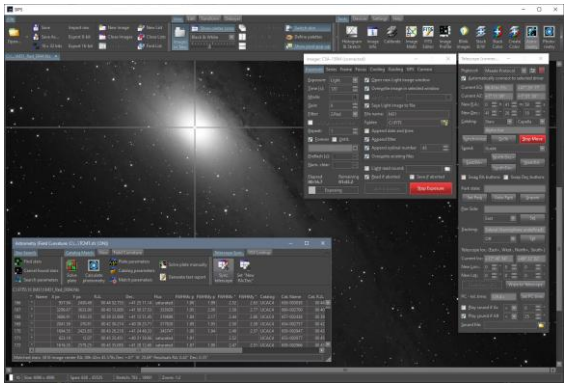
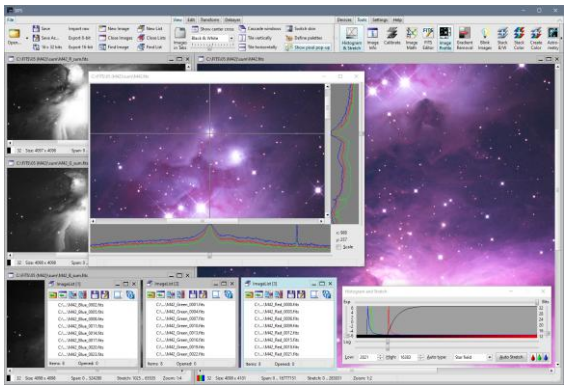
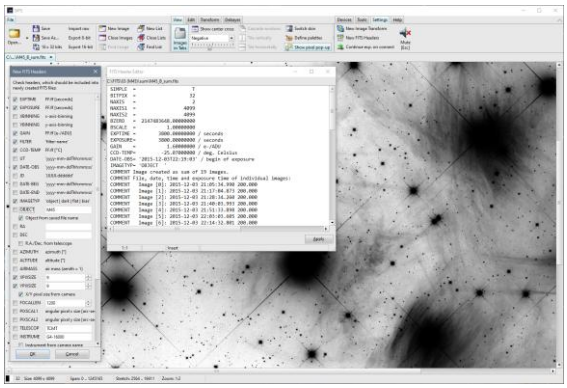
Scientific **I**mage **P**rocessing **S**ystem

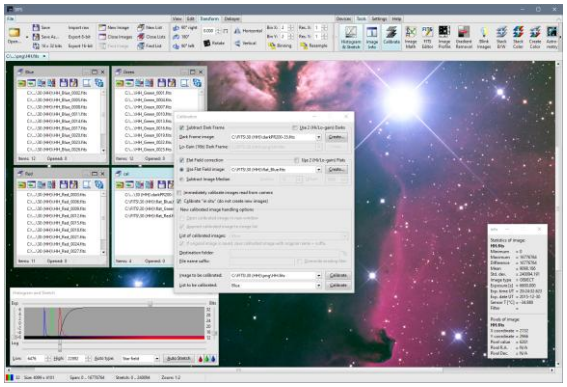
version **4**

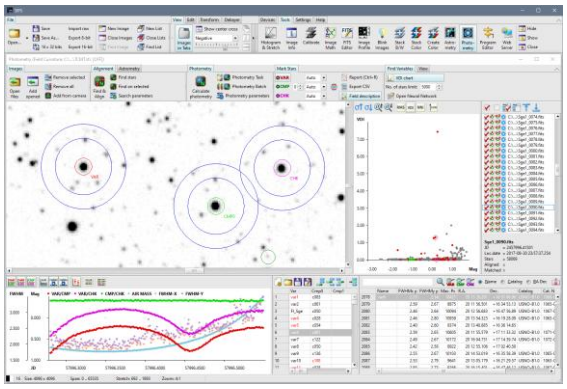
User's Guide











SIPS version 4 User's Guide

Document version 4.4.2

Modified on June 19th, 2026

Moravian Instruments
Masarykova 1148
763 02 Zlín
Czechia

phone: +420 577 107 171
web: https://www.gxccd.com/
e-mail: info@gxccd.com

Table of Contents

SIPS basics

What does SIPS mean?

SIPS stands for **S**cientific **I**mage **P**rocessing **S**ystem. SIPS does not contain many common image editing tools, like brushes, effect filters, etc. Instead, it implements operations like "dark frame subtraction" and "flat fielding", often performed by astronomers to calibrate their images (and astronomers consider themselves scientists :-). It also implements powerful tools for astrometry reduction and photometry processing—operations necessary to extract scientifically valuable information from astronomical images. The native image format used by SIPS is FITS, used among the science community to exchange data, not JPEG, TIFF, or PNG (although it is of course possible to export images to these commonly used image formats).

What is SIPS

SIPS is a software package, focused on two areas:

- Astronomical observation control, which includes **camera** and **filter wheel control** and image acquisition, automatic **guiding** as well as additional device and **observing equipment control**, like focusers, telescope mounts, observatory domes, weather stations, GPS receivers etc.

  SIPS allows controlling of any observing equipment (camera, telescope, ...) only with an appropriate driver. The native drivers for the entire range of Moravian Instruments Cx and Gx cameras are included in the installation package of SIPS. SIPS also contains interface to ASCOM standard[1] drivers, so any device (camera, focuser, telescope mount, etc.) with ASCOM-compatible driver can be used from within SIPS.

- Astronomical image scientific processing, like raw image **calibration** (dark frame subtraction, flat fielding) and basic manipulation (median combine, alignment and stacking, blinking, …). Two key image reduction workflows, used in research to analyze astronomical images, are also included—**astrometry** reduction and time-based **photometry** processing. Astrometry and photometry processing is quite advanced and allows for efficient processing of multiple targets within a field of view very quickly.

SIPS is designed to work under any 32-bit or 64-bit Windows operating system, beginning with Windows 7 and currently up to Windows 11. It should also run on subsequent Windows versions, providing these versions maintain compatibility with applications designed for Win32 API.

Basic SIPS principles

SIPS is designed to be intuitive and easy to use, but some actions require understanding of certain principles. For instance, calibration of multiple images at once is performed using image lists, therefore, the concept of image lists and some basic image list handlings (e.g., adding of images to list) should be understood.

There are four basic types of GUI elements in SIPS:

- **Command ribbons**[2], either snapped to containers, arranged around the SIPS main window borders, or floating in the independent pop-up windows.
- **Images**, which are matrices of pixels representing brightness of individual image points. Images can be displayed in overlapping windows inside the SIPS main window or in tabbed container occupying entire SIPS main window client area.
- **Image lists**, holding one or more images (number of operations can be performed on single image as well as on all images in the particular list). Lists are always displayed as overlapping windows inside the SIPS main window.

[1] See https://www.ascom-standards.org/

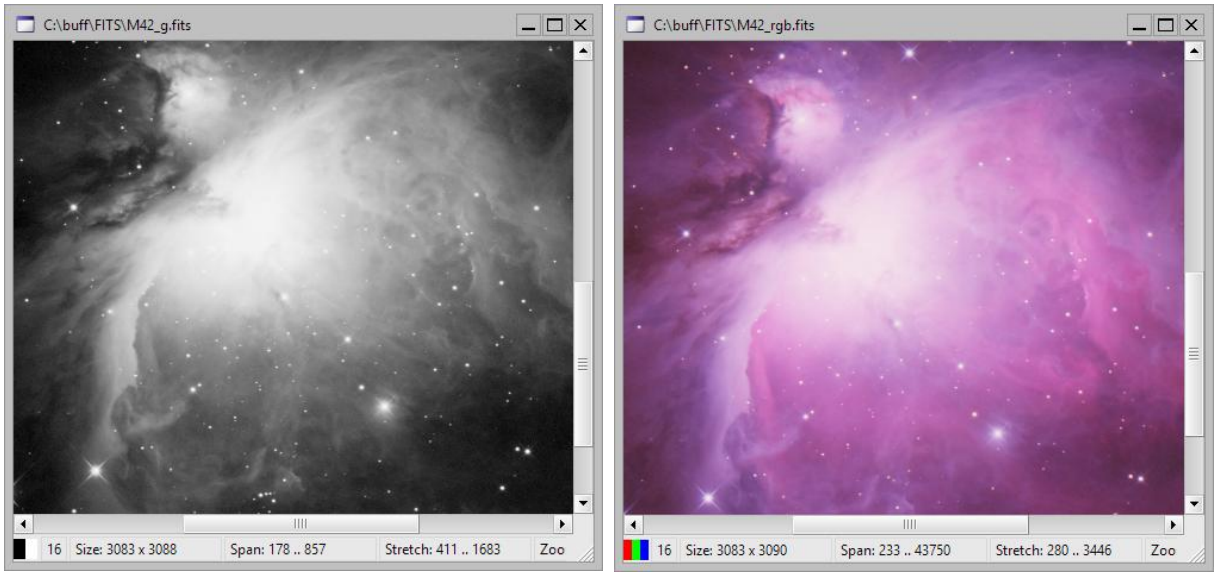
[2] SIPS v4 GUI departs from traditional menu/toolbar arrangement and introduces new and highly configurable command ribbons. How SIPS ribbons work, how they can be arranged, showed, or hidden are described in the **Appendix A: SIPS user interface**.

- **SIPS tools**, represented by tool windows. Every tool offers GUI for some operation—e.g., camera control, guiding, stacking multiple images, astrometry reduction or photometry processing etc.

While it is possible to open/create many image and image list windows, every tool window can be opened only once.

Images in SIPS

Monochrome image is simply two-dimensional array (matrix) of pixels equipped with some additional information to enable image manipulation—for example various FITS header field values, sensor temperature during image recording, exposure time etc. Image can be displayed in window, but it can exist within SIPS image list even without being visible (we will describe image lists later).



While each pixel of monochrome image is represented by single value, there are three values representing red, green, and blue components of each pixel in color image. SIPS stores color images as three matrices, one for each color. Because some operations (e.g., searching of stars within image) are independent of the particular color, SIPS stores also color-neutral luminance matrix beside RGB matrices. So, a colored image—even one using just a single color—occupies four times the memory space of a monochrome one.

Images can be not only viewed; it is possible to perform various operations on them.

- Simple operations are performed directly from the command ribbons, like for instance transformations (mirroring or rotation, …).
- More complex operations are performed through SIPS tools (tool windows), each focused to some kind of operation (note that some tools manipulate not only single images, but also image lists). Individual tools will be thoroughly described later.

Images are thoroughly described in the **SIPS Images** chapter.

Image Lists

Astronomers typically capture many images during an observing session. Images taken by a camera always require at least some degree of processing—at last they must be calibrated. Image lists allow processing of multiple images at once, but they can be efficiently used also for other purposes. SIPS supports mathematical operations defined on number of images, like computation of average or median of images. These operations are performed using image lists.

Image lists in the SIPS workspace are represented visually as a window. Image list windows display the list of images represented by the window and several tool buttons, which enable adding and removing images to/from the list.

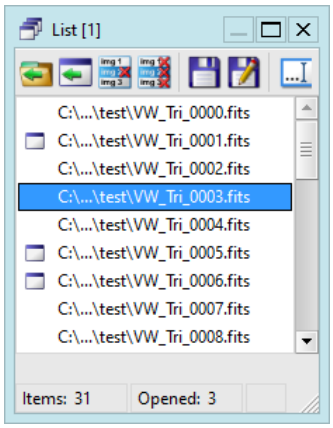
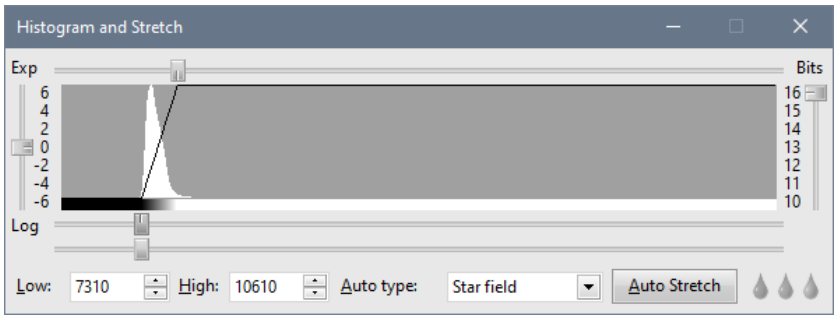


Image lists are thoroughly described in SIPS Image Lists chapter.

SIPS tools

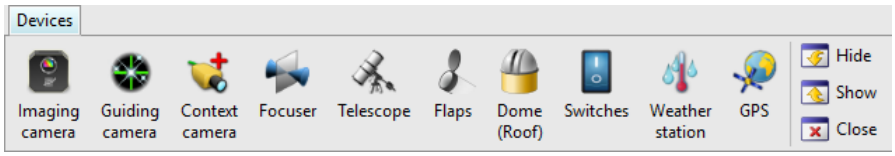
SIPS offers number of tools to control various devices (cameras, mounts, …) as well as to manipulate and process images and image lists. Individual tools will be described in detail in the next chapters of this documentation.



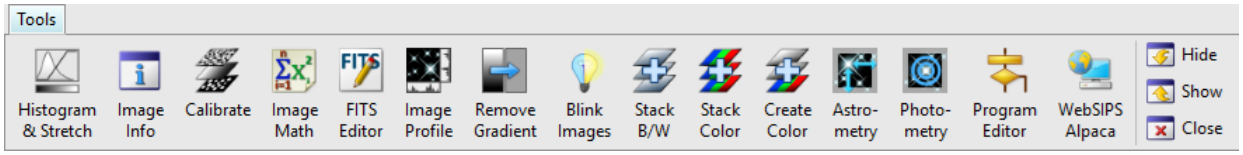
Remark:

Tool windows behave somewhat differently compared to image and image list windows. They do not exist within the SIPS workspace, but always stays above SIPS main window.

Tools can be opened from two command ribbons—the **Devices** ribbon allows opening of tools intended to control hardware devices, such as cameras, telescope mounts, etc.,



while the **Tools** ribbon contains tools designed to image manipulation and processing.



Hint:

The **Devices** and **Tools** ribbons occupy the same group in the default ribbon configuration. However, as described in the Appendix A: SIPS user interface, each ribbon can be dragged to another group or even to another ribbon container. For instance, it is possible to place the Tools ribbon vertically on the left or right side of the SIPS main windows etc.

Some tool windows are independent on other image or image list windows, e.g., the **Imaging camera** tool controlling the main imaging camera.

Other tools are related to currently active (selected) image window, e.g., the **Histogram and Stretch** or **Profile** tools. If you select another image window (e.g., by clicking it), Histogram and Stretch tool displays the histogram of newly selected image and manipulating of the stretch controls affects the selected image only.

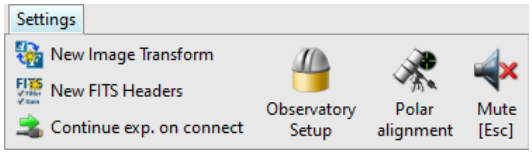
Other tools require explicit definition of which **images** or **image lists** should be manipulated, e.g., **Image Math and Filters** or **Calibration** tools.

There are also tools, which contain implicit image list. To operate such tools, it is first necessary to fill this list with images, be it from files or from another list or lists. Only then the tool can perform its tasks on the images in own list. Examples of such tools are **Blink Images**, **Combine Images**, or **Photometry** processing.

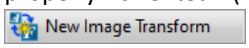
SIPS remembers the open state and position of all tool windows. When you start SIPS again, it opens tools open in previous session and places them into the same position.

Settings ribbon pane

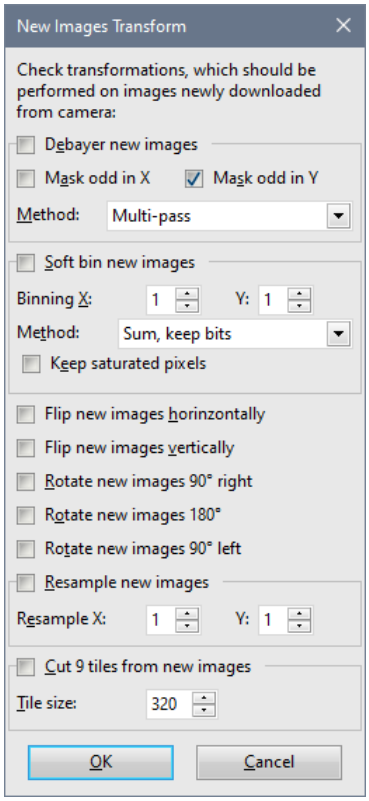
The **Settings** ribbon pane contains controls affecting global SIPS behavior and features spanning multiple tools.



New image transformation

In some cases, it is desirable to perform some image transformation immediately upon image read from a camera. For instance, when the camera is rotated 90 degrees relative to celestial north and the user wants to work with images properly oriented (north up) etc. Immediate transformation upon image read can be defined in the 

dialog box, opened from the **Settings** ribbon pane.



Available transformations are the same like the "online" transformation contained in the **Transform** ribbon. Also included is the possibility to immediately perform Debayer processing when the single-shot-color camera is used.

One exception from the above-mentioned duplicity is a transformation named **Cut 9 tiles from new images**, which is available only for newly read images. The purpose of this transformation is to allow inspection of nine parts of the image (center, all four corners and centers of the border edges) without the necessity to zoom-out the image, which hides

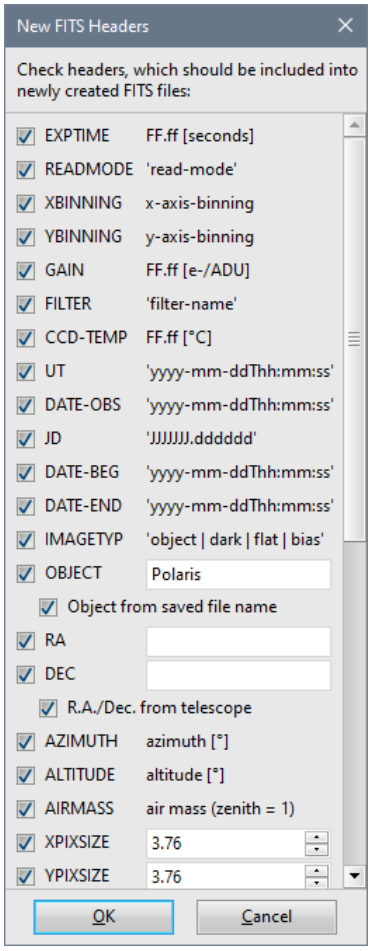
important details. This transformation is especially useful with large-resolution cameras, which image must be significantly zoomed out to be entirely visible on the computer screen. The selected nine portions of the image allow to justify focus and/or shapes of stars in all extreme image locations, e.g. when the user needs to adjust camera tilt relative to optical axis etc.

If multiple transformations are checked, then they are performed in the order defined by the top-down order of the respective checkbox in the dialog box.

New FITS Headers

FITS image format flexibility allows storing of virtually any metadata in its headers, but at the same time it leads to many variants of the same information in its headers. SIPS allows users to decide which FITS headers are to be included into every image read from the camera through the 

dialog box, opened from the **Settings** ribbon pane.



Some values, like exposure time EXPTIME or binning X/YBINNING, are always set according to the acquired image.

Other values (for instance OBSERVER) may be defined statically, and some of them may be also dynamically modified according to the connected hardware, image handling options or values defined in the global Observatory Setup:

- Image center equatorial coordinates (RA and DEC) may be set to the connected **Telescope** mount position.
- Pixel size (X/YPIXSIZE) may be set to the connected **Imaging camera** pixel size.
- OBJECT name may be set to the saved FITS file name.
- Focal length (FOCALLEN) may be set to value defined in the **Observatory Setup** dialog box.
- Telescope (TELESCOP) name may be set to value defined in the **Observatory Setup** dialog box.
- Instrument (INSTRUME) may be set to the connected **Imaging camera** name.
- Observatory (OBSERVAT) name may be set to value defined in the **Observatory Setup** dialog box.
- ORIGIN may be set to the current SIPS version.
- Geographic location (LAT--OBS, LONG-OBS, and ELEV-OBS) name may be set to values defined in the **Observatory Setup** dialog box.

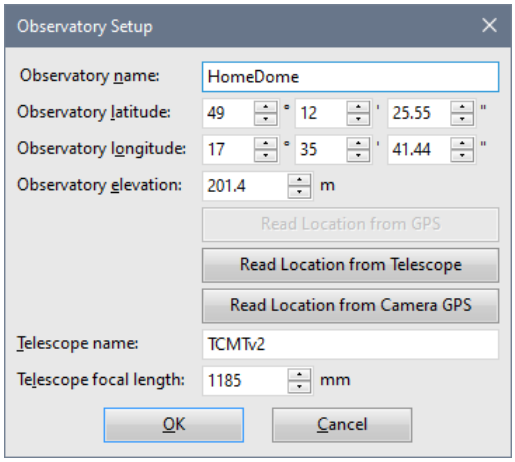
Continue exposures on camera connect

Checking of the option button 

ensures continuing of repeated exposures of the Imaging camera in the case the camera was unplugged while the exposures were running and then plugged again.

Observatory Setup

The **Observatory Setup** dialog box is a place to define some global values, used within multiple SIPS tools. For instance, observatory geographic locations may be used by the **Telescope** tool convert equatorial coordinates (R.A. and Dec.) to Altitude and Azimuth, by the **Imaging camera** tool to writhe the location to the FITS headers, by the **Photometry** tool to calculate air mass etc.



The geographic location may be defined statically, or location coordinates may be read from the connected GPS receiver, either standalone device of a receiver attached to the imaging camera. Also, location may be read from the connected telescope driver (telescope typically “knows” where it stands as it needs location information for various functions).

- **Read Location from GPS** updates the values from the currently attached standalone **GPS receiver**.
- **Read Location from Telescope** reads the values from the active **Telescope** driver (telescope mount typically contain location information e.g. to be able to transform equatorial coordinates to azimuth and altitude).
- **Read Location from Camera GPS** can be used if the used camera is equipped with **GPS receiver**. The GPS receiver integrated with camera is primarily intended to provide precise exposure timing information, but it of course also provides geographic location.

Individual buttons are enabled or disabled according to the availability (connection state) of individual device in the time of opening of the **Observatory Setup** dialog box.

Hint:

Sometimes coordinates are not provided in hours or degrees, minutes, and seconds, but in degrees in the form of decimal number (fraction of degrees). While conversion from decimal degrees to minutes and seconds is simple, it involves a few divisions and subtractions and majority of people cannot do such math without calculator or at last pen and paper. This is why the location controls are capable to perform “smart paste” operation. If a number with a fraction part (containing decimal point) is pasted to the **Observatory latitude** or **Observatory longitude** first count-box (degrees), SIPS transforms the value into whole degrees, minutes, and second and fills all three count-boxes accordingly.

Mute button

The 

button stops any alarm sound currently played by any device capable of audible alarms (for instance, the **Weather station** tool, **Telescope** tool etc.).

32-bit and 64-bit SIPS

SIPS is available in both 32-bit and 64-bit versions, the latter one naturally requires a 64-bit PC and a 64-bit operating system to run.

> Remark:
>
> Even if the 64-bit version of Windows runs on a PC with 64-bit processor, this does not mean all applications are also 64-bit. Both the 64-bit PC hardware (CPU) and 64-bit operating system (Windows) are designed to seamlessly run older 32-bit applications. So, many applications can still be 32-bit only and users may not even notice this. This is also valid for drivers intended to control cameras and other observatory equipment.

The 64-bit SIPS is not limited to 2 GB memory limit, imposed by 32-bit addressing space, and thus can be very useful when manipulating large lists of images, e.g., during photometry series processing. Also, the 64-bit code runs noticeably faster compared to the same code compiled to 32-bit application thanks to the enhancements in the instruction set, introduced in 64-bit x64 processors (also marked AMD64). On the other side, because it is not possible to combine 32-bit and 64-bit code in one process (one running application), the 64-bit SIPS version requires all used drivers to be also 64-bit, including ASCOM drivers used to control various devices, etc.

So, the 32-bit SIPS offers widest compatibility with various device drivers, which are often 32-bit only. Ability to address memory beyond 2 GB and added processing speed is not important when controlling the observing run.

As both 32-bit and 64-bit versions of SIPS can be installed at once, the 32-bit one is often used to control observation and acquire images and the 64-bit one is used to later process them.

SIPS installation

SIPS can be installed by running of its installation package on the particular computer. Installation packages are available in different version for 32-bit and 64-bit version as well as for individual language versions.

The installation packages can be either downloaded from the **Download** section of the Moravian Instruments web site `https://www.gxccd.com/` or can be found on the USB Flash Drive, shipped with every Moravian Instruments camera.

Available packages are:

- `SIPS4_EN_32-bit.exe` for 32-bit English version
- `SIPS4_EN_64-bit.exe` for 64-bit English version

Packages for other language versions differ in language identifier, e.g. the 64-bit Czech version installation package is named `SIPS4_CZ_64-bit.exe` etc.

SIPS configuration files

SIPS distinguishes two types of configurations:

- Global configuration, common for all users.
- User-specific configuration.

Driver configuration file

Driver configuration defines which hardware could be used with SIPS and which drivers control it. The configuration is stored in the simple text file `sips.ini`, which is placed in the same folder as the `sips.exe` main executable (typically in the `C:\Program Files\Moravian Instruments\SIPS4 EN 64-bit` or `C:\Program Files (x86)\ Moravian Instruments\SIPS4 EN 32-bit` folder). The file may look for example like this:

```
[Camera]
Moravian Camera (gXusb) = gxusb.dll
Moravian Camera (cXusb) = cxusb.dll
Moravian Camera on Ethernet = gxeth.dll
ASCOM Camera = ascom_camera.dll

[Filter Wheel]
Moravian filter wheel = sfw.dll
Moravian filter wheel on Ethernet = sfweth.dll
```

```
ASCOM = ascom_filters.dll

[GPS]
GarminUSB = gps18.dll
NMEA = nmea.dll

[Telescope]
NexStar = nexstar.dll
Meade = meade.dll
ASCOM = ascom_tele.dll

[Focuser]
MM4 focuser = mmfocuser.dll
ASCOM = ascom_focuser.dll

[Flaps]
ASCOM = ascom_flaps.dll

[Dome]
ASCOM = ascom_dome.dll

[Switches]
TCS2 switches = mmswitches.dll
ASCOM = ascom_switches.dll

[Weather Station]
RDH11 = fiedlermeteo.dll
ASCOM = ascom_meteo.dll
```

Individual sections define which driver would be loaded and asked to enumerate all connected devices of particular type (cameras, filter wheels, telescope mounts, ...).

SIPS is installed with `sips.ini` containing all included drivers.

Driver configuration file for 3rd party drivers

SIPS is not limited to handling of devices (cameras, focusers, …) included in its installation. The driver API is publicly available and anybody is free to implement own drivers.

However, to avoid a necessity to modify the `sips.ini` in the Program Files folder, which is overwritten with every new SIPS installation, a second driver configuration file, also named `sips.ini`, may be created in the `\Users\Public\Documents\SIPS\ini` folder. If this file exists, SIPS merges drivers listed in respective ini file sections from both Program Files and Public Documents.

> Remark:
>
> The ability to include drivers defined in `sips.ini` located in Public Documents was introduced in v4.1.

User-specific configuration file

User-specific configuration is stored in the file named `sips_v4.ini`, placed in the user-specific folder, assigned by the operating system (typically `C:\Users\%user_name%\AppData\Roaming\SIPS`, where `%user_name%` is the user's login name).

> Remark:
>
> Earlier SIPS versions used the `sips.ini` file to save user-specific configuration, without the version suffix in its name. So, if the user runs both v3 and v4 versions of SIPS, configuration is stored separately, and any parameter changes are not propagated between versions.
>
> When the SIPS v4 is run for the first time under the particular user's profile and the `sips_v4.ini` file does not exist yet, SIPS tries to open the original v3 file to use its configuration setting. However, the configuration will be saved into the v4 name upon exit for future use.

This configuration file stores hundreds of parameters, beginning from the position and open state of individual tool windows, to the preferred astrometry catalog and parameters for searching stars in images, etc.

Configuration handling command line parameters

Sometimes even single user may need different configurations, for instance if multiple SIPS instances are launched to control multiple observing setups (telescope mount, imaging and guiding cameras, domes, ...). In such cases the SIPS reads configuration of the last recently closed SIPS instance, as the last closed SIPS overwrites the `sips.ini` file, previously saved by previous instances. If another SIPS instance had to use different camera, the only choice was to manually define camera and mount driver are to be used upon each SIPS start.

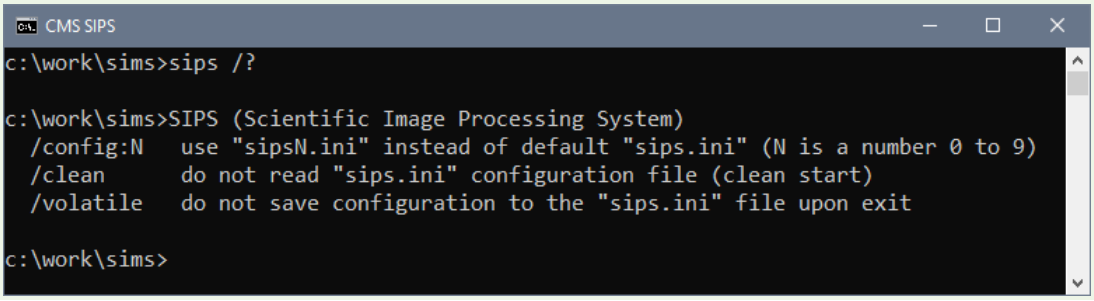
To solve this issue, SIPS accepts command-line parameters, which allow each program instance to use up to ten different configuration files. Also, other command line switches allow SIPS not to use configuration upon launch (run with all parameters set at default values), and also not to save configuration upon program exit:

- **/config:N**—use the `sips_v4_N.ini` configuration file (N is a number 0 to 9) instead of default `sips_v4.ini`
- **/clean**—do not read `sips_v4.ini` configuration file at all during start (clean start, ignores saved configuration)
- **/volatile**—do not save configuration to the `sips_v4.ini` file upon exit (configuration changes will be abandoned)

Hint:

These command line parameters can be either entered directly into command line console when launching SIPS or it is possible to create a shortcut for SIPS using specific configuration and to define parameters in the shortcut properties dialog box.

It is possible to type the command `sips /?` to display a list of available command line parameters.



SIPS Images

What is Image in SIPS?

As was already noted in the introduction to SIPS, monochrome image is a matrix (two-dimensional array) of pixels plus a bunch of additional attributes storing for instance actual image size (width and depth), image exposure date and time, sensor temperature during exposure etc.

Color images consist of three matrices of the same width and depth, one for each basic color—Red, Green and Blue[3].

Some attributes are mandatory (e.g. image dimensions), but some are optional (e.g. observer name or camera type). SIPS keeps not only its own attributes with each image, but also the information contained in the FITS image header. There can be FITS headers loaded from FITS file (and possibly written back to file when image is saved), which SIPS does not care about—the program code does not "understand" the particular headers and only stores them to ensure their persistence while image is manipulated in SIPS.

FITS Image file format

SIPS native image file format is FITS. Although FITS can support various pixel types (from 8, 16 and 32-bit integers to floating point numbers), SIPS is currently limited to 16-bit and 32-bit integer pixels, giving 65,536 or 4,294,967,296 possible values for every pixel. There are software packages, which immediately convert all FITS images into floating point pixels. This makes FITS handling easier to program (floating point numbers are less sensitive to precision lost during various mathematical operations), but such images occupy a lot of memory, and their handling is much more CPU demanding and thus slower. Although SIPS can open FITS files with floating-point pixels, it transforms them to 32-bit integer internally to handle it.

> Remark:
>
> Images converted to floating-point pixels also disallows some important operations, like distinguishing of saturated pixels or converting ADUs[4] to electrons using camera gain, especially if the floating-point numbers are normalized etc. This is why SIPS does not use floating-point pixels and keeps them in integer values.
>
> Why to bother with 32-bit pixels when many cameras provide only 12 or 14-bit resolution and even if the camera/sensor supports full 16-bit resolution, images can be rarely reliably digitized with full 16-bit precision. At last, one and more often two, three or even four least significant bits are affected by random electronics noise. This means images often contain only 14 bits (and sometimes only 10 or 11 bits) of valid information, so 16-bit precision is not limiting. But 16 bits become limiting when we stack multiple images. Just two 16-bit images with pixel values above a half of the range (greater than 32,768) would cause pixel saturation. Here 32-bit images come handy. It would be necessary to add more than 65,000 saturated 16-bit images to saturate 32-bit pixel.

FITS is very old format and some of its features are rather obsolete—it originated in the era of punched cards and the header structure reflects it. The FITS header consists of blocks of 36 lines, each 80 characters long. Forget about (CR)LF delimiters and flexible line length, every line is exactly 80 characters long and padded with spaces (to fill one punched card?). Also forget about some lexical analysis based on keywords and delimiters, particular fields on each line are defined positionally, header names are limited to 8 characters, then the equal sign '=' on column 9 must follow. Length of every string must be at last 8 characters (don't ask me why). Although SIPS extends shorter strings with spaces to fulfil this rule, it does not object if it reads string with less than 8 characters and number of other software packages ignore this rule. The header must end with line beginning with the END keyword. If the last line is not the line number 36 (or multiply of 36 if the header spans over multiple blocks), blank lines containing 80 spaces up to 36 (or multiply of 36) lines are added.

Why is such obsolete format like FITS so popular? Simply because it works. FITS handling is routinely implemented in various software packages, so why to bother with format peculiarities? If we save images to FITS files, there is a good chance that such files can be manipulated by number of other software packages (but one can never be sure :-). It is

[3] SIPS creates also luminance pixels for each color image, used for instance for astrometry etc. So, color images occupy four-times the space of monochrome one.

[4] Analog to Digital Unit—number representing each pixel brightness, usually directly the output of the sensor A/D converters.

often enough for FITS file to contain some very basic headers (keywords SIMPLE, BITPIX, NAXIS, NAXIS1, NAXIS2 and possibly BSCALE and BZERO) and image can be loaded and displayed.

On the other side, various software packages add various headers and if the header follows FITS syntax, it is valid FITS header. So, some FITS image can contain exposure time in header EXPTIME, another can use EXPOSURE header. The date of image exposure can be stored in TIME-OBS or UT header, or in the newly proposed headers DATE-BEG and DATE-END etc. This is a drawback of FITS format, but still there is no other generally accepted format other than FITS, so we have no choice.

SIPS allows users to specify which FITS header should be included into newly created images (read from camera) in the **New FITS Header** dialog. FITS headers of images existing in SIPS, either newly read from camera or loaded from disk file, can be viewed, and modified in the **FITS Header Editor** tool.

Storing Color Images in FITS

FITS format defines no standard for storing color images. However, FITS provides natural way to store multi-dimensional arrays of values (pixels), it is not limited to two-dimensional matrices only. Because color image consists of three planes, each containing red, green and blue color, it can be naturally handled as three-dimensional array with first two dimensions equal to image width and depth and the third dimension always equals to 3.

SIPS uses this way to save color images into FITS files. The FITS parameters NAXIS and NAXIS3 both equal to 3 for color images. It is not clear if such FITS files can be handled by other image processing software, but storing image into such file does not cause any information loss and can be anytime separated into three independent two dimensional matrices again.

All other FITS headers remain the same like in the case of monochrome images.

FITS Image Coordinate System

Unfortunately, the image coordinate system is not clearly defined by FITS standard. This means when the image is stored as two-dimensional matrix of pixels, it is not clear whether the first line of pixel is the top line (coordinate [0,0] in the upper left corner) or the bottom line (coordinate [0,0] in the lower left corner).

Almost all professional FITS manipulation programs suppose the coordinate [0,0] is in the lower-left corner, which corresponds to commonly accepted definition of coordinate system used e.g. in mathematics (if we draw a graph of any function y = f(x), x-axis goes right and y-axis goes up). This is also valid for SIPS.

However, coordinate system used by various graphical user interfaces (e.g. Windows GDI) places origin to the upper-left corner (x-axis goes right, y-axis goes down). This is probably the reason why most amateur software packages handle FITS like the origin is in the upper-left corner and the y-axis increases downwards.

SIPS v4 properly handles the FITS header ROWORDER, which value can be either BOTTOM-UP or TOP-DOWN. This means all images created and stored by SIPS contain this header, indicating all images are organized in the BOTTOM-UP order. Also, if a FITS file created by some other software package contains this header, SIPS loads such image with respect to originally defined row order.

If the FITS image without ROWORDER parameter is stored by a software package with [0,0] in the upper-left corner and read by software package with [0,0] in the lower-left corner (or vice versa), it appears vertically mirrored. Fortunately, this brings only minor problem for the astronomical images. Each image can be arbitrary flipped or rotated, so it is not problem to adjust each image into proper orientation.

Image Persistence

As was already noted in the introduction to SIPS, it is not necessary to save FITS images to disk file to perform any operations with them. This is maybe somewhat unusual feature—number of image processing software packages are file-only oriented. They often rely on files to perform image manipulation. Performing some operation on multiple images requires putting all image files into separate directory or at last to name particular images to comply to some wild card naming rule (SIPS solves the problem of selecting files for multiple image processing by using **Image Lists**).

On the other side it is obvious that it is necessary to save image to disk file to ensure its persistence. SIPS of course allows immediate saving of every image captured from camera etc. It also provides very flexible tools to save different image types (dark, flat, light) to different folders and/or under different names. File names can be automatically created

according to object name, date/time of exposure, filter name, ordinal number etc. (Refer to the **Imaging camera** tool description to learn more about capturing images.)

But when capturing number of images to calculate flat field or dark frame as average or median of multiple images, it is not necessary to save all files to disk. They can stay in computer memory and only the resulting dark frame or flat field can be saved. Another possibility is to capture number of images and to inspect them on screen only for e.g. tracking errors, for object placement within the chip etc. and to save only good images.

Images within SIPS Workspace

Not all images must be visible (displayed) within SIPS. For instance, images in **Image list** can exist only in the list, without being visible to the user. Image lists are described in the separate chapter **SIPS Image Lists**.

Images can be displayed in two ways in SIPS:

- Each image can have its own window. Image windows are placed inside the main SIPS window.

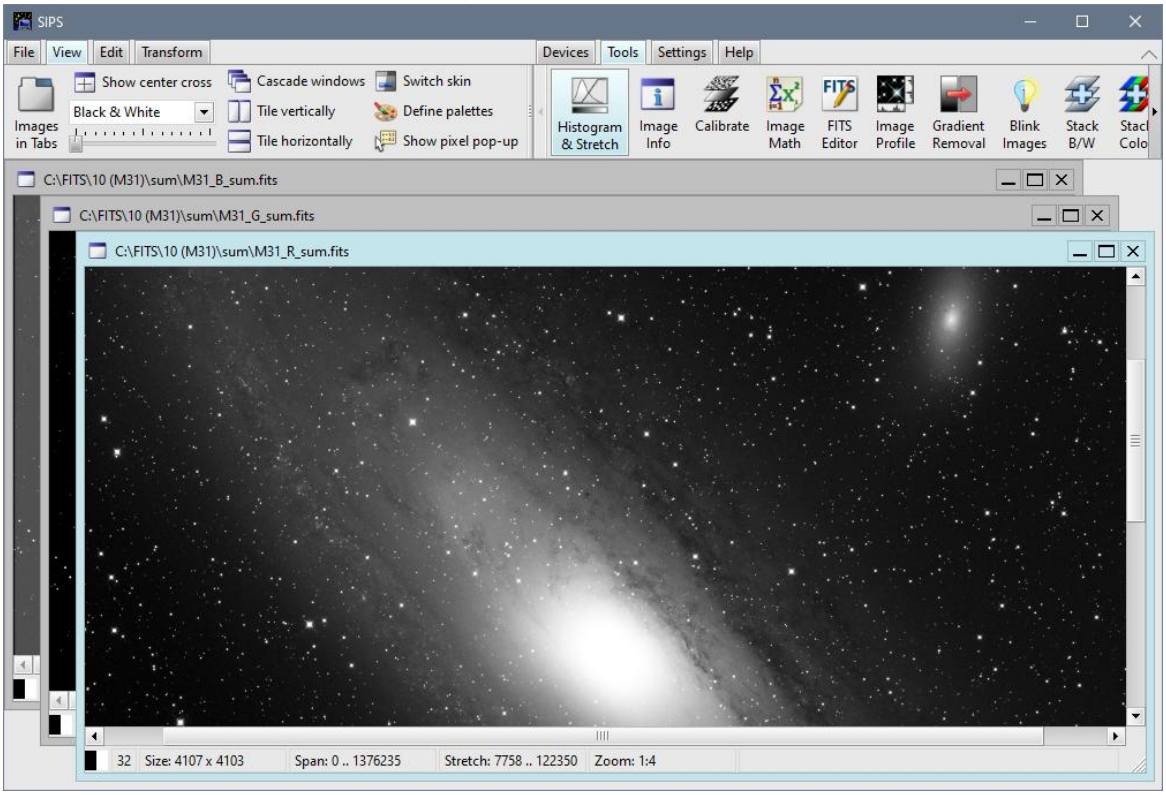
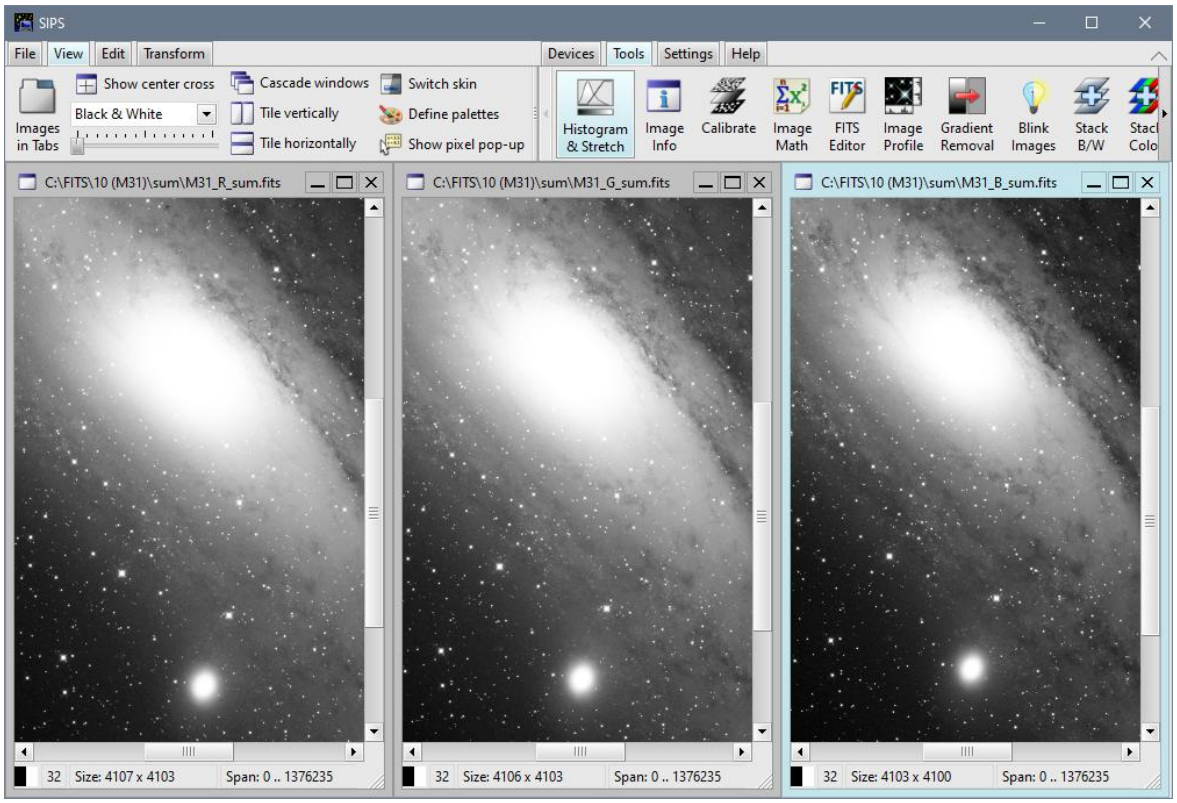


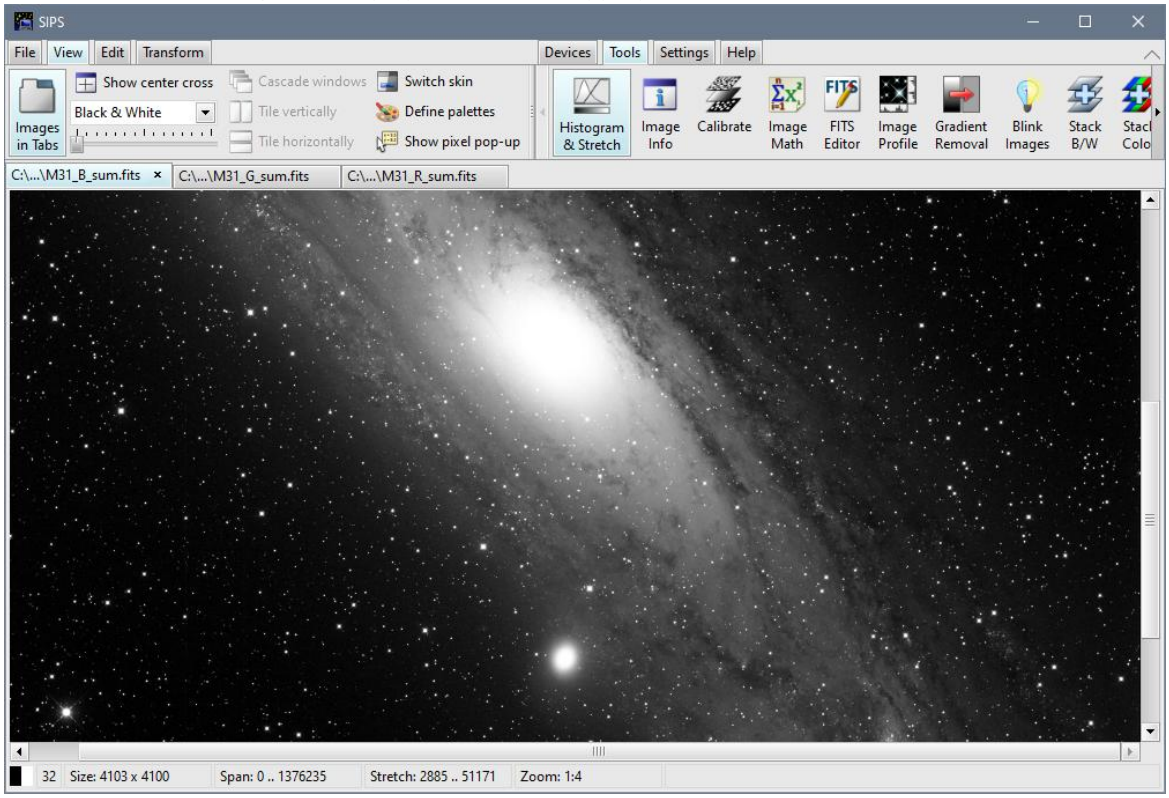
Image windows can overlap, but they cannot be moved outside the SIPS workspace. Image windows can be arranged using command buttons **Cascade windows**, **Tile vertically** and **Tile horizontally** in the **View** ribbon.



Remark:

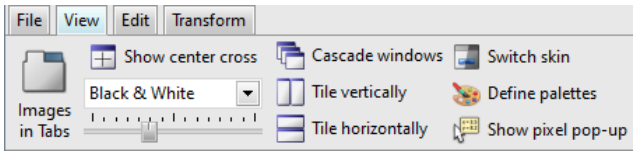
Image lists (described in the separate chapter **SIPS Image Lists**) are also represented by windows inside SIPS workspace, so their windows are arranged together with image windows.

- Images can be displayed in a tabbed pane, covering entire SIPS workspace. This means only one image is visible at a time. Different images are displayed when their appropriate tab is selected.



Switching between these two modes can be performed using **Images in Tabs** command button in the **View** ribbon. If images are displayed in individual windows, any attempt to maximize any window (be it from window menu or by

clicking the maximize button or double-clicking the window title) switches SIPS workspace to tabbed mode. However, switching back from tabbed mode to window mode is possible only by clicking of the **Images in Tabs** tool.



Remark:

Tabbed mode is available only for image windows. Image list windows are always displayed as individual windows even if images are displayed in individual tabs.

Window frame enables normal window manipulations, like zooming, moving etc. Every image window displays status bar:

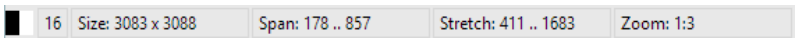
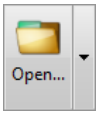


Image window status bar contains several information fields:

- Monochrome/color indication icon (black and white strips for monochrome images and red, green and blue strips for color images).
- Number of bits per pixel: 16 for 16-bit images, 32 for 32-bit images.
- Size: image width and depth in pixels.
- Span: minimal and maximal value of pixels in the image.
- Stretch: minimal and maximal stretch values used to display image. All pixels less than minimal value appear black and all pixels greater than maximal value appear white.
- Zoom: the current zoom applied to display the image. Values from 1:8 up to 8:1 are allowed.

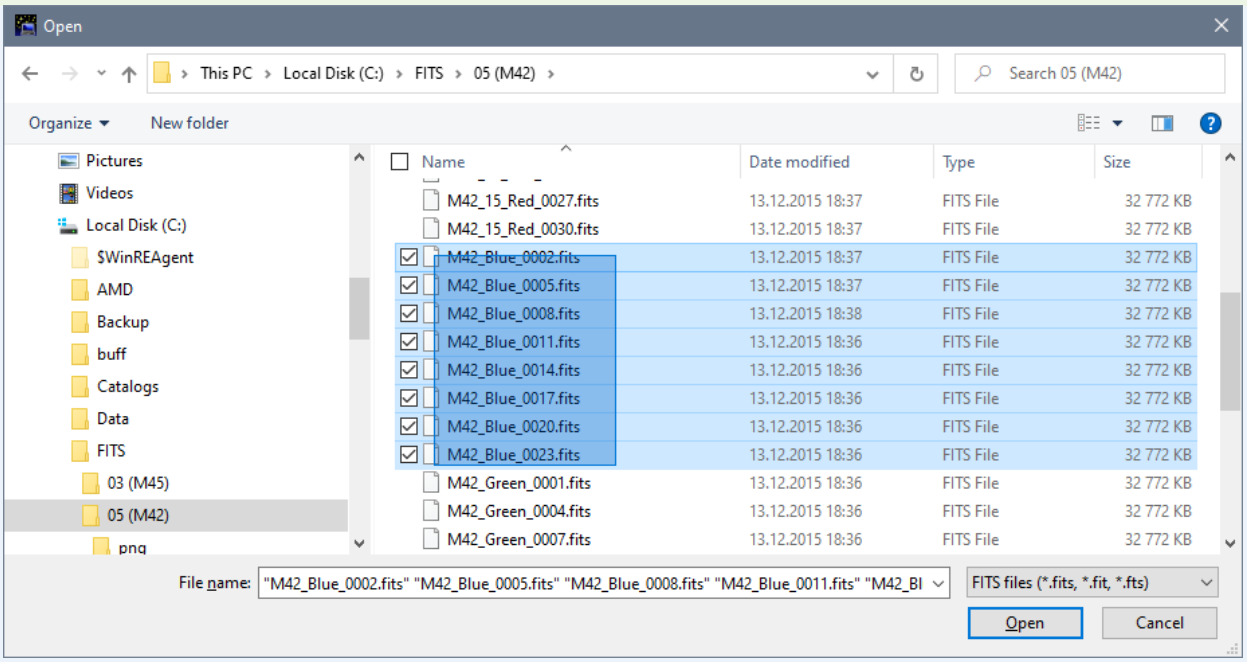
Opening and Saving Images

Images can be opened by 

tool or by pressing <Ctr O>. Open dialog also opens after the doble-click with the left mouse button on the SIPS workspace. SIPS opens only FITS images.

Hint:

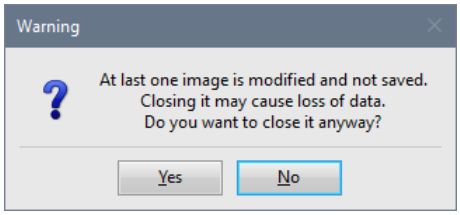
It is possible to open multiple images at once. Simply select multiple images in open dialog box. All rules for multiple file selection valid for Windows Explorer applies here.



The drop-down list box, opened by an arrow right to the open button, shows the most recently used images. Clicking any item opens the image.

If the image has been changed (e.g. its header has been updated) or newly acquired, it can be saved by the Save or Save As... tool or by pressing <Ctrl>+<S>. The default file extension is `.fits`, although extensions `.fit` or `.fts` can also be used.

The Close Images tool closes all opened images. It checks if all images are saved and displays warning if there is at last one unsaved image:



Hint:

Images can be also dragged from Windows Explorer and dropped to the SIPS workspace. Select one or more images in Windows Explorer and drag them with left mouse button pressed over the SIPS workspace. Then release the mouse button and SIPS opens all images in new image windows.

Image Export

Because common image formats (BMP, PNG, JPEG etc.) usually do not support more than 8 bits per single color, they are not suitable for storing of astronomical images. On the other side, only a few specialized programs understand FITS format. FITS cannot be displayed in web browsers, image viewers, imported into documents etc. (FITS import plug-ins for some image editing software are rare exceptions).

To enable publishing images processed by SIPS, images can be exported to PNG, JPG, TIFF, BMP and GIF formats (these formats are called “8-bit formats”, because they store each color component in 8-bit number, allowing 256 levels for each component). The **Export to 8-bit** button is located in the **File** ribbon. Desired format is defined by the file name extension (for instance “.jpg” or “.png”). You can also choose the required file type from combo-box and enter the file name without extension. The resulting image will be saved in the chosen format.

It is important to understand, that the same image transformations, used to display images on computer screen (typically also limited to 8 bits per color component), are used to transform 16- or 32-bit image to 8 bits dynamic range. So, not only the image brightness will be stored the same way it is displayed on the screen, but also possible false-color palette will be applied in the case of monochrome images. Also currently used zoom will be applied for images exported to 8-bit formats.

Remark:

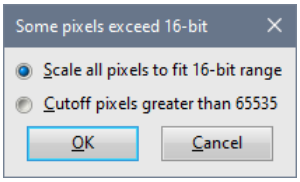
Note that image export always means loss of information—exported image always mimic the image look in the image window. This means image levels of gray are created according to currently defined stretch limits and gamma curve in the Histogram tool. This enables modification of exported images to look well, but the actual photometric information is damaged.

Two general image formats—TIFF and PNG—can also support storing images with 16 bits per color channel. The **Export to 16-bit** button in the **File** ribbon allows exporting of 16-bit images to PNG or TIFF without reduction of dynamic range and without information loss.

Because no transformation (image stretching) is applied to image prior to its export into 16-bit TIFF or PNG, only 16-bit images can be exported. 32-bit images must be converted to 16-bit ones using the 16 x 32 bits tool in the **File** ribbon prior to exporting them to 16-bit TIFF or PNG.

Changing pixel dynamic range from 32 to 16 bits, when there are some pixels exceeding 16-bit limit (65,535), unavoidably discards information. SIPS offers two ways to reduce image dynamic range:

- The scale factor is calculated as a ratio between the value of the brightest pixel and the 65,535 (the maximal 16-bit value). All pixels are then divided by this ratio. This limits the dynamic range of the image, but preserves the relative brightness of all pixels.
- Sometimes only some bright stars exceed 16-bit range and we are interested in a fine nebulosity well under this limit. It could be more useful to cutoff pixels above 65,535 and keep the dynamic range of the rest of the image.



Raw Image Import

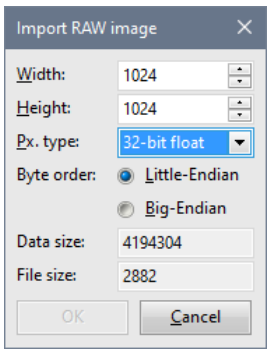
SIPS natively works with standard FITS file format. Other image formats (both 8-bits and 16-bits per pixel) are supported only for image export. But SIPS allows for importing of the raw image in the form of pixel matrix without any header.

Remark:

The "raw format" is often used by still camera manufacturers to denote image format containing pixel values read directly from the camera sensor, but without colors reconstructed from Bayer color mask and often with higher dynamic range than 8-bits per color per pixel. Such images are then processed on the PC, after they are downloaded from the camera.

Unfortunately, the actual format of stored data in "raw" files is proprietary (different camera manufacturers use different formats) and often even single company introduces more, mutually incompatible "raw" formats over time. What is more, such formats are often not documented, particular camera manufacturer does not allow other companies to use it but through supplied libraries etc.

The SIPS **Import raw** function does not concern above mentioned formats, but only pure data matrices without any header, stored in files.



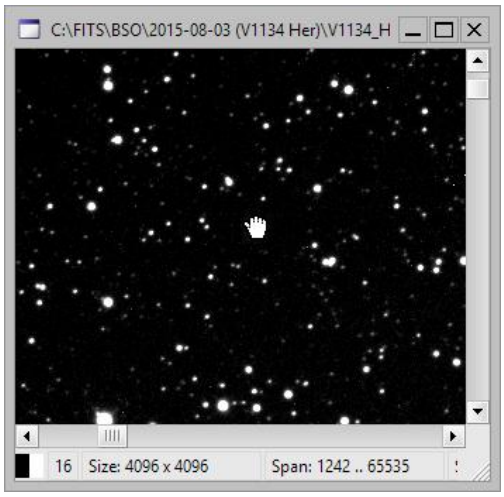
As the image header is missing, number of parameters must be defined so the SIPS understands how to interpret data file. First, it is necessary to define matrix width and depth (height). Then it is necessary to define the data format of each pixel—SIPS allows selection from 8, 16 and 32-bit signed and unsigned integers and also 32-bit floating-point numbers. The last option is byte order of individual pixels.

