kw_TH_NAXIS1'

:param pp: parameter dictionary, ParamDict containing constants
    Must contain at least:
            log_opt: string, log option, normally the program name
            kw_TH_COEFF_PREFIX: list, the keyword store for the prefix to
                                use to get the TH line list fit coefficients

:param hdr: dictionary or None, the HEADER dictionary with the acquisition
            time in to use in the calibration database to get the filename
            with key=key (or if None key='WAVE_AB')
:param filename: string or None, the file to get the line list from
                 (overrides getting the filename from calibration database)
:param key: string or None, if defined the key in the calibration database
            to get the file from (using the HEADER dictionary to deal with
            calibration database time constraints for duplicated keys.

:return ll: numpy array (1D), the line list values
:return param_ll: numpy array (1d), the line list fit coefficients (used to
                  generate line list - read from file defined)
```

13.13.9 GetLampParams

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.get_lamp_parameters

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.GetLampParams(p, filename=None, kind=None)
spirouTHORCA.spirouTHORCA.get_lamp_parameters(p, filename=None, kind=None)
```

```
Get lamp parameters from either a specified lamp type="kind" or a filename
or from p['ARG_FILE_NAMES'][0] (if no filename or kind defined)

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_LAMPS: list of strings, the different allowed lamp types
        log_opt: string, log option, normally the program name
:param filename: string or None, the filename to check for the lamp
                 substring in
:param kind: string or None, the lamp type

:return p: parameter dictionary, the updated parameter dictionary
        Adds the following:
            LAMP_TYPE: string, the type of lamp (e.g. UNe or TH)
            IC_LL_LINE_FILE: string, the file name of the line list to use
            IC_CAT_TYPE: string, the line list catalogue type
```

13.13.10 JoinOrders

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.join_orders

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.JoinOrders
spirouTHORCA.spirouTHORCA.join_orders
```

```
Merge the littrow extrapolated solutions with the fitted line solutions

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_HC_N_ORD_START: int, defines first order solution is calculated
        IC_HC_N_ORD_FINAL: int, defines last order solution is calculated
                           from

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        LL_OUT_2: numpy array (2D), the output wavelengths for each
                  pixel and each order (in the shape of original image)
        DLL_OUT_2: numpy array (2D), the output delta wavelengths for
                   each pixel and each order (in the shape of original
                   image)
        LITTROW_EXTRAP_SOL_2: numpy array (2D),
                          size=([no. orders] by [no. cut points])
                          the wavelength solution at each cut point for
                          each order
        LITTROW_EXTRAP_PARAM_2: numy array (2D),
                          size=([no. orders] by [the fit degree +1])
                          the coefficients of the fits for each cut
                          point for each order

:return loc: parameter dictionary, the updated parameter dictionary
    Adds/updates the following:
        LL_FINAL: numpy array, the joined littrow extrapolated and fitted
                  solution wavelengths
        LL_PARAM_FINAL: numpy array, the joined littrow extrapolated and
                        fitted fit coefficients
```

13.13.11 SecondGuessSolution

Defined in SpirouDRS.spirouTHORCA.spirouTHORCA .spirouTHORCA.second_guess_at_wave_solution

Python/Ipython

```
from SpirouDRS import spirouTHORCA
spirouTHORCA.SecondGuessSolution(p, loc, mode=0)
spirouTHORCA.spirouTHORCA.second_guess_at_wave_solution(p, loc, mode=0)
```

```
Second guess at wave solution, consistency check, using the wavelength
solutions line list

:param p: parameter dictionary, ParamDict containing constants
    Must contain at least:
        IC_LL_SP_MIN: int, minimum wavelength of the catalog
        IC_LL_SP_MAX: int, maximum wavelength of the catalog
        IC_RESOL: int, Resolution of spectrograph
        IC_LL_FREE_SPAN_2: int, window size in sigma unit
        IC_HC_N_ORD_START_2: int, defines first order solution is
                             calculated from
        IC_HC_N_ORD_FINAL_2: int, defines last order solution is
                             calculated from
        IC_HC_T_ORDER_START: int, defines the echelle order of
                            the first e2ds order

:param loc: parameter dictionary, ParamDict containing data
    Must contain at least:
        ECHELLE_ORDERS: numpy array (1D), the echelle order numbers
        HCDATA: numpy array (2D), the image data
        LL_LINE: numpy array, the line list wavelengths from file
        AMPL_LINE: numpy array, the line list amplitudes from file
        LITTROW_EXTRAP_SOL_1: numpy array (2D),
                          size=([no. orders] by [no. cut points])
                          the wavelength solution at each cut point for
                          each order

:param mode: string, if mode="new" uses python to work out gaussian fit
                     if mode="old" uses FORTRAN (testing only) requires
                     compiling of FORTRAN fitgaus into spirouTHORCA dir

:return loc: parameter dictionary, the updated parameter dictionary
        Adds/updates the following:
            ALL_LINES_2: list of numpy arrays, length = number of orders
                         each numpy array contains gaussian parameters
                         for each found line in that order

ALL_LINES_2 definition:
    ALL_LINES_2[row] = [gparams1, gparams2, ..., gparamsN]
                where:
                    gparams[0] = output wavelengths
                    gparams[1] = output sigma (gauss fit width)
                    gparams[2] = output amplitude (gauss fit)
                    gparams[3] = difference in input/output wavelength
                    gparams[4] = input amplitudes
                    gparams[5] = output pixel positions
                    gparams[6] = output pixel sigma width
                                        (gauss fit width in pixels)
                    gparams[7] = output weights for the pixel position
```