Hint:

As the expected data type is raw matrix of pixels, the size of imported file must equal to defined width multiplied by depth and byte size of each pixel. The **Import RAW Image** dialog box shows the calculated expected number of bytes and the actual size of the selected file. Until these two numbers are equal, SIPS does not allow to proceed with import.

Zooming and Scrolling Images

If the image size exceeds the space available in image window, horizontal and vertical scroll bars appear. Scrollbars allow selection of the image portion displayed in the image window. But there is much easier way to scroll the image—pressing the right mouse button changes the mouse cursor to the shape of hand:



Pressing right mouse button shows a hand-shaped cursor, which allows dragging of image visible portion. When the hand "holds" the image, moving the mouse with the right mouse button pressed will drag the visible portion of the image.

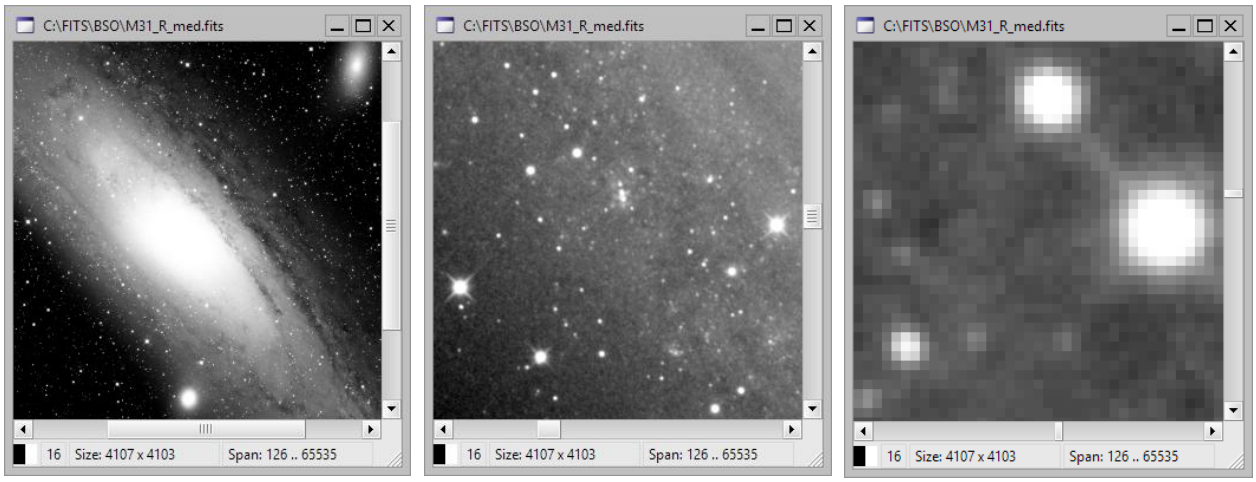
There is a slider in the **View** ribbon  , which can be used to change the image zoom from 1:8 (eight-times smaller) to 8:1 (eight-times larger).

Mouse wheel can be also using to easily zoom-in and out the selected image, as well as to scroll visible portion:

- Rolling the mouse wheel without pressing any modifier keys zooms the image.
- Holding the <Ctrl> key while rolling the mouse wheel scrolls the image vertically.
- Holding the <Shift> key while rolling the mouse wheel scrolls the image horizontally.

Similarly, two finger gestures can be used to zoom image on multi-touch capable laptop computers.

- Put two fingers on the computer touch-pad and move them towards each other (pinch). Image will zoom-out (shrinks).
- Move two fingers away from each other, image will zoom in (enlarges).



Remark:

Changing the image zoom only affects the way image is displayed, it does not affect the image information itself. Zooming the image permanently requires using of commands in the **Transform** ribbon. On the other hand, if you export image to some common 8-bit image format (e.g. PNG or JPEG), the current visual image zoom will be used to create exported image.

Handling zoom of large (4k+) images

Maximum size of Windows USER objects (Windows GUI) as well as GDI (Graphics Device Interface) bitmaps is historically limited to 32767 pixels (maximum value of 16-bit integer). This seems large enough even today, as the highest-resolution

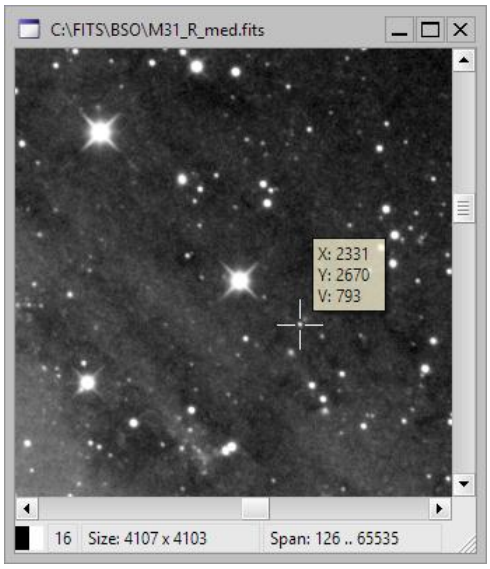
monitors are far from this dimension. But if you consider for instance C4-16000 camera, producing 4096 × 4096 images, and zoom the image 8-times, resulting size is 32768 pixels, 1 pixel more than the above-mentioned limit. C3 cameras with up to 9576 × 6388 pixels or C5 cameras with up to 14256 × 10656 pixels resolution produce images exceeding the 32767 pixels limit when zoomed only 3 or 4-times.

> Remark:
>
> SIPS of course does not allocate the bitmaps that big while zooming images, only the visible portion of the image is actually allocated as displayed bitmap, but the 32767 pixels limit is imposed by using of other internal structures.

To handle this issue, SIPS calculates the maximum zoom value for every image loaded from file or read from camera, depending on image dimensions. If the greater image dimension (width or depth) is less than 4096 pixels, maximal zoom is set to 8×. If the image dimension is greater or equal to 4096, the maximal available zoom is limited (less than 8×) to display such images correctly.

Pixel values

Value of the pixel under mouse cursor is displayed in the **Image Info** tool (providing the tool window is opened, of course). But the **Image Info** tool allows displaying of the mouse cursor coordinates and pixel value in the transparent pop-up window close to the mouse cursor. Just check the **Show pixel pop-up** checkbox. A semi-transparent window appears if the mouse is on some image window.

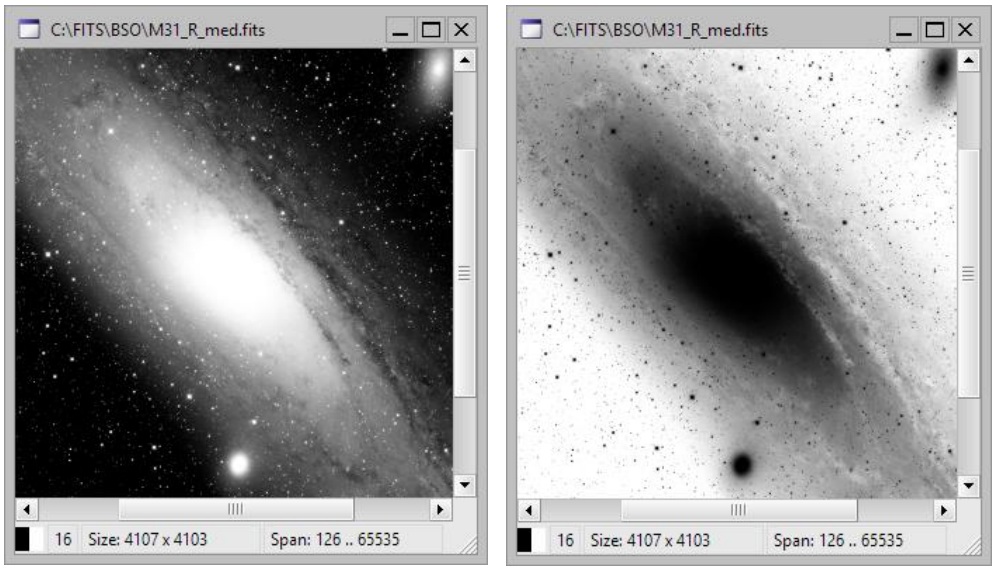


The pixel pop-up displayed above color images shows individual values of red, green and blue colors.

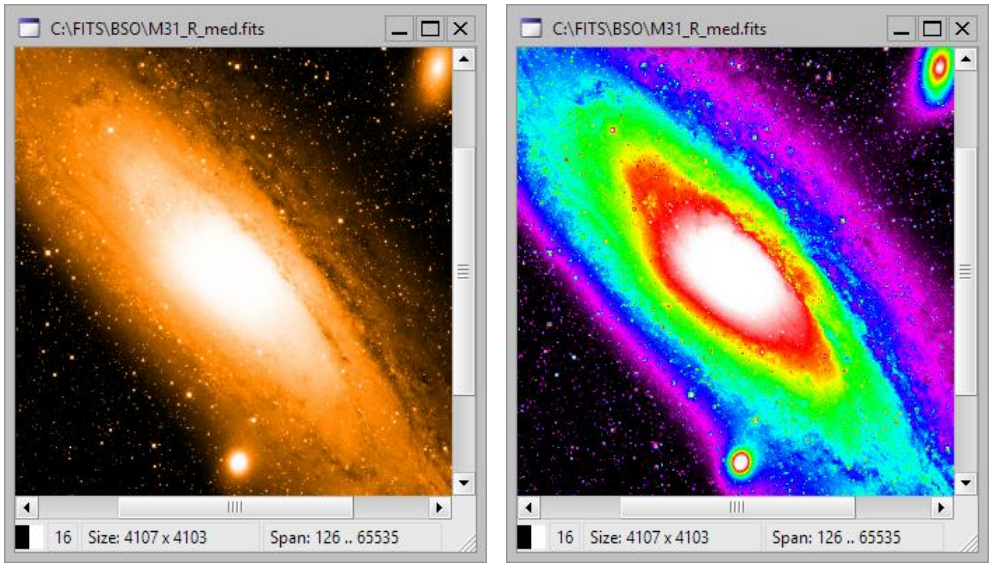
Color palettes

Images are displayed in the black and white palette by default. This reflects the view of prints of astronomical objects photographed on classical black and white film. Even if the image is taken through a color filter, the luminance of individual pixels is expressed by the level of gray.

But the color palette can be changed using the combo-box in the **View** ribbon. Some astronomers prefer negative look of images and sometimes it can be useful to display images in “false colors” to enhance subtle details etc.



Images above show standard black-and-white palette on the left and inverted (negative) palette on the right.



False-color black-orange-white palette shown on the left and “spectrum” palette on the right image.

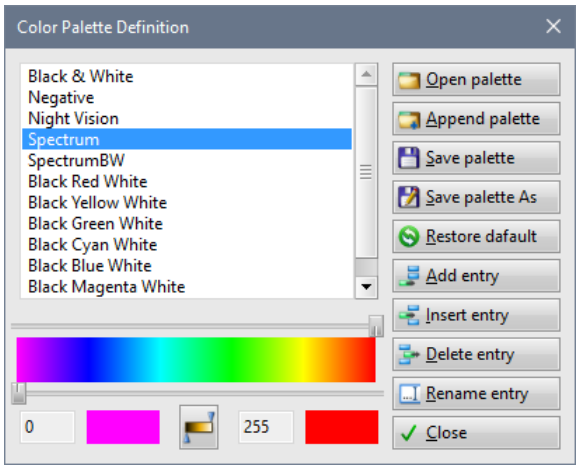
It is necessary to keep on mind that applying a color palette to image display does not change the information contained in image. Applying a palette affects only way images are displayed, not images themselves.

> Remark:
>
> Although color palette does not affect information in images, it is used when exporting images into 8-bit image formats. Exported images will have the same colors as images displayed on the screen.
>
> Palette can be applied only to monochrome images. Color images are always displayed in RGB colors according to pixels in the respective color planes.

Palette definition

SIPS contains a predefined set of color palettes, but it is possible to add user-specific palettes to the predefined ones. Palettes are handled using the dialog box, opened by the 

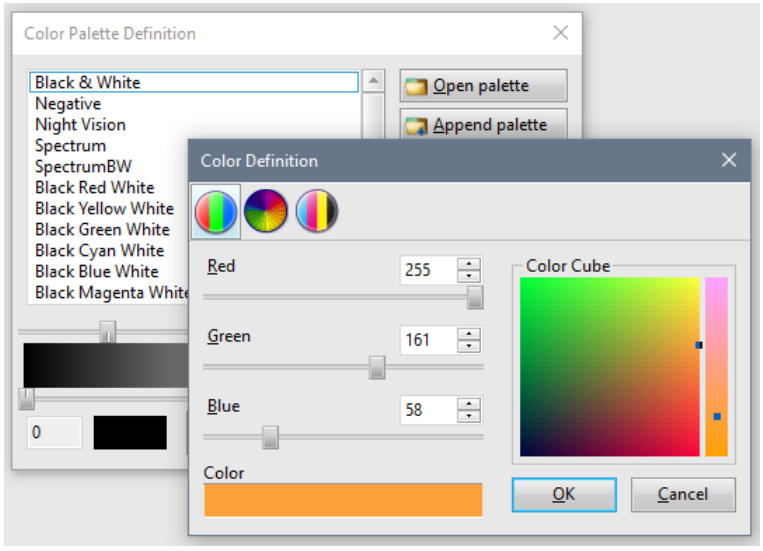
tool in the **View** ribbon.



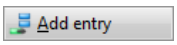
Any palette can be updated, or new ones can be added or inserted into list. It is almost impossible to define the palette one color after another, so the **Color Palette Definition** dialog box provides the interpolation (ramping) tool  .

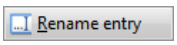
The color interpolation is always performed between two color indexes defined by the two sliders. Note that both sliders can define first as well as last color—it depends on which is more on the left side left and which is more on the right side.

The actual colors, between which other colors are interpolated, are defined by two “Color Wells” on the bottom of the Color Palette Tool window. Color Well is a GUI control displaying selected color. Clicking to Color Well opens the Color Definition dialog box, which allows changing of the Color Well color.

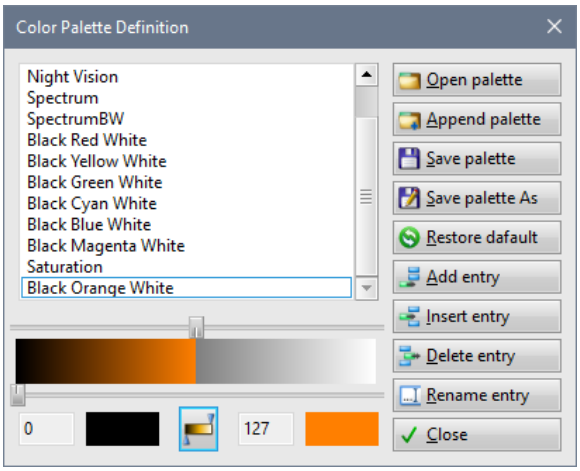


The following example demonstrates adding of new “Black Orange White” palette, often used for publications of monochrome images on the web:

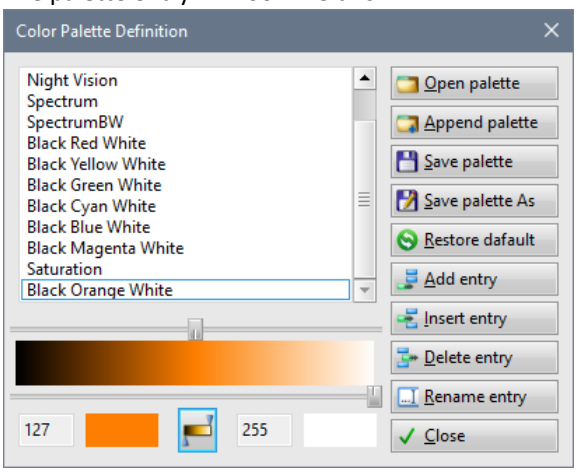
1. Append new palette entry using the 

button. The new entry is filled with the grayscale palette and named “new entry” by default.
2. Rename the item to “Black Orange White” using the 

button.
3. Leave the lower slider on the left side (color index 0) and move the upper slider to the middle of the palette (color index 127). Then define the orange color of the right Color Well (click the Color Well and define color—orange components are R = 255, G = 128, B = 0). Clicking the **Interpolate** button—the first half of the palette is filled with colors ramping from black to orange. The palette entry will look like this:



4. Keep the upper slider on its position and move the bottom slider to the right-most position. The desired color (white) needs not to be defined; the right-most color already is white. Just click the Interpolate button  . The palette entry will look like this:

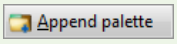


5. Note this palette entry will be discarded when the SIPS is exited. If you want to keep it for later use, save the current palette set using the 

tool. SIPS will load the last used palette file on next startup.

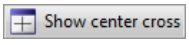
Hint:

SIPS allow to store palettes in multiple files. Last palette file opened in the **Color Palette Definition** dialog box is loaded upon SIPS startup.

It is also possible to merge multiple palette files into one file using the 

button.

Displaying the cross over image center

Checking of the 

tool in the **View** ribbon shows the cross over every displayed image center. The cross does not alter image data; it is only displayed.

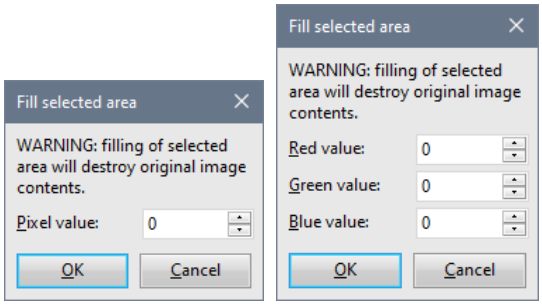
Showing the cross can be useful feature for various observing tasks, for instance:

- While performing equatorial mount polar adjustment using drift method. Placing observed star in the field center allows the user to precisely determine the star drift.
- While aligning the mount to sky, displayed cross enables precise centering of the star in the field of view.

Altering images

Although SIPS is not intended for artwork image creation and thus does not support image editing functions like brushes, pens etc., it provides some very basic editing functions. Both are related to selection areas used also for clipboard operations.

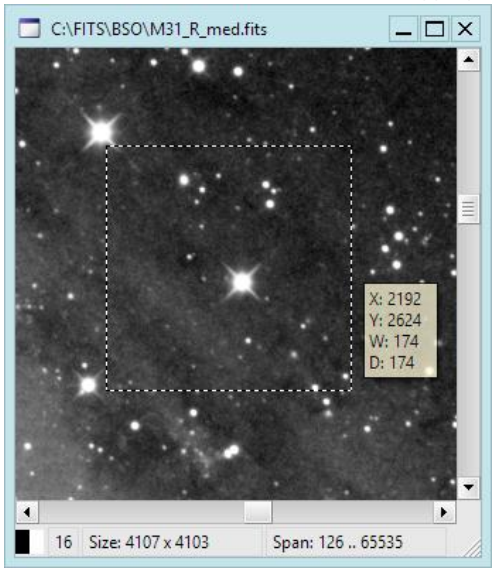
The selected image area can be filled with specified pixel value using the Fill tool. The dialog for choosing the desired pixel value is opened prior to fill operation:



The crop tool Crop allows cutting-off image borders outside the selected portion of the image.

Using the Clipboard

SIPS supports basic image manipulation using Windows clipboard. Any portion of the image can be selected by pressing the left mouse button and dragging the mouse. The highlighted rectangle appears on the image.



Hint:

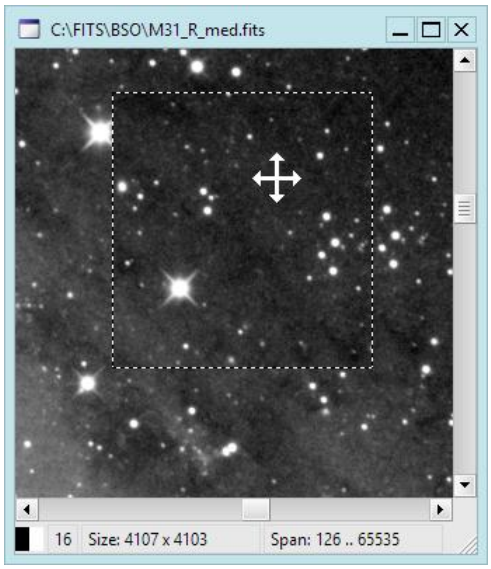
If the pixel pop-up window is displayed, it changes its contents when selecting the image. The currently selected area position and size is displayed instead of current cursor coordinates and pixel value.

It is not necessary to select the region by single mouse drag. If some region is selected, it is possible to scroll the image and again position the mouse cursor over the corner of the selection region (not necessarily the same corner used for dragging when the region was selected). Mouse cursor pointer changes shape to “arrow pointing to the corner” to indicate pressing left mouse button and dragging changes the selected region size.

The whole image can be also selected by pressing <Ctrl>+<A> keys. Image selection can be canceled by clicking anywhere into the image.

Once the image portion is selected, it can be copied to clipboard using the Copy tool or by pressing the <Ctrl>+<C>. There is also a possibility to cut the portion from the image into clipboard using the Cut tool or by pressing the <Ctrl>+<X>. Pixels in the selected area are then filled with the value 0.

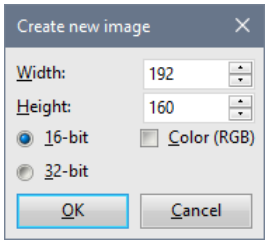
If the clipboard contains a portion of FITS image, it is possible to paste it into existing image using the Paste tool or by pressing the <Ctrl>+<V>. However, the target image is not immediately altered after the paste operation. The pasted portion, surrounded by the dashed line, is displayed floating above the target image instead. It is then possible to move the portion by mouse or by cursor keys (cursor keys move the portion by one pixel, holding the <Ctrl> key speeds the movement to 8 pixels per cursor key press).



Actual pasting (copying pixels into target image) is performed upon the <Enter> key press. Until pressing the <Enter> key the operation can be canceled by pressing <Esc> key.

Clipboard operations can also be used to create new images. New empty image can be created by 

tool in the **File** ribbon or by pressing <Ctrl>+<N>. A dialog box appears:



It is possible to choose new image dimensions, pixel data type and whether it should be monochrome or color (RGB).

> Hint:
>
> This dialog box always checks if there is some FITS image copied to the clipboard. If yes, it offers dimensions (**Width** and **Depth**) corresponding to this image size. That means if a new image is created after the desired image portion is copied into clipboard, it is not necessary to define its size.

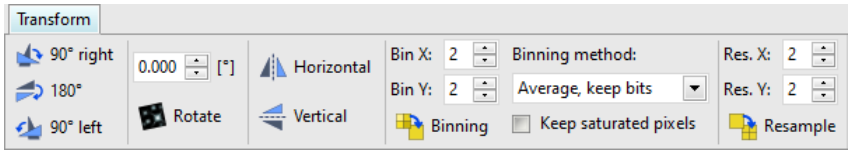
Pressing <Ctrl>+<V> or licking the 

tool then places clipboard content above the newly created empty image and following press of the <Enter> key actually copies pasted pixels to newly created image.

It is possible to paste monochrome images into monochrome ones and color images into color ones, of course. It is also possible to paste monochrome image into color one—monochrome pixels are then copied into all three (RGB) color planes and appears gray in the resulting image. But it is not possible to paste color image info monochrome one.

Image transformation

As opposed to display-related settings (image stretch, zoom, color palette), transformations are performed directly on the pixel matrices, they affect the information stored in images. Transformation commands are available in the **Transform** ribbon.



Right-angle rotation

Basic image rotation by a multiply of 90° (90° right and left and 180°) only re-arranges pixels. This means the information contained in image is not altered, it only changes shape. Astrometric of photometric reduction of images transformed this way is not altered at all.

Such transformation can be applied for instance to compensate for camera mounted rotated or to get the same axis orientation after the tube swap when the German equatorial mount crosses meridian (180° rotation) etc.

Mirroring (flipping)

Mirroring, like right-angle rotation, also do not alter image pixels at all, only arrange them differently. Mirroring can be applied e.g. when some optical element flips image (zenith mirror or prism).

Arbitrary angle rotation

Image rotation by arbitrary angle (not a multiply of 90°) is handled differently, because general rotation inevitably updates pixel values (rotation must work with sub-pixel precision). Still, the transformation is designed to preserve all important image features (e.g. total flux of individual stars) as close to original as possible.

Soft binning

Soft binning shrinks the image by adding or averaging neighboring pixels. Scales in both axes can be defined independently. Soft binning can be used e.g. when the used camera does not support binning in hardware. But even if the driver supports binning in camera, it can be very useful even when the used camera allows hardware binning, but it is desired to maintain as high dynamic range of the binned image as possible.

The **Binning method** combo-box allows to select how pixels are binned:

- Sum binned pixels into 32-bit resulting image.
- Sum pixels but keep the current image dynamic range (16-bit or 32-bit).
- Average pixels. Averaging always keeps the current image dynamic range (16-bit or 32-bit).

The traditional meaning of pixel binning implies adding of binned pixels. This originated in CCD sensors, where pixel charges were literally poured together within the sensor horizontal register and/or the output node.

For CMOS sensors with full 16-bit dynamic resolution, the negative side of binning is limiting of the sensor dynamic range, as for instance only ¼ of maximum charge in each of the 2×2 binned pixels leads to saturation of resulting pixel. CCDs eliminate this effect to some extent by increasing of the charge capacity of the output node and also by decreasing of the conversion factor in binned modes. But such possibilities are not available on CMOS detectors.

> Remark:
>
> CMOS sensors with less than 16-bit precision often just add binned pixels to fulfil the available resolution of 16-bit pixels. For instance, camera with 12-bit dynamic range can sum up to 4×4 pixels and still the resulting binned pixels will not overflow the 16-bit range.

In theory, the resulting S/N ratio of binned pixel remains the same regardless of if we add or average them. Let us take for example 2×2 binning:

- If we add 4 pixels, signal increases 4-times and noise increase 2-times—three additive operations increase noise by $\sqrt{(\sqrt{2})^2 + (\sqrt{2})^2}$. Resulting S/N increases 2 times, but only until the sum of all pixels is lower than the pixel capacity.
- If we average 4 pixels, signal remains the same, but the noise is lowered to ½ as noise is also averaged $\frac{\sqrt{(\sqrt{2})^2+(\sqrt{2})^2}}{4}$. Resulting S/N also increases 2 times, but only until the noise decreases to lowest possible 1-bit of dynamic range.

Until the camera read noise is above 2 ADU, halving it in 2×2 binning mode still keeps the read noise above the lower 1-bit limit and at the same time binned pixel will not saturate. For higher binning modes, the noise approaches lower limit, but averaging pixels still protects from pixel saturation, which is more important than possible S/N limitations caused by underflow of read noise.

If we consider that the image background noise is only rarely defined by the read noise of the sensor, as the noise caused by background sky glow is typically much higher, for 16-bit camera averaging pixels is the better way to bin pixels compared to just adding them.

The software binning brings one more possibility to bin pixels—adds pixels, but at the same time converting the resulting image from 16-bit to 32-bit dynamic range. This means S/N of the binned images always increases, pixels never saturate and read noise newer approaches lower limit. The negative side of this option is two-time bigger images.

Saturated pixels within bright stars are no issue for aesthetic astro-photography, but photometry measurement is invalid if any pixel within the measured object reaches maximum value, because it is not possible to determine the amount of lost flux. Software performing photometry (e.g. the SIPS **Photometry** tool) should detect saturation value and invalidate entire photometric point not to introduce errors.

But binning efficiently obliterates the fact that any of the binned pixels saturated (except for all binned pixels reached saturation value). So, using of binning modes for research applications (photometry and astrometry) can lead to errors caused by lost flux in saturated pixels, which cannot be detected by processing software due to binning.

The **Keep saturated pixels** checkbox causes the resulting binned pixel is set to saturation value if any of the source pixels is saturated. For aesthetic astro-photography, unchecking this option could result into slightly better representation of bright star images, but for research applications, this option should always be checked.

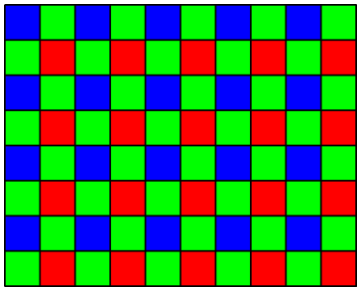
Resampling (enlarging)

Resampling scales can be set independently in both axes. SIPS implements bilinear resampling, which means pixel values are linearly interpolated from neighboring pixels. This method provides smoother images than simply enraging of each pixel, which method is used only to display zoomed images in image windows.

Image resampling can be very useful when combining images taken through color filters with unfiltered luminance image. Usually, the images taken through the filters are binned, while the luminance image is acquired without binning to achieve maximum angular resolution. It is then necessary to resample color images to enlarge them to the same pixel scale as is the scale of the luminance image.

Debayer processing

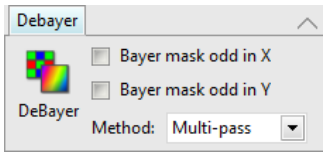
Single-shot color cameras use special detectors with red, green and blue color filters applied directly on individual pixels. Every pixel receives light of particular color only (red, green or blue). But color image consists of pixels with all three colors specified. So, it is necessary to calculate other color from the values of neighboring pixels.



Schematic diagram of color filters applied on individual pixels are displayed above. Notice there are two times more green pixels compared to green or blue ones.

> Remark:
>
> Covering pixels with such color mask and subsequent calculations of remaining colors was invented by Mr. Bayer, engineer working at Kodak company. This is why this color mask is called **Bayer mask** and the process of calculation of missing color is called **Debayer** processing.

Commands related to Debayer processing are in the **Debayer** ribbon.



Hint:

As most cameras used in astronomy are monochrome, often equipped with filters in a filter wheel, the Debayer ribbon is hidden in the default state. To show this ribbon, right-click the space with tab names above ribbons (providing ribbons are grouped in top container) and select the Debayer item from the pop-um menu.

See **Appendix A: SIPS user interface** for details about ribbon handling.

There are several algorithms for calculating missing color components of individual pixels—from simply using of color from neighboring pixels (this method provides quite coarse images) to more accurate methods like bilinear or bicubic interpolation. There are even more sophisticated algorithms like pixel grouping etc.

It is possible to perform Debayer processing immediately when the image is downloaded from the camera (color image is then immediately displayed and/or saved and no raw monochrome image is shown) or to perform this processing anytime later.

The Bayer mask can begin with pixel of any of the four colors (green color is used two times on 2×2 pixel matrix). But there are no rules specifying the color of the first pixel—in principle there can be blue or green pixel from the blue-green line on the upper-left corner, as well as green or red pixel from the green-red line.

There is no way to determine the Bayer mask organization from the image. This is why the **Bayer** ribbon provides two checkboxes called **Bayer mask odd in X** and **Bayer mask odd in Y**. Combination of these checkboxes allows specification of Bayer mask organization on the particular sensor.

Hint:

State of **Bayer mask odd in X** and **Bayer mask odd in Y** checkboxes are always updated when you connect camera with a color sensor according to the information provided by the driver. It is necessary to update them manually only if the raw color image is loaded from the disk file and needs to be processed without connected camera.

Wrong definition of these two flags results in wrong color calculation. Proper settings can be easily determined by the try-and-error method. But Debayer processing discards the original raw image so it is always necessary to backup the original raw image.

Please note the settings of the **Bayer mask odd in X** and **Bayer mask odd in Y** check boxes must be altered when any geometric transformations are applied to the raw image (e.g. mirroring, right-angle rotation, etc.). Some transformations (e.g. soft binning, resampling, arbitrary rotation) cannot be performed on raw images at all. It is always better to Debayer images first and process them later.

Also note that stacking raw color images results in loss of color information. Stacking algorithms align images regardless of if the particular pixel is red, green or blue. SIPS stacking is always sub-pixel, which can mix pixels of different colors. Images must be Debayer processed first and then stacked.

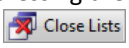
SIPS supports simple bilinear and complex multi-pass methods of Debayer processing. Refer to Advanced reconstruction of color information of single-shot-color cameras chapter for detailed description.

SIPS Image Lists

As already stated in the introduction to SIPS, image list contains references to images. The primary purpose of introducing image lists is to enable easy manipulation of multiple images at once. Instead of performing an operation on all images saved in specified folder and/or on all images saved under certain name, operations on multiple images are performed on lists in SIPS.

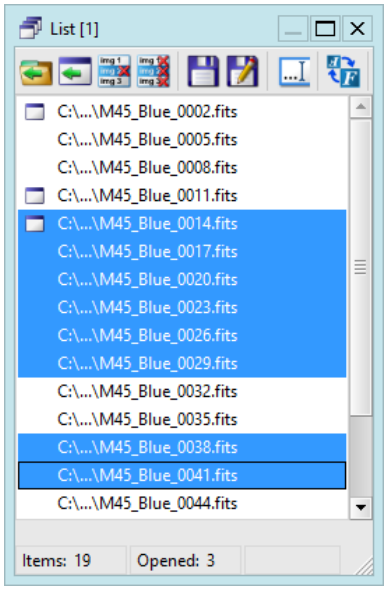
Every image existing within SIPS has a reference counter. This counter is incremented when the particular image is displayed or when the image is inserted into list. If the image window or image list is closed, reference counter of image(s) is decremented, and the image is destroyed only if the reference counter reaches zero. If the user for instance closes image window with unsaved image and the same image is also part of some image list, nothing happens—only image reference counter is decremented, SIPS still keeps the particular image in memory. Double-clicking the image in the list opens image window (and increments image reference counter again).

New list can be created by pressing the 

button in the **File** ribbon or by pressing the <Ctrl>+<L> keys. Single list can be closed as any other window, all opened lists can be closed by clicking the 

button.

Image lists are always displayed as list windows in SIPS. The window contains tools to add and remove images, but the main part of the image list window displays list of images. If an image is visible (opened in some window), its name is preceded by small icon of window. Double-clicking the image name in the list box shows the image (opens image window). If the image is already opened, it pops above other image/image list windows.



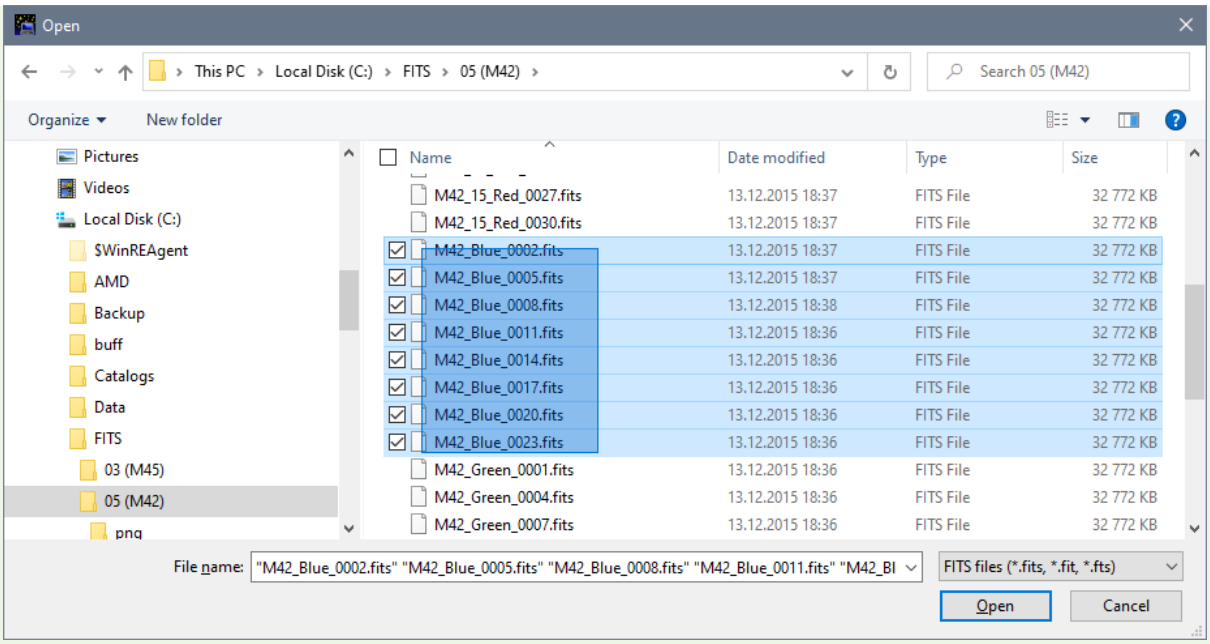
Handling list contents

Every list window offers several tools to manipulate its contents:

Add images from disk

 Standard open dialog appears and allows choosing files to be added to this list.

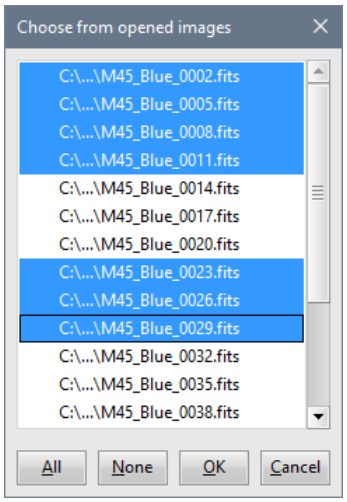
> Hint:
>
> It is possible to open multiple images at once. Simply select multiple images in open dialog box. All rules for multiple file selection valid for Windows Explorer applies here.



It is also possible to drag file or multiple files from Windows Explorer and drop them on the Image List window. In such case, images will not be opened either into individual windows or into tabbed panes, like in the case files are dropped on the SIPS workspace, but will be inserted into particular image list.

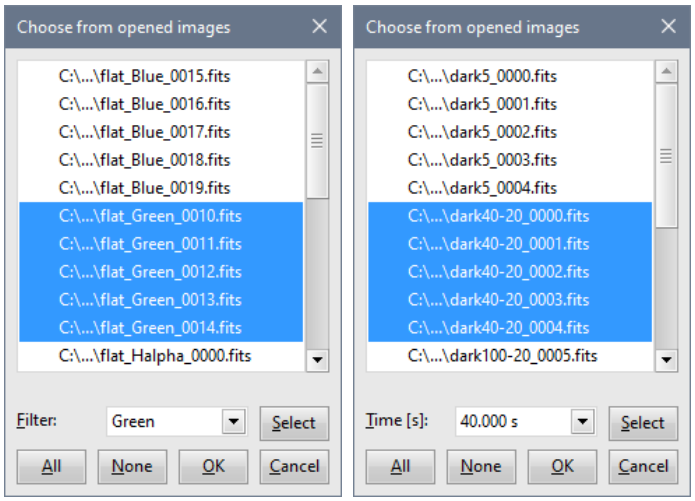
Add already opened images

If image is already opened (be it in image window or in another list), it can be added using this tool. A list of all opened images appears:

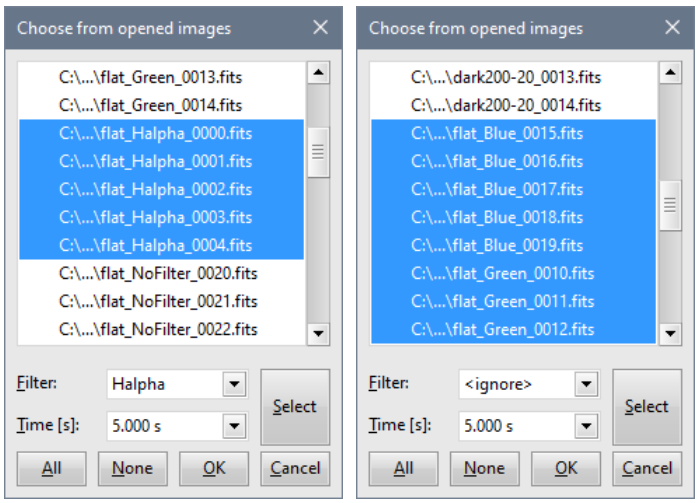


This dialog box allows selecting (and thus adding to image list) of multiple images at once.

Various image processing tasks often require list of images of the same exposure time (e.g. Calibration) and/or images taken through the same filter (e.g. Photometry). This is why the opened image pick list dialog box scans all images for exposure time and filter used for exposure and if there are images, differing in these parameters, selection of the sub-list taken with the same exposure time or through the same filter is offered:



If the already opened images differ in both exposure time and filter, the pick list dialog box offers selection of both parameters. Still, selection can be done according to either filter or exposure or filter and exposure time. If the particular parameter should be ignored, then just choose the <ignore> option in the respective combo-box:



Remove selected image(s) from the list

Currently selected image or images are removed from the list. Please note the list window list-box allows multiple items to be selected; all selected images are removed by this tool. If some of the selected images are modified (unsaved), user is prompted to confirm discarding of changes.

Empty the list (remove all images)

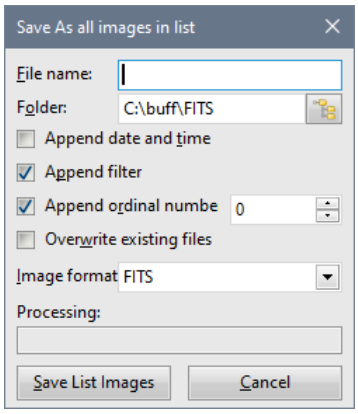
All images are removed from the list. If there modified (unsaved) images in the list, users are prompted to confirm discarding of changes.

Save all modified images

If the list contains any unsaved (modified) images, this command saves them to disk. If there are images in the list not yet saved yet (there is not file name assigned), the **Save As** file dialog box appears asking to choose the folder and file name for the particular image.

Save all images under different names

Because list contains multiple images, SIPS opens dialog box, which allows choosing of the core file name and folder, where all images should be saved.



The dialog box also allows automatic creation of more complex names, created from the core name extended with:

- **Date and time**: date and time of the image exposure is converted to text and appended to the core name.

  Flat_2021-01-10_20-32-19.fits

  This option is used only if the particular image contains information about exposure time.
- **Filter**: name of the filter used to take the exposure.

  Flat_Luminance.fits

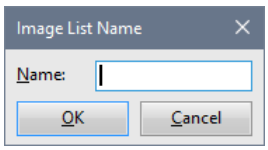
  This option is used only if the particular image contains information about filter used.
- **Ordinal number**: the index of the image in the list. The first index can be defined in the count box.

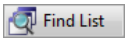
  Flat_0000.fits

  This option can be always used. It is recommended to append ordinal number, especially when the last option **Overwrite existing files** if checked. If for instance multiple images are captured within the same second (planetary images taken with short exposure is an example), they cannot be distinguished by the date and time, because the time is the same. Ordinal number ensures unique file name for every image.
- **Overwrite existing files** checkbox causes overwriting of files without questions if the newly created file name corresponds to already existing file on the selected folder.
- **Image format** allows choosing if the images in list should be saved as FITS files (the default value) or exported into some common file format—PNG, PNG16, JPEG, BMP, TIFF or TIFF16.
  The behavior varies according to selected format:
  - **FITS** just saves a copy of each image, including all headers. No information is lost in such case.
  - 8-bit image formats (PNG, JPEG, BMP, TIFF) behave exactly like single image export, which means current stretch, zoom and color palette are used to create 8-bit image.
  - 16-bit PNG and TIFF export again acts like single image 16-bit export, preserving the dynamic range of 16-bit FITS images, but loosing information in FITS headers.

Name the list


Image lists can be named to make identifying particular list easier. For instance, we can create two lists, one for dark frames and another for flat fields. Newly created image lists will be named “ImageSet [order number]” by default. If we name the first list “Dark frames” and the second one “Flat fields”, if would be much easier to direct image sequence to appropriate list than to lists named e.g. “ImageSet [4]” and “ImageSet [5]”.



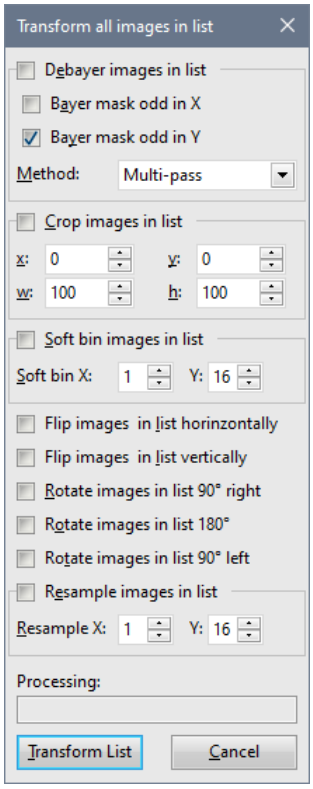
Having particular list named is also useful when using the 

tool in the **File** ribbon to locate image lists.

Transform all images in the list

 Individual images can be transformed immediately when downloaded from the camera (see the **New Image Transform** dialog box, opened from the **Settings** ribbon) or any time later using commands located in the **Transform** ribbon. But transforming every image in a list one by one is time consuming task. This is why image lists in SIPS allow applying of transformations to all images in the list.

Transformations are defined in the dialog box:

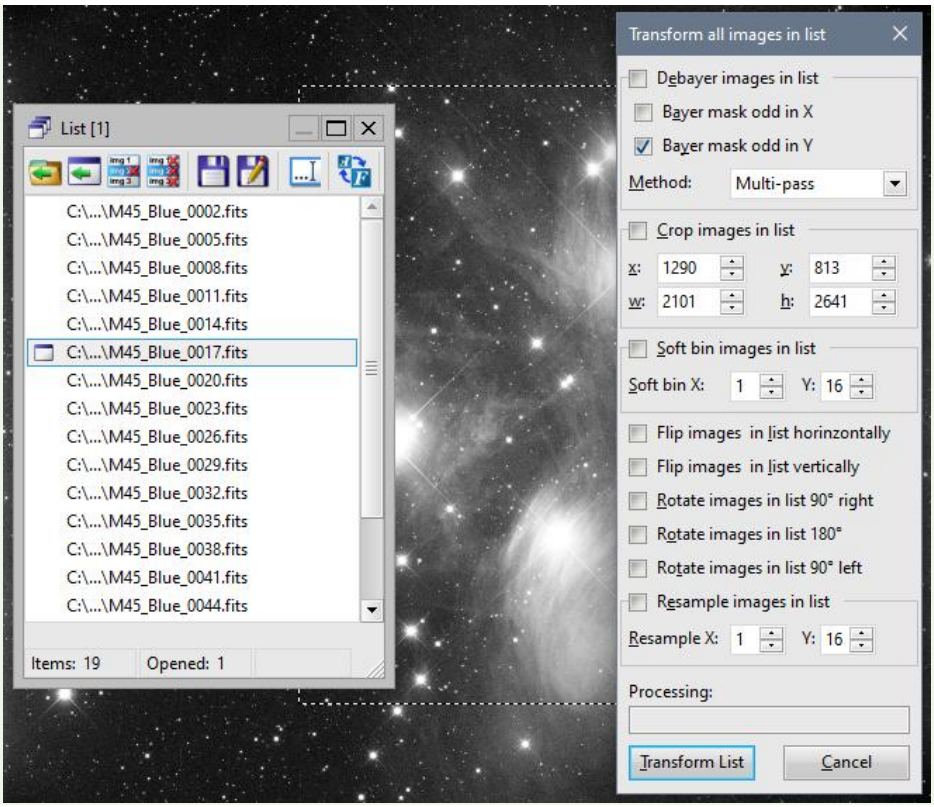


Checking of multiple checkboxes allows multiple transformations at once. Transformations are performed in the order defined by the individual controls position—operation controlled by the topmost check/box is performed first and operation controlled by the lowermost check/box is performed last.

In-depth description of individual transformations can be found in the chapter **SIPS Images**, sub-chapter **Image transformation**. The only exception is the cropping operation, which is in the case of single image performed by the 

tool located in the **Edit** ribbon, based on the selection frame.

> Hint:
>
> It is not necessary to define crop coordinates in the respective count-boxes. If there is an image from the list opened (not necessarily the first image in the list) and a frame is selected on the image, the x, y, w and d count boxes are filled with the selection frame coordinates upon the **Transform all images in list** dialog box opening.



Transformations are performed in background, and the progress bar indicates the number of images already transformed. Operations can be cancelled; in such case the rest of images will stay unchanged.

Sorting images in list

Images are stored in list in the order, in which they were added to the list. If it is desirable to reorder images, List windows offer two tools to sort image according to its name  or acquisition date/time  .

A list can contain named images (ones with associated path) as well as images not saved yet, marked with ordinal number only. If images are sorted by name, unnamed images are placed first and ordered by their number. Named images are then sorted below unnamed ones.

Hint:

By default, the size of newly created image list windows is narrower than the full length of the list window toolbar. So, the sorting command buttons are hidden behind the window edge. Just slightly zoom the list window to see these command buttons.

Sensor characterization using “Photon Transfer Curve”

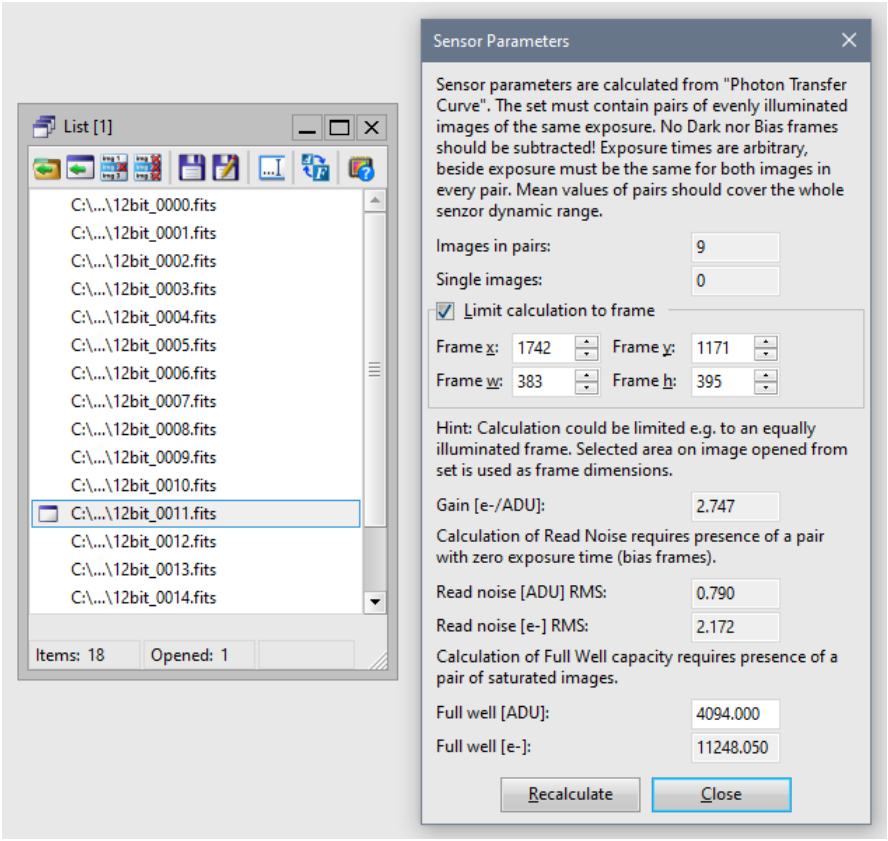
Laws of physics dictate that the variations of signal acquired by a sensor (light incoming to camera from a telescope in our case) are proportional to the square root of signal value. This law can be used to characterize every solid-state detector (both CCD and CMOS) purely on a series of uniform light images (flat fields). Even if particular manufacturer does not publish key sensor parameters like conversion factors in e^-/ADU, full well capacity in e^- or absolute read noise in e^- RMS, analyzing series of flat field images covering the sensor dynamic range and constructing so-called Photon Transfer Curve can reveal them.

SIPS image list windows have a tool button  , performing the sensor characterization. To work properly, the list must contain **pairs of evenly illuminated images** of the same exposure. No Dark nor Bias frames should be subtracted from these light images. Exposure times are arbitrary beside exposure must be the same for both images in every pair. Mean values of pairs should cover the whole sensor dynamic range.

Calculation could be limited e.g. to an equally illuminated area (sub-frame). Such sub-frame can be defined using count-box controls, or area can be framed on any image, opened from list, and the tool takes this area as sub-frame dimensions.

Calculation of **Read Noise** requires presence of a pair with zero exposure time (bias frames).

Full Well Capacity is calculated correctly only if a pair of images with higher mean value (longer exposure time) is fully saturated. However, even if such pair is not present in the list, the maximal ADU value for saturation can be entered manually (4095 for 12-bit cameras, 65535 for 16-bit cameras etc.) to calculate proper Full Well capacity.



Please keep on mind the sensor characteristics are calculated using statistical analysis, and thus results may slightly vary depending on the list of test images, their dynamic range, sub-frame selected for analysis etc.

Image list usage in SIPS tools

It is possible to find reference to image lists in various tools in SIPS. For instance, the **Imaging camera** tool, which handles camera control like exposure, image download, chip temperature adjustments etc., offers three ways to handle image downloaded from camera—you can open the image into new image window, same image to file and/or put image to specified list.

Here are some examples of how image lists can be used:

- A series of dark frame images is read from camera to image list. The master dark frame is calculated as average or median of this list. It is not necessary to save images to files or to display them to create average of multiple images.
- Similarly, number of flat field images are acquired and put to another image list. Master flat field is calculated as median of images in the list. As with dark frames, individual images need not be saved to disk files.
- Observing session begins with number of images of required target. Images are read automatically from the camera with defined interval. Images are saved under file names automatically generated according to parameters specified in the **Imaging camera** tool and also stored into yet another image list. The whole list then can be calibrated using previously created dark frame and flat field at once.

There are also SIPS tools, containing an image list, which must be filled with images prior to performing the respective tool function. For instance, the **Blink Images** tool sequentially shows aligned images, which were added to its implicit list. Similarly, **Combine Images** tool stacks images from its list, **Photometry** tool calculates photometry of series of images added to its list etc.

Remark:

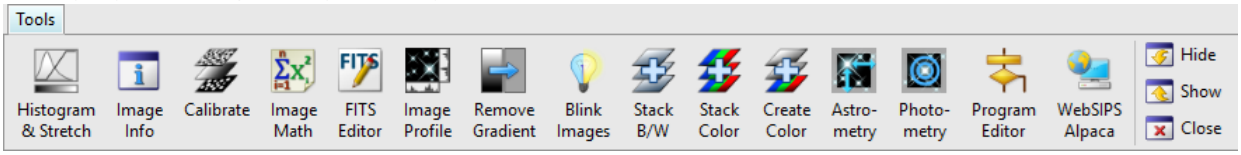
Note that image lists cannot be persistent in SIPS—it is not possible to save image list to disk file and to load it later. The list can refer to non-persistent image (image existing only in RAM) and such list cannot be created again (loaded from file) if the image itself does not exist.

SIPS Tools

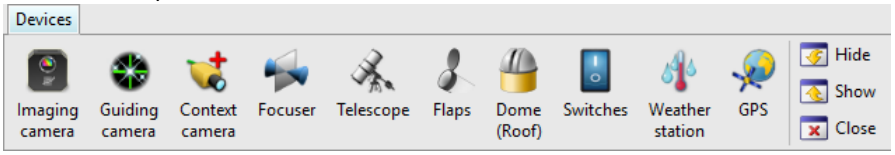
Every SIPS tool is focused to some tasks. For instance, there is a tool for capturing images from camera, tool for image blinking, tool for performing image math etc.

SIPS GUI organizes tools to two groups:

- One group contains tools intended for data handling. Tools from this group are typically used when processing already captured images. They can be opened from the **Tools** ribbon.



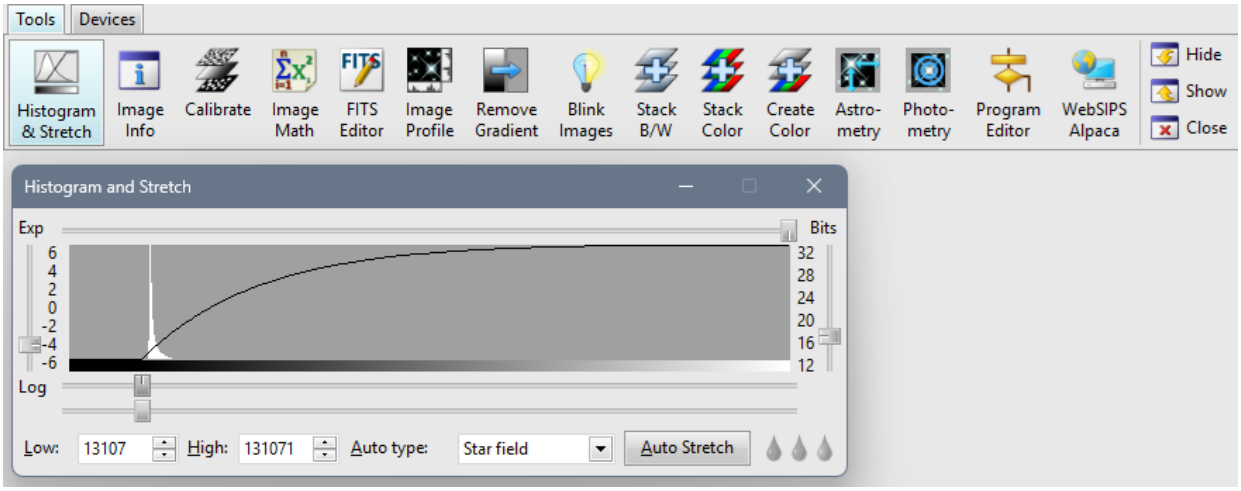
- Another group contains tools focused to device handling (camera, filter wheel, telescope mount, …). These tools can be opened from the **Devices** ribbon.



Tool windows handling

Every tool uses its own tool window. Tool windows are "floating" over SIPS main window. Because image and image list windows always live inside the main SIPS workspace, tool windows always float also above images and image lists.

Buttons in the **Device** and **Tools** ribbons open respective tool window. Ribbon buttons remain "pressed" while the corresponding tool windows remain opened. Each tool window is closed by clicking the upper-right closing button (marked with [×]) in the window title.



Especially tool windows, controlling hardware devices, perform their functions only when opened (for instance, the **Imaging camera** tool, controlling main imaging camera). Closing such tool window also stops all performed operations (closing the **Imaging camera** tool stops acquiring images, closing the **Guiding camera** tool stops guiding etc.). This is why all tool windows also have the minimizing button (marked with [–]) in the window title enabled. Clicking the "Minimize" button hides the tool window, but keeps it active. This is how to free the desktop space from not currently needed tool windows, but keep them performing their tasks.

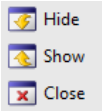
When the tool window is hidden (by clicking on the minimizing button), the corresponding button in the command ribbon remains in the "pressed" state, indicating the tool window is still opened, only not visible. Clicking the already pressed command button shows the tool window again.

Clicking the already pressed ribbon button while the corresponding tool window is displayed (not minimized or hidden), behaves differently depending on the tool window z-position. If the tool window is not on top, clicking its ribbon button

moves it to the top position above all other tool windows. Once the tool window is on top in the z-order, clicking its ribbon button hides (minimizes) it.

> Remark:
>
> Here comes one important difference between SIPS v4 and previous versions. Tool windows are opened the same way, only previous SIPS versions organized buttons into tool-bar, not into ribbons. But previous SIPS versions did not allow to hide tool windows while keeping them active, it was only possible to close them. So, clicking on the already pressed tool-bar button was an alternative way to close a tool window. Such action either shows the hidden window or has no effect if the window is already visible in SIPS v4. It is necessary to click the closing button in the window title bar as the only way how to close it.

The right side of both **Tools** and **Devices** ribbon contains three buttons, allowing to hide (minimize) all opened tool or device windows, show (restore) all opened and hidden windows and close all opened tool or device windows.

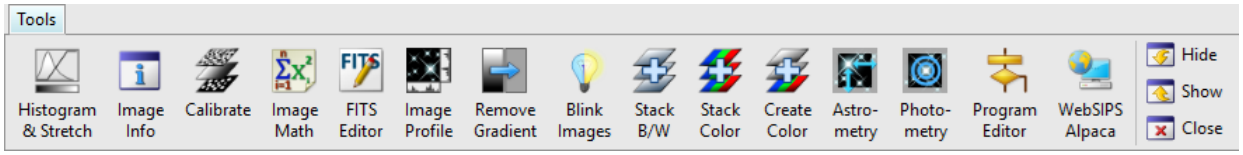


SIPS remembers the open state and position of all tool windows. When you start SIPS again, it opens tools opened in previous session and places them in the same position.

It is possible to start SIPS with all settings, including tool window positions and state, in the default state. It is necessary to specify the command line parameter **/clean** when launching SIPS. See the **Configuration handling command line parameters** sub-chapter in the Introduction to SIPS.

Data Handling Tools

This chapter describes the group of tools intended for handling and processing images, as well as tools intended to provide programmatic control (scripting), remote access to SIPS etc.



Histogram and Stretch defines how to transform high dynamic range images (typically 12, 14 or 16-bit precision) to 8-bit color depth shown of computer screen. This means how images appear on screen.

Image Info shows basic information about selected image.

Calibrate tool can subtract dark frame and apply flat-field correction either to single image or to whole list of images. This tool also supports advanced calibration for dual-gain sensors.

Image Math provides various mathematical operations on images vs. scalars, on images vs. images or on image lists etc.

FITS Editor tool allows to view and possibly update FITS headers in the text form.

Image Profile tool shows profiles in X and Y axes of the selected image.

Remove Gradient tool is designed to model the image background and subtract it from the image, so the background is flat.

Blink Images tool allows to repeatedly show images (possibly mutually aligned) in the tool implicit list with defined speed.

Stack B/W images tool allows mutually align images in the tool implicit list and combine them to sum or median.

Stack Color images tool allows does the same, but for color FITS images.

Create Color image from individual red, green, blue, and possibly luminance images.

Astrometry tool allows astrometric reduction of the image (resolving equatorial coordinates for every star or other object within the image).

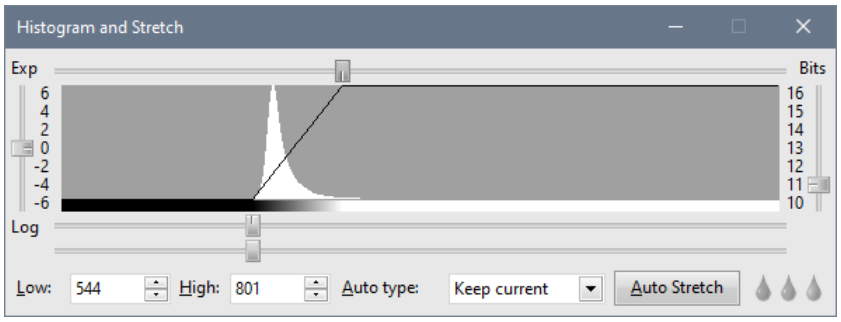
Photometry tool can perform complex astrometric and photometric reduction of images in the tool implicit list and create light curves, save photometry reports etc. This quite complex and powerful tool offers a lot of advanced features like searching for variable stars, photometry tasks for repeated processing and many more.

Program Editor tool is designed to create and run scripts written in the SIPS internal scripting language SPL (SIPS Programming Language).

Web Server publishes a RESTfull API for SIPS, which can be used to control SIPS from and remote client (for instance Python script). SIPS also contains client JavaScript application, which allows remote control of SIPS observing session from any web browser.

Histogram and Stretch

The **Histogram and Stretch** tool combines two functions in SIPS—it displays histogram profile of currently selected image, and also allows settings of image stretching used to display images on the screen. Why are these functions combined? As you will see, histogram profile provides the best hint for setting of stretching limits.



Histogram is a graphical representation of relative number of pixels with the same value in the image. The number of pixels of particular value (x-coordinate) is represented by the height of the column (y-coordinate). For instance, the image uniformly filled with pixels of value 100 has a very simple histogram—just one column at the position 100.

Number of columns in histogram should correspond to number of possible pixel values in image. But there can be up to 65,536 different pixels in 16-bit image and 32-bit images can have every pixel different (32 bits provide 4,294,967,296 combinations). This is why the histogram resolution is limited to 512 columns in SIPS and several pixel values are grouped to one column.

The range of pixels displayed in the histogram (and in the selected image) can be defined by the Bits vertical slider. Any value between 12 and 32 bits can be selected. 12 bits represent 4,096 combinations, so every histogram column represents interval of 8-pixel values (first column shows number of pixels between 0 and 7, second column shows number of pixels between 8 and 15 etc.). Number of pixel values grouped into one column increases as the bit range increase, of course. But the limitation to 512 columns does not affect the histogram information value.

The Histogram and Stretch tool changes its content according to currently selected image. If you select (e.g. by mouse click) another image, the content of this tool reflects histogram, stretch and bit range of newly selected image.

Changes to image stretch are performed in real time—there is no “OK” or “Apply” button. This allows interactive stretch settings, but can be somewhat demanding for computer speed especially when displaying images with tens thousand or even hundreds thousand pixels.

The function of Histogram and Stretch tool is as follows:

- **Low** count-box defines the low stretch limit. All pixels with values below this limit will appear black. This count-box is coupled with the slider just below the histogram profile.
- **High** count-box defines the high stretch limit. All pixels with values above this limit will appear white. This count-box is coupled with the slider above the histogram profile.
- **Bits** slider limits number of bits used for histogram profile and for limits settings.
- **Exp/Log** slider defines the profile of the stretching curve. The default is linear, but it can be exponential or logarithmic.
- The **Auto type** combo-box and Auto **Stretch button** allows definition of stretch minimal and maximal values according to predefined rules. Clicking Auto Stretch button sets stretch limits of the currently selected image according the value selected in the Auto Scale Type combo-box. At the same time all newly acquired or opened images will be also stretched according to the Auto Scale Type combo-box value.
  The available rules are:
  - **Full Scale**—minimal value will be 0 and maximal value 65,536 for 16-bit images and 4,294,967,296 for 32-bit images.
  - **Pixel Range**—minimal and maximal value will be set according to the minimal and maximal pixel in the image.

- **Star Field**—minimal value will be set to the value of maximal number of pixels in the image. It is supposed that the black background covering maximal number of pixels. Maximal value is calculated as minimal value plus the standard deviation (RMS) of pixel values.
- **Galaxy**—values are calculated similarly to Star Field, but the maximal value is 5 times the image standard deviation.
- **Nebula**—minimal value will be set to the value of maximal number of pixels in the image. Maximal value is calculated as minimal value plus the value of 95 % of all pixels in the image.
- **Keep current**—newly downloaded image will be displayed in the same stretch values as the currently selected image.

> Hint:
>
> This feature is especially important when recording a video of some planet. It is important not to adjust stretch and gamma values among individual frames. The observer optimizes these values on single frame and then records a video with the defined stretch parameters.

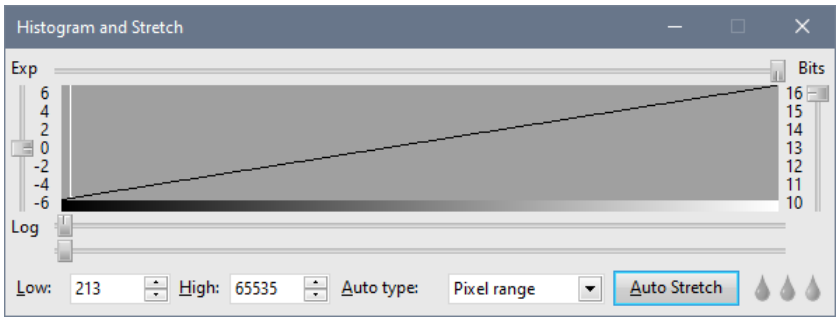
Now it is clear why the Histogram and Image Stretching is combined into one tool. The stretch low limit, representing image background, should be very often set to histogram peak value. It is always easier to do this if you see the histogram, of course.

Image Stretching

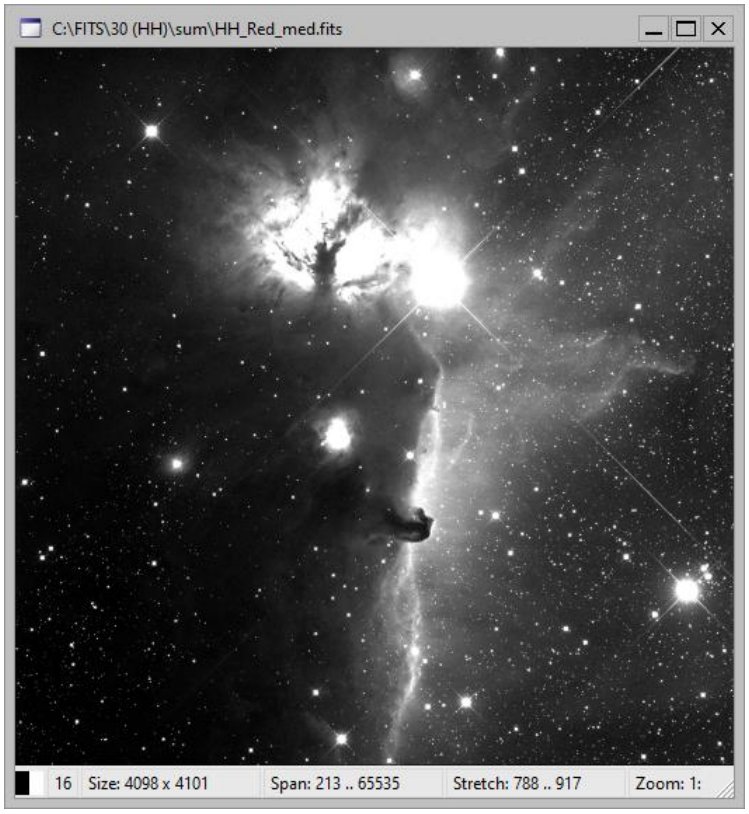
No astronomical picture has perfectly black background, regardless if taken under the darkest sky without Moon glows, not mentioning the light-polluted sky of sub-urban areas. Also, the camera itself generates some base level of counts representing "black" value. Photographers using classical films know it very well—the image after a long exposure is not black and white, but rather slightly darker gray and slightly lighter gray instead.

Consider the image of the "Horse Head" and "Flame" nebulae in Orion below. When the 65,535 levels of image dynamic range are only shrunk to 256 levels of gray on the computer screen, no details are visible, only the brightest stars dominate the field:

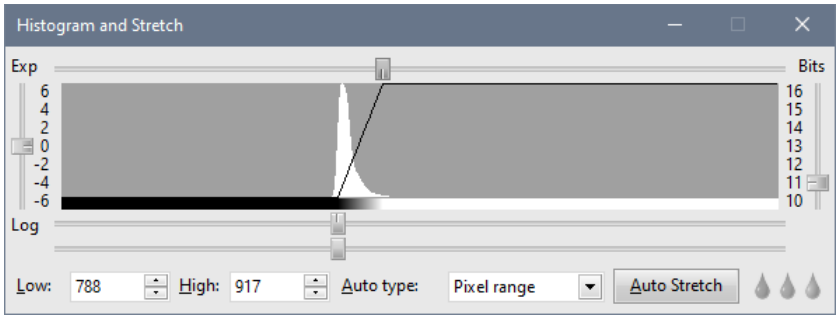
The corresponding setting of the stretch limits are 213 ADU for lower limit and 65,635 ADU for upper limit:



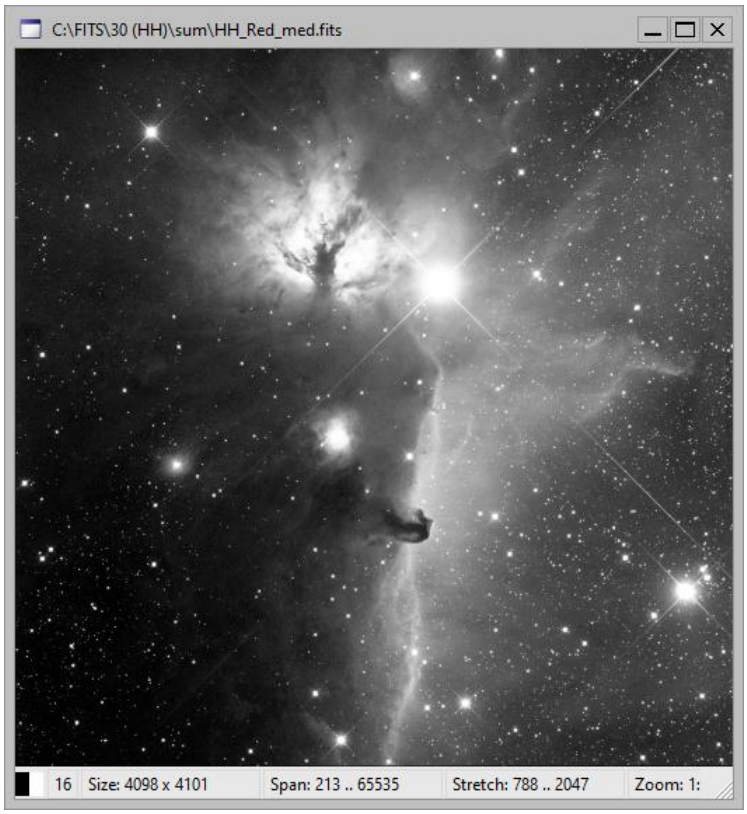
However, if we accordingly shrink the dynamic range of the original image, which will be transformed into 256 levels of gray of the screen, many more details emerge:



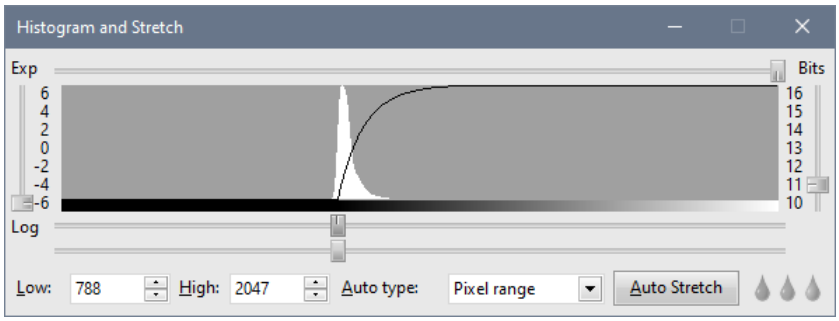
The displayed image takes only values from 788 to 917 ADU.



All original pixels below 788 are displayed black and pixels with value above 971 are white. Such aggressive stretch (only pixels within the range of 129 ADU are stretched to 256 levels of gray on the screen) highlight very dim details in the nebula, but at the same time the brightest portion of the Flame nebula is completely white, without any structure visible. The Histogram and Stretch tool offers a simple cure to this problem—non-linear stretching.



Non-linear stretching allows to more aggressively brightening of the dim details, while the bright portion of the image is stretched much slower. The image above is stretched from 788 to 2074 ADU, only the transformation is not linear, but logarithmic.



Remark:

Note the dynamic range compression, similar to simple stretch shown here, is an inevitable part of the image processing leading to beautiful images of deep sky objects, showing the faint details at the very dim parts of the image (as for instance outer spiral arms of the galaxy) as well as in the very bright portions (for instance galaxy core). But such dynamic range compression is much more aggressive compared to simple non-linear stretching shown here.

It is worth emphasizing, the image stretching done in Histogram and Stretch tool affect only the way FITS images are displayed. Altering of the stretch low and high stretch values does no modify the original 16- or 32-bit data.

The only exception is exporting image to 8-bit formats. In this case the newly created image in the chosen common image format (JPEG, PNG, …) brightens scale corresponds to the brightness scale displayed on the computer screen (and not only brightness scale, but also selected color palette is used to create common 8-bit images during export.

Histogram and Stretching of Color Images, balancing Colors

Almost no astronomical image contains proportionally represented colors. Color intensity is affected by many factors—for instance detectors are typically less sensitive to blue light and blue filters are less transparent than red filters, so the blue component is typically much less intensive than the red one. It is quite difficult and time consuming to compensate the different sensitivity by prolonging of the blue image exposures. More efficient way to balance colors is brightening

of less intensive colors (or darkening of more intensive colors) to achieve color-neutral appearance of white and/or gray colors.

Automatic white balancing can be relatively easy on normal images, where all colors are represented approximately uniformly. But this is almost impossible on images of deep-space objects. For instance, consider the image of emission nebula, dominated by deep-red hydrogen alpha lines—any attempts to lighten green and blue color to create color-neutral image result to totally wrong color representation. Astronomical images are usually color balanced manually.

As already described in the previous chapter, image can be visually brightened by altering its stretch limits. SIPS Histogram and Stretch tool displays and enables altering of stretching curve limits and shape for red, green, and blue color individually.

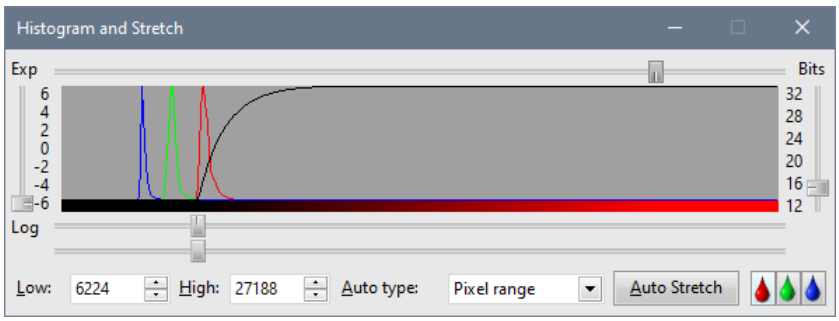


Image Info

The **Image Info** tool provides statistical information about currently selected image and pixel information of the image currently under mouse cursor.

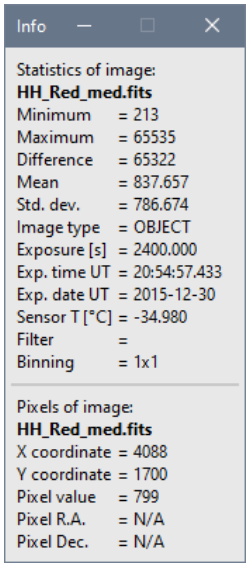


Image statistics provides following information:

- Minimum—value of lowest pixel in image
- Maximum—value of highest pixels in image
- Difference—difference between maximal and minimal value
- Mean—average of all pixels in image
- Std. dev.—standard deviation (RMS) of pixel values
- Image type—type of image (OBJECT, DARK, FLAT, BIAS)
- Exposure—exposure time of image in seconds
- Exp. time UT—UT time of exposure begin
- Exp. date UT—UT date of exposure begin
- Sensor T—temperature of the sensor in °C
- Filter—name of filter used for exposure

Pixels information provides coordinates and value of the pixel currently under mouse cursor.

- X coordinate—horizontal (x) coordinate of the mouse cursor
- Y coordinate—vertical (y) coordinate of the mouse cursor
- Pixel value—value of the pixel currently under mouse cursor
- Pixel R.A.—Right Ascension of the pixel currently under mouse cursor
- Pixel Dec.—Declination of the pixel currently under mouse cursor

Pixel value of color images is calculated as average of red, green, and blue pixel under the mouse cursor. Values of individual colors are displayed in the pixel pop-up window, described in the **Images within SIPS Workspace** chapter.

The Right Ascension and Declination are displayed only if the image was solved (see the **Astrometry** tool description).

The displayed statistical information is calculated from the whole image if no part of the image is selected (framed). When a selection is active (see the **SIPS Images** chapter), statistical information is calculated from the selected area only. This can be useful for example when checking whether a star saturates—just select a small box around the star and Maximum value in the Image Infor tool shows the greatest pixel value etc.

Remark:

Note that displayed image statistics is related to the currently selected image, but the pixel information is related to image currently under mouse cursor. This is why the names of selected image and image under mouse cursor is displayed in bold font in the Image Info tool.

Calibration

The Calibration tool allows both simple and complex image calibration. The window contains number of controls, allowing both subtraction of dark frame and applying of flat field on single image or on whole image list. Proper calibration of images taken by dual-gain cameras, where resulting HDR image is created from low-gain and high-gain frames, is also supported.

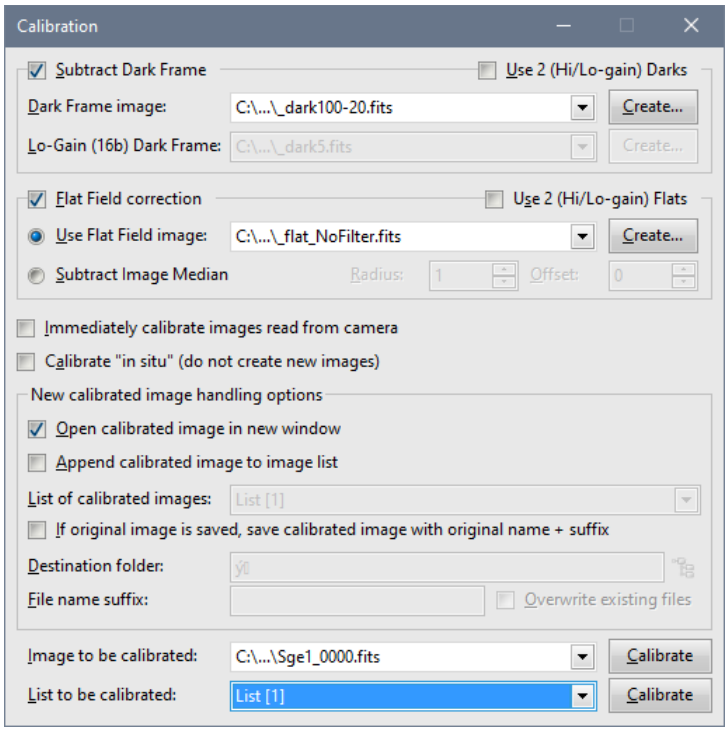


Image calibration explained

Image immediately downloaded from the camera is called raw image. It is sometimes surprising how aesthetically unpleasant raw images are, especially when compared to fully processed images, which appear in magazines and on web sites. Image processing can eliminate hot or dark pixels, remove unwanted gradients, reduce noise, sharpen image, enhance details, compress image dynamic range etc.

Such image processing can make images more beautiful, but it changes information contained in the image. It can be performed with images intended for publication, but it eliminates the possibility to gather reliable scientific data from the image (measure brightness, position, etc.). Still there are some image processing, which together with enhancing the image appearance also enhances the scientific value of raw images instead of decreasing it—image calibration. It is almost necessary to perform calibration with every raw image.

Depending on the camera, telescope (or objective lens) and object imaged, the calibration can be more or less complex. Some setups even do not require the calibration at all.

Image calibration basically consists of two steps:

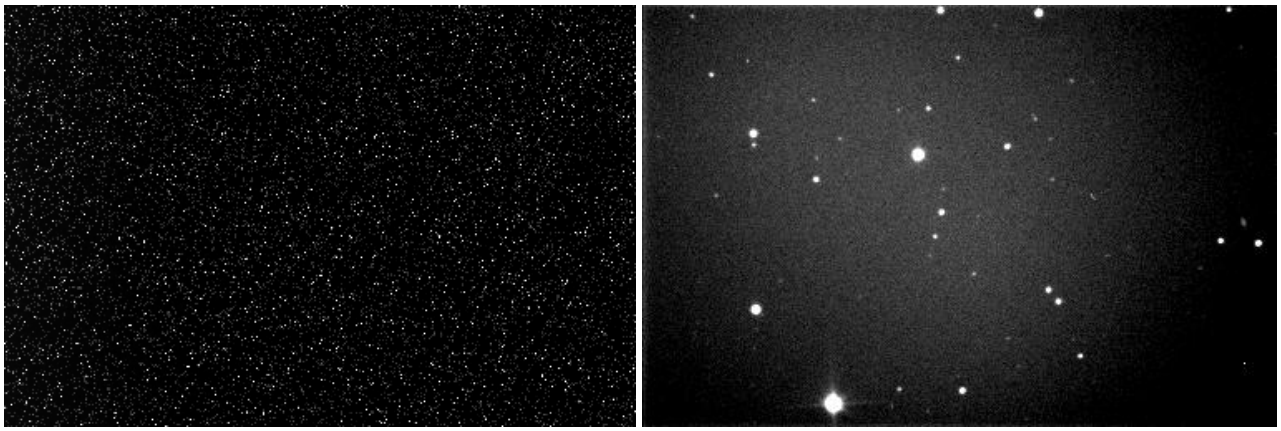
- **dark frame** subtraction.
- applying **flat field**

Dark frame

Raw image is affected by thermal noise and uneven field illumination:

The purpose of dark frame subtraction was already explained—elimination (or at last reduction) of thermal noise. Sensor dark current is proportional to temperature. Thermal noise doubles every 6 or 7 °C, depending on the sensor architecture. The charge accumulated in pixels is also proportional to exposure time (dark current is expressed in electrons per pixel per second at the defined temperature). To reduce image thermal noise, the dark frame subtracted from image should be obtained at the same temperature for the same time as is the image itself.

Dark frame corresponding to the raw image above (left) and the result after its subtraction (right):



If the dark current accumulate in pixels proportionally to the exposure time, it is possible to calculate dark frame from other dark frames taken with different exposures, at last in theory. However, procedures relying on this fact, created in the era of CCD sensors, fail with modern CMOS sensors. CMOS sensors typically keep the basic offset at predefined register value regardless of the exposure time and sensor temperature and any attempts to interpolate or extrapolate dark frames fail. Also, acquiring proper dark frame with the same exposure time and at the same sensor temperature like the light image always led to better results, even in the era of CCD sensors.

Remark:

Dependency of dark current on temperature is the reason, why cooled cameras need regulated cooling. If the camera electronics can keep the sensor temperature within a fraction of °C, it is possible to use dark frames taken at the same temperature as light images.

Let us also note that taking dark frames requires closing of the mechanical shutter (so-called electronic shutter, a property of CMOS sensors as well as FT and IT CCD sensors, does not work for dark frames). It is necessary to cover the telescope aperture (if the telescope has a truss tube, covering it can be a challenging task) by the observer every time dark frame is to be obtained. So, the unattended robotic observation is no option for cameras without mechanical shutter.

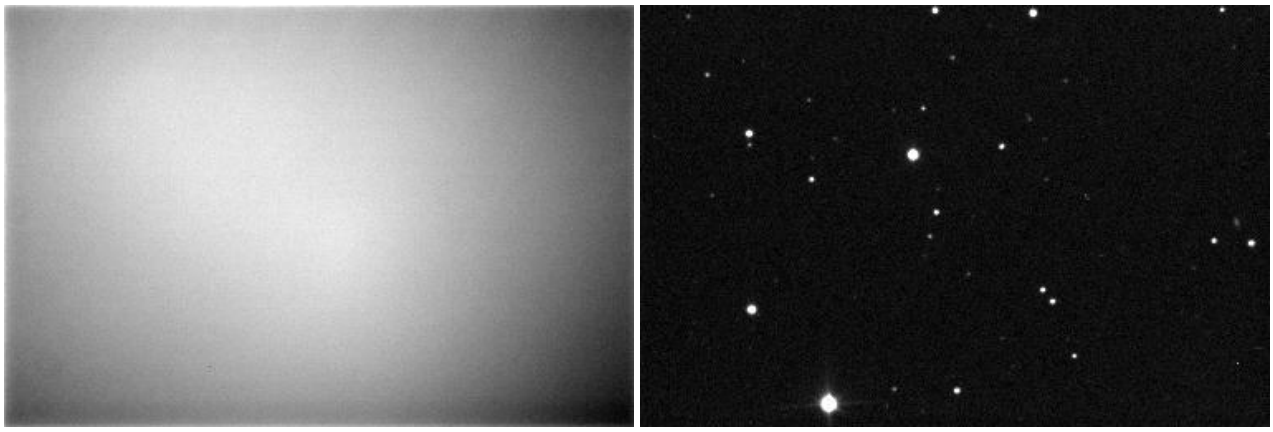
Flat field

Telescope field of view is often illuminated non-uniformly—image intensity on the borders can be lower than in the center because of the input aperture limitations, smaller telescope secondary mirror etc. What is more, dust particles on filter or camera front optical window creates ring-like shades etc.

Also, the response to illumination can be different for different pixels in the sensor array. Uniformly glowing sky background then appears as more or less bright pixels.

All these effects alter image brightness and cause not only aesthetics artifacts, but also reduces measurement precision. It is possible to eliminate these effects by applying flat field image.

Flat field corresponding to the raw image above (left) and the result after flat field correction (right):



Flat field image is an image of uniformly illuminated area. Thus, all image brightness variations on flat field are caused by telescope or camera, not by the object we image. Ideal flat field values are around one half of the image scale (average pixel count should be around 33,000 ADU for 16-bit cameras).

Applying flat field means dividing every image pixel with the appropriate pixel of the flat field. Image pixels, brighter due to telescope or camera non uniformity, are divided by greater flat field value, also brighter due to same reasons. But such division changes image scale so we also multiple each pixel by flat field average pixel value. If the operation is performed on integer values, multiplication must of course precede division, else the precision lost during integer division would destroy the image.

Remark:

The flat fields must be obtained for the current telescope setup. It is not possible to save flat fields and reuse them later, for instance after we move the camera to different telescope—dust particles pattern on the camera window will be different and other telescope has different field illumination etc.

Also note that it is necessary to take flat field for every filter we use for the same reasons.

Obtaining flat field images

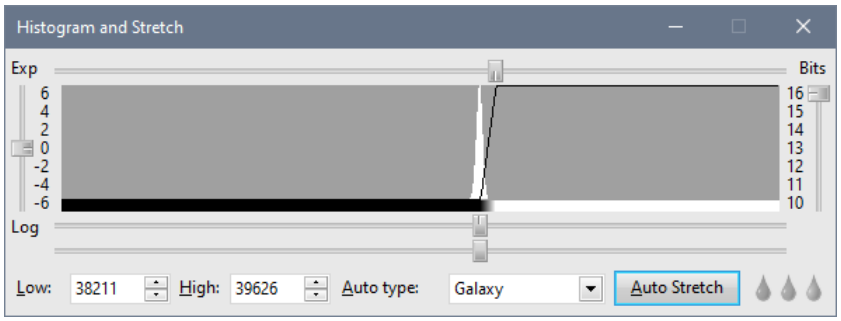
The question is how to obtain flat field images? One way of exposing flat fields is using uniformly illuminated flat surface. Such surface, located for instance inside of the observatory dome, must be illuminated by multiple light sources, possibly covered by dim glass etc.

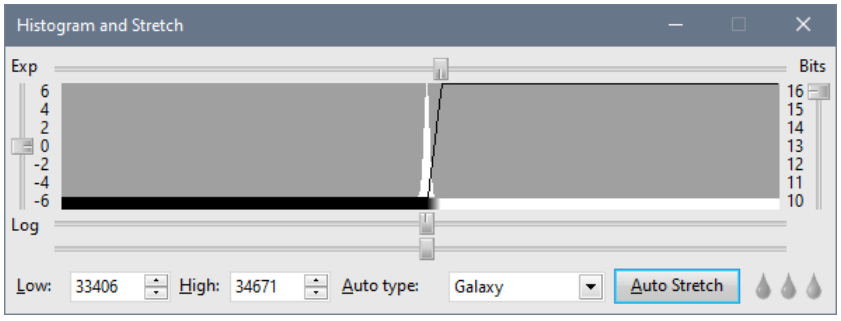
Image below shows an example of a flat field panel located in the observatory dome. The panel is illuminated by four LED sources, not shown in the image.

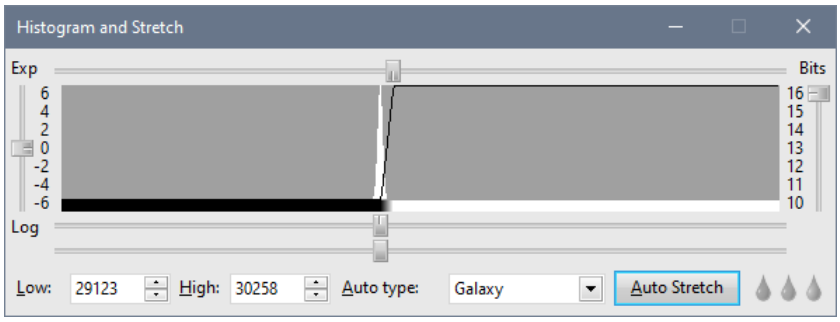
Another way of getting flat fields is to use light of the twilight or dawn sky. Especially for the narrow field of view the sky brightness gradient is not important. An empiric rule says the most uniform sky without gradients is approximately 120° from the direction to the Sun, already below the horizon. But there is a chance some star appears in the field of view, which is not allowed for the flat field. This can be eliminated by shooting multiple images while the telescope moves to nearby, but different field. Performing a median combination eliminates possible star images from the flat field.

It is important to consider the fact, that the sky brightness changes quite rapidly during dust or dawn. This means the mean value of a series of flats taken on the sky differs among individual frames. This effect is more prominent with longer exposures and slower downloads, typical for older CCD cameras relying on mechanical shutter to perform exposures. Modern CMOS cameras allow for very short exposures and very fast downloads, so the flat field series is acquired much faster. Still, the differences among mean values can eliminate the effect of median combination used to create master-flat image, a result of median combination would be virtually equal to the image taken in the middle of the series as its pixels are in the middle of the brightness scale.

Consider the histogram of the first flat image taken during dusk, compared to the histogram of the image taken in the middle of the series and the last flat image.







The first flat mean value is about 38,000 ADU, while the mean value of the last flat frame, exposed literally seconds later, is only around 29,000 ADU. So, when median-combining the master flat image, it is very important to level the mean values of individual frames first.

Flat fields are images like normal light images, so do not forget to calibrate them using corresponding dark frames prior to processing them into a master-flat image!

Creation of Calibration Images

It is generally useful to create calibration images (dark frame and flat field) as a combination of multiple images. Combining of multiple images eliminates (or at last reduces) unwanted artifacts like cosmic ray traces etc. It is desired to combine images taken with the same exposure time, at the same sensor temperature. Also binning, read mode and other parameters affecting exposure must correspond to object image.

The question is which kind of combination should be used? Three possibilities appear:

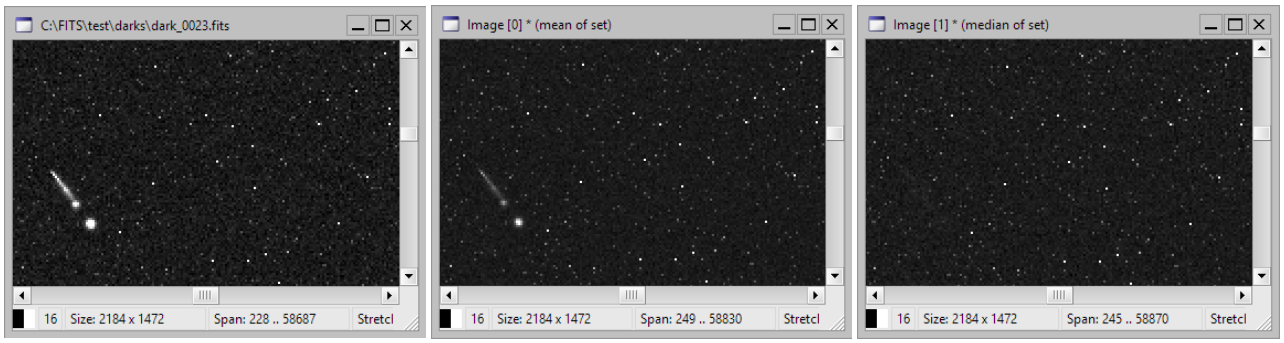
- **Mean**
  Mean of multiple images eliminates the random noise. But random artifacts, e.g., the cosmic ray trace, affects the result and introduces the error.
- **Median**
  Median operation sorts corresponding pixels of all images and picks the one, which lies in the middle of the sorted vector. This means random extremes do not affect the median (if the particular pixel on some image is the brightest one, it is on the top of the pixel vector and it is not important how bright it really is).
  But median lacks the advantage of averaging—imagine altering lower and higher values. Average will be in the middle of these values, but median will be either the lower or the higher value.
- **Mean of Median Half**
  This operation combines the positive features of both mean and median. Corresponding pixels on individual images (pixels with the same coordinates) are sorted. The first quarter and the last quarter of the sorted vector is omitted, which eliminates random extreme values. The resulting pixel value is calculated as an average of the middle half of the sorted vector.
  Averaging somewhat eliminates random noise on the one side, but the average is not affected by extremes (random artifacts, e.g. the cosmic ray trace).

Which method should be used for creation of which kind of calibration images? Simple mean is the best choice if you have very small number of images—for instance just two or three.

Using the Median of Median Half is probably the best choice for relatively small image sets of about 10 or 20 images. It eliminates extreme values caused for instance by radiation events, and somewhat eliminates random read errors.

Simple Median works perfectly on large image sets, like 30 and more images. For such sets Median combination is virtually indistinguishable from Average of Median Half. Effect of Median versus Average is demonstrated on the following three images.

Single frame dark frame (left), average of all 3 frames (middle), median of these 3 frames (right):



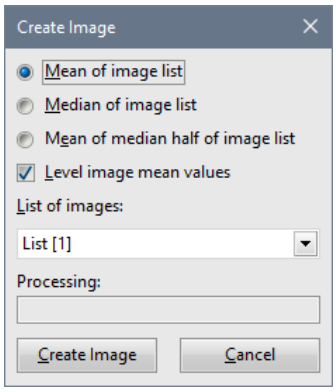
The left-most image above shows a typical radiation event trace on a single dark frame exposure. The image in the middle is a mean of 40 dark frames. Although the radiation event trace is dimmed by the averaging, it is still clearly visible. Notice the false hot pixel on the averaged image—a relic of another radiation event on another frame used for averaging. The median-combined dark frame on the fight perfectly represents the sensor noise pattern without any random effect.

Dark frame subtraction in SIPS

The first text-box contains controls allowing dark frame subtraction. The text-box title **Subtract Dark Frame** itself is a checkbox, which allows switching dark frame subtraction on and off.

The dark frame can be selected from the **Dark Frame image** combo-box. Note this combo-box contains all currently opened images (displayed in image windows and present in image lists).

Although the **Image Math and Filters** tool can be used to create median or average of multiple images, the **Calibration** tool provides direct way to perform these operations—just click the **Create** button. The following dialog box appears:



Select image list and create new image as mean (average), median or mean of median half of all images in the list.

Level image mean values check box only affects median combination. If this option is checked, every image is multiplied by a number to ensure a mean value of all combined images is the same.

It is recommended to keep this option checked when we create master flat field image else the different mean value of individual images can affect the median computation. On the other side, this option should remain un-checked when creating master dark frame.

Flat field correction in SIPS

The second text-box contains controls allowing applying of flat field. The text-box title **Flat Field Correction** itself is a checkbox, which allows switching on and off.

SIPS offers two ways of flat field corrections:

- With an actual master-flat field image
- With an artificial flat-field image, calculated as median of a box surrounding each pixel

The method is selected by two radio-buttons, acting also as labels for controls related to particular method.

Using master-flat image

The flat field image can be selected from the **Use Flat Field** combo-box. Note this combo-box contains all opened images (displayed in image windows and present in image sets). Also, the flat field image can be instantaneously created the same way like the dark frame image from multiple images in the list as an average or median (using the **Create** button to the right of the flat frame selection combo-box).

Using artificial (box median) master-flat image

If the flat field is not available and still the light images are strongly affected by uneven illumination, and artificial flat field can be created for every image. Such artificial flat field is created using median filter on a box of pixels surrounding every pixel in the image.

> Remark:
>
> Note the median filter used to create artificial flat field is different from median combination used to create master dark or master flat images from a list of individual exposures. In this case the median is not calculated from corresponding pixels on multiple images, but from a box of pixels, surrounding every pixel of image.

Median of a box of pixels represents a value in the middle of a series of pixels, belonging to the box, sorted by their values. It is supposed that most pixels in every box contains background. Only a minority of pixels contain stars and the median operation eliminates these pixels.

The box size for median calculation is a parameter entered in the **Radius** parameter. It is also possible to add a constant **Offset** to every pixel.

Note the same operation (calculation a median from a box of pixels) is available in the **Image Math and Filters** tool. It is recommended to use this tool to visualize the result of this operation to choose the best value for the Radius parameter.

Also note the median filter calculation if time-demanding operation. Using a fast PC with many processor cores significantly speeds up the calibration.

A note about (not) using Bias images for calibration

First, let us emphasize that the still occasionally used method of dark frame calibration, including subtracting of bias frames from both dark and light images, and only then subtracting of results, cannot be used with CMOS sensors, as these sensors actively adjust the “zero” offset to keep it on predefined values.

> Remark:
>
> This is also the reason why the methods used to measure the dark current of CCD sensors fail if applied to CMOS sensors, producing nonsense values.

Dividing the dark frame into “bias” and “dark—bias” frames was sometimes used to enable scaling of the “dark—bias” frame with exposure time. However, this method is only approximation as the dark current is influenced also by other factors than the reported sensor temperature, like the actual cooling performance, environment temperature affecting temperature gradients etc.

So, using biases during calibration in fact replaces the very simple operation:

Calibrated = Raw—Dark

With much more complex operation:

Calibrated = (Raw - Bias) - (Dark - Bias)
= Raw - Bias - Dark + Bias
= Raw - Dark

With exactly the same results.

Also, very important fact is that every additive operation increases the noise by square root of sum of squares of noise of both values.

$$\sigma_{result} = \sqrt{\sigma_{frame1}^2 + \sigma_{frame2}^2}$$

If both frames have the same noise, adding or subtracting them increases the noise by square root of two or approx. 1.41-times. So, naturally, the lowest number of such additive operations are used during calibration process the better.

The best way to perform dark frame calibration is to capture the set of dark frames at the same sensor temperature (ideally also at the same environment temperature) and with the same exposure time as the light frames and then to median-combine them into a master dark frame. The resulting master dark, is then subtracted from every light image with single additive operation. Both light image and master dark frame includes bias and the dark frame subtraction removes it from the light image together with the dark current. Leave bias frames for sensor performance evaluation etc., but completely avoid them during calibration process.

> Remark:
>
> The same procedure should be applied when capturing flat fields. It is recommended to capture a set of dark frames, corresponding to a flat frame exposure time and at the same sensor temperature and then to median-combine them. Resulting master dark frame for flats should be subtracted from each flat image prior me median-combining them to master flat frame.

Naturally, the read mode used for respective flat field frames must be also used to capture also the corresponding dark frames.

Calibration of dual-gain camera images

This chapter concerns calibration of images acquired with CMOS sensors employing 12-bit ADCs (Analog to Digital Converters) only, but using two sets of ADCs inside the sensor, each capable to digitize the image with different gain—one set of ADCs uses low-gain channel, while the second set uses high-gain channel (for instance the GSENSE4040 sensors, used in the C4-16000 cameras). Both 12-bit outputs are read in parallel for every exposure and can be combined into single image with true 16-bit dynamic range (such combined image is often called 16-bit HDR for High Dynamic Range).

16-bit HDR images, created from two 12-bit ones, are virtually indistinguishable from the true 16-bit ones. Differences between HDR combined image and true 16-bit image in both visual appearance (appreciated in aesthetic astro-photography) and information contents (important for research application) are beyond divergences caused by other sources.

But the fact, that every pixel of the resulting 16-bit HDR image originates either from high-gain frame or is a transformed low-gain frame pixel, poses a problem for proper image calibration. The naïve approach—just taking 16-bit HDR dark and 16-bit HDR flat and applying the to 16-bit HDR light image—is fundamentally flawed. In fact, such “calibration” can do more harm than good for image quality.

But let us start with a brief introduction of an algorithm used to combine low-gain and high-gain images into HDR one.

HDR image construction

Algorithm used to create a 16-bit HDR image is quite straightforward. The following steps are performed for every pixel:

1. If the **high-gain** image pixel **is less or equal** to a defined **threshold**, the high-gain pixel is moved to resulting image without any changes. The corresponding low-gain image pixel is not used, as the high-gain one has much better S/N ratio.
2. Else the high-gain image pixel is **above the threshold**, the **low-gain** image pixel is **transformed** to correspond to the high-gain image gain and offset and then it is put into resulting image. Transformation enlarges the low-gain pixel value from the 0 to 4095 range to full 0 to 65535 range. The corresponding high-gain image pixel is not used.

The algorithm written in pseudo-code would be:

```
for x = 0 to ImageWidth - 1 do
  for y = 0 to ImageDepth - 1 do
    if HiGainImage[ x, y ] <= Threshold then
      ResultImage[ x, y ] = HiGainImage[ x, y ];
    else
      ResultImage[ x, y ] = Transform( LoGainImage[ x, y ] );
   end_if;
  end_for;
```

```
end_for;
```

The value of **Threshold** is arbitrary set somewhere close to the upper limit of the 12-bit image dynamic range. For instance, the C4-16000 drivers use Threshold equal to 3600. This sufficiently uses the perfectly linear portion of the high-gain image dynamic range. Values close to saturation signal, and thus slightly diverging from linear response curve, are cut off.

The **Transform** function coefficients slightly vary among individual sensors. The differences are rather minor, but every C4-16000 camera undergoes individual calibration and coefficients are stored into camera permanent memory. Driver performing the HDR combination reads these values from connected camera and uses them to perform HDR combination.

Advanced Calibration

The default 16-bit HDR read mode of the C4-16000 camera may lead some users to use this camera as any other camera with proper 16-bit conversion. Unfortunately, the fact that the 16-bit HDR image is combined from two independently digitized frames, causes such naïve approach cannot be used also for the image calibration. For instance, consider this:

- The dark frame signal is generated by dark current only, which is quite low (remember, the C4-16000 camera uses sensor cooling to significantly lower dark current). So, the dark frame pixels almost never cross the 3600 ADU threshold point. The result is that the 16-bit HDR dark frames virtually contain high-gain frame pixels only. But if the light image exceeds the 3600 ADU threshold, its pixels are transformed low-gain frame pixels. When subtracting 16-bit HDR dark frame from 16-bit HDR light image of some bright object, we in fact subtract high-gain dark frame from the transformed low-gain image pixels with entirely different dark current, different hot pixels etc.
- The common practice is to make flat fields reaching approximately a half of the sensor dynamic range, which is somewhere between 30000 and 35000 ADU for 16-bit image. All pixels of such 16-bit HDR flat field image are then transformed low-gain pixels, but typical astronomical image often contains many dim portions, remaining below the 3600 ADU threshold and thus taken from high-gain frame. Then we apply flat field created from low-gain frame to image containing majority of high-gain pixels with different response to light etc.

Remark:

The fact, that the GSENSE4040 sensor consists of four quadrants, differing in bias value, dark current and response to light, makes the proper calibration crucially important to eliminate these differences.

Solution to the above-mentioned problems in calibration is rather simple in principle—it is necessary to use dark and flat field frames taken through high-gain channel when the light image pixels is below threshold and similarly, dark, and flat field frames taken through low-gain channel, but transformed the same way like the light image pixel in the process of creating 16-bit HDR image, if its value is above the threshold.

So, **no dark frame nor flat field frame could be HDR combined (16-bit)**. It is necessary to create **two master dark frames**:

- The **first master dark** frame is acquired through high-gain channel. This dark frame will be used when the 16-bit HDR raw image pixels does not reach the threshold value.
- The **second master dark** frame is read using low-gain channel, but transformed the same way like the low-gain channel is transformed during HDR combination. This is why the C4-16000 camera driver offers read mode No.4—**16-bit transformed Low-Gain**. This dark frame will be used when the 16-bit HDR raw image pixels value exceeds the threshold value.

Also, **two master flat field** frames are necessary:

- The **first master flat field** frame is acquired through high-gain channel.
- The **second master flat dark** frame is read using **16-bit transformed Low-Gain** read mode.

Remark:

The 16-bit transformed Low-Gain read mode is marked simply as **LoGain "16b"** by the camera driver for space reasons. In fact, this is 12-bit Low-Gain mode, but all pixels are transformed to 16-bit dynamic range the same way like the Low-Gain channel is transformed in HDR image combination when the High-Gain pixel exceeds the

Threshold limit. But in this case, there is no threshold limit and no High-Gain pixels are used, all Low-Gain pixels are used and transformed to create this image.

Dual-gain image calibration

SIPS calibration tool is capable to perform dual-dark and dual-flat calibration.

High-Gain master dark frame

The camera should be switched to **HiGain 12b** read mode. Capture a set of dark frames of the same exposure time like the light images. Take care to:

- If multiple exposure times are used for light images, multiple dark frame sets should be captured, one set for each used exposure time.
- Ideally, the environment temperature should be the same or at last similar to the environment temperature during imaging due to temperature gradients affecting the actual sensor temperature.
- Leave the temperature to settle for 5 or 10 minutes prior to capturing dark frames, do not start immediately when the camera indicates the target temperature was reached.
- Do not capture dark frames immediately after flat of light frames. The GSENSE4040 sensor used in C4-16000 suffers from Residual Bulk Image effect. Let the sensor several tens of minutes to dissipate RBI before you capture dark frames.

Individual frames should be median combined into master dark frame. Using median combination eliminates radiation spikes and other artifacts in master dark image.

If using SIPS **Math and Filters** to perform median combination, uncheck the **Level mean values of all images** checkbox. This option is included to compensate different mean values of individual frames e.g. when taking flat fields on sky during twilight or dawn, when the sky brightens changes among exposures.

Low-Gain 16b master dark frame

The camera should be switched to **LoGain "16b"** read mode. As opposed to **LoGain 12b** mode, the 16-bit variant performs the HDR transformation with every pixel of the image, expanding the 12-bit dynamic range to 16-bits. Beside the read mode, the procedure is the same like in the case of High-Gain master dark frame.

High-Gain master flat field

Capture a set flat field image with a camera using **HiGain 12b** read mode. Corresponding dark frames, used to create master dark frame for this flat, should be captured in the same read mode.

If the flat field images are captured on the sky and a master flat field median combination is performed using SIPS **Math and Filters**, check the **Level mean values of all images** checkbox. This option compensates different mean values of individual frames, caused by changing sky brightens among exposures.

As the used read mode is 12-bit, so the mean value of acquired frames should be **between 2000 and 2500 ADU**.

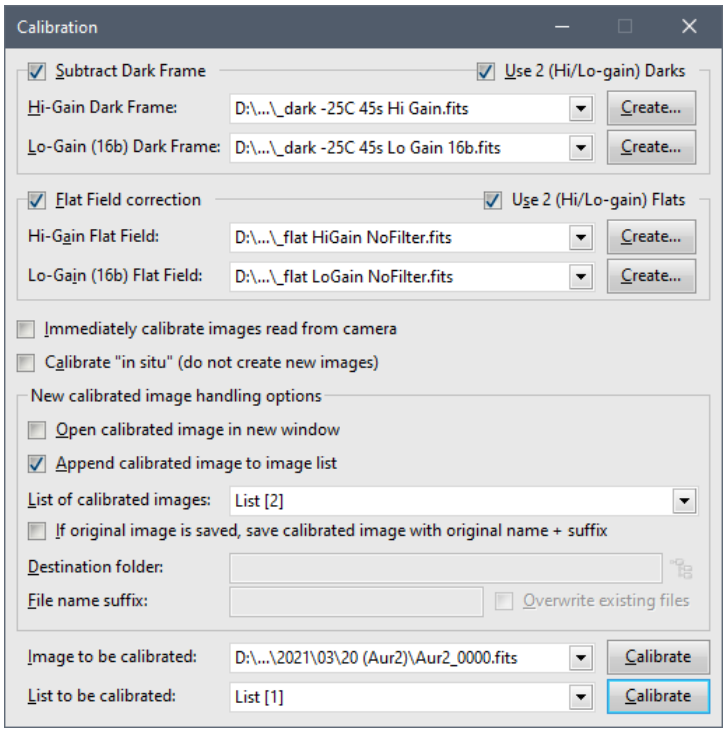
Low-Gain 16b master flat field

As in the case of Low-Gain 16-bit master dark frames, the camera should be switched to **LoGain "16b"** read mode, which is of course true also for corresponding dark frames.

While the low-gain frame is also 12-bit only, the used **LoGain "16b"** read mode transforms every pixel into 16-bit dynamic range. So, the mean value of acquired frames should be **around 33000 ADU**.

SIPS Calibration tool

When all 4 calibration frames are ready, calibration of any 16-bit HDR raw image or image set can be performed using the SIPS **Calibration** tool.



Warning:

The calibration of dual-gain cameras should always be performed at once. When calibrating single-gain raw images, regardless if 12 or 16-bit deep, it is possible to subtract dark frame from raw images and to apply flat field to these intermediate images anytime later.

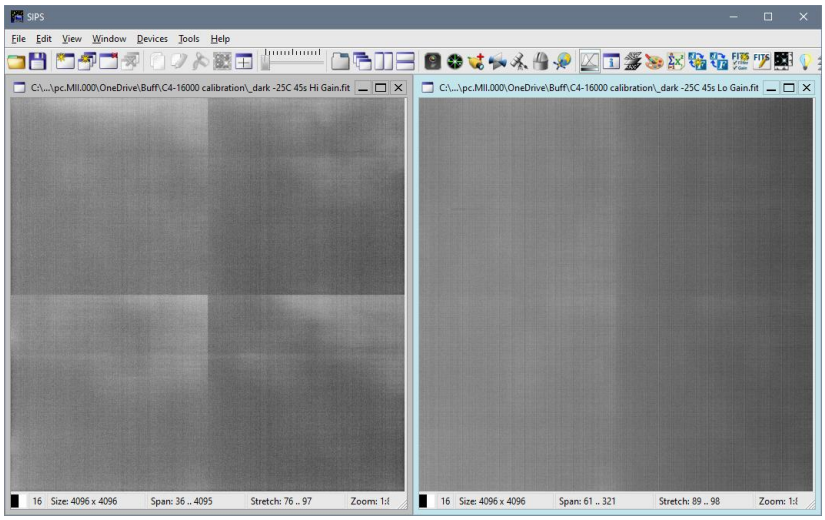
But as described above, the advanced calibration of dual-gain camera depends on testing of the threshold. But subtracting of the dark frame always lowers the pixel values. So, even pixels created by transformation of the low-gain ones during HDR combination, may be below the threshold value after dark frame subtraction. This would lead to using of high-gain flat field instead of transformed low-gain one.

So, if both dark frame and flat field calibration are to be performed on 16-bit HDR raw frames, always perform the calibration in one step. SIPS calibration code handles this situation correctly.

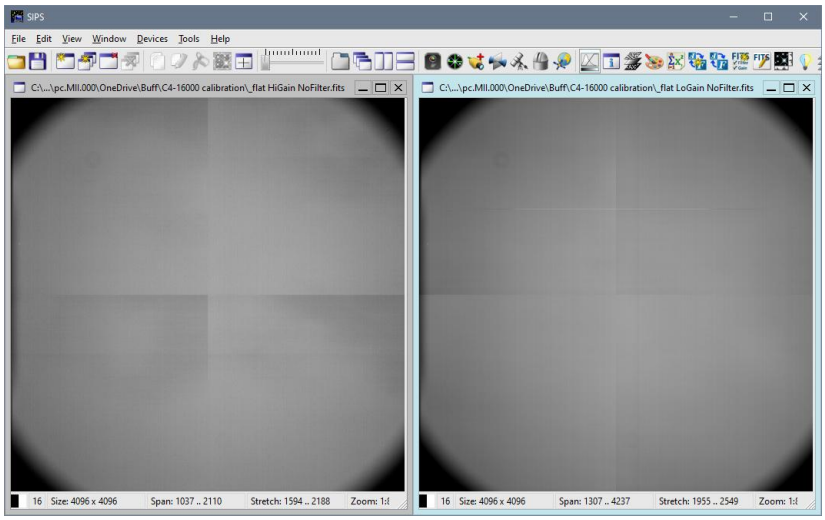
To perform advanced calibration, simply check respective checkboxes **Use 2 (Hi/Lo-gain) Darks** and **Use 2 (Hi/Lo-gain) Flats**. Then select respective calibration frames and choose other options in the SIPS Calibration tool the same way like in the case of standard calibration.

Example calibration frames

The sample calibration frames, showed below, clearly demonstrate differences in individual quadrants of the GSENSE4040 sensor.

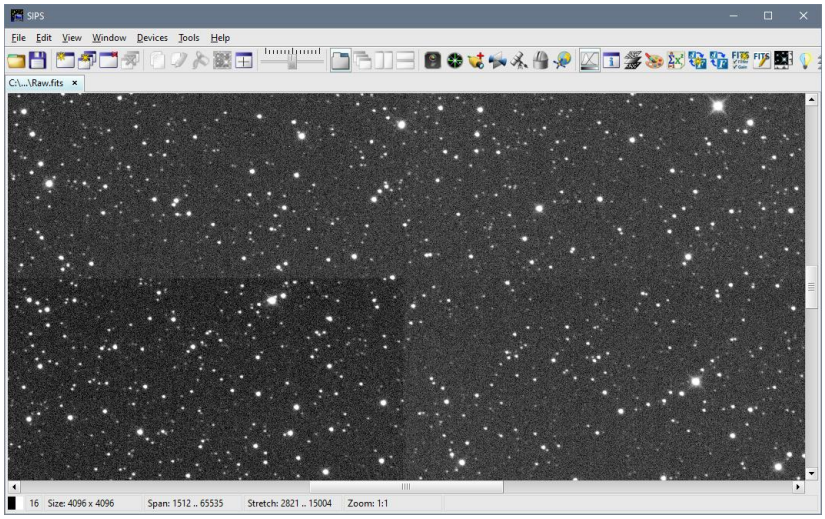


The image above shows an example high gain (left) and low-gain(right) dark frames.

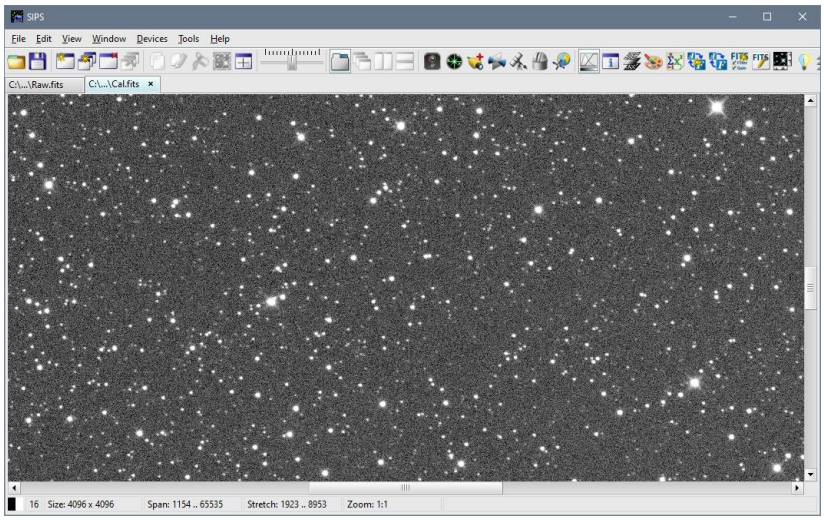


Similarly, the high gain (left) and low-gain(right) flat field frames are shown.

The raw and calibrated image example blow are only cropped section around image center of real series of 45s long exposures of a star field.



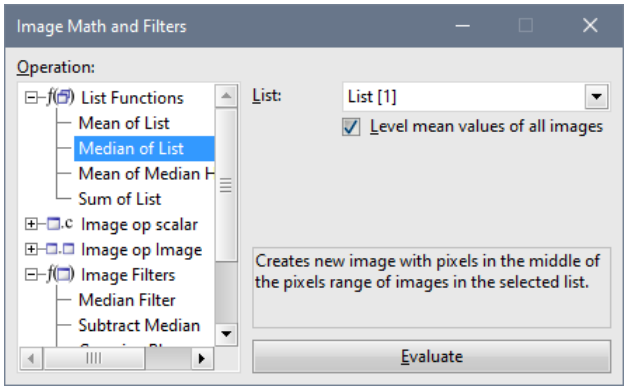
Single highly stretched 16-bit HDR raw image shows non-uniform response of GSENSE4040 sensor.



Single highly stretched 16-bit HDR image, calibrated using the SIPS Advanced Calibration tool with four calibration frames, virtually eliminates any traces of the GSENSE4040 non-uniformity.

Image Math and Filters

The **Image Math and Filters** tool provides various mathematical operations on images vs. scalars, on images vs. images or on image lists etc.



The Image Math window offers image operations on the left-side tree and operation parameters on the right-side pane. Individual operations are divided into categories according to number and type of operands:

- $f(\square)$ Operations performed on whole image list:
  - Mean of list—all images in one set are averaged.
  - Median of list—corresponding pixels on individual images (pixels with the same coordinates) are sorted and the pixel value in the middle of the sorted vector is selected to create resulting image.

    Median (as well as Mean of Median Half) operation offers leveling of average values of all images as option. Because median is calculated as middle value of sorted pixel values, leveling the mean value of all images is very important. If for instance images differ only in the bias level, median computation is negatively affected by the bias.

  - Mean of Median Half of list—corresponding pixels on individual images (pixels with the same coordinates) are sorted. The first quarter and the last quarter of the sorted vector is omitted and the resulting pixel value is calculated as an average of the middle half of the sorted vector.

    This operation combines positive features of both Average and Median operations. It somewhat eliminates random noise by averaging of multiple samples. But the average is not affected by extremes (random artifacts, e.g. the cosmic ray trace) like the median.

  - Sum of list—all images in one set are added.

    These operations are performed on pixel-to-pixel basic, without any image alignment. They are appropriate e.g. for creation of dark frames of flat field images, not for stacking of multiple object exposures. Use the **Combine Monochrome Images** Tool to stack object exposures with sub-pixel alignment.

- $\square.c$ Operations between image and scalar value:
  - Add—add a number to every image pixel.
  - Subtract—subtract a number from every image pixel.
  - Multiply—multiply every image pixel by a number.
  - Divide—divide every image pixel by a number.
  - Cutoff—replace every pixel above defined threshold with the threshold value.
- $\square.\square$ Operations between two images:
  - Add—add two images.
  - Subtract—subtract two images.
  - Interpolate—interpolate two images. If the interpolation ratio is between 0 and 1 then every pixel of the new image will be a linear interpolation between pixels of the two images. Ratio less than 0 or greater than 1 cause extrapolation of pixel values.

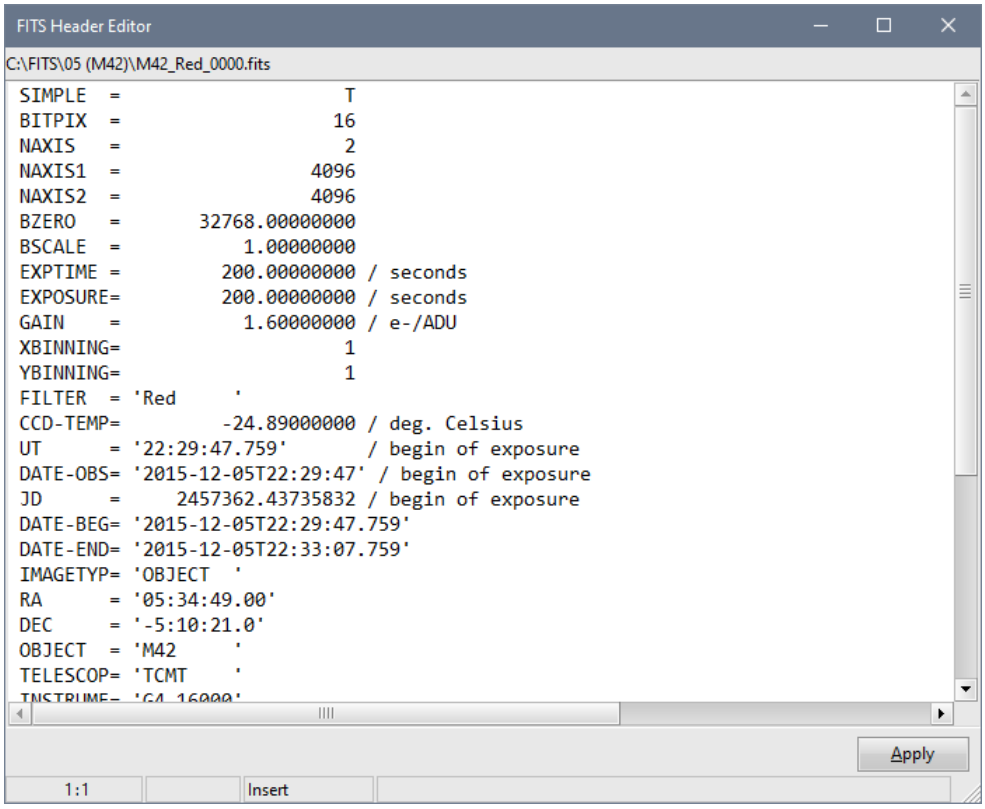
-  Operations on simple image:
    - Median filter—replaces all pixels by the median value of surrounding pixels, defined by the radius parameter. This filter removes all brightness changes smaller than the defined radius (e.g. individual stars). Only the background features greater than radius remains (e.g. nebulosity or galaxy background).
    - Subtract median—calculates median image and subtracts it from the image. The offset parameter ensures no pixel value will be clipped to zero if the median value is greater than original value. Effect of this filter is inverse to median filter—all features greater than radius will be removed (e.g. gradient, nebulosity, etc.) and only stars remain.
    - Gaussian blur—blurs the image by the Gaussian kernel filter of defined radius. This operation will soften the image, eliminating noise and sharp edges.
    - Gaussian unsharp mask—subtracts image filtered by Gaussian blur from the image. This operation will sharpen image, enhancing edges but also the noise.
    - Remove extreme pixels—calculates the standard deviation from 3 × 3 pixel neighborhood of each pixel and replaces the center pixel with average of surrounding pixels if it exceeds the average by a defined multiply of standard deviations.

      This operation can be used to eliminate dark (black) and hot (white) pixels from the image (e.g. if the dark frame for the particular image is not available to eliminate hot pixels by its subtraction). The default value of standard deviation (RMS) multiply is 1.5, but you can experiment with this number to achieve optimal results for the image.

Clicking the **Evaluate** button performs the chosen operation with the defined parameters. Keep on mind that some operations, like median filter, can take a while (up to several minutes on large images and/or on slow computers). The new image window is always opened, the chosen source image remains untouched.

FITS Header Editor

The **FITS Header Editor** tool allows to view and possibly update FITS headers in the text form.



Changes performed in the editor are not immediately propagated to the FITS header of selected image. It is necessary to press the Apply button else all changes will be canceled when the tool is closed or another image is selected.

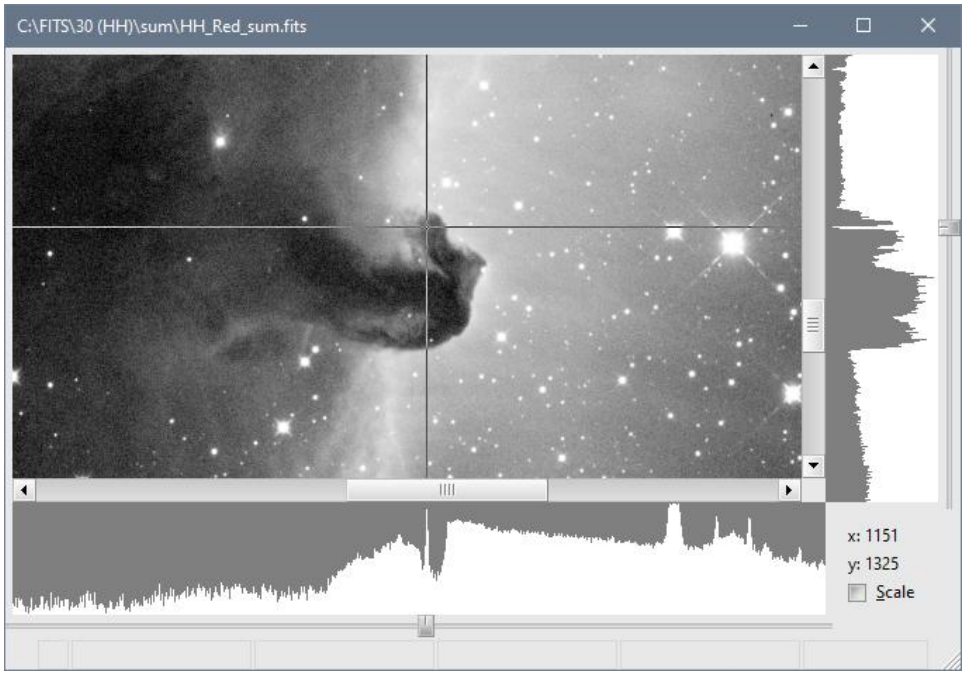
The FITS Header Editor checks the text before the update and prohibits updating of the image if some header is corrupted. Although the FITS header is basically text-based, number of rules apply to the header (e.g. every line must be exactly 80 characters long, strings should be padded to multiply of 8 characters by spaces, number of lines in header must be multiply of 36 lines and blank lines must be added to fill the 36 lines etc.). It is not necessary to keep these rules if editing text in the editor—longer lines are cut, shorter lines are padded, empty lines are added etc. by the FITS Header Editor during update. Only general rules must be followed, like the “SIMPLE = T” first line and “END” last line etc.

Remark:

Some FITS headers may be not important at all, but some are critical for the image. Change them only if you perfectly understand why and what are you doing.

Image Profile

The **Image Profile** tool shows profiles in X and Y axes of the selected image.



The Image Profile tool shows horizontal and vertical profiles of the currently selected image. The profiles range (minimal and maximal value) is defined by the current image stretch (see the Histogram and Stretch Tool). Anything displayed on the image black (because pixel value is under low stretch limit) or white (because pixel value is above high stretch limit) is outside the profile display.

To view the profile covering the complete pixel range, stretch the image to full pixel range first.

The **Scale** checkbox in the lower right corner of the tool window extends the profile to the full profile space (if it does not fill it already).

The pixel row and column displayed in the horizontal and vertical profile views is defined by the inverted cross-hair in the image. The cross-hair can be moved by mouse click (it is also possible to drag the mouse while the left mouse button is pressed).

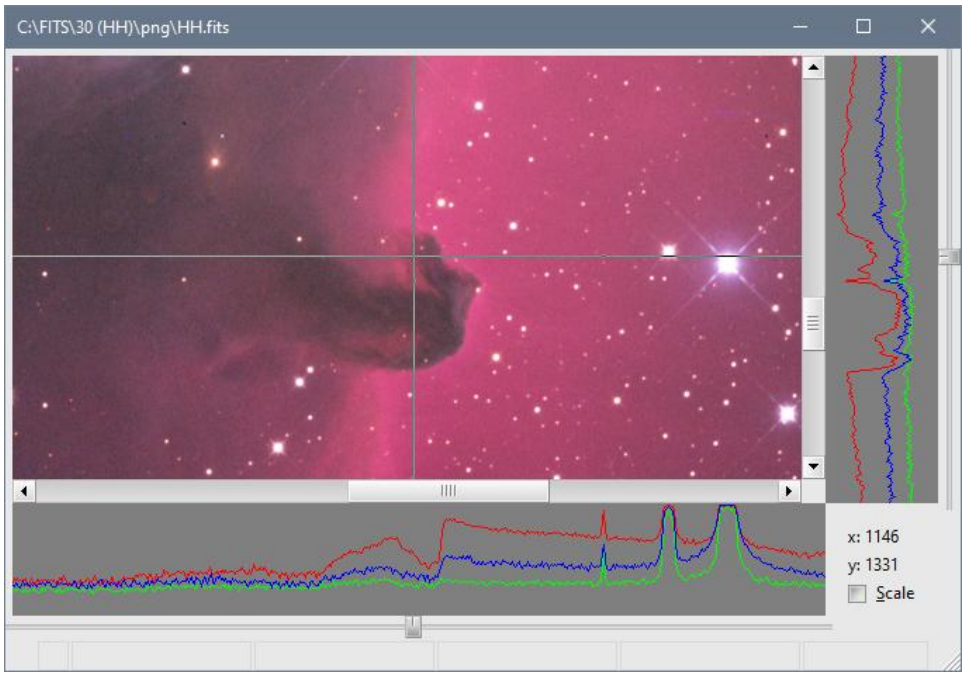
The cross-hair can be also moved by cursor keys:

- <Up>, <Down>, <Left> and <Right> keys move the cross-hair one pixel.
- Combining cursor keys with the <Ctrl> key speeds up the movement to 8 pixels.

Remark:

The image displayed in the Image Profile tool can be manipulated similarly to the ordinary image window—it can be zoomed by mouse wheel and the visible portion of the image can be dragged by the right mouse button.

In the case of color images, the Profile tools displays individual profiles of red, green, and blue colors.


C:\FITS\30 (HH)\png\HH.fits
x: 1146
y: 1331
Scale

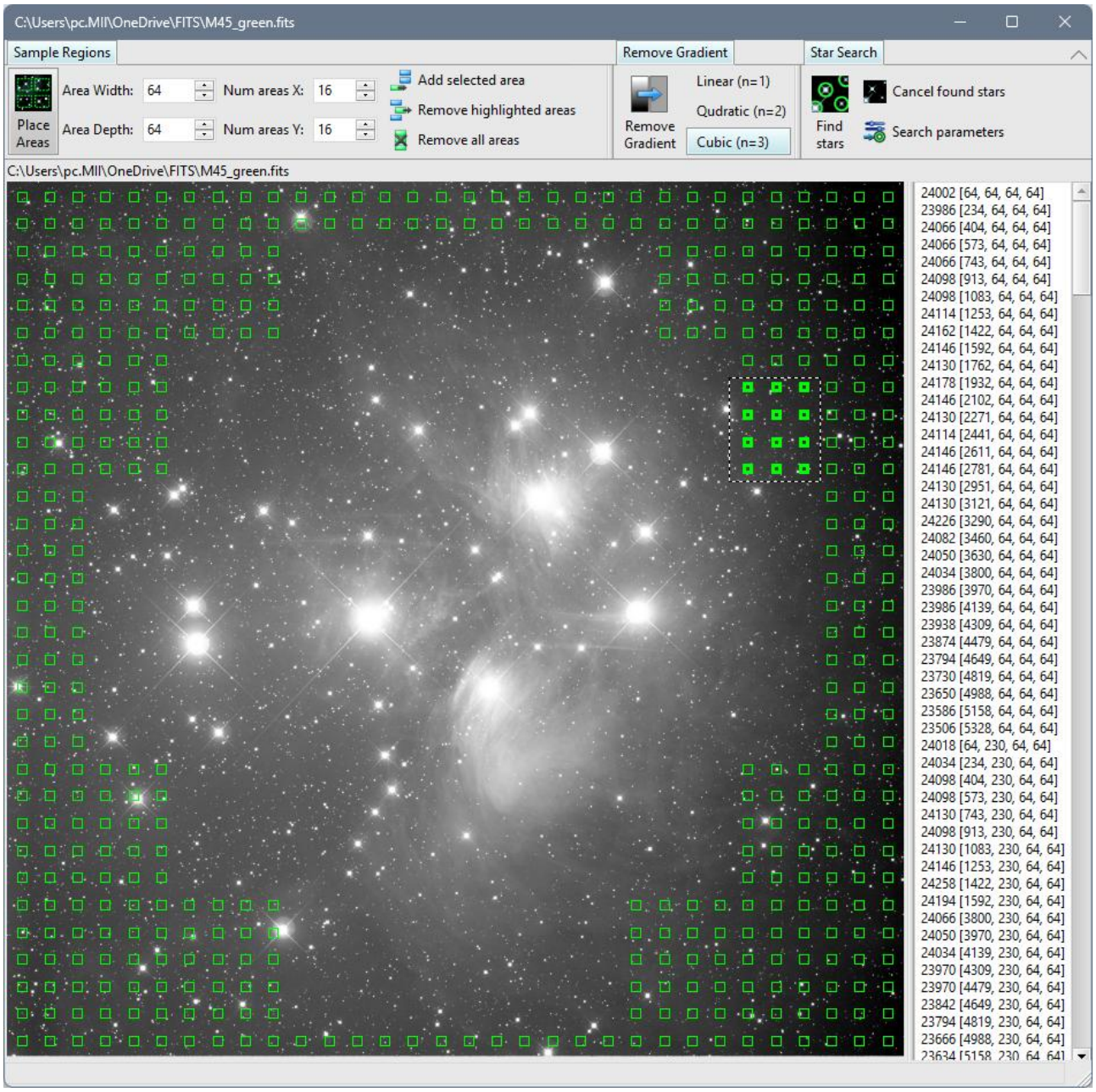
Remove Gradient

The **Remove Gradient** tool is intended to eliminate differences in image background values. Unfortunately, the sky background is never uniform, as it is affected by light pollution, Moon position relative to the field of view, height above horizon etc. The wider the field of view, the bigger the differences in sky brightness in different parts of the image. But the sky background often changes so rapidly that even narrow field-of-view images exhibit visible unevenness.

Hint:

The uneven background does not affect the scientific data reduction (e.g., the photometry) that much, as the used algorithms calculate with it. It is mainly an aesthetic issue, often not allowing us to create a nice image, especially if the background gradients differ in individual color channels.

The **Remove Gradient** tool is capable to model the image background using 2D polynomials and to subtract the model from the image.



There are three types of 2D polynomials available, differing in the order:

- Linear (1st order) 2D polynomial with three free parameters:
$$offset = a \cdot x + b \cdot y + c$$
- Quadratic (2nd order) 2D polynomial with six free parameters:
$$offset = a \cdot x^2 + b \cdot y^2 + c \cdot xy + d \cdot x + e \cdot y + f$$
- Cubic (3rd order) 2D polynomial with ten free parameters:
$$offset = a \cdot x^3 + b \cdot y^3 + c \cdot x^2y + d \cdot xy^2 + e \cdot x^2 + f \cdot y^2 + g \cdot xy + h \cdot x + i \cdot y + j$$

The background values are sampled using a grid of areas. The median value of each area is used as a reference background value in the area center.

The **Remove Gradient** tool supports two methods of placing the sampling areas over the image:

1. It is possible to manually select the region in the image. Clicking the 

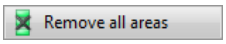
button adds the current selection into list of sample areas. The median value as well as area coordinates are added to the area list on the right side of the tool window.
2. The 

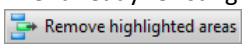
button places sampling areas over the image according to defined parameters. The **Area Width** and **Area Depth** define the size of each area; **Num areas X** and **Num areas Y** define the number of areas to be evenly distributed in the respective axis.

   Hint:

   If there is a selected (framed) portion of image, the **Place Areas** button distributes new areas only within the selection.

Obviously, not all background variations are unwanted gradients—nebulae or galaxies are in fact also "background gradients", but we naturally want to keep them in image. So, it is desirable that the background gradient is determined only from image parts without a nebulosity of galaxy.

The already existing areas can be removed, either the 

button removes them all or the 

button removes only the highlighted ones. Areas can be highlighted either in the list on the tool window right side or simply by selecting a portion of the image—areas within the selected portion will be highlighted.

So, the portions of the image to be used for gradient calculation (the ones containing sample areas) can be either defined by removing previously evenly placed areas (highlight the portion of the image with sample areas to be removed and click **Remove highlighted areas** button) or by adding areas to an empty image (highlight the portion of the image and click **Place Areas** button) or by combination of both ways.

Skipping stars

As mentioned above, each sampling area contributes to the background polynomial fitting with single point—its coordinates are the area center x and y and offset is the median of all pixels in the area region. Using of median should eliminate all extremes, including pixels belonging to a star, diffraction spike, hot pixel, or any other artifact.

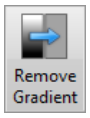
Still, sometimes some bright star can fill the significant portion of the area, especially if the areas are small, and thus affect the median value. This is why the **Remove Gradient** tool allows to find stars within the image. Found stars are then highlighted with rings indicating their respective diameters.

If stars are found in the image, the median of each area is then calculated only from respective area pixels not belonging to any star.

Hint:

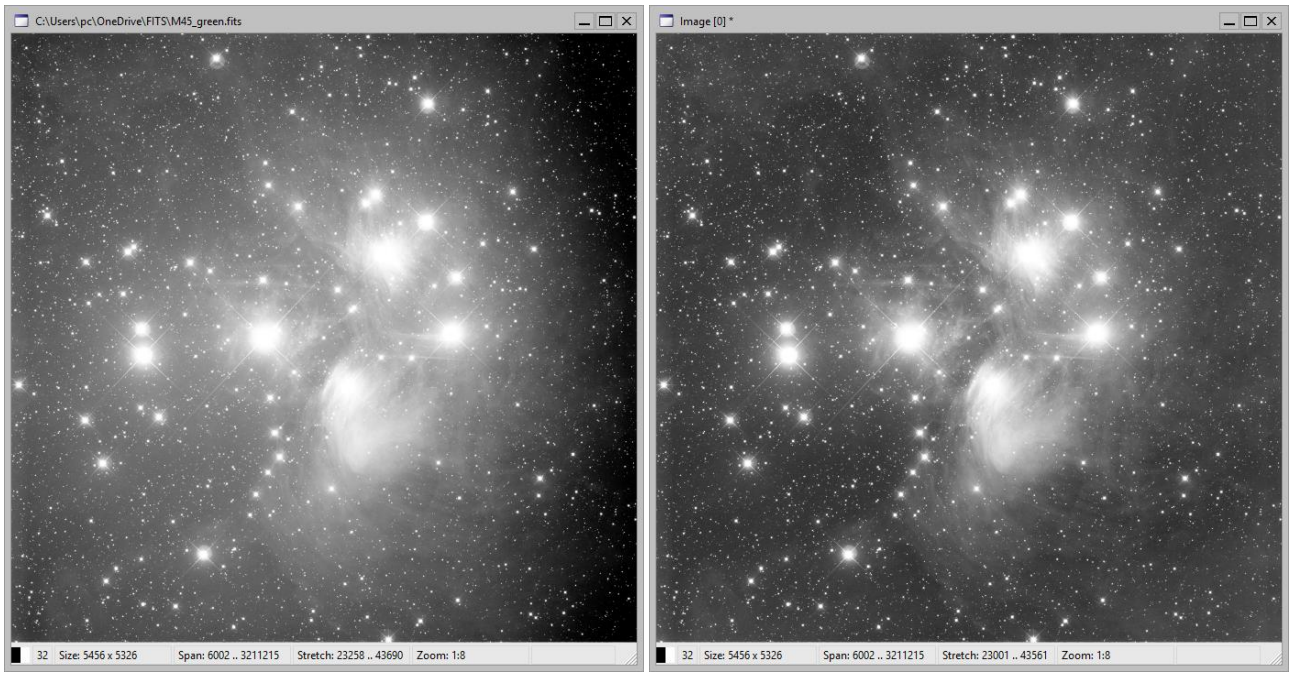
Especially in dense fields, highlighting of all found stars in the image with circles may slow down the image repaint. So, it is more convenient to find stars only after all sampling areas are placed. The area median values are recalculated when stars are found.

Removing the gradient

When the sample areas are properly distributed over the background areas of the image, the 

button performs actual polynomial fitting and the subtracts the resulting 2D polynomial from the selected image. The polynomial order is selected using three “radio buttons” in the Remove Gradient ribbon pane.

The example below shows the M45 Pleiades captured through photographic green filter, exhibiting significant background gradient over the field of view (left).

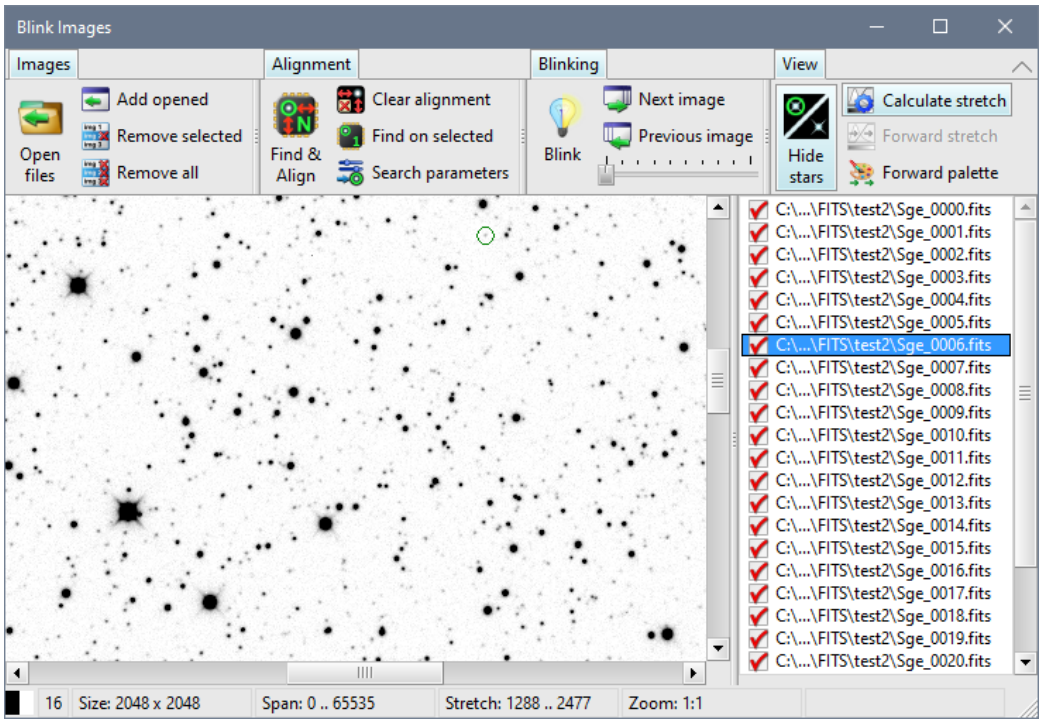


With the 3rd order gradient removed, the image background is even (right).

Blink Images

The **Blink Images** tool allows to repeatedly show images (possibly mutually aligned) in the tool implicit list with defined speed. The resulting effect is like the optical instrument called “blink-comparator”, intended for searching of moving objects. It does not work with glass plates, of course, but with FITS images. The idea behind blink comparison is in rapid switching of two or more images with stars aligned to each other. If there is some moving object (e.g. minor planet), its image will jump, while images of stars remain at the same position.

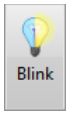
Compared to opto-mechanical instruments, software tool can blink any number of images, not just two, and a user can include/exclude individual images to/from blink sequence. It can also automatically align images, so the moving object is more easily detectable because stars appear static. It is very easy to alter blink speed. Image appearance (stretching) can be adjusted to be the same in all images not to disturb the observer etc.

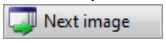


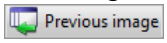
The Blink Image tool includes implicit image list, holding images to be blinked. It can include already opened images as well as load images from disk files. The image list is displayed on the right side of the Image Blink tool window.

The **Images** ribbon handles the Blink Images tool implicit image list contents. It contains the same controls like the standalone image list window.

The **Alignment** ribbon deals with mutual alignment of all images in the tool’s implicit image list, so the stars do not move when images are blinked. The alignment is based on matching of stars found on images. Mechanisms and controls of star alignment are described in the following sub-chapter.

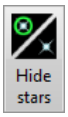
The **Blink** ribbon controls blinking itself. The 

button turns blinking on and off. Blink speed can be set using the horizontal slider. When the blinking is off, images can be switched manually using the 

and 

buttons.

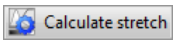
> Hint:
>
> If the image list on the right-side of the tool windows is selected, images can be also switched by changing the highlighted image using arrow keys or by mouse click.

The **View** ribbon alters how images are displayed. The first option button 

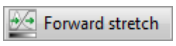
controls if the stars fond with the image are marked using green circles (image alignment requires finding of stars in image). As drawing of hundreds or even hundreds of thousand circles, depending on the star density and field of view, is time-consuming and so many green circles can almost hide the image behind, it can be turned on and off.

Remark:

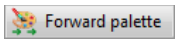
The Hide/Show stars option is handled globally for the whole SIPS. This means if the user turns the option on or off in the Blink Images too, it also affects other tools (Astrometry, Photometry, …) and highlighting of found stars is turned on or off for all images in SIPS.

It is important all blinked images are similarly stretched, otherwise the visual differences among individual images (background level, contrast, …) distract the user. The 

option button causes stretch limits are always calculated according to auto-stretch type predefined in the **Histogram and Stretch** tool when the images is selected and displayed in the left-side panel. This can be useful when images are loaded with different auto-stretch calculation option and stretch limits need to be re-calculated for all images.

The 

option button does not cause re-calculation of the stretch limits, but only copies stretch limits of the current image, possibly manually adjusted, to the stretch limits of the newly selected/displayed image. The disadvantage of such simple forwarding of stretch limits is that it does not adopt to changing imaging conditions of individual images. For instance, the sky can get darker after twilight or brighter closer to dawn and thus the background value moves and fixed stretch limits do not respect these changes.

The last option button 

ensures the next selected image uses the same palette like the currently selected one.

Identifying stars on image

Mutual image alignment depends on the program ability to identify individual stars in the image.

The star centroid searching algorithm can be modified by a set of parameters. These parameters can be defined in the dialog box opened by the 

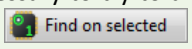
. Please refer to the **Astrometry** chapter for the detailed description of star search parameters.

However, there are a few differences when stars are searched on images for the purpose of **Astrometry** or **Photometry**, where it is important to find also the very dim stars, and when stars are searched only to allow mutual image alignment. As thoroughly described in the **Astrometry** chapter, the difficulty of finding both brightest and weakest stars is overcome by optional usage of two apertures. But for mutual image alignment, proper finding of brightest stars is enough. Typically, only a few tens of brightest stars are used for matching.

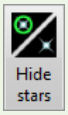
This is why the SIPS always uses only the greatest aperture defined not to miss the brightest stars:

- If only one aperture is defined, it is used also for finding stars for matching.
- If two apertures are defined, only the second (greater) one is used. This could lead to missing of weak stars if they are close to bright ones, but this is irrelevant for matching.

Hint:

It is not necessary to try to align all images in list during fine-tuning of these parameters. The **Alignment** ribbon contains the 

button, which tries to search the stars on the single selected image only. The user can judge if the brightest as well as dimmest stars were properly detected and possible to alter the parameters.

Also, do not forget to release the 

option button to see search results. When the star search parameters are tuned, it is better not to highlight fond stars to save time needed to redraw images as well as to see the images themselves, not only a bunch of green circles.

Automatic image alignment

Proper image alignment requires proper detection of stars in all images. The algorithm then determines the transformation (translation and rotation) among the reference image and all other images. Other images are the displayed in the transformed form to appear in the same orientation like the reference image.

General transformation is capable to cover field drift as well as field rotation caused by tracking errors, unprecise polar alignment of the telescope mount etc. There are no limits on the transformation, if for instance half of images are captured after the mount swap and thus are approximately 180 degrees rotated in regard to the reference image, they are rotated back to the original orientation.

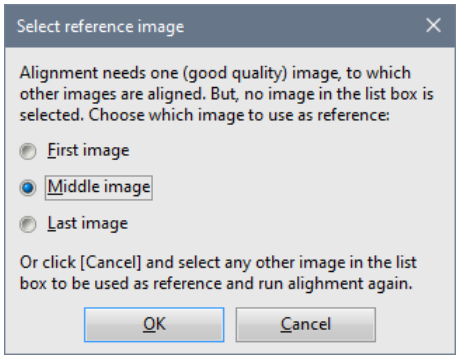
> Remark:
>
> The transformation is used only to display images, original bitmaps are not affected.

Proper selection of the reference image may affect alignment of all images in the list. This is particularly important for poor quality image sets with big differences of the same star shapes among individual images. The differences may be caused by changing of focus during observing session, e.g. because of thermal expansion of telescope tube and/or primary mirror. Not properly collimated telescope and/or camera sensor aligned to optical axis may produce different star shapes over the field of view and in combination with drifting filed or possibly GEM swap. Also, if the first images are taken low over horizon on relatively bright sky during twilight (or last images during dawn), their quality are often noticeably worse than quality of images taken around midnight.

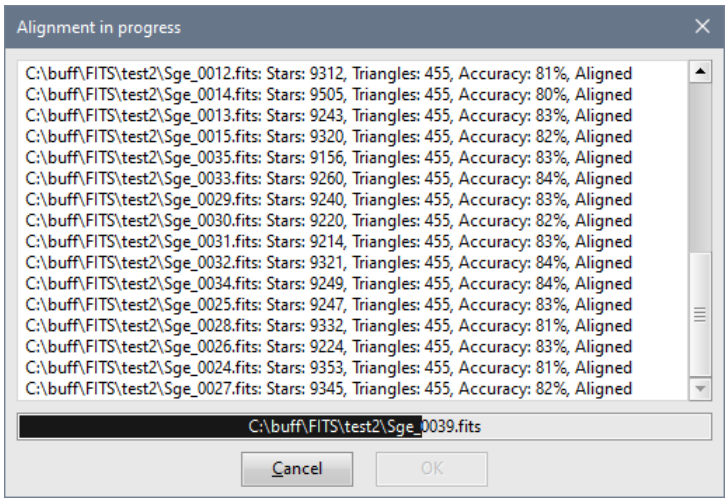
It is always the best practice to select a good quality image with the best focus as a reference one. There is a good chance majority of stars are properly detected on such image and the matching with other images produce the best results.

The **Blink Images** tool takes the actually selected image (displayed in the left panel) as the reference one. So, it is just necessary select one of the best images in the list (probably around the middle of the series) prior clicking the 

button. If no image is selected and align command is executed, the Blink Images tool opens a dialog box asking the user which image is to be used as the reference one.



The alignment progress is displayed in the dialog box, opened upon clicking of the **Find & Align** button.



The dialog shows progress on individual images in the list, including the number of detected stars, number of created triangles, matching accuracy and overall align result of every image.

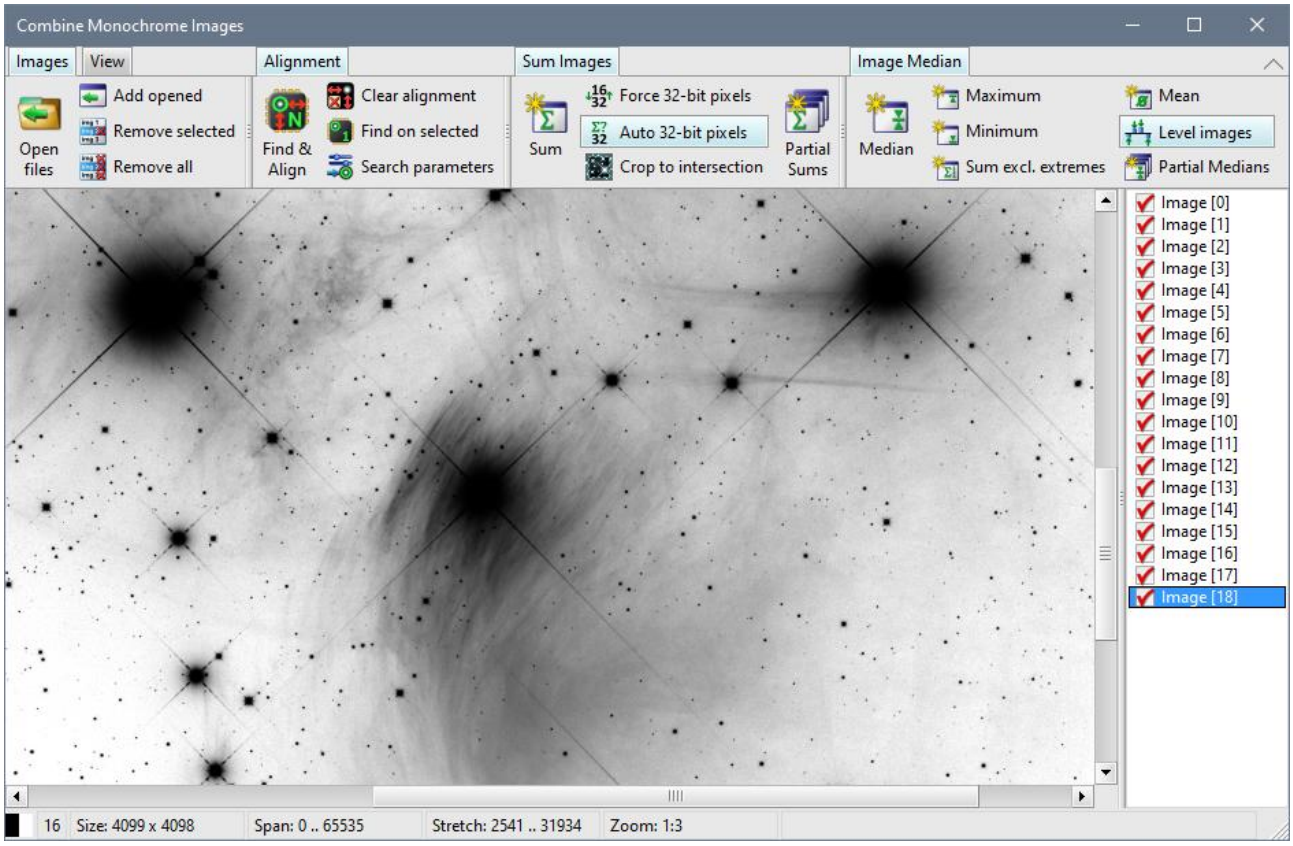
Remark:

SIPS is well optimized for modern personal computers equipped with many processor cores. The aligning task is always distributed to all available CPUs, which is the reason the images are shown finished in different order than they are in the list. This is caused by the fact, that some core finishes its task and continues with next image faster than other core, depending on many factors like access to main memory etc.

This is also the reason clicking the **Cancel** button does not react immediately, but the tool first needs to wait for all cores to finish the image they are currently working on and only then the whole alignment is canceled.

Combine Monochrome Images

The **Combine Monochrome Images** tool allows to mutually align images in the tool implicit list and combine them to sum or median.



I addition to simple math operations Sum of image list or Median of image list (see the **Image Math and Filters** tool description), the **Combine Monochrome Images** tool is capable mutually align individual images prior to creating of a sum or median. The mutual image alignment works the same way like in the case of **Blink Images** tool. Refer to the **Blink Images** tool description for align process explanation, please.

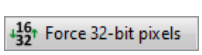
> Remark:
>
> While simple image stacking using the **Image Math and Filters** tool is suitable for creating of master calibration images (master dark and master flat), stacking of actual astronomical object images requires previous image alignment, as every image is more or less shifted and/or rotated relative to reference frame. Even in the case of perfectly aligned mount and very good tracking, at last sub-pixel differences among images exists and aligning eliminate them.

When images are aligned, the **Combine Monochrome Images** tool is capable to create new image as sum or median of individual images. Both operations can be modified by various options.

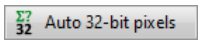
Sum Images

Sum Images ribbon contains controls involved in creation of a new image as sum of individual frames.

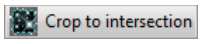
button creates new image as a sum of individual frames.



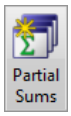
option ensures resulting sum image will be always 32-bit, regardless if it fill the 16-bit dynamic range or not.



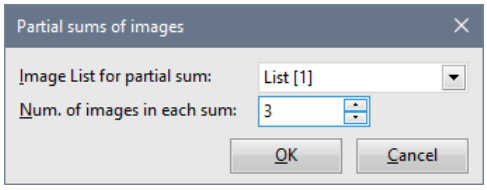
option can be used to automatically create 32-bit resulting image id the dynamic range of individual stacked images exceeds 16-bits.



option controls the size of the resulting image. If this option is left unchecked, the resulting image dimensions are enlarged to contain all, possibly mutually shifted, stacked images. Checking of this option causes the resulting image is cropped to the size of the area contained on all stacked images.

The 

button allows to fill a list of images with partial sums. This means the tool stacks a defined number of images and puts resulting sum into a list. Then it stacks another batch of images and adds the result into list etc.

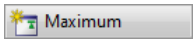


The dialog above allows definition of the image list, into which newly created sums are to be added, and number of images in each partial sum.

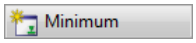
Image Median

Image Median ribbon contains controls involved in creation of a new image as median of individual frames, but also related operation like creation of image containing maximal or minima values of all stacked images etc.

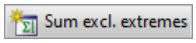
button creates new image as a median of individual frames.



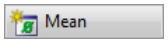
button creates new image, which pixels are the maximum of corresponding pixels of all combined images.



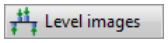
button creates new image, which pixels are the minimum of corresponding pixels of all combined images.



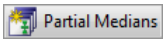
button creates new image as a sum of all combined images, but maximal and minimal pixels will be excluded from the sum.



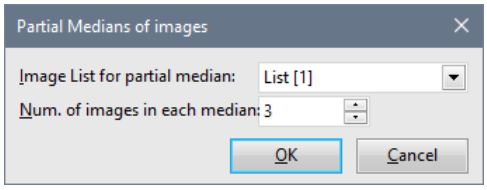
button creates new image as a mean of all combined images.



option ensures the mean value of all images are equalized for all images. Leveling of mean values of all images could be important, because median is calculated as middle value of sorted pixel values.

The 

button allows to fill a list of images with partial medians. The function is like **Partial Sums**, only median combination is used instead of adding images.



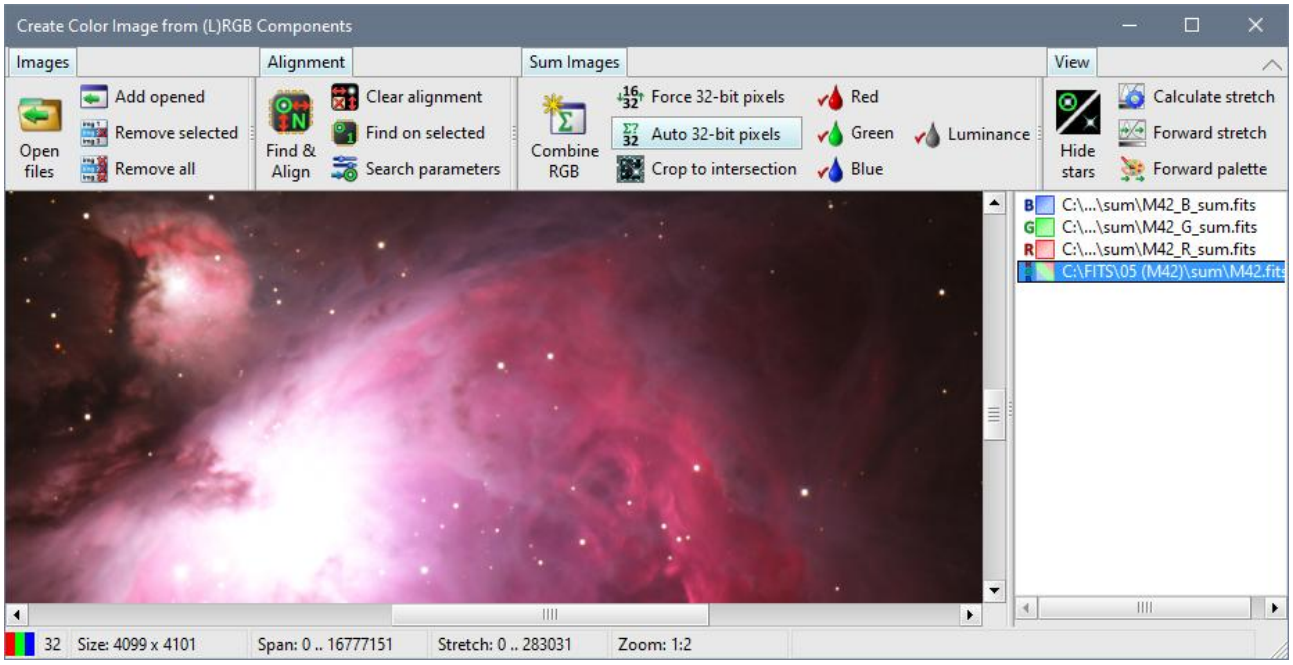
The dialog above allows definition of the image list, into which newly created sums are to be added, and number of images in each partial median. Note while the minimum number of images, stacked into partial sums, is two, partial medians can be created from at las three individual images.

Combine Color Images

The **Combine Color Images** tool functionality is the same like the **Combine Monochrome Images** tool, but image stacking or median combination is performed on color FITS images. Because of difference between monochrome and color images such functionality cannot be handled by one tool.

Create Color images from (L)RGB Components

The **Create Color images from (L)RGB Components** from individual red, green, blue, and possibly luminance images.



This tool combines the functionality of both Combine Monochrome Images tool and Combine Color Images tool. It always creates color images, but it accepts both monochrome and color ones.

- All three red, green, and blue planes of color images are added with corresponding color planes of other images.
- Monochrome images can be marked as red, green, or blue. Images marked as particular color are added with other images marked the same color or with the same color planes of color images:
  - Red mark red image
  - Green mark green image
  - Blue mark blue image
- It is also possible to mark monochrome image as Luminance:
  - Luminance mark luminance image

Hint:

SIPS scans the FILTER property of all added images for the “red”, “green”, and “blue” strings and if one of these keywords is found, the corresponding color channel is assigned to the image automatically. Even if the FILTER FITS header is missing (information about the used filter is not available), SIPS then scans image file name (providing the image is saved and thus a file name is assigned) for the same keywords and possibly assigns appropriate color.

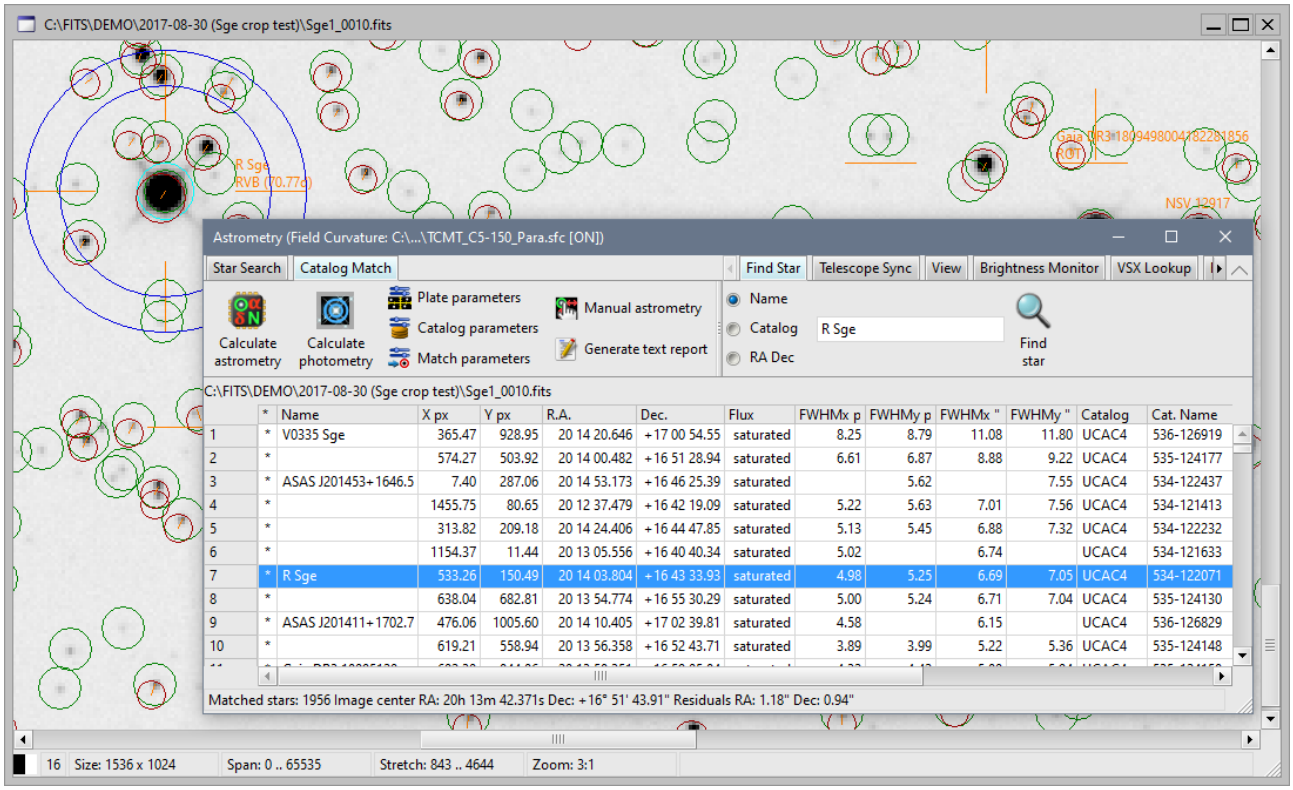
Individual images, representing colors channels, must be mutually aligned prior to combining them into resulting color image.

Astrometry (and Photometry) of single image

Note:

The **Astrometry** tool name reflects its evolution from a tool performing just astrometric reduction of single image. Over time, full **Photometry** processing, including **Ensemble Photometry**, was added to this tool and thus the more appropriate name would be **Astrometry and Photometry of single image**. Unfortunately, such name is too long for launch buttons and the tool window title, so the original name was kept. But keep on mind both astrometry and (ensemble) photometry of single image can be performed.

The **Astrometry** tool allows astrometric reduction[5] of the image (resolving equatorial coordinates for every star or other objects within the image).



	*	Name	X px	Y px	R.A.	Dec.	Flux	FWHMx p	FWHMy p	FWHMx "	FWHMy "	Catalog	Cat. Name
1	*	V0335 Sge	365.47	928.95	20 14 20.646	+17 00 54.55	saturated	8.25	8.79	11.08	11.80	UCAC4	536-126919
2	*		574.27	503.92	20 14 00.482	+16 51 28.94	saturated	6.61	6.87	8.88	9.22	UCAC4	535-124177
3	*	ASAS J201453+1646.5	7.40	287.06	20 14 53.173	+16 46 25.39	saturated		5.62		7.55	UCAC4	534-122437
4	*		1455.75	80.65	20 12 37.479	+16 42 19.09	saturated	5.22	5.63	7.01	7.56	UCAC4	534-121413
5	*		313.82	209.18	20 14 24.406	+16 44 47.85	saturated	5.13	5.45	6.88	7.32	UCAC4	534-122232
6	*		1154.37	11.44	20 13 05.556	+16 40 40.34	saturated	5.02		6.74		UCAC4	534-121633
7	*	R Sge	533.26	150.49	20 14 03.804	+16 43 33.93	saturated	4.98	5.25	6.69	7.05	UCAC4	534-122071
8	*		638.04	682.81	20 13 54.774	+16 55 30.29	saturated	5.00	5.24	6.71	7.04	UCAC4	535-124130
9	*	ASAS J201411+1702.7	476.06	1005.60	20 14 10.405	+17 02 39.81	saturated	4.58		6.15		UCAC4	536-126829
10	*		619.21	558.94	20 13 56.358	+16 52 43.71	saturated	3.89	3.99	5.22	5.36	UCAC4	535-124148

Note the Astrometry tool works with currently active (selected) image within SIPS workspace. Selecting individual images also changes the context of the Astrometry tool.

Images of individual stars (or asteroids, comet nuclei etc.) on an image typically cover multiple pixels with varying brightness (the profile of star image resembles Gaussian curve). Calculating weighted average of all pixels, where pixel brightness is the weight, allows to determine the star image centroid position with sub-pixel precision. Then the equatorial coordinates of celestial bodies can be calculated with sub-pixel precision, too.

Remark:

Let us note that image is often called “plate” when talking about astrometry from historical reasons. Similarly, pixel coordinates within the image are called “plate coordinates” etc. Astrometry was historically performed on emulsion-coated glass plates and star centers were measured on micro-metric machines. While it is much easier to handle digital images by software than to measure coordinates on glass plate, the goal of “plate solution” remains the same—to retrieve celestial coordinates of observed objects.

[5] Astrometric image reduction is sometimes also called “plate solution”

Astrometry (astrometric reduction) is performed in several basic steps:

1. SIPS must be able to find stars in the image. The whole astrometry procedure relies on this ability—found stars must be matched with stars in catalog. So, pixel coordinates of stars on the image must be known prior to any astrometric reduction.

   Remark:

   Also, other algorithms within SIPS rely on finding of stars, e.g. mutual alignment of images used in **Blink Images** or **Combine Images** tools, telescope mount guiding etc.

2. Plate parameters must be defined. Plate parameters are (at last approximate) equatorial coordinates of image center and the pixel scale—the angular size of single pixel. Alternatively, it is enough to define camera pixel size and possibly used camera binning, together with the used telescope focal length. Also, pixel scale (or pixel size and focal length) may be only approximate.

   Hint:

   SIPS allows storage of these parameters in the FITS headers, so it is not necessary to define them explicitly prior to astrometry calculation. See the **New FITS Headers** dialog box description in the **SIPS Images** chapter.

   While these parameters can be defined statically, the **New FITS Headers** dialog box also allows to read pixel size from the connected camera as well as to read image center coordinates from the telescope mount each time the image is downloaded.

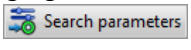
3. There must be at last one supported astronomical catalog present on the local PC. SIPS currently supports UCAC4, UCAC5, USNO-A2.0 and USNO-B1.0 catalogs. The **Catalog Parameters** dialog box allows to define catalog type and folder, where catalog files are located.
4. Astrometric reduction can be further parametrized through the **Matching Parameters** dialog box, despite default values should be OK for majority of cases.
5. After successful catalog matching with detected stars, equatorial coordinates are calculated for every object within the image.
6. A bit out of scope of astrometry reduction is calculating of photometry of found stars. Photometry calculation finds flux of stars (proportional to the amount of light captured from each star) as well as to calculate FWHM of each star in both pixel units and arc-seconds.

Let us go through individual steps in more detail.

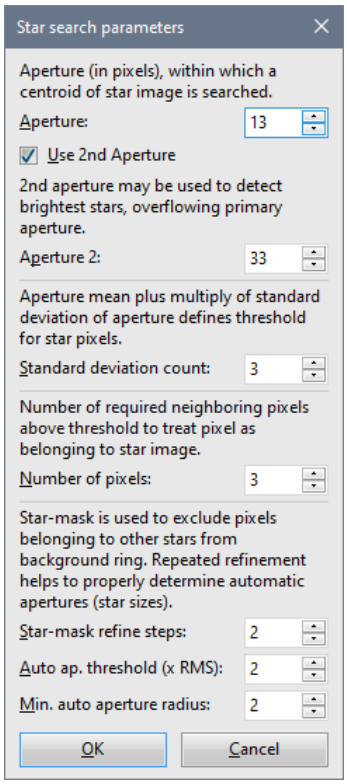
Identifying stars on image

Reliable identification of all stars in the image is a crucial step in astrometry reduction. As mentioned earlier, astrometry is based on matching stars on image and in catalog.

- If for instance the brightest stars are omitted on image (e.g. the search aperture is too small to contain very bright stars), astrometry may fail as the brightest detected stars on image and the ones read from the catalog do not match.
- If the dim stars are not found on image, SIPS cannot assign their counterparts from the catalog nor calculate their coordinates.

The star centroid searching algorithm can be modified by a set of parameters. These parameters can be defined in the dialog box opened by the 

button.



Searching stars is based on statistical analysis of image pixels in the box of defined size, which moves through the image. Star is detected if a group of neighboring pixels reaches values a defined amount of the box standard deviation above the box mean value. The box size, minimal number of neighboring pixels and threshold (a multiply of standard deviation) are all user-definable parameters:

- **Aperture** is the diameter (box size) of pixel area, in which a star centroid is searched. General rule is that the star image should well fit in the box else the star will not be recognized. Too big aperture on the other side covers more pixels than necessary. Aperture 11 or 13 is good value for typical well-sampled astronomical images.
- **Use 2nd Aperture** check box and **Aperture 2** value can be used to overcome the problem with very different star diameters in single image:
    - If the single aperture is significantly less than the aperture of bright stars, then such bright stars are not detected. The star is detected only if the group of pixels, fulfilling the “above the threshold” condition, does not touch the box border. This condition ensures the star is detected only when the box is positioned that all pixels belonging to the star are within it. Detecting the star prematurely, when only a portion of its pixels are in the moving box, would result into centroid offset from real location.
    - Aperture significantly greater than the aperture of dim stars slows-down the algorithm, but does not prohibit proper detection of dim stars, providing there is no bright star within the area of the box. Presence of a bright star increases the box standard deviation and mean value and causes the pixels belonging to the nearby dim stars do not surpass the threshold and the star is missed.

  Cure for this problem is the introduction of two apertures. The first aperture should be a good fit for the dimmest stars within the image. Thanks to the limited aperture size, these stars are detected even if they are located close to the bright stars. The second aperture should be big enough to contain even the brightest stars in the image. Proper detection of even the brightest stars is important for image alignment (and also for catalog matching as explained in the **Astrometry** and **Photometry** tool description). Alignment is based on finding similar triangles among images and tringles are constructed beginning with the brightest stars. If the brightest star or stars are detected on one image and missed on another, constructed triangles are very different and alignment may fail.
- **Standard deviation count** defines the threshold, above which a pixel is considered to belong to a star image.
- **Number of pixels** defines how many neighboring pixels above threshold must be detected to consider them star pixels. This condition eliminates single bright pixels above threshold like traces of sensor hot pixels etc.

The algorithm of searching stars depends on the single-bit-depth bitmap of the same size as the image itself, called "star mask" in SIPS. This bitmap contains 1 if the corresponding image pixel belongs to a star or 0 otherwise. After the first pass of the algorithm through the image, the 1-bits only roughly represent the actual star images, because of varying mean and threshold values as the detection box moved through the image—the box statistics depend on the presence of other stars or diffraction spikes, background gradient etc. Naturally also the star centroid position can be slightly out of the ideal position because of varying number of star pixels detected during first pass.

Remark:

The centroid detection precision decreases with decreasing image quality. Out-of-focus star images, possibly affected by optical aberrations, are more prone to bad detection of star pixels than well focused, round stars.

To achieve the best possible centroid positions and representation of stars, SIPS needs to iterate through several steps to refine the star mask. At least one iteration is necessary, but more than three iterations may not bring noticeable refitment. Two iteration steps (the default value) look like optimal compromise between detection precision and processing speed.

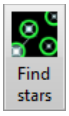
The star-mask-refine steps also include detection of apertures of found stars images (automatically determined star apertures play important role in the **Photometry** tool, described later).

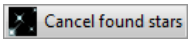
The algorithm used to determine aperture of stars scans the profile from the centroid in the four directions (up, down, left, and right) and measures the distance from the centroid to the first pixel lying below the defined threshold. Typically, values from 1 to 2 multiplies of standard deviation are used, but the optimal value depends on the stellar profile, which can be affected by many factors such as quality and type of optics, focusing precision, tracking precision, seeing, etc.

It might happen that automatically determined radii of nearby stars overlap. This is not desirable especially when automatic apertures are used to calculate the fluxes. SIPS detects and eliminates overlapping radii and calculates new radii as the fraction of the Euclidean distance of star centroids corresponding to the ratio of the fluxes of the two stars.

The last three parameters in the dialog box concerns automatic star aperture calculation and star-mas refining:

- **Star-mask refine steps** parameter was described above.
- **Auto ap. threshold** defines the value at which the scanning of the star profile ends.
- **Min. auto aperture radius** defines the minimal radius for the cases the statistics result into unrealistically small apertures for the very dim stars.

Finding stars itself is initiated by the 

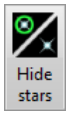
button. If star search parameters are updated, just clicking this button again cancels all found stars and finds them again. However, it is possible to explicitly cancel all found stars without finding them again using the 

button.

Notice the **Astrometry** tool window lists all stars, found on image, in a table (sheet) once the stars are searched. The table contains only X and Y coordinates (in pixels) of all stars, leaving other columns empty. Stars are sorted by its flux (brightness), but please note the flux determined in this phase may not be very precise as it is affected e.g. by nearby bright stars etc. Precise flux can be calculated later during Photometry processing, but the difference is rather small.

Highlighting stars

Once the stars are found in any image, SIPS highlight them with green circles. This can be particularly useful when fine-tuning stars-search parameters for the used telescope and camera setup. Missing green circles around brightest stars indicates it is necessary to enlarge aperture or to use 2nd (greater) aperture etc.

However, green circles around every found star partially obstruct original data and make the image hard to perceive. Also, drawing many circles takes some time and slows the zooming and scrolling of the image. Once the parameters of star search algorithm are fine-tuned and SIPS reliably finds all stars on images, green circles around found star are superfluous. The **View** ribbon contains 

option button, which allows to hide the green circles around stars.

Hint:

This option button is synchronized among all tools, containing controls to find and highlight stars—**Blink Images**, **Combine Images**, **Astrometry** and **Photometry**. Star highlighting can be turned on or off in any of these tools and changes are propagated to other opened tool windows.

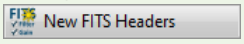
Plate parameters

Astrometry in SIPS cannot perform all-sky search for the image position and scale. The algorithm needs at last approximate information about the image:

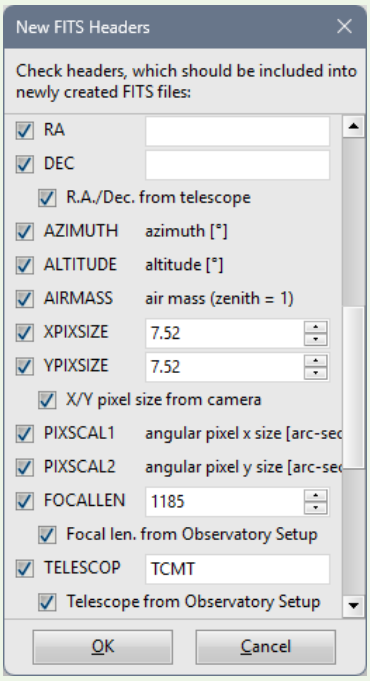
- Equatorial coordinates of the image center.
- Image scale (angular size of pixels). The image scale can be defined:
  - Directly (by specification of pixel angular size).
  - Indirectly (by specification of pixel physical dimension and telescope focal length) and SIPS then calculates pixel angular size itself.

Relying on such "starting" information, despite only approximate, significantly speeds up the astrometry solution.

Hint:

As mentioned earlier, SIPS can store all this information into FITS headers automatically for later usage in astrometry reduction. In typical usage scenarios, image center coordinates are read from the telescope mount upon image acquisition (the mount must be connected to SIPS using appropriate driver) and pixels physical dimensions are read from camera, while focal length of the used telescope is defined in the **New FITS Headers** dialog box, opened from SIPS **Setting** tab by the 

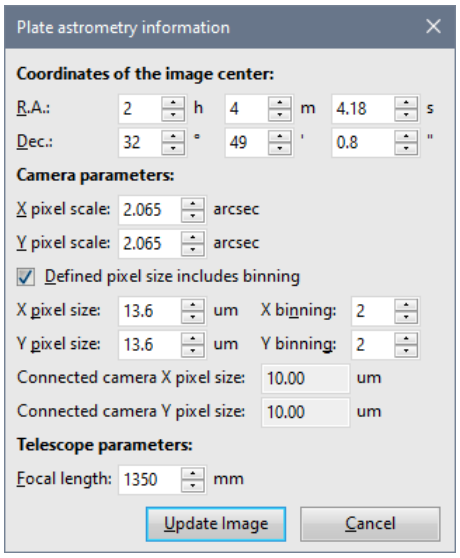
button.



The telescope focal length itself can be either defined explicitly in the **New FITS Header** dialog box or taken from the **Observatory Setup** dialog box.

If plate parameters information is to be reviewed or modified, the 

opens the **Plate astrometry information** dialog box.



Hint:

Sometimes coordinates are not provided in hours or degrees, minutes, and seconds, but in degrees in the form of decimal number (fraction of degrees). While conversion from decimal degrees to minutes and seconds is simple, it involves a few divisions and subtractions, most people cannot do such math without calculator or at last pen and paper.

This is why the equatorial coordinates controls are capable to perform “smart paste” operation. If a number with a fraction part (containing decimal point) is pasted to the **Dec.** degrees count-box, SIPS interprets it as a declination expressed in decimal degrees. It transforms the value to whole degrees, minutes, and second and fills all three count-boxes accordingly.

In the case of right ascension, the value, containing decimal point, pasted to **R.A.** hours count-box, is also interpreted as decimal degrees as it is customary to use degrees not hours also for right ascension when equatorial coordinates are expressed in the fraction form. So, the value is first divided by 15 to convert it to hours and only then it is transformed to whole hours, minutes, and second.

There is a kind of dispute in the astronomical community whether the pixel physical dimension, stated in the FITS header, refers to the pixel prior to binning or after the binning. To satisfy both approaches, SIPS allows to define if the **Defined pixel size includes binning**.

- If this option is set, the binning information becomes superfluous (the **Plate astrometry information** dialog box still displays the binning defined in FITS headers and allows to modify it, but they do not influence astrometric solution).
- Otherwise, the astrometry algorithm multiplies the stated pixel size with binning value in respective axes to get resulting physical pixel dimensions and use it for astrometric solution.

Warning:

The **Plate astrometry information** dialog box modifies metadata of the actually selected image. Closing the dialog box with the **Update Image** button modifies the image and it must be saved to make these modifications persistent.

But the **Defined pixel size includes binning** option is not stored in any image, this is a global SIPS option, stored in the SIPS configuration file. It instructs SIPS how to handle X/YPIXSIZE and X/YBINNING FITS headers of all processed images, regardless is the dialog box is closed with **Update Image** or **Cancel** button. We understand this can be slightly misleading, but the **Defined pixel size includes binning** option is so closely related to the **Plate astrometry information** dialog content, that placing this option elsewhere could cause even more confusion.

Also, this option controls how the already existing FITS files should be handled, it does not affect the X/YPIXSIZE and X/YBINNING headers content when SIPS creates new FITS files. SIPS always sets the X/YPIXSIZE values to physical sensor pixel dimension regardless of the used binning, else the X/YBINNING headers are superfluous.

Astrometry catalogs

SIPS needs an astrometry catalog, located at the computer file system (on some drive), to be able to match stars on image with stars in catalog. Catalogs available online are not used for performance reasons as well as to be able to perform astrometry solution without the access to Internet.

There are four catalogs currently supported in SIPS. Each catalog description below is accompanied with sample of information for one star, contained in respective catalog.

- **UCAC4**: very good catalog overall, consuming 8.41 GB of drive space. Provides rich information about every star (2MASS and APASS magnitudes, proper motions, …). Majority of bugs in the catalog concerns false positives (presence of non-existing stars), which does not hurt the astrometry. UCAC4 contains stars up to approx. 16 Mag, so another catalog must be used for deeper images.

```
UCAC4 ID          614-005736
R.A. [D.d]        30.9565686
Dec. [D.d]        32.6952500
Model fit mag.    11.363
Aperture mag.     11.349
Mag. error        0.03
Prop. motion R.A. -0.1
Prop. motion Dec. -14.9
2MASS id.         140753965
2MASS J mag.      9.695 (0.02)
2MASS H mag.      9.169 (0.02)
2MASS K_s mag.    9.054 (0.02)
APASS B mag.      12.691 (0.02)
APASS V mag.      11.568 (0.02)
APASS g mag.      12.099 (0.01)
APASS r mag.      11.207 (0.04)
APASS i mag.      10.902 (0.03)
```

- **UCAC5**: contains more precise star coordinates, based on Gaia DR1, but otherwise reduces the amount of information about every star. Catalog needs only 5.25 GB of drive space, but better precision brings no significant benefits for typical usage (e.g. photometry processing), so the UCAC4 is still a better choice.

```
UCAC5 ID            614-005472
Gaia DR1 ID         301739800644289152
Gaia R.A. [D.d]     30.9565744
Gaia Dec. [D.d]     32.6951961
Gaia R.A. err [mas] 0.3
Gaia Dec. err [mas] 0.1
Prop. motion R.A.   1.0 (1.3)
Prop. motion Dec.   -13.3 (1.1)
Gaia DR1 G mag.     11.180
UCAC model mag.     11.386
NOMAD R mag.        11.180
2MASS J mag.        9.695
2MASS H mag.        9.169
2MASS K_s mag.      9.054
```

- **USNO-A2.0**: Rather outdated catalog, generated by digitizing of Palomar Sky Survey plates. Occupies just 6.11 GB space but contains only “red” and “blue” magnitudes, originating in blue and red filters[6], used for Palomar Sky Survey. SIPS converts these non-standard magnitudes to get approximate B-V index, but the color information is not reliable. The greatest weakness of this catalog is the absence of many bright stars, which makes astrometry reduction very difficult (bright stars found on image are not found in catalog). USNO-A2.0 is therefore not recommended for plate solving.

```
USNO-A2.0 ID   1200-0868585
R.A. [D.d]     30.9565778
Dec. [D.d]     32.6954167
Red mag.       11.4
Blue mag.      12.6
```

[6] The Red and Blue filters, used for Palomar Sky survey, do not correspond to Johnson-Cousin R or B filters. These filters are not part of any common color photometric system.

- **USNO-B1.0**: Also generated from Palomar Sky Survey plates, but with more sophisticated algorithms. Brightness information is also based on red and blue color filters and transformed to standard colors in SIPS with limited precision. USNO-B1.0 occupies 124 GB of drive space and contains stars up to 18+ Mag, which is significantly more than other catalogs. This makes this catalog the only option for weak stars.

```
USNO-B1.0 ID      1226-0033688
R.A. [D.d]        30.9566028
Dec. [D.d]        32.6952889
R.A. err [mas]    0.157
Dec. err [mas]    0.032
Prop. motion R.A. -0.002
Prop. motion Dec. -0.018
Red 1 mag.        11.22
Blue 1 mag.       12.53
Red 2 mag.        11.18
Blue 2 mag.       12.08
N mag.            10.81
```

Remark:

The above-mentioned catalogs are used also in various modified forms. For instance, some planetarium programs use reduced versions of such catalogs to save drive space, eliminating information not used within the planetarium program. SIPS always needs the original form of the catalog to work properly, the compressed and/or reduced versions cannot be used with SIPS.

Support for two catalogs

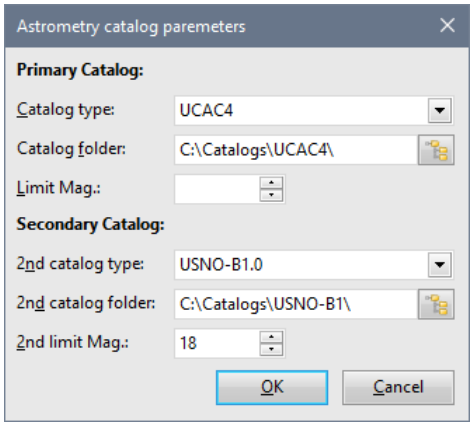
If all detected stars are covered by the UCAC4 catalog (no star of interest is weaker than ~16 Mag), using UCAC4 only brings no disadvantages. However, if there are stars weaker than ~16 Mag, astrometry cannot provide cross-identification of such weak stars with catalog, which makes e.g. publication of such stars problematic as such cross-identification must be performed later manually. Using USBO-B1.0 catalog solves this issue, but we lose important information about bright stars in the field (color indexes etc.).

The above dilemma is solved in SIPS by the ability to use two catalogs for plate solving.

1. Astrometry is performed with **Primary** catalog. Also, stars on image are matched with all available stars in primary catalog.
2. Then a **Secondary** catalog is used, this time not to perform astrometry (it was already performed using primary catalog), but only to match remaining stars and obtain their cross-identification.

The **Astrometry catalog parameters** dialog box, opened by the 

button, allows specification of the primary and secondary catalogs. If no secondary catalog is to be used, select the **undefined** value as the catalog type.



Sometimes a catalog contains too many stars, for instance if a camera with a small photographic lens is used to get very wide field images. Lowering the number of stars loaded from a catalog by imposing magnitude limit saves memory and processing time, as the catalog plate, corresponding to image, does not contain many weak stars, which are not detected on images either way.

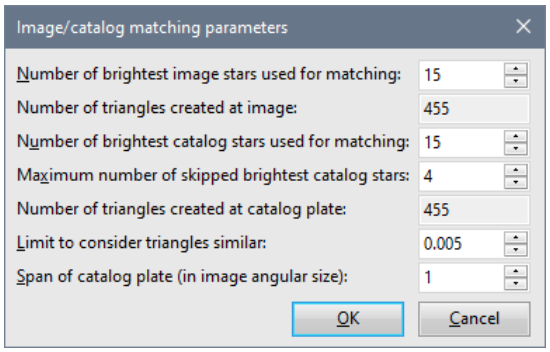
The **Astrometry catalog parameters** dialog box also allows to define limiting magnitude for stars from particular catalog. If no limit should be used, clear the **Limit Mag** count-box.

Astrometry calculation

Astrometric match is performed similarly to the mutual image alignment, except that one set of triangles is artificially created from tangentially projected star coordinates from the catalog files.

If a matching triangle is found, a transformation matrix is calculated using the least-squares method and tested on all stars used to create triangles. If 50% or more stars are matched, another transformation matrix is calculated from all matched stars. For astrometry purposes, every detected star in the image is searched in the catalog and if the corresponding catalog star is found, the pair is recorded. Then a final transformation matrix is calculated from all matched stars.

Parameters for image-catalog matching can be modified in the **Image/Catalog matching parameters** dialog box, opened by the Match parameters button.



For majority of cases, the default values of should lead to reliable astrometry matching. However, sometimes it would be necessary to adjust some of them to overcome possible issues.

Number of stars, used to create triangles for each image and catalog plate, can be set using **Number of brightest image/catalog stars used for matching** count-boxes. Non-editable edit boxes labeled **Number of triangles created at image/catalog plate** show the $\binom{N}{3}$ number or how may triangles will be created from the defined number of N stars.

The **Maximum number of skipped brightest catalog star** parameter allows to overcome possible matching issues caused by an inability to match the brightest stars from catalog with image. While such situation is not typical when we set the **Star search parameters** properly, it may happen e.g. when the difference between detected bright star position and corresponding position in catalog exceed the allowed threshold etc.

> Hint:
>
> Bright stars captured by legacy CCD cameras with non-ABG[7] sensors are often affected by blooming and thus are not recognized as stars by SIPS, regardless of star search parameter values. In such case, the matching could fail because SIPS creates triangles from the brightest star in catalog, while this star is missed on image.

If this parameter is greater than zero, SIPS repeats matching attempts with the same catalog plate, but skips up to defined number of brightest catalog star, until the match is found or maximum number of catalog stars are skipped and no match is found.

If the used camera does not suffer from blooming and even the brightest stars in the field of view are always properly recognized by SIPS, this parameter can be kept at 0.

The **Limit to consider triangles similar** parameter defines the maximal difference between image and catalog triangle side ratios to be considered similar.

The last parameter **Span of catalog plate** defines area, which is searched in catalog to find the match. If the image center coordinates are always stated precisely enough (say within 1/3 of the field of view), then this parameter can be kept at

[7] Anti-Blooming Gate

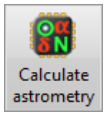
1. SIPS then tries only one catalog plate to match it is image. But if the image center coordinates uncertainty can be comparable or even greater than the field of view, increase this parameter. The astrometry algorithm then tries defined area in the catalog to find a match.

Hint:

Increasing of the **Span of catalog plate** parameter can be useful when astrometry is used to precisely point the telescope mount.

The typical procedure involves taking an image after telescope **Go To** operation. If the match is found, then the image center coordinates are synced with telescope mount to inform the mount where it actually points. The subsequent **Go To** command then only moves the telescope to the desired coordinates and because the move is very short, resulting error is also very small. Depending on the mount, pointing error can be eliminated to a few pixels.

So, the initial long-distance **Go To** precision should determine the **Span of catalog plate** parameter value. If the mount error remains a small fraction of the field of view, the value can be 1. But if the mount error is comparable with or exceeds the field of view, increasing of this parameter ensures **Astrometry** reliably finds the telescope equatorial coordinates.

Actual astrometry calculation is invoked by the 

button.

Hint:

If the stars are not searched yet, the **Solve plate** button does the search first. So, for regular usage, just click the **Solve plate** button and it does all necessary steps. Explicit **Find stars** button is present mainly to enable search parameters tuning etc.

When the astrometry matching succeeds, plate parameters of the image are updated. This means image center coordinates as well as pixel scale are precisely calculated and stored in the image. This is why image appears as modified (asterisk follows its name) after astrometry solution. If the image is then saved, the updated information appears in the respective FITS headers.

Once image is properly matched with the catalog, the **Astrometry** tool fills most of the remaining columns in the table of all found stars. At last equatorial coordinates, corresponding to the detected centroid of each star, are shown in the table.

Hint:

After the image is solved and the **Image Info** tool is opened, the equatorial coordinates of the pixel under the mouse cursor are displayed.

If any star on image is matched with a corresponding star in catalog, the first column in the table is filled with an asterisk character and also other columns are filled (used catalog and identifier in the catalog, catalog equatorial coordinates, possibly color indexes etc.).

Hint:

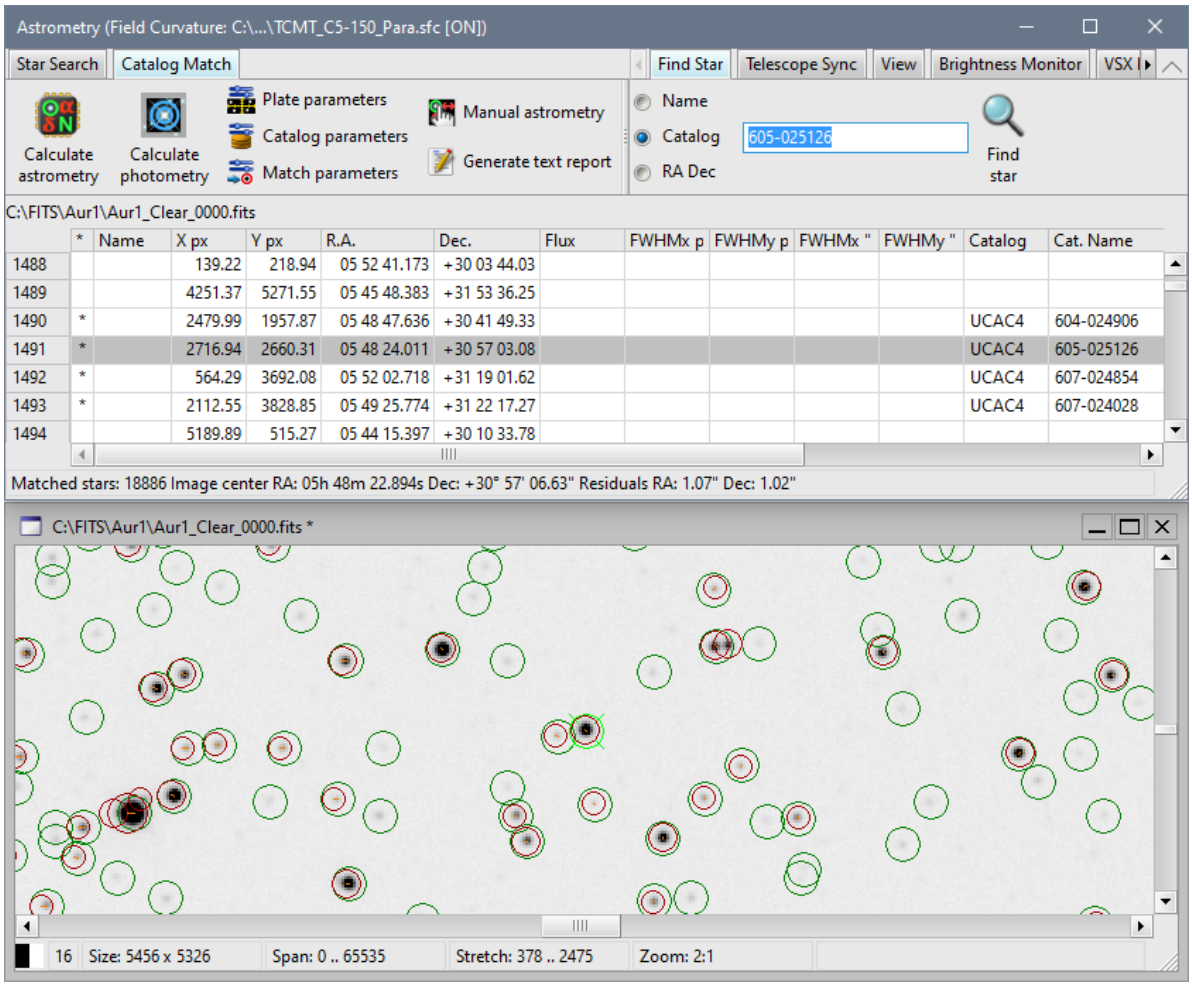
The <Ctrl>+<K> hotkey copies a string formatted “RA: HHh MMm SS.sss Dec: +DD° MM' SS.ss" Rot: DDD.dd°” with image center coordinates and rotation to clipboard.

Astrometry tool and image windows interactions

The selected window with a solved image and the **Astrometry** tool, showing a table of found and matched stars, mutually react to selection of a star.

- If and star is highlighted at the image by left mouse click, the **Astrometry** tool window highlights a line with the selected star (the table is possibly scrolled to show the line).

- Similarly, if any line is selected in the **Astrometry** tool window, the star is highlighted in the corresponding image. Depending on the current image zoom, the star is centered (or the image is scrolled to its edge) in the visible area of the image.



	*	Name	X px	Y px	R.A.	Dec.	Flux	FWHMx p	FWHMy p	FWHMx "	FWHMy "	Catalog	Cat. Name
1488			139.22	218.94	05 52 41.173	+30 03 44.03							
1489			4251.37	5271.55	05 45 48.383	+31 53 36.25							
1490	*		2479.99	1957.87	05 48 47.636	+30 41 49.33						UCAC4	604-024906
1491	*		2716.94	2660.31	05 48 24.011	+30 57 03.08						UCAC4	605-025126
1492	*		564.29	3692.08	05 52 02.718	+31 19 01.62						UCAC4	607-024854
1493	*		2112.55	3828.85	05 49 25.774	+31 22 17.27						UCAC4	607-024028
1494			5189.89	515.27	05 44 15.397	+30 10 33.78							

Finding stars in the star sheet

Stars within the Astrometry window sheet can be selected by searching according to various criteria, using the tools available in the **Find Star** ribbon:

- By object **Name**: text defined in the **Name** column of the sheet is searched. Each star name is empty by default, but the Astrometry tool allows the user to enter name of any detected object. Such name can be the found.

  Hint:

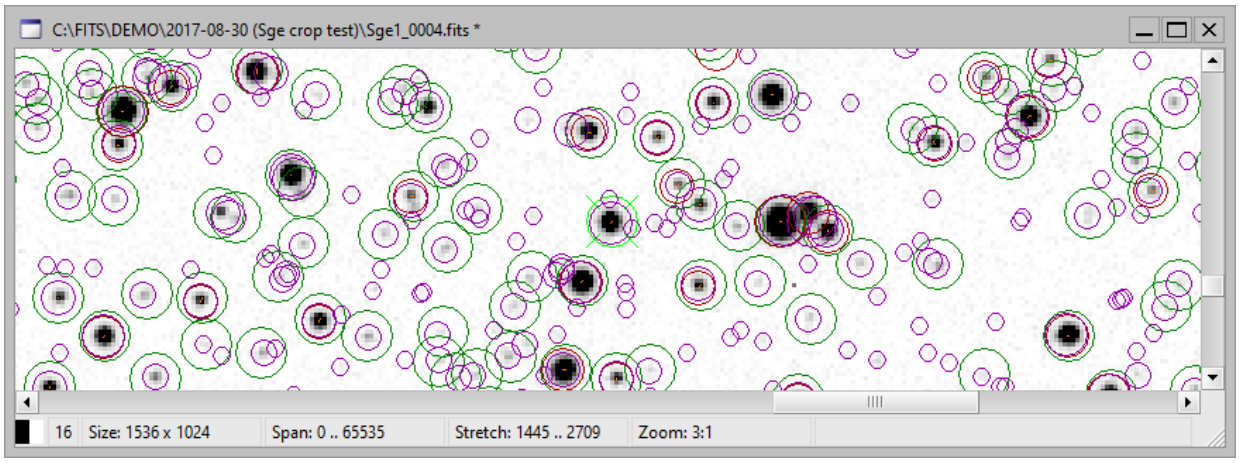
  The VSX database lookup fills the names with the object primary name within the VSX catalog. Such name can be then found using this option.

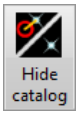
- By **Catalog** identification: stars, matched with catalog, can be found by the catalog identifier (content of the **Cat. Name** column).
- By equatorial coordinates **RA, Dec**: the entered test is parsed for R.A. and Dec. values and the coordinates are then searched. The coordinate format is arbitrary; SIPS tries to properly parse coordinates entered in various formats:
  - Two decimal numbers are interpreted as R.A and Dec. in decimal degrees.
  - Two strings delimited by blank(s) are interpreted as R.A and Dec. in “HH:MM:SS.ss +DD:MM:SS.ss” form. The delimiters can be arbitrary non-numeric characters, so also “HHhMMmSS.ss +DDdMMmSS.ss” and similar delimiters are accepted.
  - Six strings delimited by blanks are interpreted as R.A and Dec. in “HH MM SS.ss +DD MM SS.ss” format.

Highlighting catalog stars

Once the image is solved, SIPS highlights matching catalog stars by red circles. Also, relations between stars found on image and catalog stars are illustrated by orange lines.

When using also secondary catalog for matching, stars from the secondary catalog are highlighted by magenta circles, in addition to red circles from primary catalog. The image below demonstrates matching with UCAC4 as primary catalog and USNO-B1.0 as a secondary one.

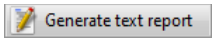


Similarly to highlighting found stars by green circles, also red circles showing catalog star positions partially obstruct original data, make the image difficult to perceive and slows image drawing. Another **View** ribbon option button 

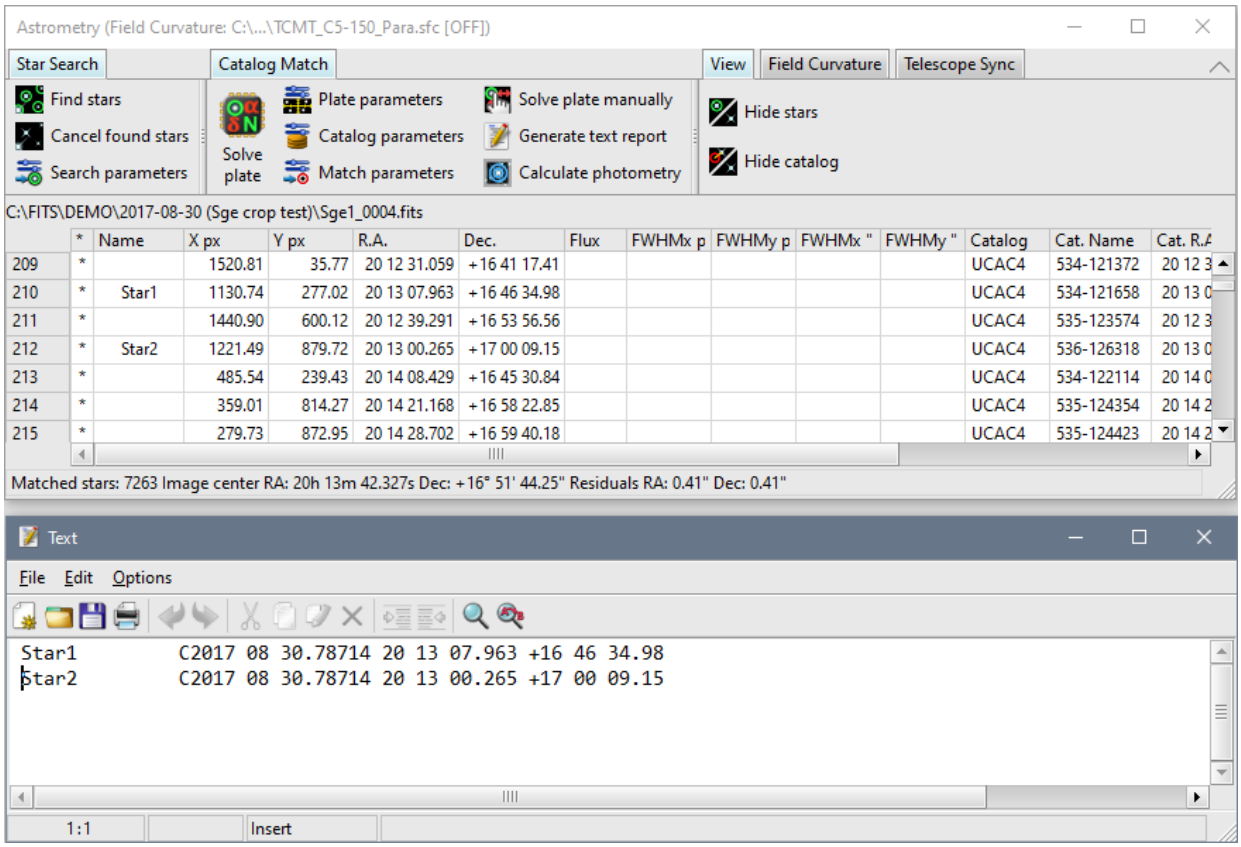
allows to hide the red circles around catalog stars and orange relation lines.

> Hint:
>
> This option button is synchronized between **Astrometry** and **Photometry** tool windows. Catalog highlighting can be turned on or off in any of these tools and changes are propagated to the other opened tool window.

Astrometry report

The 

button opens a new text editor window with a text report containing measured coordinates. However, the report does not contain a list of all detected stars as it would be typically too long. Instead, only named objects are included into text report. Object can be arbitrary named using the **Name** column of the table displayed in the **Astrometry** tool window.



	*	Name	X px	Y px	R.A.	Dec.	Flux	FWHMx p	FWHMy p	FWHMx "	FWHMy "	Catalog	Cat. Name	Cat. R.A
209	*		1520.81	35.77	20 12 31.059	+16 41 17.41						UCAC4	534-121372	20 12 3
210	*	Star1	1130.74	277.02	20 13 07.963	+16 46 34.98						UCAC4	534-121658	20 13 0
211	*		1440.90	600.12	20 12 39.291	+16 53 56.56						UCAC4	535-123574	20 12 3
212	*	Star2	1221.49	879.72	20 13 00.265	+17 00 09.15						UCAC4	536-126318	20 13 0
213	*		485.54	239.43	20 14 08.429	+16 45 30.84						UCAC4	534-122114	20 14 0
214	*		359.01	814.27	20 14 21.168	+16 58 22.85						UCAC4	535-124354	20 14 2
215	*		279.73	872.95	20 14 28.702	+16 59 40.18						UCAC4	535-124423	20 14 2

The text report format follows the Minor Planet Center formatting rules. Each line begins with object name, then a capital letter C indicates the measurement were taken from CCD image (not sure if MPC distinguishes between CCD and CMOS cameras). The time of the image acquisition, Right Ascension and Declination follows.

Syncing with the telescope

The **Telescope Sync** ribbon of the **Astrometry** tool window allows high precision **Go To** operations of the telescope mount. Depending on the telescope mount quality, precision of its polar alignment, telescope tube conic error etc. the long-range **Go To** operation may end of the target by many minutes or even degrees. But even if the telescope **Go To** ends up within a fraction of field of view off the target, it is always useful to precisely center the target object within the field of view.

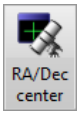
Performing astrometry on an image, captured when the telescope **Go To** move finishes, finds the actual equatorial coordinates, to which the telescope points. The Astrometry tool then allows us to perform three actions:

- 

button invokes the **Sync** command on the telescope mount with newly determined **image center** coordinates. This command sets the mount controller current coordinates, which were of by an error caused by above-mentioned uncertainties. So, after the **Sync** command, the mount knows exact coordinates to which it is pointing and subsequent (very short) **Go To** command to the original coordinates is usually capable to place object in the field of view center with very good precision (the originally set **New R.A.** and **New Dec.** coordinates remained from the original **Go To** command).

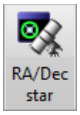
  Hint:

  If is of course possible to repeat this procedure several times to achieve **Go To** precision on the very mechanical limits of the telescope mount.

- 

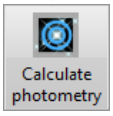
button overwrites the telescope mount **New R.A.** and **New Dec.** with determined image coordinates. It is then upon the user to manually synchronize the mount with new coordinates. While the mount is then

synchronized with the sky, the disadvantage of this procedure is the coordinates of the target object, stored in the **New R.A.** and **New Dec.** controls prior to **Go To** command, are overwritten.

- 

button overwrites the telescope mount **New R.A.** and **New Dec.** with coordinates of the star selected in the active image (or in the **Astrometry** window star sheet – see the Astrometry tool and image windows interactions). This command can be used to center the desired star in the field of view. After the star is selected and the **Ra/Dec star** command is invoked, the **Go To** command should center the star. If the star is not in the image center, precision **Go To,** involving repeated astrometry and telescope mount synchronization with image center (the **Sync center** command), can be used.

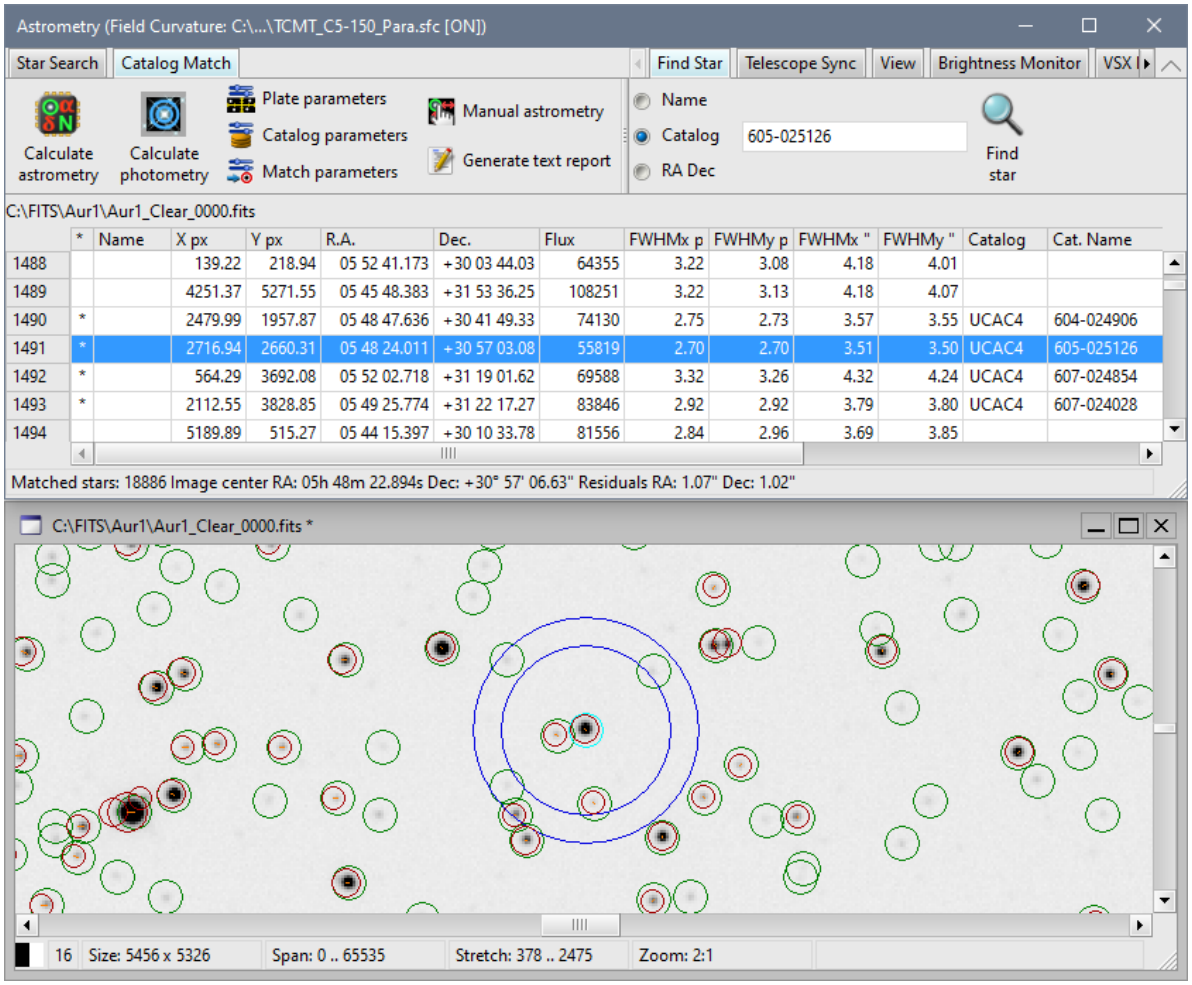
Photometry processing

Despite the photometry processing of a series of images is handled by a dedicated Photometry tool, also the **Astrometry** tool allows calculation of photometry on single image using the 

button. The main purpose of the photometry calculation is to show FWHM of all detected stars. Flux for each star, calculated precisely during photometry processing, is an indication of the amount of light captured from individual stars.

Remark:

Photometry calculation is affected by several parameters, for instance the radii of a ring used to determine a background value etc. These parameters can be defined in a dialog box, opened from the Photometry tool. **Astrometry** tool uses photometry processing only as an optional step in addition to astrometry and thus it does not offer controls for photometry parameter settings.



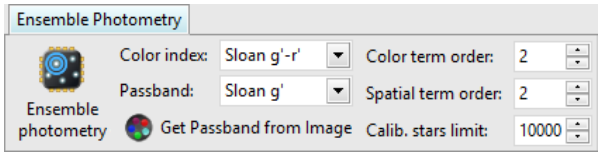
	*	Name	X px	Y px	R.A.	Dec.	Flux	FWHMx p	FWHMy p	FWHMx "	FWHMy "	Catalog	Cat. Name
1488			139.22	218.94	05 52 41.173	+30 03 44.03	64355	3.22	3.08	4.18	4.01		
1489			4251.37	5271.55	05 45 48.383	+31 53 36.25	108251	3.22	3.13	4.18	4.07		
1490	*		2479.99	1957.87	05 48 47.636	+30 41 49.33	74130	2.75	2.73	3.57	3.55	UCAC4	604-024906
1491	*		2716.94	2660.31	05 48 24.011	+30 57 03.08	55819	2.70	2.70	3.51	3.50	UCAC4	605-025126
1492	*		564.29	3692.08	05 52 02.718	+31 19 01.62	69588	3.32	3.26	4.32	4.24	UCAC4	607-024854
1493	*		2112.55	3828.85	05 49 25.774	+31 22 17.27	83846	2.92	2.92	3.79	3.80	UCAC4	607-024028
1494			5189.89	515.27	05 44 15.397	+30 10 33.78	81556	2.84	2.96	3.69	3.85		

When photometry is processed the table displayed in the **Astrometry** tool window is filled also with each star FWHM, calculated independently in the x and y axes, and expresses in both pixels and arc-seconds.

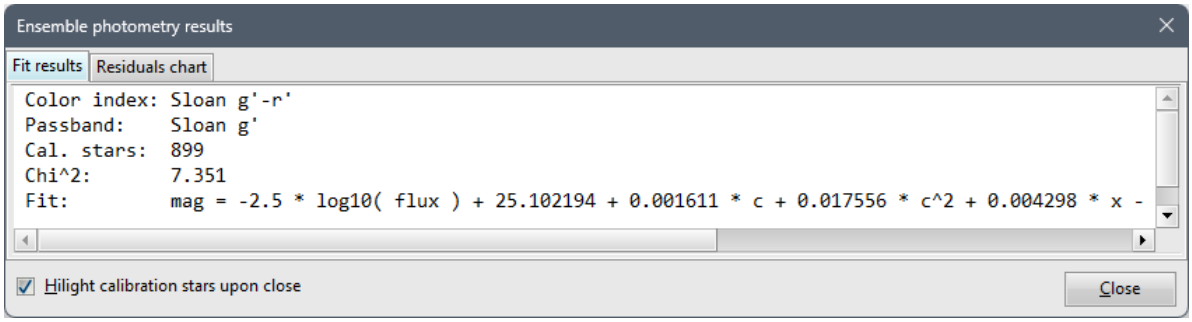
Also, the highlighted star is shown with automatically calculated aperture as well as the background ring. See the **Photometry** tool description for more details.

Ensemble photometry

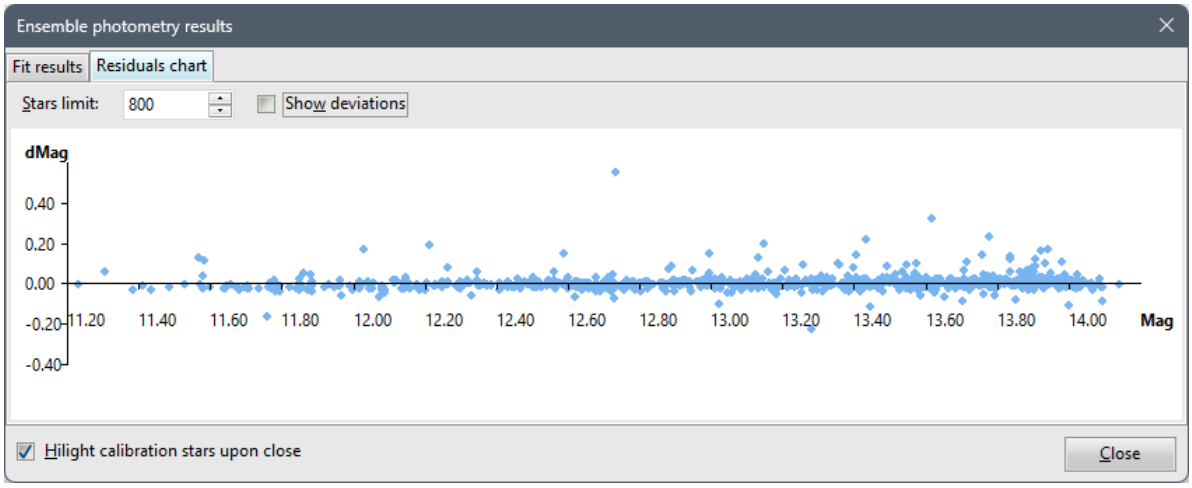
The **Astrometry** tool also supports Ensemble photometry calculation (see the **Photometry** tool description as well as **Appendix C: A short introduction to photometry** for details about ensemble photometry).



The ensemble photometry calculation shows the results in the dialog box, containing two tabs. The **Fit results** tab shows input parameters (Passband, Color index), number of stars used for calibration and also the resulting transformation function.



The Residuals chart shows a dependence of the residuals (calculated magnitude vs. catalog magnitude) on the magnitude:

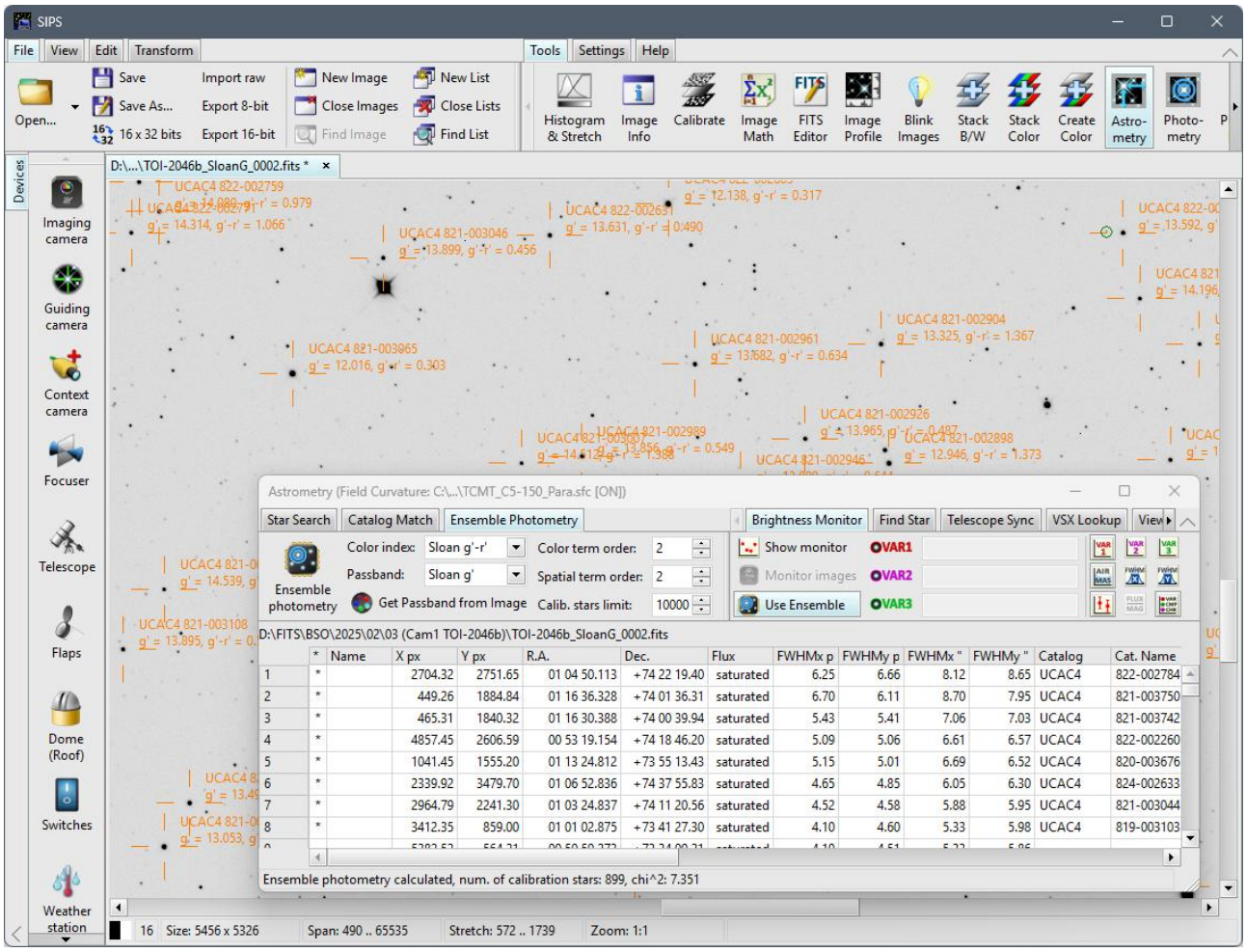


The **Stars limit** count box allows us to limit the chart content to a defined number of brightest stars in image.

Hint:

It is possible to click to any point in the chart. The Astrometry tool then selects the corresponding star in the star sheet and also highlights it in the image.

The **Highlight calibration stars upon close** option can be used to highlight stars in the image:



Remark:

Highlighting calibration stars cancels possible VSX star highlighting.

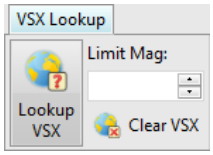
VSX database lookup

The VSX database lookup function is designed for variable star observers. The VSX is a database of known variable stars, so marking stars in the image, which are registered in VSX, can be helpful for observers targeting particular variable star. VSX database query is of course possible only after the plate is successfully solved and photometry is calculated on the image.

Remark:

The VSX database lookup requires internet connection to query the database online.

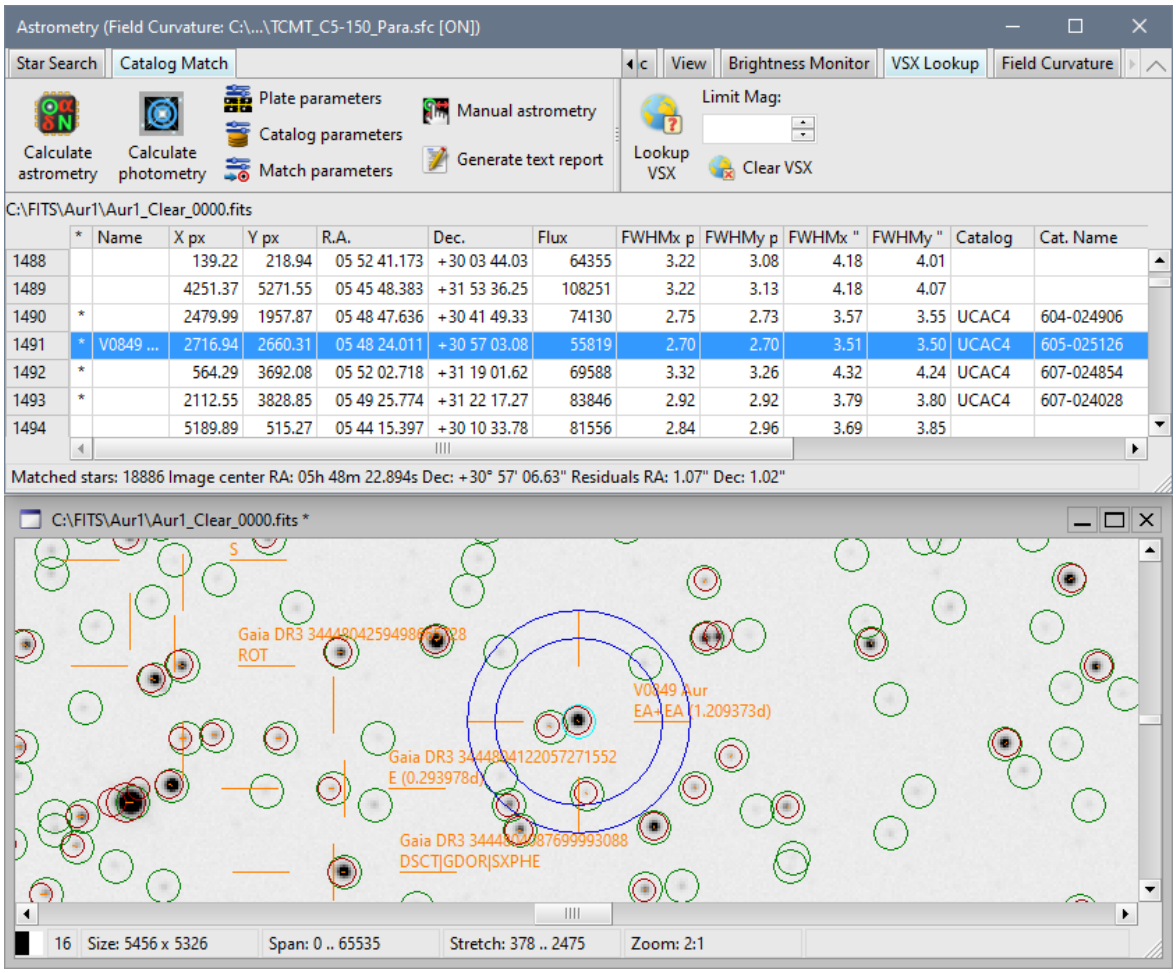
Controls allowing the VSX database query are grouped in the **VSX Lookup** ribbon.



Hint:

The **Name** column is filled with the primary name of the star in the VSX database, providing the Name of the object was empty (not already defined by the user). The name can be then used to find the star using the Find Star ribbon tools.

The **Lim.Mag** count box control allows setting of limiting brightness and is particularly useful when the image is a wide field, containing a lot of stars.



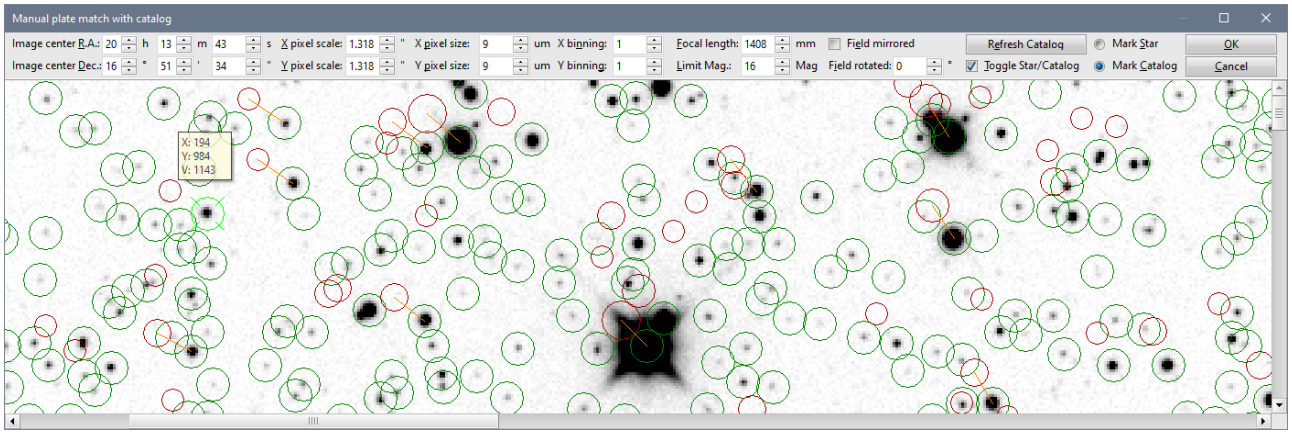
	*	Name	X px	Y px	R.A.	Dec.	Flux	FWHMx p	FWHMy p	FWHMx "	FWHMy "	Catalog	Cat. Name
1488			139.22	218.94	05 52 41.173	+30 03 44.03	64355	3.22	3.08	4.18	4.01		
1489			4251.37	5271.55	05 45 48.383	+31 53 36.25	108251	3.22	3.13	4.18	4.07		
1490	*		2479.99	1957.87	05 48 47.636	+30 41 49.33	74130	2.75	2.73	3.57	3.55	UCAC4	604-024906
1491	*	V0849 ...	2716.94	2660.31	05 48 24.011	+30 57 03.08	55819	2.70	2.70	3.51	3.50	UCAC4	605-025126
1492	*		564.29	3692.08	05 52 02.718	+31 19 01.62	69588	3.32	3.26	4.32	4.24	UCAC4	607-024854
1493	*		2112.55	3828.85	05 49 25.774	+31 22 17.27	83846	2.92	2.92	3.79	3.80	UCAC4	607-024028
1494			5189.89	515.27	05 44 15.397	+30 10 33.78	81556	2.84	2.96	3.69	3.85		

Remark:

The VSX star highlight remains active only while the image is selected. Selecting another image causes the VSX star highlight is canceled.

Manual plate match

If for some reasons automatic plate solving fails, for instance because some bright stars are missing from the catalog or there is a bright diffuse object in the field of view (e.g. core of some galaxy), causing “blind spot” in the catalog etc., it is possible to manually mark pairs of stars on image and stars in catalog. The Solve plate manually button opens the **Manual plate match with catalog** dialog box.



If enough pairs are marked (at last tree), astrometry is calculated the same way like in the case of automatically found star pairs.

The dialog box allows for definition of all plate parameters (image center coordinates, image scale). The **Refresh Catalog** button loads the catalog plate according to defined plate parameters and shows it using red circles in place of catalog stars. To make visual pairing of image and catalog stars easier, the manual matching dialog box also allows for catalog plate mirroring and rotation. Do not forget to click the **Refresh Catalog** button after each change in either parameter (plate center, scale, angle, …) to update the catalog star position.

Corresponding star pairs are marked by clicking a star from image and the clicking the corresponding star on overlaid catalog. Pair is highlighted by orange line, connecting both stars. If some pair is marked by mistake, deleting the relation is performed the same wat like it was created—mark both stars in sequence and the relation is removed.

SIPS marks stars from image only, ignoring catalog stars, when **Mark Star** radio button is selected. And if the **Mark Catalog** radio button is selected, only catalog stars are marked. The **Toggle Star/Catalog** check box allows automatic switching between image and catalog after selection of any particular star.

This mechanism works well when star pairs are marked one after another, but if it is necessary to unmark some pair, the Image/Catalog must be typically selected manually.

Hint:

The image displayed in the manual matching tool window can be manipulated the same way like any other image displayed in window—zoomed by mouse wheel, dragged by right-mouse button etc. E.g. zooming-in makes marking of star pairs much easier.

Field curvature

The image created by a telescope tends to lose quality with increasing distance from the optical axis due to laws of geometrical optics. Telescopes typically create properly focused image on a curved (spherical) surface, while the sensors are flat. With narrow field of view, these aberrations can often be overlooked. But as the field of view increases, these aberrations must be corrected by some refracting elements. Wide field refractors need more lenses to create high-quality wide field images and reflectors need some refracting correctors in front of the camera.

Unfortunately, while the correction elements cause properly focused stars all over the field of view, the position of these stars moves somewhat from the ideal tangential projection. The actual difference between ideally projected star position and real star position is not a simple (linear) relation.

If the actual position difference all over the field of view is only a fraction of pixel, it is not necessary to take it into account. But if the difference is many pixels, it is not possible perform astrometry on such image as the stars on the image are significantly off their catalog positions.

To solve this problem, SIPS includes a capability to calculate the difference between ideal tangential projection and actual projection of the used optics, model it with 3rd order polynomial surface (2D polynomial) and then to use this 2D polynomial to correct the measured star positions. In fact, there are two 2D polynomial surfaces created—one models the differences in X-coordinate, another in Y-coordinate.

SIPS allows usage of two types of polynomials:

- Monomial polynomials
- Legendre polynomials

The Legendre polynomials are designed to be easier to fit. However, typically both polynomial types can fit the field curvature without issues.

Remark:

Please note the Legendre polynomials were introduced in SIPS v4.1.3. If the field curvature description should be useable also in older SIPS versions, always selected the Monomial type.

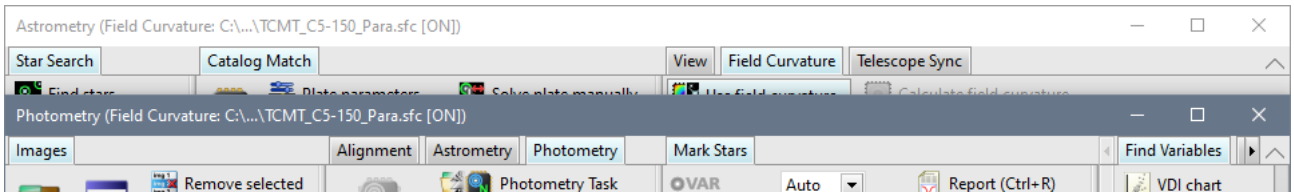
The correction polynomials are calculated by the least-squares method from the differences between actual star positions and corresponding catalog position. But if the differences are too high, it is not possible to match stars with catalog without knowing the corrections and we get kind of Catch-22 problem—to match stars with catalog, we need correction polynomials, but to calculate correction polynomials, we need matched stars with catalog.

The manual plate match, mentioned in the previous sub-chapter, provides a solution. Even if automatic astrometry is not possible, it should be always possible to manually mark image and catalog star pairs (human brain can distinguish matching patterns regardless of field deformation). Of course, it is not necessary (it is even not desirable) to mark all pairs within the image, only a few tens of bright stars, evenly distributed over the field of view, are necessary. It is useful to mark more star pairs especially in the areas, where the difference changes quickly (typically image corners).

All controls related to field curvature correction are available in the **Field Curvature** ribbon on the **Astrometry** tool window.

After the image is manually matched, it is possible to use the Calculate field curvature button to calculate the mapping polynomials. The calculated polynomials can be saved into `*.sfc` file using the Save field curvature button for later usage. The Load field curvature button loads previously saved polynomial definitions.

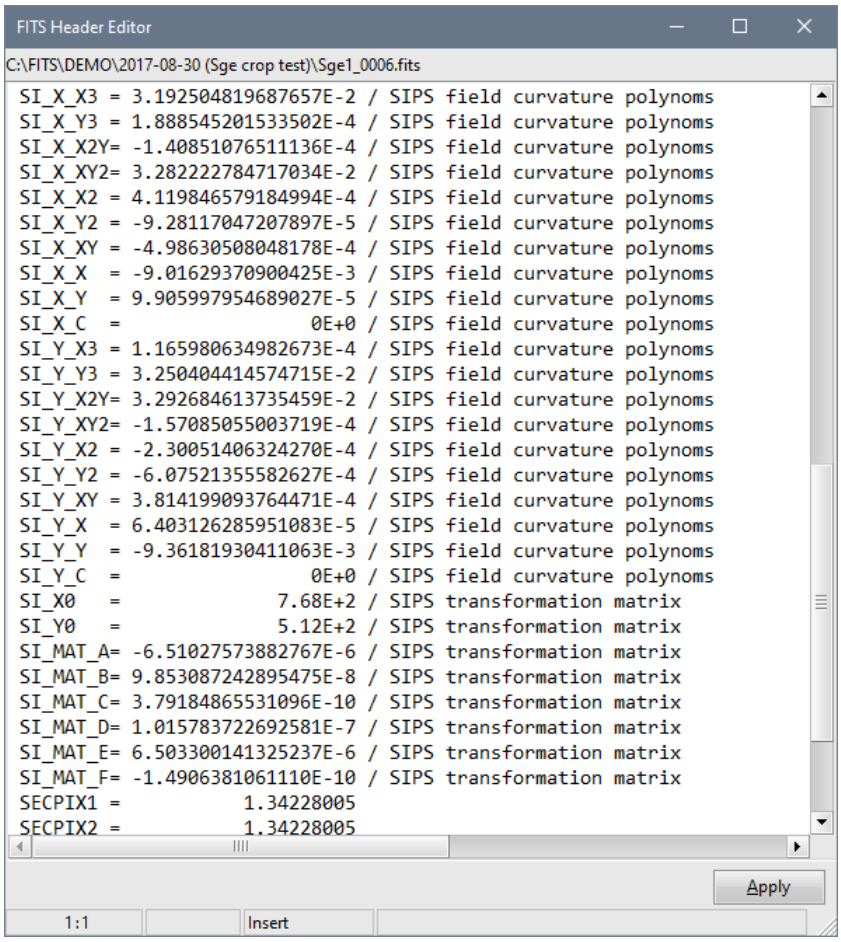
SIPS keeps the currently loaded SFC file in its state and reloads it each time you run SIPS. So, once the field curvature is calculated/loaded, SIPS uses if for all astrometry reduction. The currently active field curvature file is indicated in the title of both **Astrometry** and **Photometry** windows.



Still, it is possible to turn the usage of current field curvature On and Off using the Use field curvature button. The current status (the field curvature is used or not) is again indicated in the **Astrometry** and **Photometry** window title bars.

The Delete field curvature button causes all field curvature information is deleted and not corrections will be applied regardless of the On/Off state.

Note when an image is solved with a filed curvature On, the 2D polynomial coefficients (as well as the transformation matrix between image and catalog stars itself) are stored into FITS headers, so it is always obvious which field curvature description was used to solve particular image.

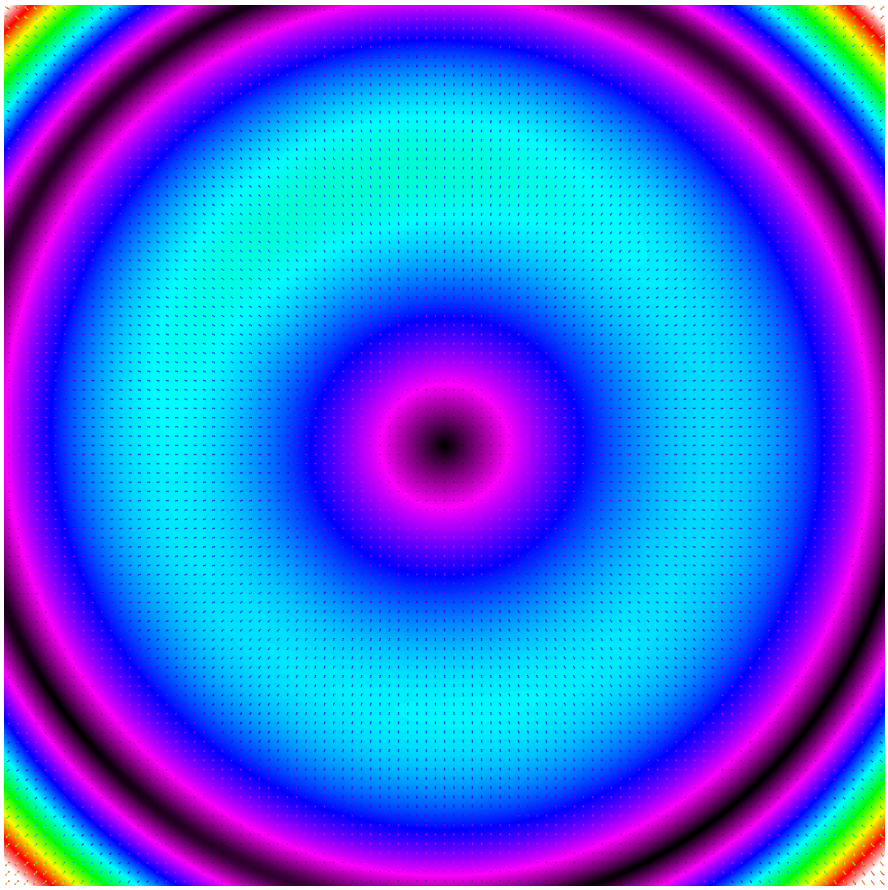


SIPS is able to visualize the actual filed curvature by generating a FITS image, which values represent an absolute difference between ideal tangential projection and actual projection of the used optics. Also, a grid of abscissas of the same length and direction, as is the ideal/real position, is generated into FIST matrix. The abscissas are generated using zero pixels, and thus the other pixels are offset by 32767 ADU (a half of the 16-bit dynamic range).

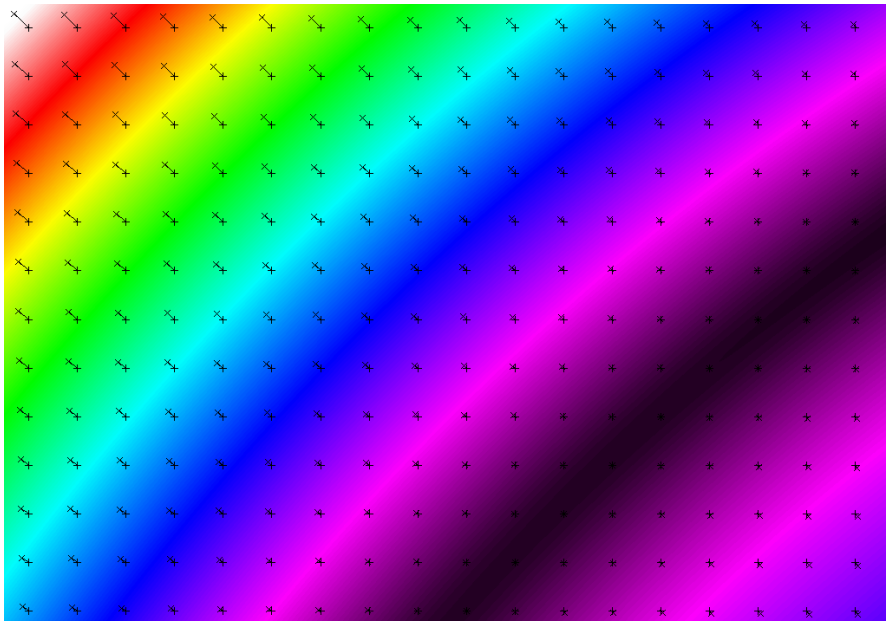
Hint:

False-coloring of the field curvature FITS may help to visually enhance the shape of field curvature.

An example of the filed curvature generated by the TeleVue ParaCorr BIG coma corrector on the 30 cm, f/4 Newtonian telescope.



Crop of the same image showing an upper-left corner of the field deformation illustration with black abscissas illustrating actual image shift from ideal tangential projection.



Hint:

Filed deformation also acts like an indicator of the whole telescope-corrector-camera collimation. Rotationally symmetrical pattern indicates good alignment, while horseshoe-like and otherwise non-symmetrical patterns indicate optics and camera misalignment.

Brightness Monitor

The Brightness Monitor is a feature designed for interactive monitoring of brightness of selected star during observing run. This function is appreciated by researchers (amateur or professional), following some variable star or exoplanet

transit. The light curve, displayed in real time during the course of observation, informs the observer about the current state of the expected brightness variation, like for instance if the transit is still in progress or already finished etc.

The **Brightness Monitor** not only shows the light curve, but also air-mass and FWHM[8] of the monitored star in both x and y axes.

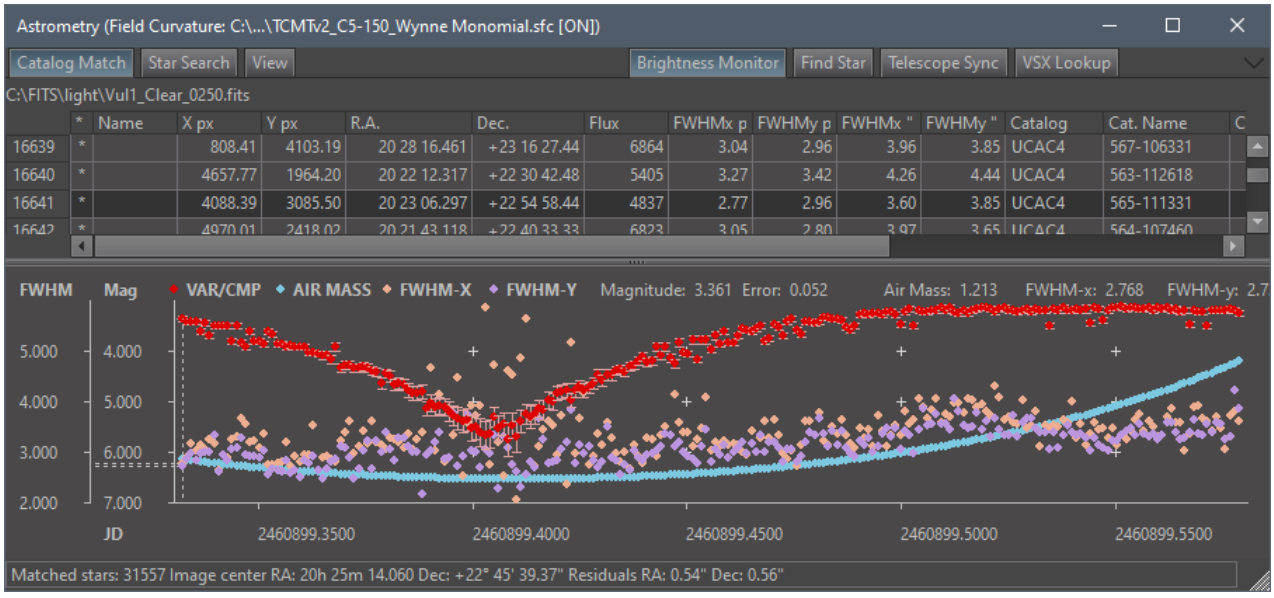
Remark:

To some extent, the **Brightness Monitor** functionality duplicates the **Photometry** tool function to perform photometry on images, newly acquired from imaging camera. It also plots a light curve of the selected star, compared to selected comparison star, through the observing run.

But the **Photometry** tool implements this function for offline processing of the photometric time series, which means all images are kept in memory, including all additional information (matched catalog stars, flux information in multiple apertures etc.) for each image, which could be almost as big as the image itself. The **Photometry** tool needs all this data available to be able to switch variable, comparison, and possible check star, as well as to change used photometric apertures, magnitude or flux representation etc. anytime. This makes the **Photometry** tool particularly demanding for computer resources, like the computer memory. But computers used to control the observation run are typically older and less equipped than computers used for offline image processing. Keeping the full-featured **Photometry** running may be beyond their capabilities and jeopardizes data acquisition by consuming all available memory.

This is why the **Brightness Monitor** feature was introduced into the **Astrometry** tool. As opposite to the **Photometry** tool, no images are kept in memory, and it is not possible to change the marked variable or comparison star and see the different history data. It keeps only a few values for each acquired image (brightness, air-mass, FWHM) ant its memory consumption is then negligible.

The Brightness Monitor is entirely based on astrometry reduction[9]. If the astrometry cannot be performed on acquired images, this function is not available.



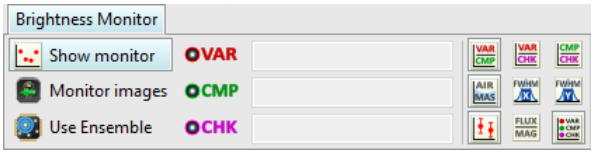
Hint:

The image above shows the **Astrometry** window during real observation, with hidden ribbon panes to preserve screen space and in the dark skin, used by many users at nighttime. See the **Appendix A: SIPS user interface** for ribbon pane controls handling.

[8] Full Width Half Maximum – essentially the "diameter" of a star image, indicating the image angular resolution.
[9] In principle, the brightness history could be implemented by simply matching new images with a stored reference one, without astrometric reduction. But using of astrometry eliminates the need storing of reference image and allows for more flexibility.

To activate the Brightness Monitor, follow these steps:

1. Open the **Astrometry** tool windows and select the **Brightness Monitor** ribbon.



Hint:

If the **Brightness Monitor** ribbon is not visible, right-click the ribbon name bar and select (check) the desired tab name from the pop-up menu opened. This shows the selected ribbon pane. Refer to the Appendix A: SIPS user interface for more details about SIPS GUI handling.

2. Click the Show monitor button to show the **Brightness Monitor** light curve pane below the sheet of stars detected within the active image.
3. Wait until the first image, containing **variable** and **comparison** stars (and if desired, the **check** star, too).
4. Perform the astrometry using the Calculate astrometry button ad wait until the astrometry is calculate din the background.
5. Select the **variable** star, either by clicking it within the image itself or in the star sheet, and mark it as variable using the VAR button. The selected star catalog Id. Is displayed in the text bar following the selection button in the Brightness Monitor ribbon pane.

Hint:

Star can be also found in the sheet based on various criteria, see the Finding stars in the star sheet chapter.

6. Similarly, select the **comparison** star and mark it with the CMP button.
7. While the variable and comparison stars are necessary to show the light curve, selection of the **check** star using the CHK button is optional.
8. The monitoring itself is initiated with the Monitor images button. If the monitoring is turned on, then the **Astrometry** tool waits for each image read from the main imaging camera, then it performs astrometry and photometry reduction of the image and shows the relative magnitude point in the light curve pane. Note there must be at last two images read to show two brightness points, no point is shown after the first image.

Monitoring can by paused at any time by simply unchecking the **Monitor images** button, e.g. when the telescope tube is swapped when the meridian is crossed etc. Checking the button again commences the monitoring.

Note hiding the entire light curve pane, as well as selecting a different star as variable of comparison, clears the light curve history. Once history is cleared, there is no way how to restore it as the **Astrometry** tool do not remember all acquired images. The **Photometry** tool should be used later to process the entire image series to get proper time-based photometry.

Remark:

The quality of the light curve shown by the **Brightness Monitor** is typically worse compared to the light curve generated by the **Photometry** tool from several reasons. For instance, the **Brightness Monitor** does not use the robust mean to calculate photometry ring background values from performance reasons. Also, the automatically determined apertures are not unified across the entire observing series, which limits the precision.

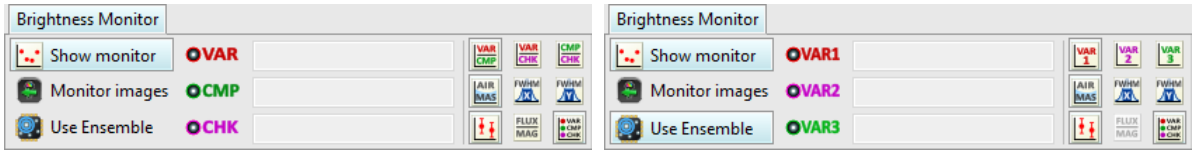
The **Brightness Monitor** ribbon pane allows modification of the light curve pane appearance, like the Light curve pane in the **Photometry** tool. The functionality of 9 buttons in the **Brightness Monitor** ribbon pane is identical to the **Photometry Light Curve** pane tool bar.

Ensemble photometry in Brightness Monitor

The **Brightness Monitor** ribbon pane allows using of Ensemble photometry for brightness monitoring (see the **Photometry** tool description as well as **Appendix C: A short introduction to photometry** for details about ensemble photometry).

> Remark:
>
> Just keep in mind that Ensemble photometry needs some conditions to be met, like observing through supported photometric filter and using astrometric catalog containing magnitudes in the used filters (preferably the UCAC4).

If this option is selected, the **Brightness Monitor** ribbon changes the meaning of some tools:



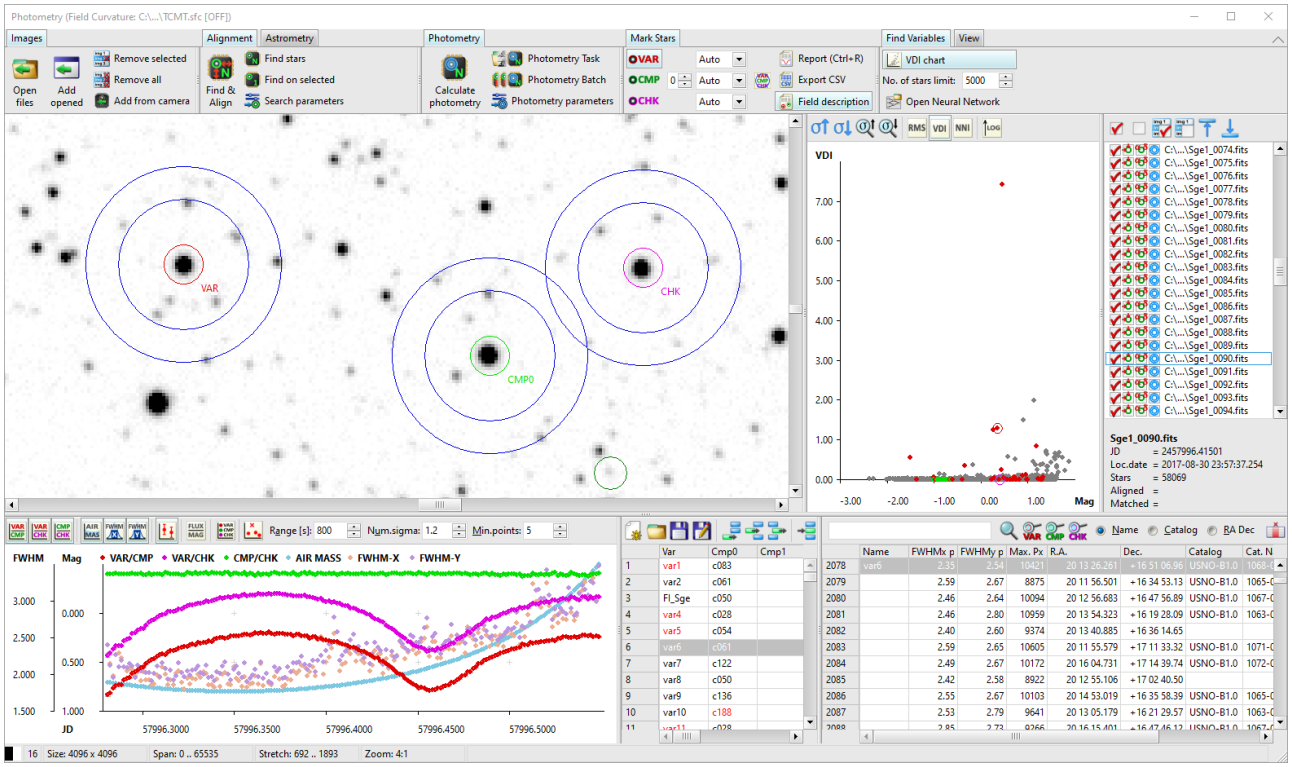
As the ensemble photometry provides absolute magnitudes and does not use comparison stars, the CMP and CHK buttons are useless. So, the **Astrometry** tool changes the respective button function. In the Ensemble photometry mode, the **Brightness Monitor** allows displaying of light curves of three stars, marked VAR1, VAR2, and VAR3, at once. It is upon the user which stars are selected—it is possible to select three variable stars, a variable star and a non-variable one as check star etc.

Photometry (and Astrometry) of image time series

Note:

The **Photometry** tool name reflects its evolution from a tool performing photometry processing of image time series. Over time, full **Astrometry** processing was added to the pipeline and thus the more appropriate name would be **Astrometry and Photometry of image time series**. Unfortunately, such name is too long for launch buttons and the tool window title, so the original name was kept. But keep in mind also astrometry of image time series can be also performed, despite the astrometry step is still optional.

The **Photometry**[10] tool can perform complex astrometric and photometric reduction of a series of images loaded into the tool implicit list and create light curves, save photometry reports etc. This quite complex and powerful tool offers a lot of advanced features like searching for variable stars, photometry tasks for automated processing, creation of field description lists for automatic processing of many variable stars in the field of view and many more.



Time-series photometry is rather complex task, involving many steps—calibration of image series, matching stars on individual images and proper calculation of fluxes (flux is an amount of light, acquired by the sensor from a star during the exposure time) of all detected stars on all images in the list. With the knowledge of individual star fluxes, it is possible to compare the selected star with another comparison star (or stars) to evaluate changes of the star brightness over time. Majority of stars keep the brightens unchanged (possible measured changes are caused by secondary effects unrelated to the star itself, like changes of weather conditions, atmospheric extinction differing in different heights above horizon etc.), but brightness of some stars varies, which is why we call them “variable stars”. There are various causes of star brightness variability (we cannot discuss them here). Either way, the time-series photometry of variable stars reveals huge amount of information about the star (or stars in the multiple-star systems or star and its orbiting planet etc.). And acquiring of time-series photometry is the main task of the SIPS **Photometry** tool.

As the **Photometry** tool is a part of the whole SIPS program, it naturally implements only tasks related to time-series photometry, other functionality (FITS input/output, image calibration including creation of calibration images etc.) are performed by other SIPS tools. Basic FITS image manipulation is described in the **SIPS Images** chapter and image list concepts and operations are described in the **SIPS Image Lists** chapter.

[10] SIPS astrometry and photometry algorithms are described in the Appendix A of the article [Pejcha O., Cagaš P., et al., 2022, A&A, 667, A53]. If the SIPS is used for research work, this article can be cited as reference.

An important part of the photometric processing can be astrometric identification of observed objects (determining of equatorial coordinates and possibly assigning corresponding star catalog identifiers). While astrometry on single image is performed using the dedicated **Astrometry** tool, the **Photometry** tool is also capable to perform astrometry solution, this time on all images in the list. However, some special astrometry-related function (e.g. manual star field matching and calculation of the optics field curvature) can be performed only in the **Astrometry** tool.

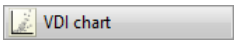
Photometry tool window

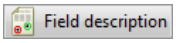
The **Photometry** tool window consists of several panes, which mutual proportions can be altered by horizontal and vertical split bars. Four panes are always present, two other panes can be shown or hidden upon user's request.

Always visible panes include:

- **Image pane** (upper-left) shows the currently selected image. All image manipulations can be applied on this image—for instance user can choose normal or inverted palette, easily zoom (with a mouse wheel), and drag (with right mouse button) the image to overview the whole field or to inspect desired detail etc. (see the **Image pane and image list** chapter below).
  Also, variable/comparison/check stars are marked on the image shown here (see the **Light curve pane** chapter below).
- **Images list** pane (upper-right) shows all images in the current Photometry tool implicit image list. Icons in front of each image indicate the state of processing. Also, more details about the selected images are displayed below the list (see the **Image pane and image list** chapter below).
- **Light curve** pane (lower-left) shows the light curve of the selected variable star compared to selected comparison star. Also, light curves of stars compared to selected check star (if any) and comparison-check stars relations as well as air mass can be displayed (see the **Light curve pane** chapter below).
- **Star sheet** pane (lower-right) shows all known information about all detected stars in the currently active image (the one selected in the Image list pane and displayed in the Image pane). As the processing continues, new information is added (e.g., matched catalog ID, coordinates and magnitude is shown after astrometry, fluxes are added after photometry calculation). Star sheet functionality is described in the **Star sheet pane** chapter.

Optionally displayed panes are:

- **Find variables** pane (upper center). It is displayed when the 

button is pressed and hidden when it is released (pressed again). Selecting of any other image in the list also hides this pane (see the **Finding variables** chapter below).
- **Field description** pane (lower center). It is displayed when the 

button is pressed and hidden when it is released (pressed again). Functionality of this pane is described in the **Field description** chapter.

Photometry processing workflow

The **Photometry** tool implements two ways leading from a set of raw FITS images to light curve of variable star of interest.

1. Experienced users probably prefer the shortest and fastest route—define **Photometry task**, possibly save it for future reuse, and run it.
2. The second (longer) way of photometry processing involves invoking of individual steps manually. This way allows checking results of individual steps and tuning of parameters.

Using photometry task is effective only if the whole sequence of photometry processing steps is already tested and tuned. This means the parameters for searching stars are properly defined for the used telescope and camera (image sampling, quality of optics and focus, …). If necessary, field curvature is calculated and used. Astrometry catalog(s) are defined, and astrometry works well on images (FITS files contain at last approximate coordinates of image center, as well as approximate image scale or pixels size, binning and focal length are defined) etc.

Photometry task

Photometry task performs complete photometry processing:

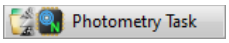
- Loads images into image list.
- Possibly calibrates them with defined master calibration images (dark frame and flat field).

- Mutually aligns all images.
- Possibly performs astrometry solution of all images.
- Calculates photometry.
- Possibly calculates Ensemble photometry.
- The task could end with a field description file loaded and ready for use. So, the user can just inspect light curves of variable stars defined in the field description and save photometry protocols.

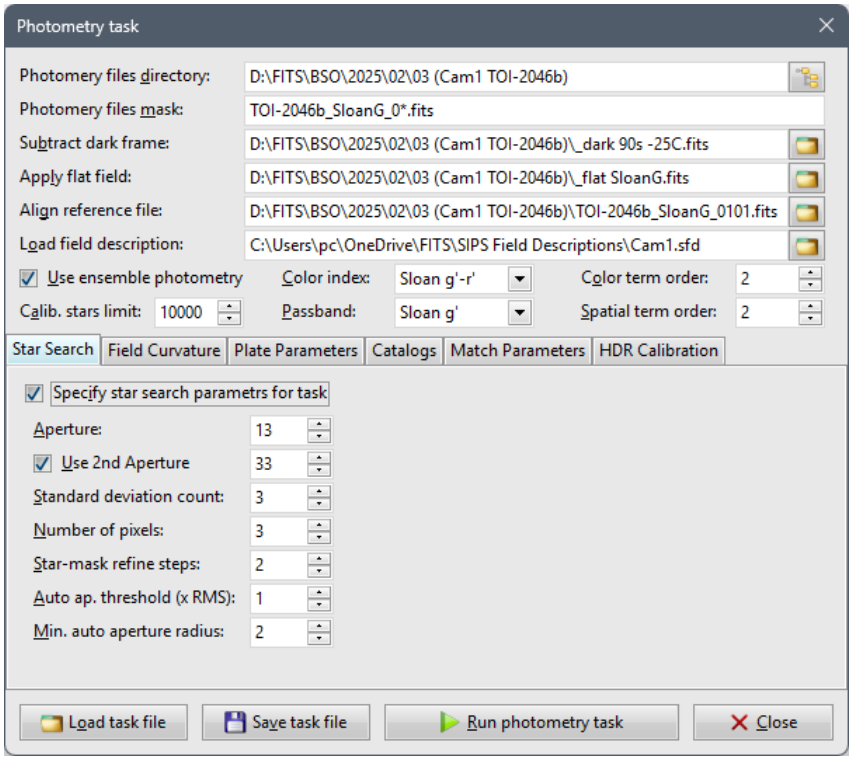
To do so, the photometry task needs to know the location (folder) of images and file name mask of data FITS files (to exclude possible other FITS files like dark and flat images, located in the same folder) to be processed. If the images are not calibrated, it is possible to define paths to master-dark and/or master-flat images.

Then it is necessary to define reference frame for alignment. If the reference frame is not defined, the task run is interrupted by a dialog box, asking the user to choose reference frame (see the **Automatic image alignment** sub-chapter of the **Blink Images** tool description for details).

Optionally it is possible to define field description file to be loaded upon task finish.

All these parameters can be defined in the **Photometry task** dialog box, opened with 

button in the **Photometry** ribbon.



Hint:

The **Photometry task** dialog box analyzes the content of the **Photometry files directory** upon choosing some directory and tries to fill subsequent controls automatically, providing respective controls are still undefined (empty):

Photometry file mask is created if more than a half of all file names in the specified directory begin with a common sub-string.

Subtract dark frame is filled with a file containing a substring “dark” in its name, if any.

Apply flat field is filled with a file containing a substring “flat” in its name, if any.

Align reference file is filled with a file in the middle of all files matching the **Photometry file mask**.

Photometry task can be saved into *.spt file for later reuse. Running the task closes the dialog box and any unsaved task description will be lost.

Hint:

SPT files are text files, following standard INI file conventions. It is possible to edit them, but the knowledge of individual parameters meaning is necessary not to make the SPT file unreadable for the SIPS Photometry tool.

As the photometry task performs numerous operations, affected by various parameters (e.g. star search apertures, astrometry catalogs, …), the task by default uses currently set values of all parameters. However, different camera/telescope setups may require different parameters. Also, some necessary data may be missing in processed images. This is why the task description is capable to override default parameters and define different values, used during the processing of the particular task only.

These task-specific parameters can be defined in individual tabs at the bottom of the Photometry task dialog box. Note every set of parameters begins with a checkbox, which defines if the corresponding set of parameters is defined by the task or the values currently defined in SIPS are to be used.

- **Star Search** tab allows definition of parameters affecting searching of stars in images.
- **Filed Curvature** tab can be used if currently chosen filed curvature may belong to different camera/telescope than the ones used to capture processed images.

  Hint:

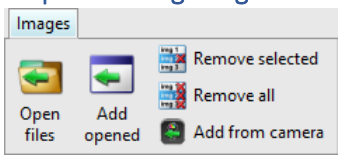
  It is also possible not to define filed curvature for a task while some field curvature description is used in SIPS by default. Just check the **Define field curvature** checkbox and leave the **Field curvature file** box empty.

- **Plate Parameters** tab is useful if the processed images do not contain information necessary for astrometry solution (image center coordinates and pixel scale). It is possible to define these parameters for the first image as well as to let the Photometry tool to reuse them for all subsequent images in the processed list.
- **Catalog** tab allows definition of astrometry catalogs.
- **Match parameters** tab allows modification of parameters affecting image and catalog matching.
- **HDR Calibration** tab is intended for processing of data set taken with dual-gain cameras. The default photometry task description defines only one master-dark and one master-flat file, while dual gain cameras need low-gain and high-gain dark and flat files. See the **Calibration** chapter for details.

Step-by-step photometry processing

The step-by-step processing needs the user to invoke individual steps using command buttons and other controls in the respective ribbons.

Step 1: Loading images into list



Similarly to some other SIPS tools (e.g., **Blink Images** tool or **Combine Monochrome Images** tool), the **Photometry** tool contains implicit image list (the upper-right pane). Time series photometry is performed on images contained in this list.

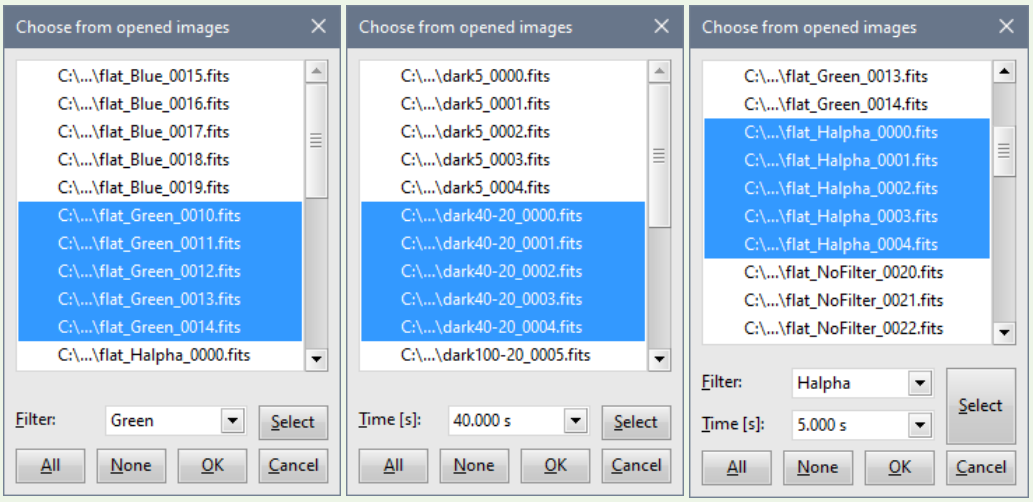
It is necessary to load calibrated images into **Image list** of the **Photometry** tool. Some users calibrate images, store them on disk and discard raw data. In such case it is enough just to load already calibrated images to image list using the **Open files** button. However, calibration is very fast in SIPS and typically needs a fraction of second or a few seconds, depending on number of images and their resolution. It is worth storing raw images and calibrate them prior to processing, as any possible error (e.g., using of wrong dark or flat file etc.) does not permanently harm data. Image calibration is described in the chapter **Calibration** above.

The Result of image calibration is typically a list of calibrated images. These images should be just added to the Photometry tool Image list using the **Add opened** tool button.

Hint:

The **Add opened** function includes images already opened in SIPS, be it in another list or as image windows. As the photometry is performed on images taken with different filters individually, it is necessary to include only images

taken with single filter into Photometry tool image list. The **Choose from opened images** pick list allows selection based on filter and/or exposure time.



More details are explained in the SIPS Image Lists chapter.

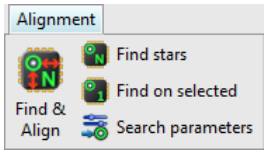
If the **Add from camera** option button is checked, images downloaded from the camera will be added to **Photometry** image list and processed. If variable and comparison stars are already marked, new brightness point is added to displayed light curve. This allows monitoring of the star brightness while the observation is running.

Remark:

The image added to photometry is calibrated according to current state defined in the Calibration tool. Make sure the defined dark and flat images correspond to the images read from camera.

Be aware the images added to Photometry list box remain in computer memory. Make sure the computer controlling the observation has enough memory to keep all acquired images during observing session. Also, data structures keeping found stars on the image and corresponding catalog stars etc. need a room in computer memory.

Step 2: Aligning images

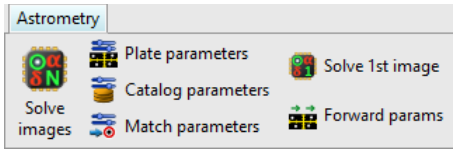


The next step is finding stars in images and align all images in the list. Both finding stars and aligning of images can be done by clicking the **Find and Align** button in the **Alignment** ribbon. Proper alignment depends on properly found stars in all images.

Hint:

Alignment is thoroughly described in the Blink Images tool description, which also relies on matching of images in list. The Finding Starts and Automatic image alignment sub-chapters explain all details, it is recommended to read them first.

Step 3: Astrometry solution



Astrometry ribbon invokes astrometry of one or all images in the list.

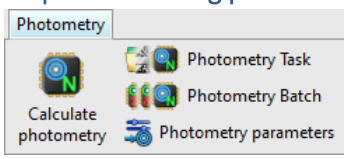
While the Astrometry tool provides a workaround for the case the image to be solved does not contain necessary information—the **Plate parameters** dialog box allows explicit definition of such parameters. But as photometry processes a list of images (typically hundreds), it is not feasible to update plate parameters for all images by editing

them in dialog box. So, the **Photometry** tool allows definition of plate parameters for the first image in the list only and propagating them to all following images in list—click the **Forward parameters** option button.

This step is optional. Light curve can be obtained by manual marking of variable star, comparison star (or multiple stars) and possibly check star in the image without astrometry solution of images. However, some advance functions like using of **Field description**, **VSX lookup** etc. will not be available.

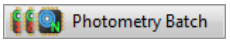
For thorough description of the astrometry in SIPS, see the **Astrometry** chapter, please.

Step 4: Calculating photometry



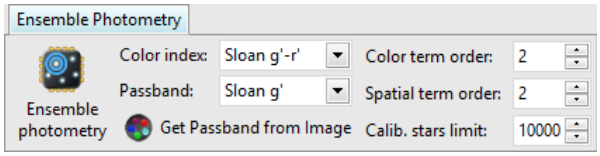
The **Calculate photometry** button in the **Photometry** ribbon calculates fluxes of all stars on all images. Fluxes are always determined for a defined aperture around star. Proper aperture for each star should be great enough to contain all pixels belonging to the star, but also should not be greater, as the presence of background pixels in the aperture increases flux noise.

Steps 2, 3, and 4 at once: Photometry batch

The 

button is intended to speed-up photometry processing, providing an image list is already filled with calibrated images. It invokes image alignment, astrometry solution and photometry calculation without the need to confirm each step result and starting next step by the user.

Optional step 5: Ensemble photometry calculation

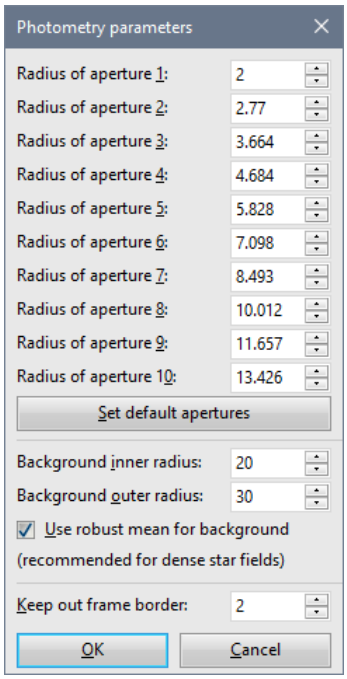


The ensemble photometry provides directly absolute magnitude values for all detected stars within all images. No comparison stars are used, which makes further processing much easier.

But keep in mind that ensemble photometry requires some prerequisites, like observing through supported photometric filter and using astrometric catalog containing magnitudes in the used filters (preferably the UCAC4). Also, a wide field setup, which provides images with enough calibration stars, is a big advantage for ensemble photometry. Refer to the **Appendix C: A short introduction to photometry** for details about ensemble photometry.

Photometry calculation

Because each image contains stars of various brightness and they appear of different diameters on each image, there are 10 predefined apertures (or radii), for which a flux of each star is calculated. It is then upon the user to choose proper aperture for each star.



Remark:

Theoretically the Point Spread Function (PSF) of every star on image, bright or dim, has the same profile width (typically similar to Gaussian curve), only the bright stars PSF is significantly higher than the PSF of dim stars. So, in theory, the aperture for all stars should be the same. But in reality, this is not the case, the wings of the Gaussian profile of dim star decreases below the background noise or even below the theoretical limit (read noise of the camera) much faster than the wings of the bright star profile. So, bright star diameters are much greater compared to diameters of dim stars.

SIPS determines optimal star aperture (aperture containing all pixels belonging to the star, but no more background pixels) during the process of finding star on every image, called **automatic aperture**. In addition to 10 predefined apertures, the photometry calculation also calculates flux for the automatic aperture of every star. Typically, the automatic aperture is the best one to be used for photometry.

The algorithm used to determine automatic aperture for a star scans the profile from the centroid in the four directions (up, down, left, and right) and measures the distance from the centroid to the first pixel lying below the user-defined threshold (see the **Finding stars on image** sub-chapter of the **Astrometry** tool description).

It might happen that automatically determined radii of nearby stars overlap. This is not desirable in photometry, as including of the flux of nearby star significantly compromises photometry precision. SIPS detects and eliminates overlapping radii and calculates new radii as the fraction of the Euclidean distance of star centroids corresponding to the ratio of the fluxes of the two stars.

There is also the lower limit for the automatic aperture in the case the calculation results in too small value. The smallest auto aperture default value is set to two pixels, but this parameter is also adjustable by the user.

To find the automatic aperture for a set of images, SIPS sorts the images according to the radius of the automatically determined aperture for every star. Then it chooses the third largest aperture. This empirical rule is intended to select the largest aperture to cover the whole star when, for example, seeing changes during an observation session. At the same time, the algorithm removes up to two extreme values caused, for example, by telescope mount tracking glitches or similar random events.

Background inner and **outer radii** define the ring around each star, from which the background mean value and standard deviation is calculated. The inner ring radius is always larger than the biggest aperture used to calculate photometry.

The SIPS Photometry offers two methods of background mean and standard deviation calculation.

- The first method uses the previously calculated bitmask, which allows skipping pixels belonging to stars. Background statistics are then calculated only from pixels between the two radii, which are also not flagged in the corresponding bitmask (that means, are not part of another detected star). To eliminate random and mostly single-pixel artifacts (e.g. residuals after hot pixels etc.), all pixels above or below the average plus or minus 3-times the standard deviation are rejected as outliers. The newly calculated mean and standard deviation are then taken as valid values for background.
  While this method is fast and reliable for sparse star fields, it is not robust enough for dense fields with many stars within the background ring or when artifacts (e.g., diffraction spike from a nearby bright star, sub-atomic particle, or satellite trace) significantly affect the background ring.
- If the **Use robust mean for background** is checked, SIPS uses slower but more robust mean calculation using the Hampel influence function. This method is especially recommended for dense star fields.

The **Keep out frame border** parameter allows definition of number of pixels around the frame, which are excluded from any calculations (star fluxes, background statistics). Especially older CCD cameras return images with various artifacts close to the image edges (e.g. in the first few digitized lines). This parameter ensures such artifacts do not negatively influence the photometry.

Ensemble photometry calculation

The Ensemble photometry calculates directly absolute magnitudes using a flux-to-magnitude transformation function, calibrated on every captured image. SIPS can use several transformation functions, differing in **color** and **spatial term orders**.

The simplest transformation function available is:

$$Mag = -2.5 \cdot log_{10}(F_V) + q + c_1 \cdot CI$$

The F_V is the variable star flux and *CI* is its color index. The transformation with the higher order terms available, including the spatial terms, looks like:

$$Mag = -2.5 \cdot log_{10}(F_V) + q + c_1 \cdot CI + c_2 \cdot CI^2 + x_1 \cdot X + y_1 \cdot Y + x_2 \cdot X^2 + y_2 \cdot Y^2 + xy \cdot XY$$

Again, the F_V is the variable star flux and *CI* is its color index; but the transformation above uses also an object position within image *X* and *Y*. Free parameters are:

- q: offset
- c_1: first order color coefficient (mandatory)
- c_2: second order color coefficient (optional)
- x_1, y_1: first order spatial coefficients (optional)
- x_2, y_2, xy: second order spatial coefficients (optional)

The ensemble photometry relies on astrometry catalog, so the astrometry calculation is a mandatory step is ensemble photometry is to be used.

> Remark:
>
> An astrometry catalog, used for astrometry solution, must contain magnitudes for selected color index and passband. USNO catalogs, created from “red” and “blue” Palomar plates, do not contain standard magnitudes. UCAC5 catalog contains Gaia G magnitude, which unfortunately does not correspond to any standard passband, and Cousins R magnitude, which is not enough for ensemble photometry. So, UCAC4 is currently the only catalog, containing magnitudes in many Johnson-Cousins and Sloan passbands, available in SIPS.

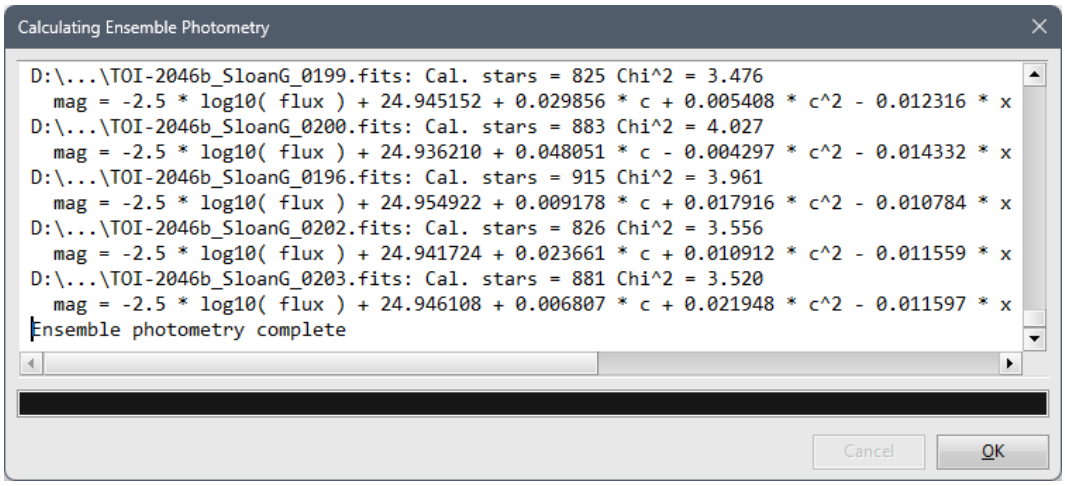
The Ensemble photometry ribbon contains controls allowing the user to select passband and color index, as well as the order of both color and spatial terms.

- **Color index** can be selected from:
  - Johnson B – V
  - Sloan g’ – r’
  - Glass J – K
- **Passband** can be chosen from:
  - Johnson-Cousins U, B, V, Rc, Ic

- Sloan u', g', r', i', z'
- Glass J, H, K

- The **Get Passband from image** button analyzes the FILTER string of the image and tries the determine used passband. As the filter naming is not standardized and the actual FILTER string may vary (for instance, the Sloan g' filter string may be just "g", "g'", "Sloan g", "Sloan g'", "Sloan G", "SloanG", "SG", etc.), always check if the passband was selected correctly.
- The **Color term order** can be in the range [1..2]. This means the transformation contains at last linear dependance on color index.
- The **Spatial term order** can be in the range [0..2]. If the spatial term order = 0 then the star position within the image does not affect the transformation. Spatial term order = 1 calculates with linear dependence and is suitable e.g. for gradients. Spatial term order = 2 calculates with quadratic dependence and can cover for instance vignetting.
- The **Calibration stars limit** defines the maximum number of calibration stars used for transformation function fitting. However, typically the issue is a lack of enough calibration stars, not too many of them. Keep this parameter at last on sever thousand stars, which ensures good ensemble photometry precision.

The Ensemble photometry calculation progress is indicated in the dialog box, showing results for each processed image:

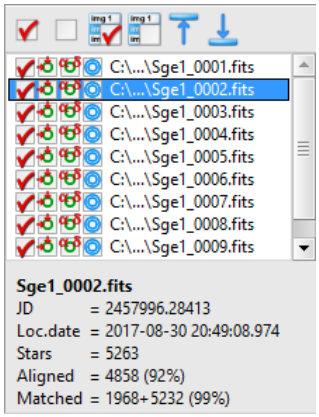


Number of stars used for calibration, χ^2, and fitted transformation functions are displayed for each image in the time series.

Image and Image List panes

The **Image pane** (upper-left) and **Image list pane** (upper-right) are mutually connected. The implicit image list should contain all images to be processed for time-series photometry, while the **Image pane** always displays the currently selected image.

The **Image list pane** consists of a tool-bar, image list itself and image information pane.



Tool bar contains list-related commands:

- Check all images in list. Keyboard shortcut <Ctrl>+<A>.
- Un-check all images in list.
- Check images currently highlighted in list.
- Un-check images currently highlighted in list. Keyboard shortcut <Ctrl>+<U>.
- Select (show in **Image pane**) first image in list. Keyboard shortcut <Ctrl>+<F>.
- Select (show in **Image pane**) last image in list. Keyboard shortcut <Ctrl>+<L>.

Individual image names in the set are preceded with an icon, consisting of four boxes.

- The first box indicates whether the particular image is included into time-series photometry. This means if the image has a point in the light curve, is included into text report etc. Toggling the check box on and off is possible with mouse double-click or by pressing the <Enter> key when the particular image (line in the list) is selected.

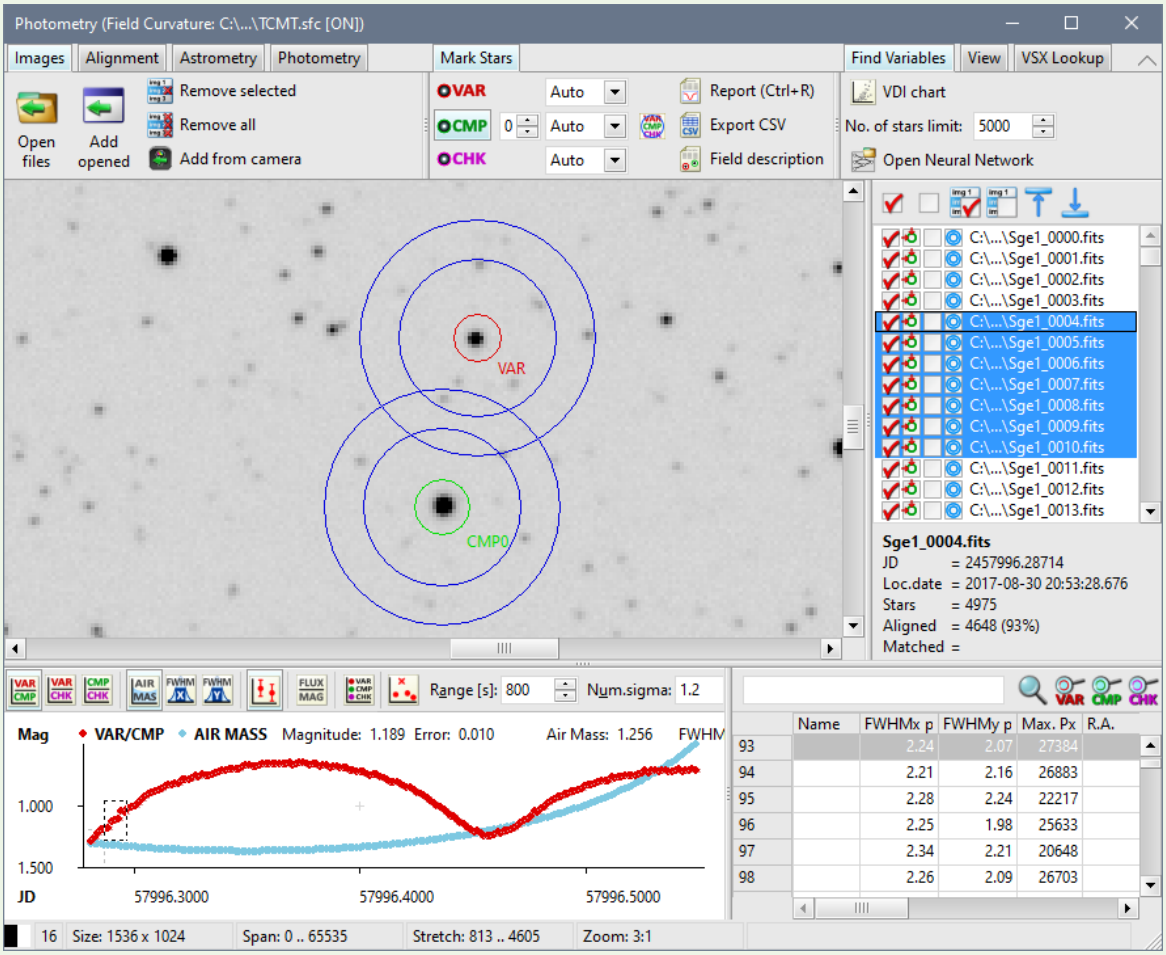
  C:\...\test PM\TY_Tri_0000.fits
  C:\...\test PM\TY_Tri_0001.fits

  The first image is included into light curve, the second one is excluded.

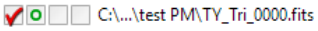
  Hint:

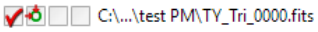
  There are other ways how to exclude any specific image from processing—it is possible to double-click the point corresponding to the image in the **Light curve** pane. This unchecks the image and it is no longer used.

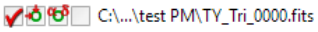
  Another possibility is to frame group of points in the **Light curve** pane using mouse—press the mouse left button and draw a frame around points in the **Ligh curve** pane, images corresponding to the framed light curve points will be selected. Pressing <Ctrl>+<U> unchecks (excludes from light curve) all framed points.

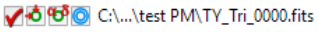


The **Image list pane** also contains toolbar allowing check or uncheck all images or just selected images (image list allows for selection of multiple images).

- The second box indicates whether the stars are found within the image with a green circle:
  

  If stars are found and images are mutually aligned, a green circle is supplemented with arrows:
  

- If the image is successfully solved and matched with catalog, icon with α and δ close to green circle in the third box occurs:
  

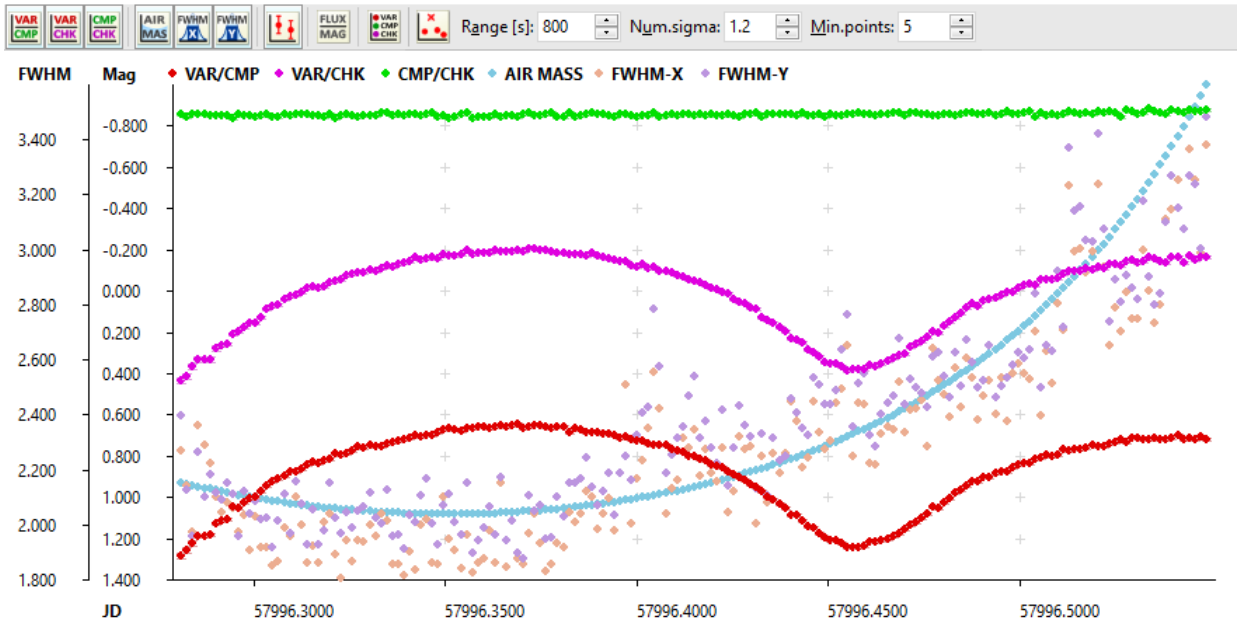
- And finally, when a photometry of every star is calculated, a symbol of two blue circles in the fourth box of every icon is shown:
  

There is an information pane below the list box itself, showing detailed information about currently selected image:

- The pane always displays selected **image name** first.
- **JD** shows Julian date of the middle of image exposure time.
- **Loc. date** shows the date and time of the middle of image exposure, but in local date and time.
- **Stars** shows total number of stars detected in the selected image.
- **Aligned** shows how many stars were aligned with the reference image.
- **Matched** shows how many stars were matched with the catalog. If astrometry uses two catalogs, number of stars matched from both catalogs are shown individually.

Light curve pane

The **Light curve pane**, located in the bottom-left portion of the **Photometry** tool window, shows the charts of the instrumental magnitude or relative flux of the selected variable star through the time series.



Remark:

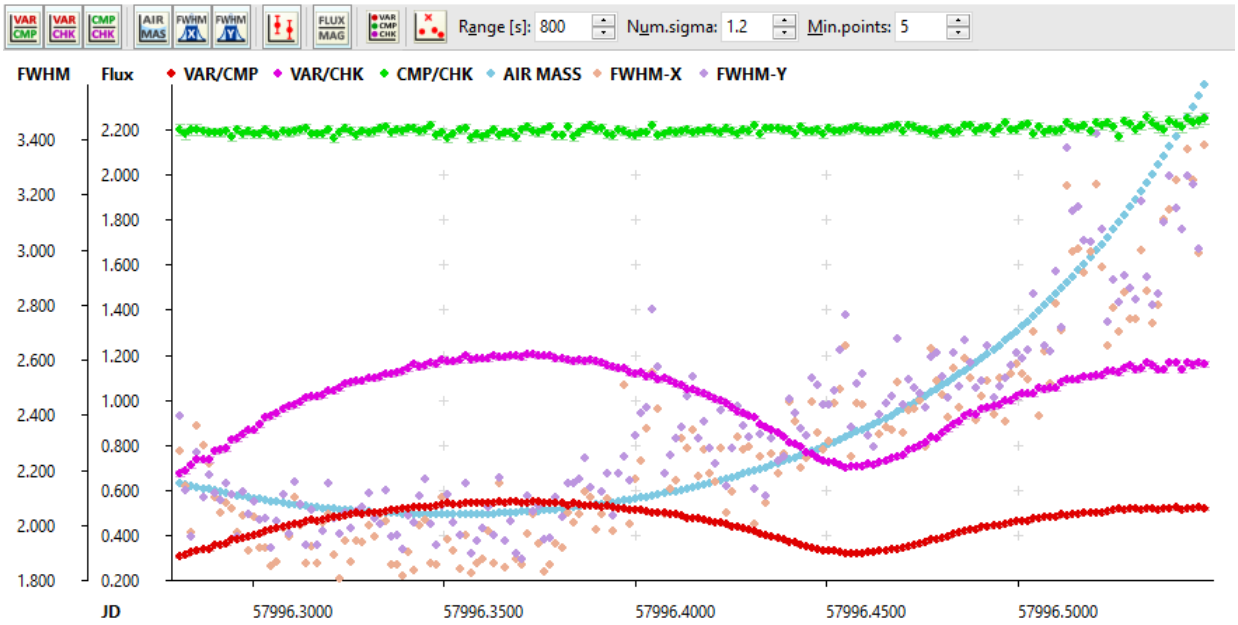
The **Light curve pane** needs the images in the **Image list** are at last aligned, photometry is calculated and variable and comparison stars are marked. Otherwise, no light curve is shown.

Light curve may not be show also because the marked variable and/or comparison star saturate on all images. If even a single pixel of variable of comparison star saturates, SIPS invalidates the photometry point because there is no way how to determine the amount of lost flux in the saturated pixels and thus the point is inevitably wrong.

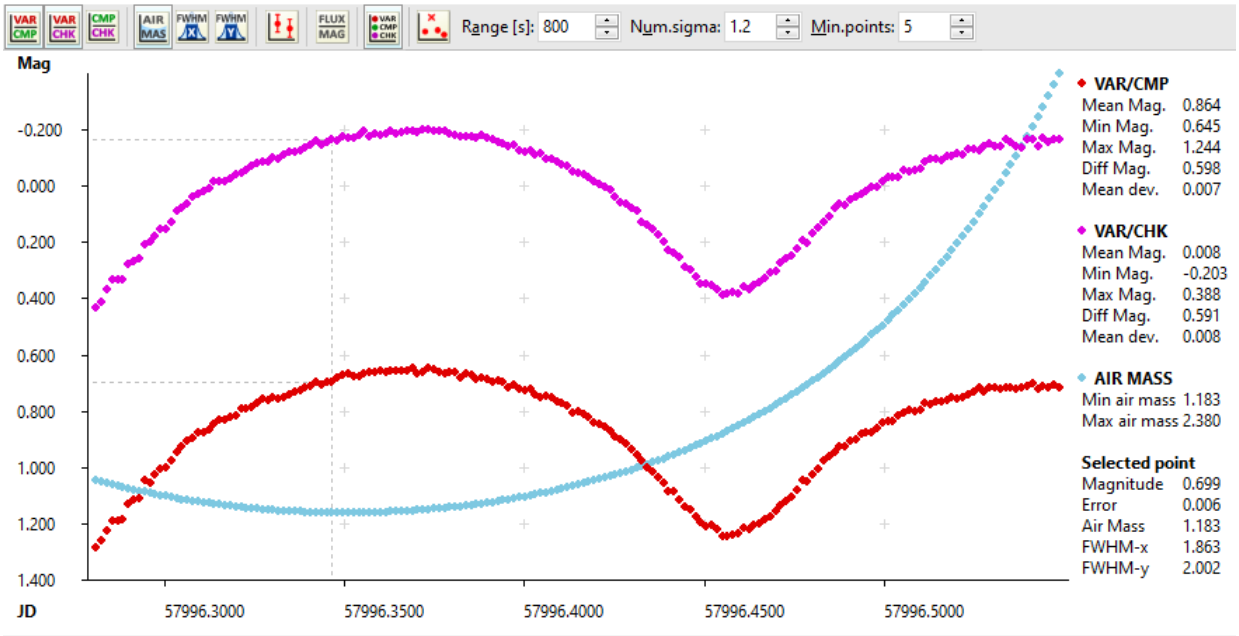
The toolbar on the top of the pane contains numerous switches, affecting the appearance of the charts shown.

-  Show chart of variable star compared to comparison star.

- Show chart of variable star compared to check star.
- Show chart of comparison star compared to check star.
- Show chart of air mass for every point shown. Air mass is calculated only if equatorial coordinates of every image is known and the location of the observatory is defined. Only then it is possible to calculate azimuthal coordinates of every image and the air mass can be calculated.
  The air-mass is calculated using the Pickering (2002) formula from the angular height above horizon.
- Show the star FWHM (in pixels) in the image x-axis direction.
- Show the star FWHM (in pixels) in the image y-axis direction.
- Show error (uncertainty) of each photometry point.
- Switches the used brightness unit from magnitudes to relative flux. Because the magnitude is a logarithmic unit, it also affects the shape of light curve. Also, because logarithm of multiply is a sum of logarithms, scaling brightness expressed in magnitudes is performed by adding a number, while scaling of flux need multiplication by a number.
  For illustration, compare the example of light curves in magnitudes, shown on the image above, with the same light curves shown in relative fluxes:

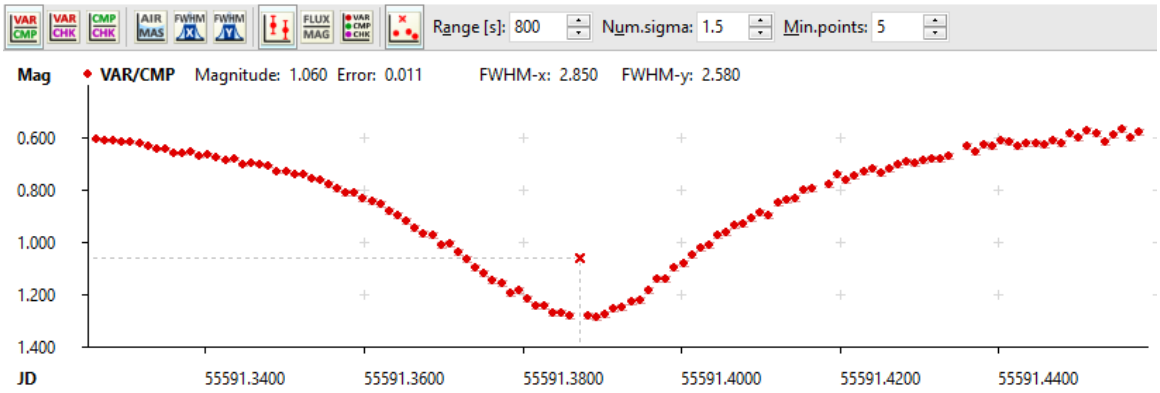


- Switches if the char legend is displayed in the brief form on the top of the light curve pane or in more details on the right side of the pane. The example below shows detailed legend:

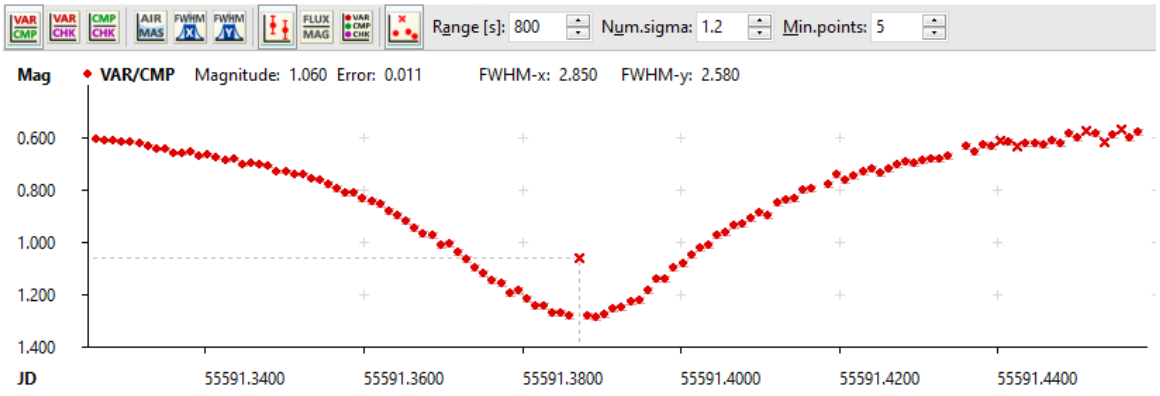


The legend displays ranges of individual series and if a photometry point is selected, its detailed values are also shown in the legend. The chart also displays gray dashed lines showing the exact position of the selected point.

- Turn on outlier filtering. The filtering is based on simple evaluation of the distance of photometric point from the quadratic function $y = ax^2 + bx + c$, fitted to an interval of photometric points. The interval is defined as time range (in seconds) around the current point. The standard deviation is calculated for the distance of all points within the defined time range from the fitted parabolic function and if the current point exceeds its distance by the defined multiply of standard deviation, it is marked as outlier.
  Points fulfilling the outlier criteria (which are too far from the fitted parabola) are marked with cross instead of circle.



The image above illustrates the obvious outlier marked with cross. If the outlier filtering is turned on, points marked as outliers are excluded from the **Photometry protocol**.
Lowering the multiply of standard deviation causes more photometric points do not fulfill the criteria and thus marked as outliers and excluded from the light curve.



The sigma multiply was lowered from 1.5 to 1.2 in the image above, which causes ale some points at the end of the light curve are marked as outliers.

Warning:

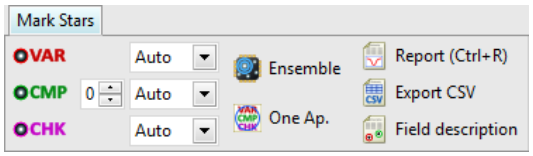
Use outlier filtering with caution. Sometimes it is necessary to adjust the range (for instance, the range should be smaller for fast physical variables and longer for slowly changing, long period variable stars) and multiply of standard deviation to suit the series sampling cadence etc. Also, the filtering could possibly cut real phenomena, like very short star flares.

Marking stars

Marking stars differs in **Relative photometry** and **Ensemble photometry** mode.

Marking stars in Relative photometry mode

It is necessary to properly mark individual types (variables or VAR, comparison or CMP, check or CHK) of stars to display light curve. The **Mark Stars** ribbon contains controls related to tasks performed after successful photometry calculation.



SIPS Photometry tool allows for three ways how to mark any star type.

- The first and the most natural way to mark a star is clicking on the desired star in the **Image pane**, shown on the upper left corner of the **Photometry** tool window. There are three checkbox buttons in the Mark stars ribbon, which indicate the type of star to be marked:
  - VAR the clicked star will be marked as variable star
  - CMP the clicked star will be marked as comparison star
  - CHK the clicked star will be marked as check star

  If all three buttons are unchecked (released), clicked star is just marked, but no longer used to plot a light curve. SIPS Photometry allows selection of multiple comparison stars. In this case any star compared with comparison star is in fact compared with average flux of all selected comparison star. Due to ability to select multiple comparison stars, the small count box indicating which of 10 possible comparison stars is currently selected CMP 0. However, the logic behind multiple star selection is not simple clicking on the first comparison star, then the second one etc. Instead SIPS always selects only the star with the index in the count box. If a comparison star is selected (say star CMP0) and another star is then clicked, originally selected CMP0 star is deselected and new CMP0 star is selected. If a second comparison star is to be selected, increment the index in the count box following the button to 1. Click to another star then selects second comparison star CMP1.

  Hint:

  It is not necessary to mark comparison stars with increasing indexes. It is possible e.g. to select index 9 and then mark comparison star CMP9 without prior selecting of stars CMP0 to CMP8.

  Star marked as VAR, CMPx or CHK can be unselected by three ways:
  - if another star of this type is selected
  - if the same star is selected as different type
  - if it is selected again as the same type of star

  Remark:

  If VAR and CMPx stars are selected, light curve should appear. If it does not, it may be because one or both selected stars cannot be used for photometry. The most common reason is the star contains some saturated pixel within its aperture. Inspect the star sheet for **Max. Px** values—if its value reaches 65535 (for 16-bit camera, lower resolution cameras saturation value is lower), star cannot be used for photometry.

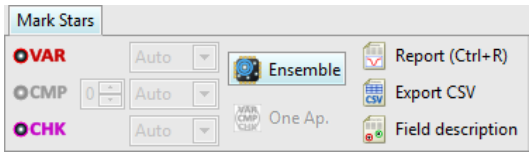
- Clicking (highlighting) any line in the **Star sheet pane** also selects the corresponding star (the star is also centered in the **Image pane**). The **Star sheet pane** is described in detail later (see the Star sheet pane sub-chapter).

  Star type (VAR, CMPx, CHK) selected by selecting of the star sheet line is determined by the VAR, CMP, and CHK button states like in the case of selecting of the star on image.
- The **Star sheet** is also capable to find and mark particular star according to its user-defined name, catalog identifier or equatorial coordinates (ratio buttons in the tool-bar define whether **Name**, **Catalog** identifier or **RA Dec** coordinates are to be found). The coordinate format is arbitrary, SIPS tries to properly parse coordinates entered in various formats:
  - Two decimal numbers are interpreted as R.A and Dec. in decimal degrees.
  - Two strings delimited by blank(s) are interpreted as R.A and Dec. in "HH:MM:SS.ss +DD:MM:SS.ss" form. The delimiters can be arbitrary non-numeric characters, so also "HHhMMmSS.ss +DDdMMmSS.ss" and similar delimiters are accepted.
  - Six strings delimited by blanks are interpreted as R.A and Dec. in "HH MM SS.ss +DD MM SS.ss" format.

  Searching with the tool also respects the selected VAR, CMPx and CHK button states. But it is also possible to find stars and mark the as particular type using VAR, CMP and CHK buttons, regardless of the main star type selection button states. This feature significantly speeds-up proper marking of stars.
- The third possibility how to select VAR, CMPx and CHK stars is using a **Field description**. File description is an advanced feature of SIPS Photometry, which can save huge amount of work needed for photometry time series processing. Details are described later in the Field description chapter.

Marking stars in Ensemble photometry mode

If the Ensemble photometry is calculated, the **Ensemble** option button in the **Mark Stars** ribbon pane is enabled.

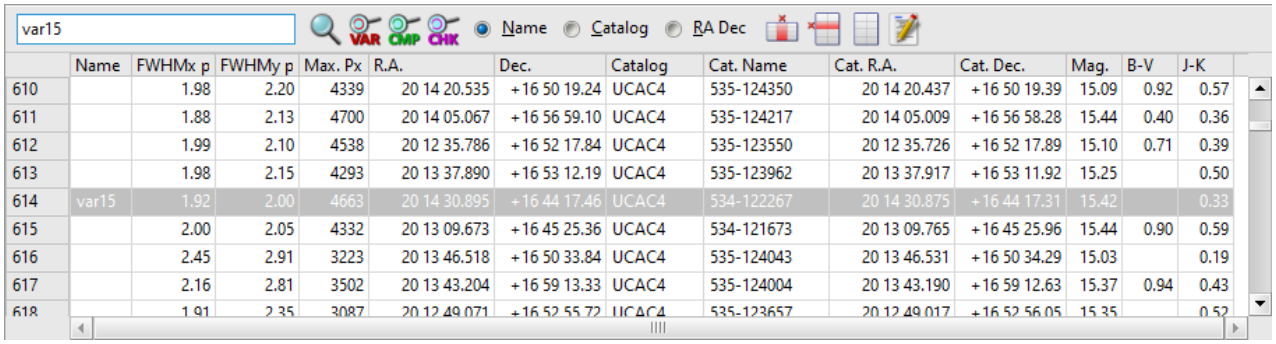


Checking the **Ensemble** button switches the **Photometry** tool to the Ensemble photometry mode. As the ensemble photometry does not work with comparison stars, it is not possible to mark comparison stars, only variable and check stars may be marked.

> Hint:
>
> If the Ensemble photometry is calculated, it is possible to freely switch between Relative and Ensemble photometry modes any time.

Star sheet pane

The **Star sheet pane** (lower-right portion of the **Photometry** window) is the table showing all stars detected on the actual image.



	Name	FWHMx p	FWHMy p	Max. Px	R.A.	Dec.	Catalog	Cat. Name	Cat. R.A.	Cat. Dec.	Mag.	B-V	J-K
610		1.98	2.20	4339	20 14 20.535	+16 50 19.24	UCAC4	535-124350	20 14 20.437	+16 50 19.39	15.09	0.92	0.57
611		1.88	2.13	4700	20 14 05.067	+16 56 59.10	UCAC4	535-124217	20 14 05.009	+16 56 58.28	15.44	0.40	0.36
612		1.99	2.10	4538	20 12 35.786	+16 52 17.84	UCAC4	535-123550	20 12 35.726	+16 52 17.89	15.10	0.71	0.39
613		1.98	2.15	4293	20 13 37.890	+16 53 12.19	UCAC4	535-123962	20 13 37.917	+16 53 11.92	15.25		0.50
614	var15	1.92	2.00	4663	20 14 30.895	+16 44 17.46	UCAC4	534-122267	20 14 30.875	+16 44 17.31	15.42		0.33
615		2.00	2.05	4332	20 13 09.673	+16 45 25.36	UCAC4	534-121673	20 13 09.765	+16 45 25.96	15.44	0.90	0.59
616		2.45	2.91	3223	20 13 46.518	+16 50 33.84	UCAC4	535-124043	20 13 46.531	+16 50 34.29	15.03		0.19
617		2.16	2.81	3502	20 13 43.204	+16 59 13.33	UCAC4	535-124004	20 13 43.190	+16 59 12.63	15.37	0.94	0.43
618		1.91	2.35	3087	20 12 49.071	+16 52 55.72	UCAC4	535-123657	20 12 49.017	+16 52 56.05	15.35		0.52

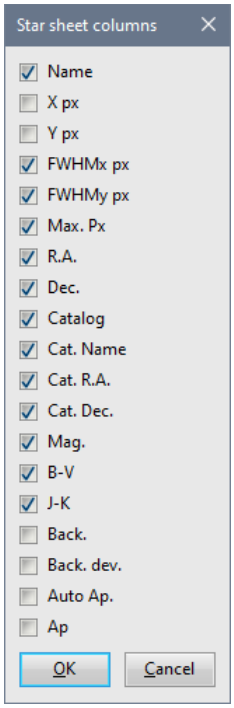
The table shows all information the SIPS Photometry calculated or gathered about all stars. This means results of the individual processing steps performed during image list processing—finding stars (coordinates of star centroid withing

the image, star FWHM, …), astrometry and catalog matching (equatorial coordinates, catalog identifier, brightness, color indexes, …), and photometry (fluxes for all apertures, background ring values, …).

Another Star sheet purpose is to enable selection of VAR, CMPx and CHK stars. This functionality is described in point No.2 of the **Marking stars** sub-chapter above.

Filtering columns

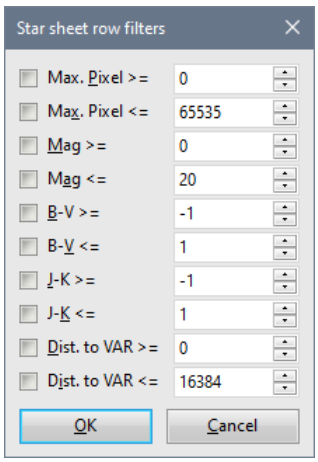
The number of columns is high and not all values are important when working with the Photometry tool. So, the **Star sheet** allows the user to select only columns of interest and hide others. The  button opens column selection dialog box.



Filtering rows

Depending on the field of view of the used camera/telescope as well as the field position on the sky (proximity to the Mily Way plane), the number of detected stars and corresponding rows in the Star sheet may var from a few tens to more than hundred thousand. Browsing through such huge table when e.g. searching for appropriate comparison star of proper color index (to eliminate trends caused by atmospheric extinction), proper brightness (to get as high S/N as possible but not to saturate) and not too far from variable star (to eliminated effects of varying background gradients) may be challenging.

The **Star sheet** offers filtering of displayed lines using number of criteria, defined in the row filter dialog box, opened by the  button.



The row filtering may be based on:

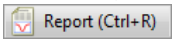
- Maximal pixel value (to ensure star does not saturate).
- Magnitude.
- B-V color index.
- J-K color index.
- Distance to variables star expressed in pixels.

Remark:

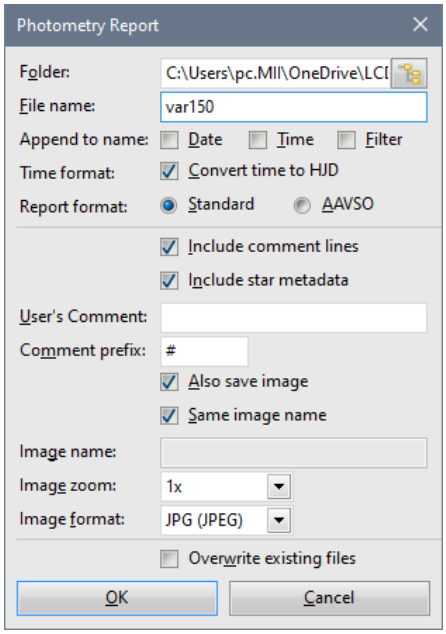
The row filtering requires the astrometry to be performed on images. Also, brightness and color indexes are read from catalog. Not all catalogs contain color indexes (e.g. USNO-B1.0) and even if the catalog generally contains them (e.g. UCAC4), color indexes may not be available for all stars.

It is possible to cancel row filtering without the necessity of un-checking all options in the row filter dialog box in the same image by clicking the  button.

Photometry protocol export

The displayed light curve can be exported into text report file and an accompanying image showing selected VAR, CMPx and CHK stars using the 

button or by using the <Ctrl>+<R> hotkey. SIPS opens a dialog box to conform all parameters.



The **Photometry Report** dialog box is optimized to be able to save the report as quickly as possible without overhead of subsequent opening and confirming of standard “Save As” dialog boxes, confirming of image format and zoom etc. All required information (file names, folders, image format and zoom, ...) is entered in single dialog box and remembered until the user changes it. So, typically saving a report consists of only opening the dialog and clicking the OK button.

Remark:

The saved time needed to store a report it is not that important for single light curve, but for tens or more than a hundred light curves the spared time is significant.

SIPS Photometry supports two formats of the photometry report, distinguished by the **Report format** radio-buttons.

- **standard** format
- AAVSO-specific format

Common parameters include:

- **Folder** is the name of folder where the text file and accompanying image is to be stored.

- **File name** defines the name for text file (and possibly also the image file).

  Hint:

  If the particular variable star has a **Name** defined in the **Star sheet** (this is often the case if the **Field description** is used), the report dialog fills the name into **File name** edit box, so it is not necessary to enter the file name manually.

- **Append to name** checkboxes allow extending of the file name with **Date**, **Time** and **Filter** of the first photometry point in the light curve.
- **Time format** defines if the SIPS Photometry should apply the heliocentric correction to Julian Date marking the time of individual photometry points. Using of Heliocentric Julian Date (HJD), not affected by light travel time differences caused by Earth orbit around Sun, inherent to Geocentric Julian Date (GJD), is necessary for any analysis of light curves. However, some researchers may prefer to perform such correction themselves and save the protocol in Geocentric GJD.

  Remark:

  Because heliocentric correction depends on the object coordinates, HJD time stamps can be used only if the astrometry solution is performed on the images.

  If the **Convert time to HJD** option is checked, the fourth column (beside the HJD, brightness and error) is added to the standard protocol, containing the heliocentric correction applied. So, even if the heliocentric conversion is performed, it is possible to reconstruct the original GJD time and convert the JD to other system, e.g. BJD (Barycentric Julian Date) etc.

  The JD format (GJD or HJD) is indicated in the attributes line of the protocol text file after the "JD:" keyword. The values of the JD attribute are "geocentric" or "heliocentric".
- If the **Overwrite existing files** check box is left unchecked any of the saved files already exists, the **Photometry report** first asks the user to confirm overwriting of existing file(s).

Standard report format

Standard report is an ordinary text file containing three or four columns of numbers, delimited by spaces:

1. Julian Date either containing HJD or HJD (indicated by the "JD:" attribute in protocol header).
2. Brightness value either in magnitudes of fluxes (indicated by the "Unit:" attribute in protocol header, containing either "magnitude" or "flux" identifier).
3. Value error (uncertainty).
4. Possibly the heliocentric correction applied to get HJD in the first column from the GJD stored in image.

The standard report format typically begins with several comment lines. Comment lines do not start with number, which distinguishes comment lines from data lines. The comment lines contents and also prefix (if any) can be modified parameters defined in the **Photometry report** dialog box.

Standard report dialog box parameters:

- **Include comment lines** checkbox defines if comment lines will be present in the protocol.
- **Include star metadata** checkbox causes including of metadata line for each star used to create light curve (VAR, CMPx, CHK). This option requires previous option checked.
- **User's comment** allows including of any text line into photometry report protocol.
- **Comment prefix** field is intended for any special character or character sequence, which is placed on the beginning of every comment line.

  Remark:

  The comment line prefix is typically some special character, like '#' or '$'. Such prefix is necessary if the **User's comment** starts with number and so it could be mistaken for data line. But standard comment lines always begin with letters and it is possible to distinguish them from data lines, which begin with number. So, the **Comment prefix** purpose is more aesthetic, some special character clearly indicates "this line is a comment, not data" mainly for human readers.

- **Also same image** checkbox defines whether accompanying image has to be saved or not.
- **Same image name** allows using of the same name for image (only the file name extension will be different). This is typically the case, only if there is some reason to name both files differently, uncheck this box to be able to fill the next parameter.
- **Image name** allows to define different file name for accompanying image if desired.
- **Image zoom** defines the scale from 1/8 to 8× of the accompanying image (map). Depending on the camera resolution, scaling down the image saves lots of drive space, but at the expense of less details included.
- **Image format** allows choosing from common image format for the accompanying image (map). Using of a loss-less format like PNG ensures images free of compression artifacts, but files are bigger. Lossy format like JPEG provides smaller files, but image details may be compromised by compression.

Example of a first few lines of the standard protocol file:

```
# HJD V-C s1 HJD-GJD
# VarAperture: 9.01 CmpAperture: 18.00 JD: heliocentric Unit: magnitude Filter: Clear
# VAR Name: CzeV800 RA: 21:38:11.557 Dec: +58:13:46.70 Catalog: UCAC4 CatalogId: 742-073727
# CMP0 Name: cmp5 RA: 21:37:42.943 Dec: +57:36:31.39 Catalog: UCAC4 CatalogId: 739-073746
2460235.320737 3.4704 0.0228 0.002447
2460235.321792 3.4742 0.0230 0.002447
2460235.322847 3.4822 0.0229 0.002447
```

Remark:

The comment lines corresponding to star meta-data are cropped on the protocol file example above to keep them on single line.

The first comment line describes data line column data.

The second line contains attributes related to light curve:

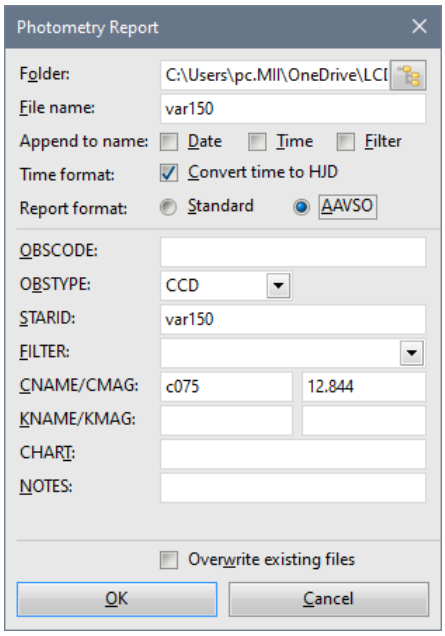
- “VarAperture:” aperture applied for variable star
- “CmpAperture:” aperture applied to comparison star
- “JD:” Julian Date used “geocentric” or “heliocentric”
- “Unit:” brightness unit used “magnitude” or “flux”
- “Filter:” used to capture this light curve

The next lines contain meta-data of stars used to create this light curve. Each line begins with identification of the star (“VAR” for variable star, “CMPx” for comparison stars number 0 to 9 and “CHK” for check star, despite check star does not affect the light curve at all).

- “Name:” name of the star, assigned in the **Star sheet**
- “RA:” right ascension of the star, determined by astrometry solution of the image
- “Dec:” declination of the star, determined by astrometry solution of the image
- “Catalog:” catalog used for astrometry solution
- “CatalogId:” identifier in the above catalog, providing the star was matched
- “CatalogRA:” right ascension of the star in catalog
- “CatalogDec:” declination of the star in catalog
- “CatalogMag:” magnitude of the star in catalog
- “CatalogB-V:” B-V color index of the star in catalog
- “CatalogJ-K:” J-K color index of the star in catalog

AAVSO report format

The AAVSO (American Association of Variable Star Observers) uses proprietary data format. The AAVSO format is a bit extensive (a lot of information is repeated for every photometry point), but users, who want to post the light curve to the AAVSO database, must provide the protocol in this format. This is why SIPS Photometry allows using of AAVSO format for photometry report.



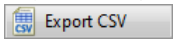
Example of a first few lines of the AAVSO protocol file:

```
#TYPE=EXTENDED
#OBSCODE=
#SOFTWARE=SIPS v 4.0 64bit (x64)
#DELIM=;
#DATE=HJD
#OBSTYPE=CCD
#NAME;DATE;MAG;MERR;FILT;TRANS;MTYPE;CNAME;CMAG;KNAME;KMAG;AMASS;GROUP;CHART;NOTES
var150;2457996.285329;17.8767;0.1411;na;NO;STD;c075;-13.2622;na;na;1.2715;1;na;…
var150;2457996.286787;17.6862;0.1106;na;NO;STD;c075;-13.2716;na;na;1.2676;1;na;…
var150;2457996.288370;17.6919;0.1043;na;NO;STD;c075;-13.2788;na;na;1.2635;1;na;…
var150;2457996.289875;17.9162;0.1228;na;NO;STD;c075;-13.2729;na;na;1.2597;1;na;…
```

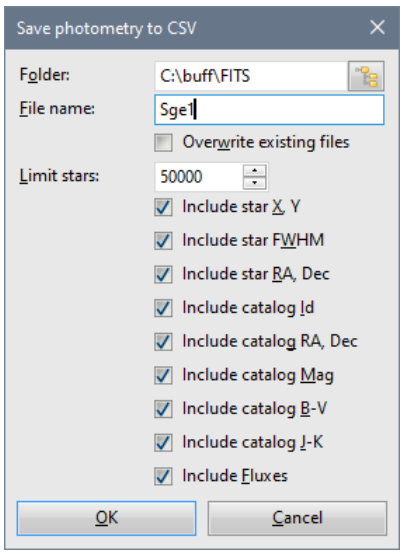
Remark:

Lines ending with … in the example above were cut to keep the example illustrative.

Exporting photometry to CSV

The SIPS **Photometry** tool can export all information contained in the **Star sheet** into CSV file for every image in the implicit image set. So, the number of generated CSV files equals to the number of images in the image list. The 

button opens the **Save photometry to CSV** dialog box.



Dialog parameters are:

- **Folder** defines folder where the individual CSV files will be created.
- **File name** is the fila name prefix for CSV files. Individual files are numbers by adding the ordinal number ("Sge1_0000.csv", "Sge1_0001.csv", … in the example above).
- In the case a file with the same name already exists in the defined folder, the **Override existing file** turns off confirmation if the existing file is to be replaced with a new one.
- **Limit stars** parameter allows limiting of the number of stars (CSV table lines) included in the files. The exported CSV files are typically intended for processing by some other software package and not all detected stars are suitable for such processing. Typically, only the brightest stars (be it just tens or tens of thousand) in the field of view offer great enough S/N to be useable for scientific research. So, including all detected stars typically only wastes drive space and makes CSV filed unnecessarily big.
- The set of checkboxes allows to define which columns are to be included into CSV files. These columns correspond to **Star sheet** table of the **Photometry** tool.

Each CSV file begins with two columns of keywords and values, providing meta-data for an image, from which the CSV file was created. Then a line of cells beginning with keyword "Name", containing names of data columns. Following lines are numbers, copying values in the **Star sheet**.

Example of a first few lines of the exported CSV file:

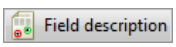
```
sep=;
ExposureBegin;2017-08-30T18:43:56.227
Exposure;2017-08-30T18:44:46.231
ExposureEnd;2017-08-30T18:45:36.235
ExposureTime;100.00000000
JDBegin;2457996.28051189
JDEnd;2457996.28166939
GJD;2457996.28109064
HJD;2457996.28532935
Filter;NoFilter
CenterRA;20 13 42.357
CenterDec;+16 51 43.66
CenterRAHour;20.22843240
CenterRADeg;303.42648594
CenterDecDeg;16.86212638
Width;1536
Depth;1024
PixelScaleX;1.34460101
PixelScaleY;1.34460101
Object;
Instrument;
Telescope;
Observer;
Software;SIPS Version 4.0 64bit (x64)
Name;X;Y;FWHMX;FWHMY;RA;Dec;Catalog;CatalogId;CatalogRA;CatalogDec;CatalogMag;CatalogBV;…
;7.75;287.23;;6.10;20 14 53.222;+16 46 25.07;UCAC4;534-122437;20 14 53.339;…
;1454.82;81.55;5.65;5.92;20 12 37.450;+16 42 18.67;UCAC4;534-121413;20 12 37.301;…
;314.27;209.82;5.46;5.75;20 14 24.403;+16 44 47.86;UCAC4;534-122232;20 14 24.334;…
```

Remark:

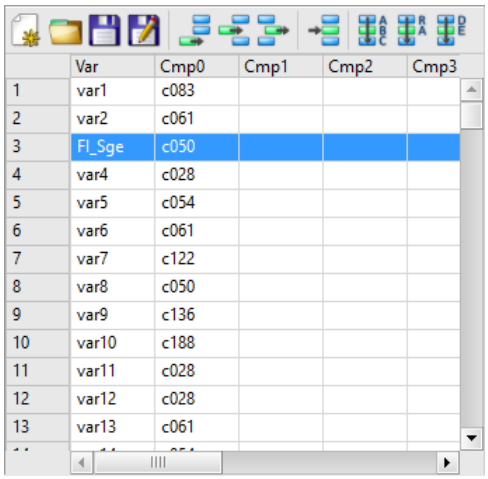
Lines ending with … in the example above were cut to keep the example illustrative.

The fist cell of the CSV file contains string "sep=;", which indicates the values of individual cells are not separated by commas, but by semicolons. This extension of the CSV format ensure compatibility with Microsoft Excel.

Field description

Field description pane is opened (and closed) with the 

button in the **Photometry** ribbon. Its main purpose is to save all marked stars (variables, comparison, and check stars) into description file and later just use this file to rapidly generate light curves and reports of all stars of interest in the field.



	Var	Cmp0	Cmp1	Cmp2	Cmp3
1	var1	c083			
2	var2	c061			
3	FI_Sge	c050			
4	var4	c028			
5	var5	c054			
6	var6	c061			
7	var7	c122			
8	var8	c050			
9	var9	c136			
10	var10	c188			
11	var11	c028			
12	var12	c028			
13	var13	c061			

Remark:

Individual stars are identified with their equatorial coordinates, so the Field description works only on images, on which a successful astrometric solution was performed.

It is highly desirable to use always the same astrometry catalog, with which the description was created. The Catalog name (identifier) is used as a primary star identification, so using of a field description created with different catalog brings significant limitations.

Even if the particular star is not included in the used catalog, it can be still included into field description, it is only identified by coordinates and not by catalog name.

Construction of the field description consists of two steps:

1. All stars of interest, be it variable, comparison or check stars, must be uniquely named. Star can be named in the **Star sheet** by entering its name into the first column.
2. Then it is possible to add another item into the field description table, where names of variable star and comparison and check star(s) are stored.

Field description is saved into a text file with default ".sfd" extension, following the INI file conventions. The text file structure reflects the way the description is created.

```
[Description]
version = 1
catalog = UCAC4

[Stars]
cmp2 = 5.395547204E-1; 5.691312848E-1; 614-005727; 5.395552059E-1; 5.691316911E-1
cmp3 = 5.38700184E-1; 5.701288131E-1; 614-005718; 5.387024041E-1; 5.701280414E-1
v4 = 5.410245319E-1; 5.721339355E-1; 614-005744; 5.410203177E-1; 5.721310928E-1
cmp1 = 5.402964787E-1; 5.706426903E-1; 614-005736; 5.402940474E-1; 5.706397623E-1
v2 = 5.414696827E-1; 5.72068233E-1; 614-005749; 5.414679461E-1; 5.720669083E-1
v1 = 5.414845401E-1; 5.718940758E-1; 614-005750; 5.41483305E-1; 5.718923996E-1
v3 = 5.412030003E-1; 5.721243677E-1; 614-005747; 5.41198191E-1; 5.721220898E-1

[Variables]
v1 = cmp3; ; ; ; ; ; ; ; ;
v2 = cmp1; ; ; ; ; ; ; ; ;
v3 = cmp2; ; ; ; ; ; ; ; ;
v4 = cmp3; ; ; ; ; ; ; ; ;
```

The **[Description]** section contains the header, currently containing the version number and the catalog used.

The **[Stars]** section lists all used stars. Defined star name (assigned by the user) is followed by star equatorial coordinates, determined by the astrometry solution. The catalog identifier (if any) follows with the equatorial coordinates defined in the catalog. When the Field description is loaded, the **[Stars]** section is projected into the **Star sheet**—individual stars are found and defined names are added into the sheet.

And finally, the **[Variables]** section contains list of name sequences. The first name is always the variable star name. The following names are (up to) 10 comparison stars and a check star name is the last one. When the Field description is loaded, the **[Variables]** section is projected into the sheet in the **Field description pane** itself.

Remark:

The Field description sheet does not allow direct modification of individual items (lines) by the user. Each item can be created or modified only by selecting VAR, CMPx and CHK stars and then this particular state can be stored into the field description. This mechanism helps keep the whole field description consistent, without errors caused by typos etc.

The Field description pane tool bar offers the following tools:

-  Clear current field description and start with a new empty one.
-  Open new file description from .sfd file. Current field description will be cleared.
-  Save current file description into .sfd file.
-  Save current file description into .sfd file under different file name.
-  Add new item into the field description. The currently selected VAR, CMPx and CHK stars will be added as a new line.

  Remark:

  All selected stars (VAR, CMPx, CHK) must have a name in the **Star sheet**. If there is any unnamed star, error message is displayed and no item is added.

-  Works like Add item, but inserts item into currently selected position.
-  Remove currently selected item from field description.
-  Update (replace) currently selected item in the field description with selected stars.
-  Sort field description according to variable star name.
-  Sort field description according to variable star Right Ascension.
-  Sort field description according to variable star Declination.

Hint:

The Photometry tool always searches stars defined in the field description in the currently selected image. If for instance some star, defined in the **[Stars]** section, is not found on the current image, all definitions referring to this particular star become invalid. This is typically not the case for comparison stars, as these stars are typically chosen to be bright and not in a close proximity to another star, and thus easily detectable on all images. But if some dim variable star is not found on selected image, it could be detected on another image, taken for instance at higher altitude, while the weather was more favorable etc.

It is always possible that some star is not found on the current series of images (e.g. it is out of field of view) or the field description is not consistent (this means some item in the **[Variables]** refer to name of star, which is not defined in the **[Stars]** section). Field description sheet notifies the user with color used to display particular star name:

- Black names are stars properly found on the image.
- Blue names refer to stars from the **[Variables]** section, which were not found in the **[Stars]** section.
- Red names identify stars, which were not matched in the actual image. This means star is too weak (below detection threshold) or it is out of the field of view.

Selecting stars using field description

The main purpose of the Field description pane is to allow fast selection of VAR, CMPx and CHK stars in the field without a necessity to search them in the image and selecting them with a mouse. Saving of reports for all star in the field of view is the matter of minutes instead of hours.

The **Field description** sheet selects stars defined in the respective columns (Var as VAR, Cmp0 as CMP0 etc.) when any item (line) is selected. Also, **Light curve pane** is updated, so a light curve of a variable can be immediately observed.

Hint:

The image in the **Image pane** is centered around of the selected variable stars, providing the whole line in the **Field description** table is selected, e.g. by clicking the row number cell on the left.

But it is possible to select every cell in the table independently. In such case a star named in the selected cell is centered in the **Image pane**.

Field description in the Ensemble photometry mode

The ensemble photometry calculates each star absolute magnitude from a transformation function, fitted on every image individually. So, no comparison stars are used. However, it makes sense to mark a check star in addition to a variable one. The check star should not be variable and its brightness should be constant over time.

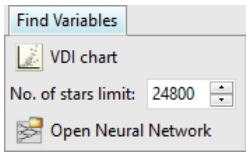
All comparison stars are hidden from the field description sheet in the Ensemble photometry mode, only the VAR and CHK columns remain visible. Also, the **Photometry** tool no longer insists on marking and naming at last one comparison star prior to adding of variable star to the field list, in fact there is no way how to even mark comparison star in this mode.

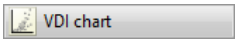
Remark:

While filed descriptions created in the **Relative photometry** mode remain usable in the **Ensemble photometry** mode, only comparison stars are hidden and not used, field description entries created or modified in the Ensemble photometry mode cannot be used in Relative photometry mode.

Finding variables

SIPS Photometry supports finding of variable stars among all stars detected in the field of view. Controls related to finding variables are grouped on the **Find Variables** ribbon.



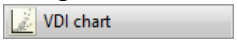
SIPS currently supports three ways of detecting variable stars, described in detail below. Two methods are based on a statistical analysis of a light curve, the third one uses neural network. Results from both statistical analysis and neural network response are shown in a chart, allowing the user to pick up variable star candidates. The chart is displayed in the **Find Variables** pane within the **Photometry** tool window. Because calculating of the statistics or inferring the neural network response usually takes some time, especially for dense and wide fields with thousands of stars, the **Find Variables** pane is opened only upon user request using the 

button.

Remark:

The “VDI” abbreviation stands for Variability Detection Index. Depending on the chosen detection method, it can be just standard deviation, a result of more complex and robust analysis, or a neural network response.

The chart in the **Find Variables** pane is calculated for all stars detected on the currently selected image. A comparison star (or stars) must be selected, so the SIPS can calculate chart points (the light curves of all stars are related to currently selected comparison star). Also, the response is affected by the chosen aperture of both variable and comparison stars.

So, selection of different comparison star, changing of aperture or selection of another image initiates recalculation of the chart points, which causes pausing of the **Photometry** tool response. It is recommended to close the **Find Variables** pane (by another clicking of the 

button) prior such changes.

Remark:

Because of strong dependence of the relative flux or instrumental magnitude on the used aperture, it is always recommended to use automatic aperture for variable star when finding new variables to eliminate as much artifacts as possible. Aperture of comparison star can be fixed or automatic, depending on the user preferences.

The **Find Variables pane** toolbar commands are:

- Select star with nearest greater variability index. The <Up> or <Left> keys are shortcuts for this function, providing the **Find Variables pane** is selected.
- Select star with nearest lower variability index. Shortcut keys are <Down> or <Right>.
- Select star with nearest greater variability index, but not yet marked as variable (this means possibly new variable). This function relies on the **Field description**, which lists known variable stars in its first column. Shortcut keys are <Shift>+<Up> or <Shift>+<Down>.
- Select star with nearest lower variability index, but not yet marked as variable (this means possibly new variable). This function relies on the **Field description**, which lists known variable stars in its first column. Shortcut keys are <Shift>+<Down> or <Shift>+<Right>.
- RMS, VDI and NNI check-buttons define variable search method (see the explanation below).
- LOG turns y-axis into logarithmic scale.

The <Page Up> and <Page Down> keys also move the active (selected) star, if the **Find Variables pane** is selected. This time all stars, whose dost are on the same level (y-coordinate) to the right in the chart are skipped and a brightest star with a dot in the above or below line is selected.

Variable star detection methods

The supported variability detection methods are:

- RMS The **RMS method** calculates standard deviation of the time series for every star. For steady stars, the standard deviation increases with magnitude. Variable stars (and unfortunately also artifacts) show standard deviation above the average of steady stars of certain magnitude. Higher than average RMS of time series indicates possible variability.
- VDI The **VDI method** (Variability Detection Index—name chosen in SIPS to denote this method) is based on a method originally proposed to detect the Blazhko effect of RR Lyrae variable stars[11]. In SIPS, the VDI method is slightly modified to better indicate fast physical variable stars, which light curve crosses the average brightness many times during a single observation.
- NNI The **Neural Network method** uses simple tree layer neural network, trained on shapes of light curves of various types of variable star. Details of neural network functionality and its training is discussed below.

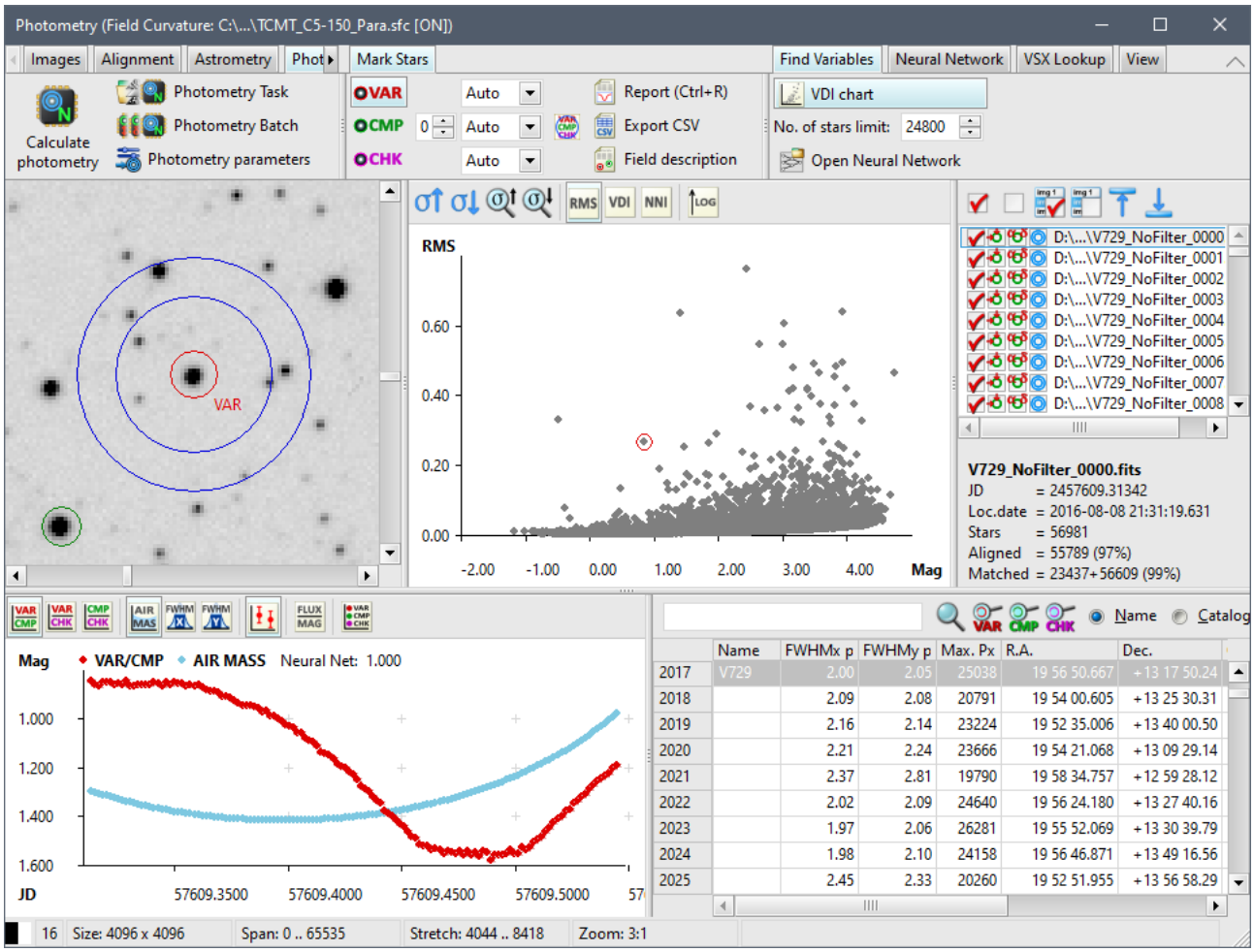
VDI pane chart

The **Find Variables pane** shows a dot for every star detected in the currently selected image, compared to selected comparison star (or stars) using defined apertures. The chart x-axis increases with magnitude (decreasing brightness), y-axis represents a probability (index) that the particular star is variable.

If the RMS method is used, dots representing individual stars create typical “hockey stick” shape, as dimmer star (on the right side of the chart) show higher RMS (on the upper side of the chart). However, variable stars of certain brightness have typically higher RMS than non-variable stars, so the dots corresponding to variable star are located above the “hockey stick” shaped cloud of dots of non-variable stars.

The example below shows the V0729 Aql minimum in the RMS chart.

[11] Jaimes, R. F. et al, Variable stars in the globular cluster NGC 7492, A&A, 2013, 2013A&A...556A..20F

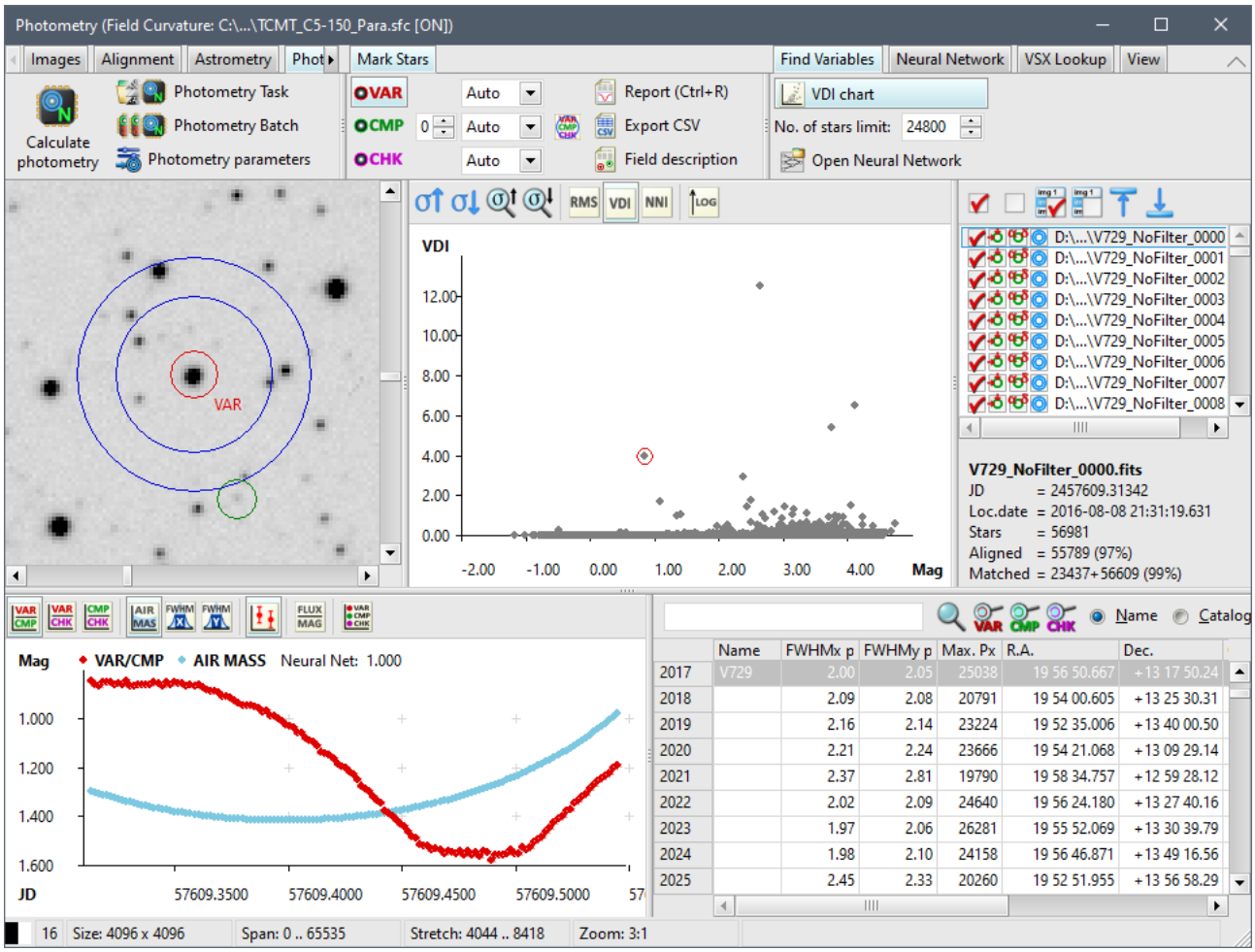


	Name	FWHMx p	FWHMy p	Max. Px	R.A.	Dec.
2017	V729	2.00	2.05	25038	19 56 50.667	+13 17 50.24
2018		2.09	2.08	20791	19 54 00.605	+13 25 30.31
2019		2.16	2.14	23224	19 52 35.006	+13 40 00.50
2020		2.21	2.24	23666	19 54 21.068	+13 09 29.14
2021		2.37	2.81	19790	19 58 34.757	+12 59 28.12
2022		2.02	2.09	24640	19 56 24.180	+13 27 40.16
2023		1.97	2.06	26281	19 55 52.069	+13 30 39.79
2024		1.98	2.10	24158	19 56 46.871	+13 49 16.56
2025		2.45	2.33	20260	19 52 51.955	+13 56 58.29

Of course, denser fields with more detected stars suffer from more artifacts. Numerous stars exhibit brightness changes, caused by wrong star detections, diffraction spikes of nearby bright stars etc. All these false positives populate the chart space above the “hockey stick”, making distinguishing of variable star particularly difficult.

The VDI method is more sophisticated compared to simple RMS method, especially in eliminating of false positives.

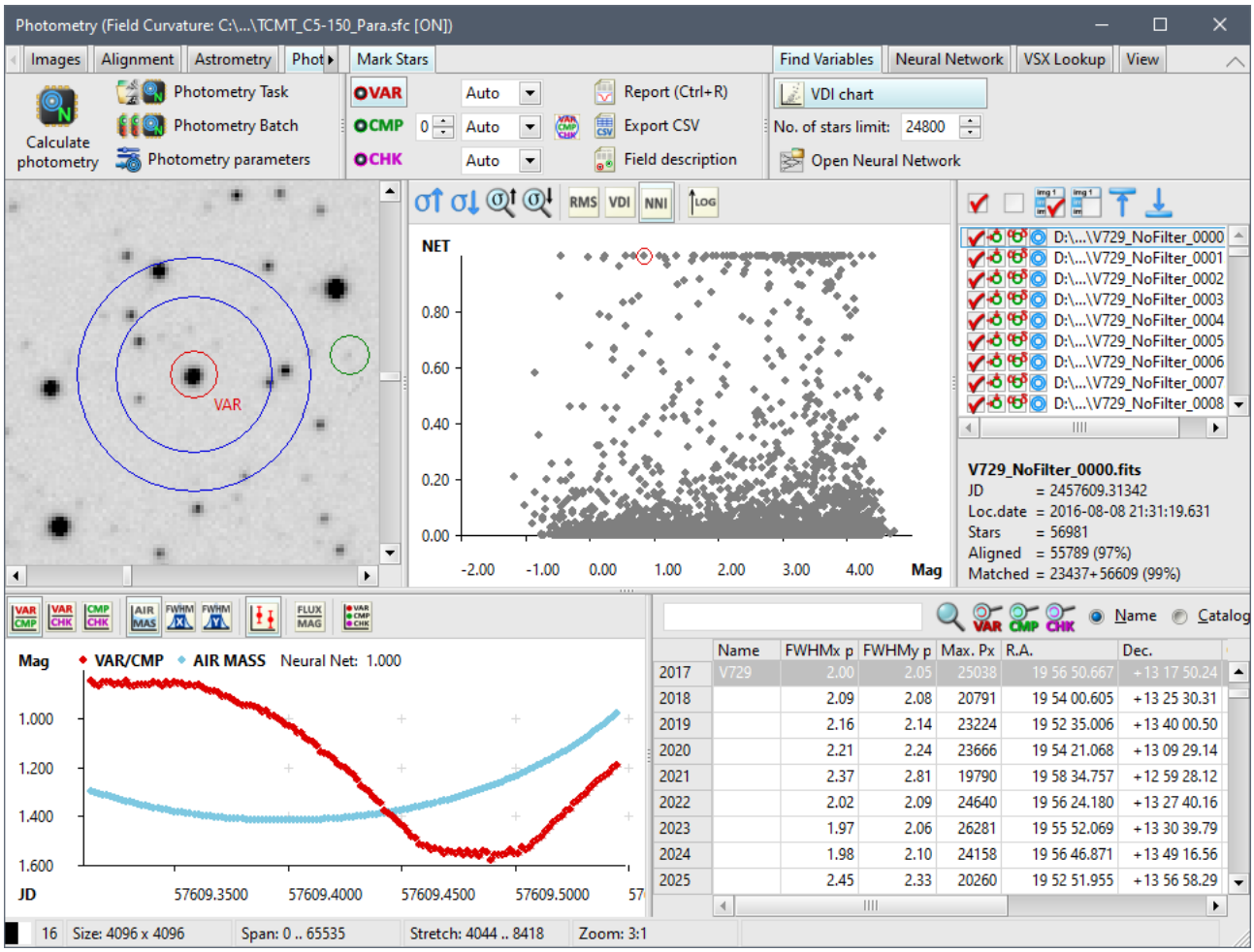
The example below shows the V0729 Aql minimum in the VDI chart.



The Neural network works differently compared to statistical method. The variable star detection mechanism is similar to the mechanism used by humans—the network simply “sees” the light curve similarity to some of thousands of variable star light curves, presented to it during the learning phase. Of course, the simple network is much more prone to false detection than humans as it is incomparably simpler than human brain. But on the other side, computer running the neural network never gets tired or inpatient.

The response of the neural network is a probability that the particular light curve is a variable star light curve, expressed as a number from 0 (definitely not a variable) to 1 (certainly a variable).

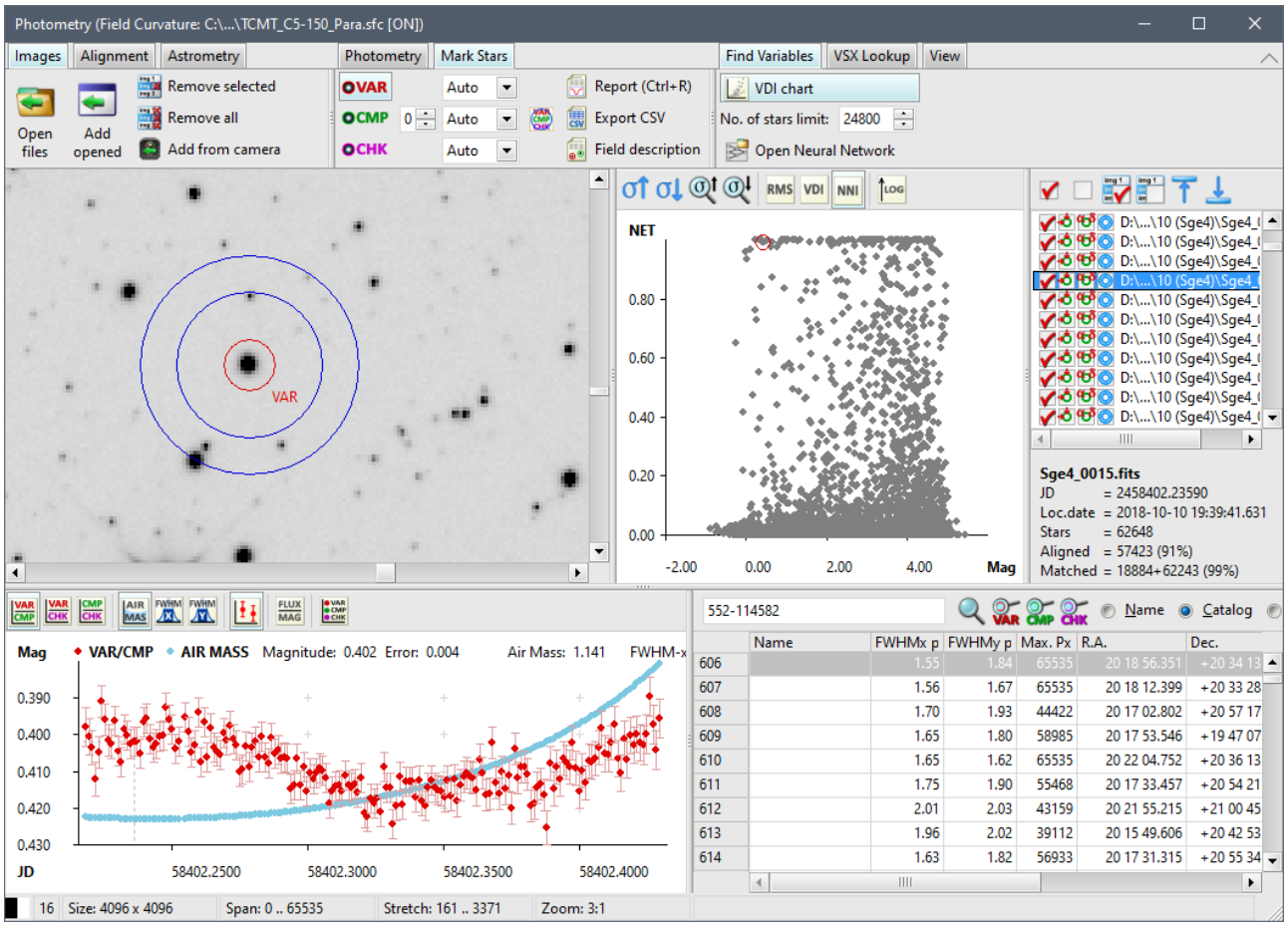
The example below shows the V0729 Aql minimum response of the neural network. The value 1.000 returned by the neural network indicates this light curve definitely belong to variable star.



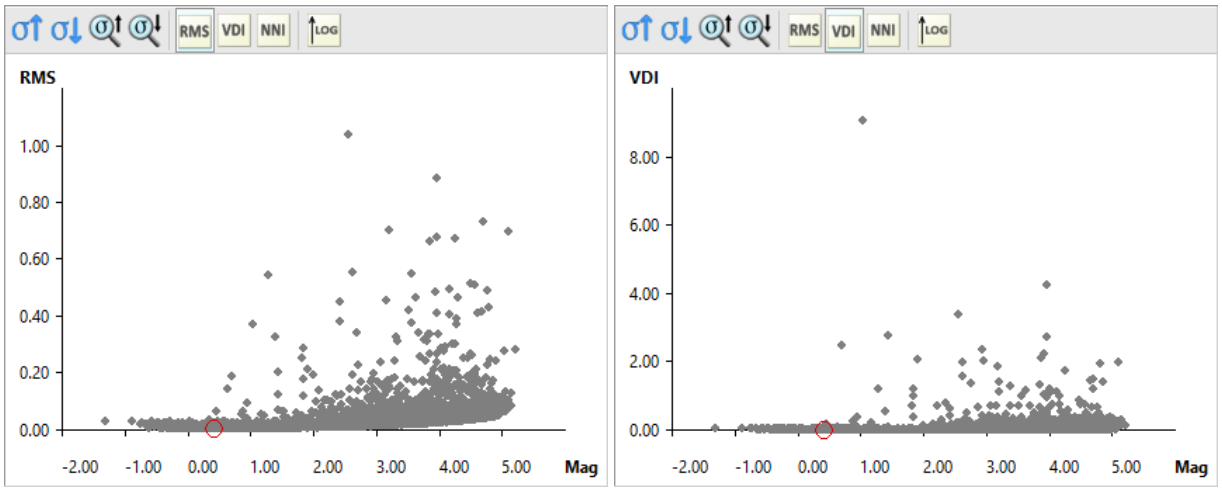
The neural network effectivity strongly depends on the data quality (amount stars exhibiting rapid brightness changes due to various artifacts).

The fact, that the neural network pre-scales the amplitudes of all light curves into -1..1 range, makes the network particularly insensitive to minima amplitudes (as opposite to statistics-based variability detection methods). So, the neural network is suitable for detection of variable star with shallow minima like for instance exoplanet transits. Also, depending on the training set, neural network can detect only partial light curves.

Let us take an example of the secondary minima of 12.82 Mag variable star CzeV2494 (UCAC4 552-114582), observed on October 10th, 2018. The secondary minimum depth is only 0.016 Mag. Still, the neural network identifies the light curve to be a variable star with 98% probability and the CzeV2494 is on the top of the NNI generated variability index chart, because the light curve shape is very similar to shapes of many minima, used to train the neural network.



Statistical methods, on the other side, completely overlook such shallow minimum and place the star on the very bottom of the VDI charts. The RMS method response is on the bottom-left and VDI method response on the bottom-right images (a dot corresponding to CzeV2494 is marked with a red circle).

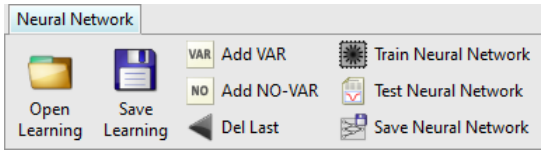


Neural network training

The SIPS package is installed with a variability detection neural network already trained and ready to use (the network parameters are stored in the `NeuralNetwork_FindVariables.txt` description file in the SIPS installation folder). So, users need not to bother with the rather complex process of creation of a learning set, setting of training parameters etc. This chapter is intended for users with some knowledge of neural network principles, willing to train their network on own data.

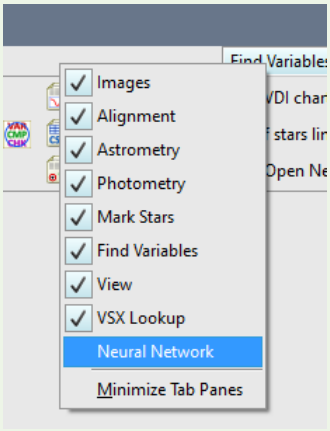
As mentioned above, neural network operation significantly differs from statistical analysis. The neural network ability to recognize a light cure belongs to a variable star is gained through a process of learning. It is necessary to provide ideally tens of thousands, but at last thousands of light curves while the network is learning. Also, light curves of non-

variable stars and various artifact should be included in the learning set, so the network can distinguish them. The SIPS Photometry tool supports the learning process by numerous tools, grouped in the **Neural Network** ribbon.



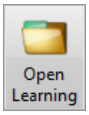
Hint:

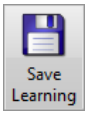
The Neural Network training is performed so rarely, that the corresponding ribbon is by default hidden not to occupy space of the **Photometry** tool window. To show this ribbon, open the ribbon list by right-clicking the tabs space of the ribbon container. Then check the **Neural Network** item.



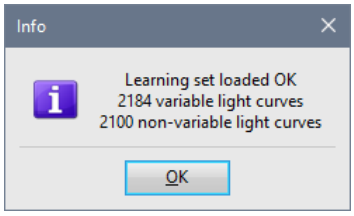
See the **Appendix A** for more detailed description of SIPS GUI.

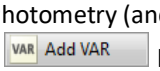
The **Neural Network** ribbon tools enable creation of a variable star detection learning set. The learning set is a simple text file, with each line containing desired output (1 for variable star, 0 for non-variable star) followed by 512 equidistant brightness points, scaled to range from -1.0 to 1.0.

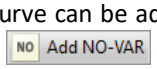
As the learning set should contain at last thousands of sample light curves, it is virtually impossible to create it with data from one observing night. The 

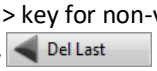
and 

buttons allow to save it and again load prior to adding light cures from the next processed night. When the learning set is loaded, a message box shows number of variable and non-variable light curves present.



After an observing night is processed with SIPS Photometry (and possibly opening an already existing learning set file), a light curve can be added to learning set using 

button or by pressing the key <Y> as variable star light curve or 

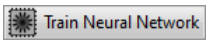
button or <N> key for non-variable star light curve. The last light curve added to the learning set unintentionally may be removed by 

button or <D> key. But SIPS does not provide any more sophisticated means of editing of the learning set.

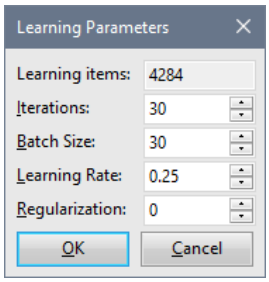
Hint:

We have to emphasize that the network recognizes only shapes, which are at last similar to light curves presented to it during learning. So, it is necessary the learning set included as wide range of variable star types (detached,

semi-detached and contact eclipsing binaries, pulsating star of various periods, …) as possible. Also, partial light curves, containing only beginning or end of minima are desirable.

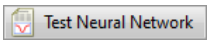
The network is trained using the 

button. A dialog box with training parameters appears.

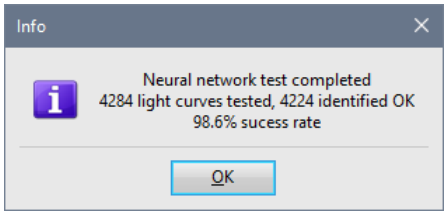


Remark:

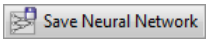
It is beyond the scope of this manual top explain neural network learning process in detail.

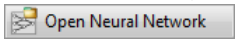
The trained network can be tested 

to show the success rate.



It is worth noting that the proper neural network testing requires using of different set than the one used for training. Testing the network on the used training set is possible, but should be taken with caution.

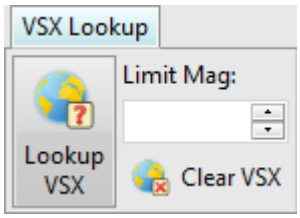
The trained neural network configuration should be saved 

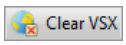
into file. Once the parameter file is opened using the 

button in the **Find Variables** ribbon, SIPS stores file name into configuration and uses it until another neural network parameter file is opened.

VSX database lookup

The online query to VSX database allows marking of all variable stars registered in VSX. This functionality is identical to VSX lookup implemented in the **Astrometry** tool. Refer to the **Astrometry** tool description for detailed description of this function.

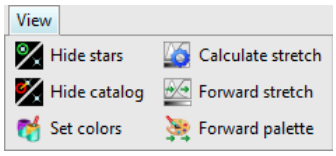


However, implementation of the VSX lookup in the **Photometry** tool differs from the Astrometry implementation in handling of the VSX star highlighting. While VSX registered stars are highlighted in the image displayed in image window only until another image is selected, stars highlighted in the **Image pane** of the **Photometry** tool remain highlighted even if different images of the list are selected. The 

button must be used to hide VSX registered star labels.

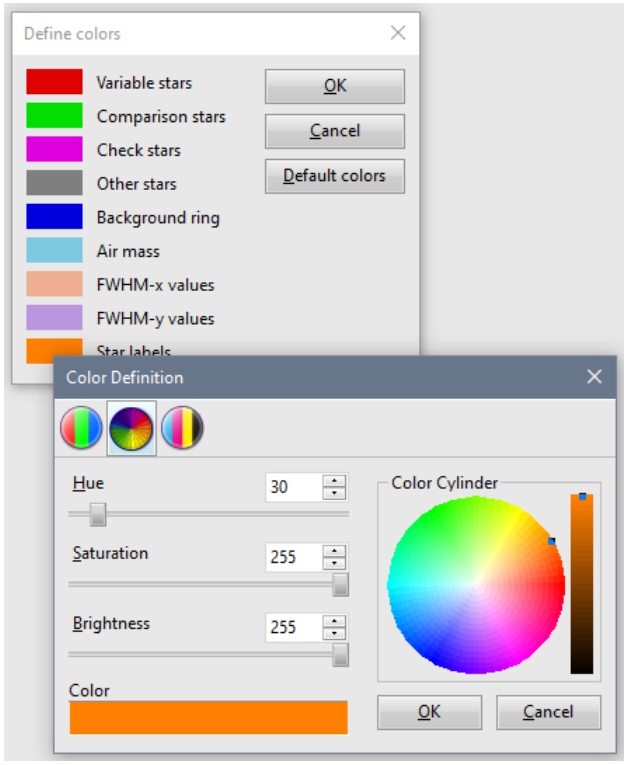
User color definition

The **View** ribbon of the **Photometry** tool contains control affecting image display. Their functionality equals to same controls in for instance Blink Images or Astrometry tools.



The only exception is the  button, which opens a dialog box allowing definition of colors, used to draw aperture circles around variable, comparison, and check stars as well as background ring.

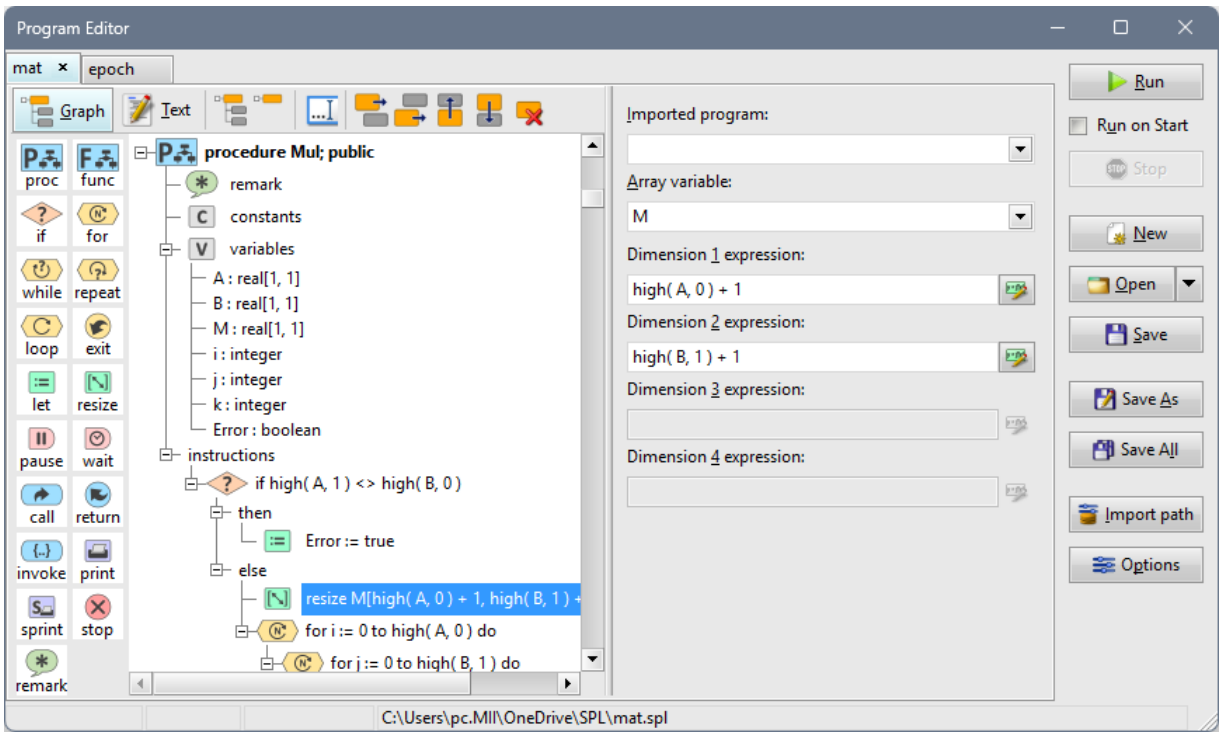
Left-click the sample color box on the left of the color name to open the **Color Definition** dialog box.



Refer to the **SIPS Images** chapter, Palette definition sub-chapter for description of the **Color Definition** dialog box functionality.

Program Editor

The **Program Editor** tool is designed to create and run scripts, written in the SIPS internal scripting language **SPL** (**S**IPS **P**rogramming **L**anguage), to control observation sequences.



The SPL is designed to provide all necessary features to express any algorithm, but to be understandable even by beginners without previous programming knowledge. Still, to be useable and productive, SPL needs to implement many concepts like typed variables, procedures and functions with parameter passing, program module imports etc. These concepts need to be understood by any programmer using SPL to create scripted observing sessions.

Basic concepts

SPL syntax may seem a bit "chatty" for experienced programmers, especially the ones used to a brief and rather cryptic syntax of the C language. But programmers spend only a short time typing the code itself, usually much more time is needed to debug the code, maintain it, or try to understand someone else's algorithm (or even own's algorithm, written long time ago) to implement changes. So, the clearly understandable, yet a bit longer, code is much more useful than short but cryptic code consisting mainly of delimiting characters instead of self-explaining keywords. Here comes the SPL motto: **The program must be comprehensible to humans; computers can always handle it.**

SPL is a **strongly typed language**. Every variable or constant has a type and conversion among types is a responsibility of a programmer (the only exception is an automatic conversion between integer and floating-point number types). This concept also differs from weakly typed languages (e.g. JavaScript), which may seem as more convenient at first. But as the code prolongs, more functionality is added and algorithms become more complex, strange behavior and hard-to-find bugs, caused by often unexpected implicit type conversions, occur. Fixing them is more time consuming than proper assignment of data types from the beginning.

The basic SPL unit is a **program**. Each program is stored in a file with ".spl" extension. Each program consists of several blocks:

- Every program starts with a keyword **program**, followed by program name. The program name should equal to program file name (minus the ".spl" extension), else the program cannot be imported into other programs.
- Then a keyword **version** defines a SPL version, which was used to create this program. Versions are used to check compatibility of the code with the SIPS Program Editor used to edit and run it and possibly to prompt the user to upgrade SIPS to newer version.

- Optional **remark .. end_remark** section corresponds to the **remark** instruction within the code (despite remark is technically not an instruction). Program global remark is intended to provide comments related to entire program.
- Optional **import** block defines which other programs are used by the code. It is possible to use various objects from other programs (constants, variables, procedures, and functions), providing these objects are marked as **public** in respective programs.
- **Constants** block defines global constants—data objects of certain type, which value ifs fixed (does not change during program execution). Global constants are visible by all procedures and functions in the current program and constants marked **public** can be read in other programs, too.
- **Variables** block defines global variables—data objects of certain type, which value may be changes by the algorithm. Global variables are visible by all procedures and functions in the current program and variables marked **public** can be read and written in other programs, too.
- **Procedures** and **Functions** then contain all algorithms, implemented by the program. There is no code outside some procedure or function. Procedures and functions marked **public** can be invoked from other programs.
  SPL differentiates between procedures and functions.
    - **Procedures** are invoked using a control instruction **call**. Each procedure can have none, one or many input variables as well as none, one or more output variables. Code calling the procedure may optionally **set** input variables and, also optionally **get** output variables at its will. See the **call** statement description later.
    - **Functions** are invoked from within expressions during evaluation. So, function cannot be invoked outside of some expression. This means functions can have none, one or many input variables like procedures, but each function must return one value of defined type. See the description of expressions later.
- Program ends with a keyword **end_program** followed by dot.

SPL is a **case-sensitive** langue. Small letter and capital letter are different characters, e.g. variables named 'a' and 'A' are two different variables.

Example programs

SIPS is installed with example programs, which can be immediately loaded into program editor and run. Also examining the example programs may inspire the user how to code SPL programs.

Example programs are installed into two folders:

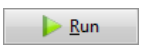
1. "\Users\Public\Documents\SIPS\programs\" folder contains programs intended to be reused (imported) in other programs. They fulfil a role of libraries containing useful code (procedures and functions).
2. User specific "Documents\SIPS\programs\", where are some examples using the public programs installed. Also, this folder is by default offered to user for storing their own programs.

Warning:

Windows Explorer shows the "\Users\Public\Documents\" folder as "\Users\Public\Public Documents\", probably in a (failed?) attempt to make things easier to find by general users?

Running a program

SPL does not use a concept or "main program" or "main function", which is started upon running of the program. Any procedure, marked **public**, implemented in any program can be run as an entry point of the program.

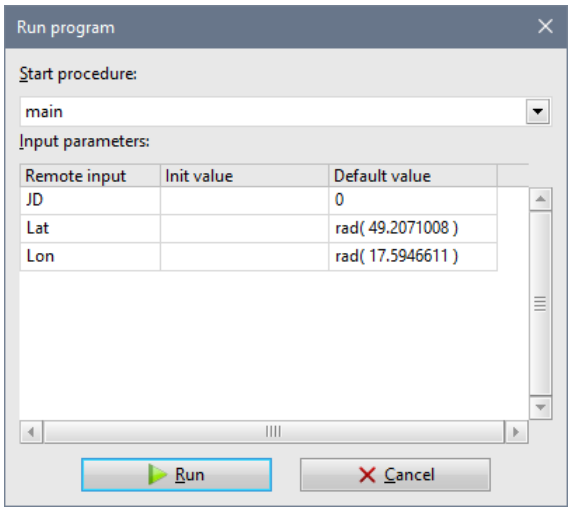
Clicking the 

button thus behaves differently according to the program content.

Program is immediately run, if:

1. Contains only one **public** procedure. Because only public procedures can be invoked from outside of the program, single public procedure is the only possible entry point.
2. The public procedure has no **input** variables. This means there is no way how to modify the program behavior by setting non-default value of some input variable.

Otherwise, the **Program Editor** tool opens the **Run program** dialog box, which allows the user to choose which procedure should be run and set values of input variables of the selected procedure.



The **Run program** dialog box allows the user to select which procedure to run as well as to define values of input variables of the selected procedure (if any). Of course, the input variables can be left unchanged with their default values.

Hint:

Similarly to procedure and function calls inside the program code, also the main program procedure may determine which input variable was assigned in the **Run program** dialog box and which was left on default value using the embedded boolean function **assigned(paremeter_name)**.

The **Run program** dialog box tries to help the user with selection of procedure to run.

1. If there is a public procedure named “main” in the program, this procedure is offered first to run (“main” procedure is selected in the **Start procedure** combo-box).

   Remark:

   Please note this rule does not mean the procedure to be run as a program entry point must be named “main”. Only if there is a procedure named “main”, the **Run program** dialog box pre-selects it as a first choice.

2. Otherwise, the last public procedure in the program is selected. Placing the program entry point procedure as the last one is typical, so it can call all other in-forward procedures (each procedure can call only procedures, which are defined prior to the procedure itself).

Users are of course free to select any other public procedure as the program entry point.

As SPL programs are typically bound to real time (astronomical observations of course cannot be performed anytime), programs typically run for a long period of time. However, a running program can be stopped by the user using the 

button in the command pane. Force stopping of the program works immediately regardless of the program state and all unsaved works is lost, devices are left in the current state (for instance, a camera may be just exposing, telescope moving etc.).

Force stopping is logged with a message “Program Terminated by User”, as opposite to the “Program Stopped” message, which is shown when the normally terminates, either by executing of the last instruction or **return** instruction in the program entry point procedure or reaching the **stop** instruction anywhere within the program.

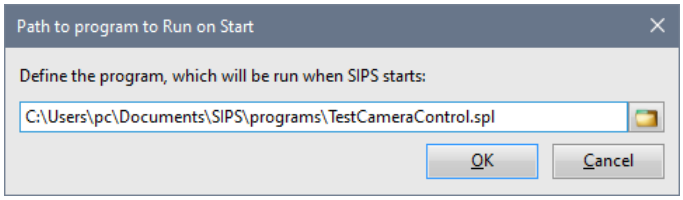
Running a program on SIPS start

Checking of the **Run on Start** check box in the **Program Editor** command pane causes SIPS to perform three operations on startup:

1. Opens the **Program Editor** tool window (if not already opened, because it was left opened during last SIPS session).

2. Loads the defined program into editor (if not already loaded).
3. Runs the defined program.

The program to be run is defined in a dialog box, which is opened when the **Run on Start** check box is checked. It allows definition of the program to be loaded and run on start.



SIPS never opens the **Run program** dialog box when running a program on startup.

- If the program contains more public procedures, the first public procedure is always used. So, the best practice is the program intended for running on startup contains only one public.
- If the first procedure contains some input variables (parameters), their default values are always used.

Remark:

SIPS can run either a procedure or the REST API **Web Server** on startup, but not both. When the **Run on Start** check bock in the **Program Editor** is checked, the same check box in the **Web Server** tool is unchecked.

Data types

As mentioned earlier, every constant and variable must have a type. SPL works with four data types:

- **Boolean** type holds only two logical values. Literals representing values of these types are keywords **true** and **false**. The Boolean type is required for instance in conditional statements etc.
- **Integer** type holds whole numbers (without decimal point). SPL does not distinguish between signed and unsigned integers; all integers are signed with the 32-bit resolution. So, the integer type rage is from -2147483648 to 2147483647.
- **Real** type is for 8-byte wide (double-precision) floating point numbers. Real range is approx. $\pm 1.7 \times 10^{308}$, but as the floating-point numbers are stored with exponent, also range around zero can be important $\pm 5.0 \times 10^{-324}$.
- **String** type holds a string of characters. SPL handles string lengths internally, users need not to care about allocating of space for strings or about overflow of string lengths. String variables have always the length needed to hold its current content.

As mentioned earlier, SPL is strongly typed and conversion among types must be explicitly done by programmer. The only exception is conversion between integer and real types. There are several embedded functions available performing conversions between numeric and string types.

Literals

Any occurrence of a value of certain type is denoted as literal.

- There are only two literals of **boolean** type, denoted by keywords **true** and **false**.
- **Integer** literal are numbers without decimal point. Optionally, a sign can precede the number, the + sign is optional. For instance, 0, 1, +1, -1, -128, 127, +65535 are integer literals.
- The rules to write **real** literals are a bit more complex. Real literals are distinguished by the presence of:
    - Decimal point in the number. For instance, 3.14 or -2.718 are real literals.
    - The letter ‘E’ (or ‘e’) delimiting the exponent part of the number, expressed in semi-logarithmic notation. For instance, 5E+34 represents a number 5×10^{34}. While the SPL compiler accepts both letter ‘e’ and capital ‘E’ to delimit the exponent, code generated by the **Program Editor** tool always use capital E.
    - Of course, real literals can have both decimal point and exponent part, like -2.5E-20.
- String literals are arbitrary string of characters. But to be readable by SPL compilers (and distinguishable from other keywords, identifiers, numbers etc.), each string literal must begin and end with quotes. “Hello world” is a string literal.

As the quotes delimit begin and end of string literal, they cannot be used inside the string (the occurrence of quote ends literal). To overcome the limitation, quotes inside strings must be doubled. Double quotes inside a string literal is not interpreted as the literal end but is replaced with single quote and the string literal continues. If, for instance, the word “world” should be in quotes in string literal, it should be written as “Hello ””world””.

Identifiers

Identifier are names of various elements of the SPL program. Every program name is identifier, constants, and variables as well as procedures and functions are named with identifiers, too. Embedded functions and embedded library programs are also named with identifiers.

Because identifiers are used within the program code, it cannot consist of arbitrary sequence of characters. Identifiers must be clearly distinguishable by SPL compiler and thus must comply to several rules:

- Identifier is a sequence of small and capital letters, underscore characters ‘_’, and numbers. No other characters (e.g. blanks or special characters like parentheses, percent signs etc.) are allowed within identifier.
- Identifier must not start with number.
- Maximal identifier length is 64 characters.

Identifier scope

All indenters, used to name SPL objects, must be **unique**. It is not possible to define different object with already existing name, regardless of the object kind. For instance, it is not possible to define constant or declare variable with the same name as the program itself, procedure cannot have the same name of already existing constant or variable etc.

The inability of to use the same identifier is always valid only for certain **scope**. So, constants, variables, procedures, or functions of the same name can exist in different programs as each program has its own scope.

Also, it is possible to declare a local variable in the procedure with the same name like the global variable in the program, because each procedure and function also has own scope. But such local variable disables access to the global variable (or constant, function, procedure). The used name now represents the local object and the global object is hidden for the code inside the procedure or function.

Keywords

Keywords are identifiers reserved by the SPL compiler to denote statements. Such keywords are e.g. **program** and **end_program**, conditional statement keywords **if**, **then**, **else**, and **end_if** etc. Keywords used for individual statements are described later.

Keyword identifiers are reserved, this means it is not possible to use a keyword to name any other element in the SPL program, like constant or variable etc.

Constants

Constants are named data objects of certain data types, which values are defined in the program code. The value of constant cannot be changed during program execution.

The main purpose of constants is to allow definition of values, which occur multiple times in the code, only once. Say, for instance, the code needs to use a speed of light on multiple occasions. Instead of typing the speed of light each time, it is useful to define a constant for the speed of light and use the constant identifier instead of constant literal.

```
constant
  c = 299792458;

instructions
  …
  E := m * c * c;
  …
```

Remark:

As already mentioned, every constant has a type. But the type in constant definition is optional, it may or not be explicitly defined in the program code. If the type is not defined, it is derived from each constant’s initialization expression.

Let us note the type is mandatory if the constant is an array. Differences between array and scalar values are explained later.

An important benefit of defining a constant instead of typing its literal directly into program code multiple times is the fact, that if the value has to be changed for some reasons, it can be changed in one place instead of finding and replacing every occurrence in the program code.

Constants can be defined at the beginning of the program after the keyword **constants**, in front of any procedure or function. Such constants are **global**, this means a code in any procedure of function may use it.

If the **global constant** is marked **public**, such constant can be also used by procedures and functions in another programs, which import this program.

Making the speed of light constant, used in the example above, available for other programs using the **public** attribute:

```
constant
  c = 299792458; public;
```

Constants can be also defined inside the procedure of function, also after the keyword **constants**, but prior to the beginning of procedure of function code, started with the **instructions** keyword. Such constants are **local** and can be used only by the code inside the procedure or function, in which they are defined.

As already mentioned, local constant hides all global objects of the same name.

Constants initialization is typically just a literal, but it can be a **constant expression** (expressions are described in detail later). This means, it can use not only literals, but also previously defined constants and function calls with constant parameters.

```
constants
  TwoPi = pi + pi;
  HalfPi = pi * 0.5;
  DegAngle = 180.0;
  RadAngle = rad( DegAngle );
  …
```

Constant types

Constant definition does not require an explicit type. If the type is omitted, types are derived from the expression, which defines the constant value. So, keep on mind that SPL distinguishes between **integer** and **real** types, despite making them mutually compatible. If, for instance, the following code:

```
constants
  a = 5;
  b = 2;
variables
  r : real;
instructions
  r := a / b;
  print r, crlf;
```

sets the **r** variable to 2, not 2.5, because both **a** and **b** constants are integers and thus the entire expression **a / b** is of type integer and integer division is used. Only after the expression is evaluated, the value is converter to real and assigned to variable **r**.

One option is to initialize any of the constant (or both) to explicit real value:

```
constants
  a = 5.0;
```

Another option is to convert constant to real number prior to division:

```
  r := real( a ) / b;
```

Both options make the entire expression real and real division is used. Yet another option is to include explicit type to the constant definition, similarly to declaration of variables:

```
constants
  a : real = 5;
```

Hint:

Explicit types in constant definitions are optional; the SPL compiler can read the constant definitions with or without types. However, the user can choose if explicit constant types are generated into program source text when switching from the Program editor **Graph** mode to **Text** mode. See the Code generation from Graph mode for details.

However, despite type definition is optional in constant declaration, type is always generated even for constants in the following cases:

- Constant is an array (declaration of constant arrays is described later in the Array constants sub-chapter).
- The numeric constant type differs from the type of numeric expression, which defines constant value. For instance, consider the constant declarations:

```
constants
  i : integer = 3.14;
  r : real = 1;
```

If the type is omitted when generating program source text from Graph mode, constant types would change, which would affect code execution (e.g. integer and real divisions differ significantly). So, type explicit definition is generated for these constants.

Variables

Variables are named data objects of certain data type, which values can be both read and written by program code. So, variables hold all data, with which the program works and which are modified during program execution.

Variables can be declared at the beginning of the program after the keyword **variables**, in front of any procedure or function. Such variables are **global**, this means a code in any procedure or function may use it.

If the **global variable** is marked **public**, such variable can be also used by procedures and functions in another programs, which import this program. Variables are made accessible from other programs using the **public** attribute, the same way like the constants.

```
variables
  RA : real; public;
  Dec : real; public;
```

Variables can be also declared inside the procedure of function, also after the keyword **variables**, but prior to the beginning of procedure of function code, which starts with the **instructions** keyword. Such variables are **local** and can be used only by the code inside the procedure or function, in which they are defined.

```
variables
  index : integer;
  …
instructions
  …
  index := 10;
  while index > 0 do
    …
    index := index - 1;
  end_while;
  …
```

Like constants, also local variables hide (make them inaccessible) all global constants or variables of the same name.

Local variables can be also used as procedure or function interface, through which the procedure exchanges data with the caller code. This functionality is in details described later.

Variable initialization

Every variable has an initial value. SPL allows to define the variable initial value by an expression, following the type in the variable declaration:

```
variables
  ErrorString : string = "Error";
  Angle : real = 2 * pi;
```

```
…
```

Like values of constants, initial values of variables are also defined by **constant expressions**. This means, it can use not only literals, but also previously defined constants and function calls with constant parameters.

The default initial values depend on variable type:

- Boolean variables are initialized to **false**.
- Number (integer and real) variables are initialized to **0**.
- String variables are initialized to empty string **“”**.

Remark:

While including the default initialization values to variable declaration of course does not violate syntax rules, it is superfluous and the **Program Editor** tool removes such initialization values from the variable declaration when generating source text from program graphic representation.

Variable initialization differs between global and local variables:

- **Global variables**, located in the program itself, are initialized only once, upon the start of the program.
- **Local variables** are initialized each time the procedure is called or function is evaluated.

This means the local variable cannot hold any value between two calls of the same procedure or two evaluations of the same function. If some variable needs to keep its value during the whole program execution, it should be global one.

However, sometimes it is desirable to hide the variable from other procedures and functions (so, it should not be global), but at the same time its value should be kept among individual calls. To do so, local variables can be marked with a **static** attribute.

```
variables
  Offset : real = 1.0; static;
  …
```

Static variables are visible only inside the procedure or function, in which they are declared, but they are initialized only upon the first procedure or function call and their value is preserved after the procedure or function returns. When the procedure or function is invoked again, static variable value is as it was left last time.

Remark:

The **static** attribute is valid only for local variables. Global variables are initialized only once at program start either way.

Scalars and arrays

Single instance of a data type (constant or variable) is called **scalar** value. However, algorithms often benefit from handling multiple instances of the same data type at once, using single identifier. Such data type is called **array**.

Individual values in arrays may be organized into single row (**vector**), into multiple rows of the same length (**matrix** or **2D matrix**), into multiple 2D matrices of the same size (**3D matrix**), etc. Number of items in a row, number of rows in a matrix, etc. are called array **dimensions**.

SPL syntax distinguishes scalar types and array types by the presence of array dimension(s) in square brackets, following the type name. So, let’s take for instance scalar and array of type **real**:

- **real;** declares a scalar real type
- **real[10];** declares a vector of ten real values
- **real[3 ,3];** declares a matrix of real values, containing 9 real values (three vectors of three values each)

Remark:

For efficiency and performance reasons, the SPL compiler currently limits the maximum number of dimensions to four. Vast majority of algorithms do not need more dimensions, so such limit is a reasonable tradeoff.

Accessing array items

As mentioned above, each array has one identifier. So, to access individual items of the array, it is necessary to specify the item index (in the case of vector) or multiple indexes for multi-dimensional arrays. Indexes are stated in the square brackets, following the array identifier. The example below shows how to sum values from a single vector:

```
variables
  a : real[3];
  sum : real;
instructions
  sum := a[0] + a[1] + a[2];
```

Notice arrays in SPL are always indexed from zero. So, highest index available is one less than the number of items in the array.

> Warning:
>
> An attempt to access array elements outside of their range leads to runtime error and premature program termination.

It is possible to index array items with any expression of **integer** type (expressions of **real** type can be used, too, but the results are truncated to integers).

```
variables
  a : real[3];
  sum : real;
  i : integer;
instructions
  sum := 0;
  for i := 0 to 2 do
    sum := sum + a[i];
  end_for;
```

The **for** instruction, used in the example above, will be explained later.

Arrays with more the one dimension require multiple indexing expressions in the square brackets, delimited by comma:

```
variables
  A : real[3,3];
  i : integer;
instructions
  for i := 0 to 2 do
    A[i, i] := 1.0;
  end_for;
```

Array constants

If all array items should be set to the same value, even array may be defined with a scalar constant initialization expression:

```
constants
  a : real[3] = 5;
  m : real[2, 2] = 1;
```

Constant array **a** will have three items (indexed from 0 to 2) and all of which will be set to value 5. The real 2×2 matrix **m** will be filled with number 1.

> Hint:
>
> While type is optional in scalar constant definition, it is mandatory in the case of arrays. So, any constant array must define the type and sizes in individual dimensions in square brackets.

If individual array items should have different values, it is necessary to use **array literal**.

```
constants
  a : real[3] = [1, 2, 3];
  m : real[2, 2] = [[1, 2], [3, 4]];
```

Matrices in SPL are organized row-first. So, the matrix **m** in the example above will be initialized to $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$.

Array literals need not define all items. If a list of item values is shorter than the number of items in array, SPL compiler simply reuses the last item value to fill the rest. So, for instance the constant:

```
constants
  m : real[3, 3] = [[1], [0, 1], [0, 0, 1]];
```

Creates a matrix **m** as $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$.

It is even possible to omit entire vectors, but item values must always be enclosed in a number of square brackets equal to number of dimensions. Sor example:

```
constants
  m : real[3, 3] = [[1, 2]];
```

Creates a matrix **m** as $\begin{bmatrix} 1 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$.

Let us note that single initialization value for entire array:

```
constants
  m : real[3, 3] = [[1]];
```

will be interpreted by SPL compiler as scalar initialization and after regeneration of the source code when switching from Graph to Text mode, the definition will be:

```
constants
  m : real[3, 3] = 1;
```

Expressions

Expressions are prescripts for evaluation (or calculation, if the expressions are numeric) of resulting value from other values (literals, constants, variables) and values resulting from function evaluation. These values are connected with operators, determining the operations to pe performed with values. Typical expression is for instance 1 + 1. Evaluation of this expression leads to the result 2.

Like literals, constants, and variables, also expressions have unique **data type**. Expressions can be not only numerical, but also logical (boolean) and string. As SPL is strongly typed language, it is not possible to freely mix data types and rely on some hidden implicit conversions. The only exception is **integer** and **real** types, which can be mixed. But it is necessary to keep on mind, that converting real to integer cuts the fraction part of the real number and may lead to loss of dynamic range, as real numbers can hold much greater range of values than integers.

Examples of numeric expressions:

```
• e + 0.00256 * cos( Omega )
• rad( NormalizeAngle( angle := Long - 0.00569 - 0.00478 * sin( Omega ) ) )
• frac( JD + 0.5 ) * 24
```

Examples of boolean expressions:

```
• alt > pi
• length( ResultString ) < MaxLength
• (a > b) and (a > c) or not d
```

Examples of string expressions:

```
• "Error: " + ErrorString
• "Air mass = " + str( AltToAirMass( Alt := Alt ), "*.###" )
```

Operator priorities

Operators have different priorities. That means some operators with higher priority are evaluated earlier than operators with lower priority, regardless of their position in the expression.

Operators with the same priority are evaluated as they appear in the expression from left to right.

Priority 1 parentheses (any portion of an expression enclosed into parentheses is evaluated first):

- ( )

Priority 2 unary signs:

- + unary plus (has no meaning, as numbers are positive by default, but SPL syntax allows it)
- - unary minus

Priority 3 multiplication operators:

- * multiplication
- / division
- % modulo
- & bitwise and (integer expressions)
- **and** logical and (logical expressions)

Priority 4 addition operators:

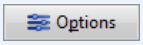
- + add (numeric expressions)
- + concatenate (string expressions)
- - subtract (numeric expressions)
- | bitwise or (integer expressions)
- **or** logical or (logical expressions)
- ^ bitwise exclusive or (integer expressions)
- **xor** logical exclusive or (logical expressions)

Priority 5 relational operators:

- = equal
- # not equal (<> is also accepted)
- > greater
- >= greater or equal
- < less
- <= less or equal

Remark:

Bitwise and logical operators use different lexical symbols (e.g., keyword **and** vs. delimiter **&**) by default, but the SPL compiler accepts both forms for both bitwise (integer) and logical (boolean) expression types.

The 

button, located at the **Program Editor** command pane, opens a dialog box, which allows setting of the user's preferences of code generation rules. So, it is possible to choose which form of bitwise and logical operators are to be used when generating the program source code from the graph representation. See the **Code generation from Graph mode** for details.

Integer and floating-point division

The / delimiter is used for both integer and floating-point division. The operator type (**integer** or **real**) is determined from its operands, like in the case of other bitwise operators. However, as opposite to addition, subtraction, and multiplication[12], integer and floating-point division may provide different results.

There are certain situations when both division operands are integer, but the algorithm needs floating-point result to operate correctly. One such example is conversion of hours into fraction of day:

```
variables
  Day : real;
  Hour : integer;
  …
instructions
  …
```

[12] We do not take possible integer overflow into account here.

```
Day := Day + Hour / 24;
```

The code above adds zero to the **Day** variable (providing **Hour** is less than 24) as both operands are integers and integer division of a number less than 24 by 24 is zero. To ensure the intended functionality of the example above, at last one (or both) division operands must be converted to floating-pointe format to ensure floating-point division is performed:

```
Day := Day + real( Hour ) / 24;
```

Or it is enough just to use a floating-point number literal:

```
Day := Day + Hour / 24.0;
```

The **24.0** number literal informs the compiler that the number is of type **real**, not **integer**, and the compiler then automatically converts also the **Hour** variable to the **real** type prior to division.

Function calling

A function is a block of code, which have zero, one or more input parameters and returns **one value** of certain type. The type of the returned value is also used as a type of the entire function.

Functions in SPL cannot be called (invoked) outside of some expression. All function calls are from within expressions, which evaluation causes calling of the function code and using of the returned value.

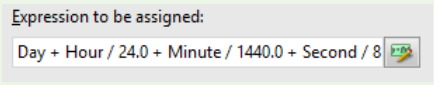
While SPL allows definition of arbitrary functions, certain set of the most frequently used functions is implemented as embedded ones. Embedded functions are readily available for programmers without a necessity to import another program that implements them etc. This also means the identifiers of the embedded functions are reserved (like program keywords) and cannot be used for naming of any other object in SPL.

There are significant differences in the syntax of parameter passing between embedded functions and functions implemented in SPL code, be it in the same program or imported from another program.

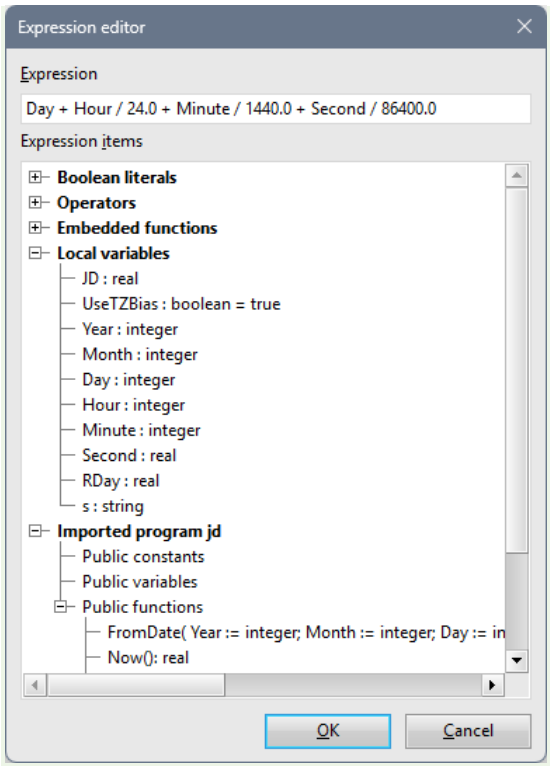
- **Functions implemented in SPL** code, be it in the current program or imported from another programs, use the same parameter passing like passing of input parameters in the **set** section when calling a procedure (procedures are explained later). Because functions have no output parameters, just single return value, the keyword **set** is omitted, and input parameters of the functions are defined in parentheses immediately after function name.
    - Input parameters are identified by their name, to which the passed values are assigned. This means assignment of input parameters may be done in arbitrary order.
    - Every input parameter has a default value, so assigning a value by a caller is optional. If no value is assigned to a parameter, the parameter is initialized to its default value.
    - Like in the case of the **call** instruction **set** section, also parameters passed to functions may optionally use the **shortened syntax** parameter passing.
- Input parameters of **embedded functions** have no default values neither have names.
    - The caller must always define all parameters, none can be skipped.
    - Parameter values must be set in the order defined by the embedded function prescription.

Hint:

The SIPS **Program Editor** tool allows the user to write a program in standard text editor or to create algorithms in graphical representation. Both **Graph** and **Text** modes of the **Program Editor** are thoroughly described later, for now let us just mention that every expression edit-box in the **Graph** mode contains a button opening **Expression Editor**.



The **Expression Editor** dialog box shows all possible expression items available, including all operators, embedded functions etc. So, it is not necessary to browse this manual to get the proper name of some embedded function.



Embedded functions

Boolean embedded functions:

- **assigned(v : local_in_variable)**
  This special function accepts only single local input variable of any type as an argument (as opposed to other functions, accepting any expression of required data type). Function **assigned** returns boolean value (**true** or **false**) indicating if the input variable, passed as a function argument, was assigned by the calling code, or the variable was left on its default value upon current procedure or function call.

Integer embedded functions:

- **abs(i : integer)**
  Returns absolute value.
- **high(a : array)**
  high(a : array; dimension_index : integer)
  Returns greatest index of the array **a**. The first argument is an identifier of array, but without indexes in square brackets. Because arrays are indexed from 0, the number of items is the array is high(a) + 1. The second argument is optional for one-dimensional arrays (vectors have one dimension only, so the dimension index would be always 0). For arrays with two or more dimensions, the second argument, specifying the dimension index, is mandatory.
- **integer(r : real)**
  Converts real argument to integer. Decimal part is truncated.
- **length(s : string)**
  Returns length of string in characters.
- **max(i1, i2 : integer)**
  Returns greater number of two arguments.
- **min(i1, i2 : integer)**
  Returns a smaller number of two arguments.
- **ord(s : string)**
  Returns ordinal number (ASCII code) of the first character of string.
- **pos(source, pattern : string)**
  Position of the **pattern** string in the **source** string. If the **pattern** string is not found, function returns -1. Positions are indexed from 0.
- **round(r : real)**

Returns rounded real value converted to integer.

- **sgn(i : integer)**
  Returns sign of the argument: 1 if argument > 0, -1 if argument < 0 and 0 if argument is 0.
- **shl(i : integer)**
  shl(i, by : integer)
  Shift bits left. With single arguments, bits are shifted by 1. If the second argument **by** is present, bits are shifted by its value left. The **by** variable may be from 1 to 31.
- **shr(i : integer)**
  shr(I, by : integer)
  Shift bits right. With single arguments, bits are shifted by 1. If the second argument **by** is present, bits are shifted by its value left. The **by** variable may be from 1 to 31.
- **val(s : string; base : integer)**
  Converts string to integer number. String is supposed to contain integer number with radix **base**. If the conversion fails, value 0 is returned.

Real embedded functions:

- **abs(r : real)**
  Returns absolute value.
- **acos(r : real)**
  Returns arcus cosine in the range 0 to π.
- **asin(r : real)**
  Returns arcus sine in the range $-\pi/2$ to $\pi/2$.
- **atan(r : real)**
  Returns arcus tangent in the range $-\pi/2$ to $\pi/2$.
- **atan2(y, x : real)**
  Returns arcus tangent of y/x in the range $-\pi$ to π.
- **ceil(r : real)**
  Returns nearest higher whole number.
- **cos(r : real)**
  Returns cosine of angle in radians.
- **cosh(r : real)**
  Returns hyperbolic cosine of angle.
- **deg(r : real)**
  Converts radians to degrees.
- **dms(degree, minute, second : real)**
  Converts angle expressed in degrees, minutes, and seconds into radians.

  Remark:

  For negative angels, make sure the first non-zero value is negative. For instance, the angle -0° 1’ 2”, passed as dms(-0, 1, 2) will return positive angle, as it is not possible to distinguish the negative sign (-0 is evaluated as +0). So, use dms(0, -1, 2) or -dms(0, 1, 2).

  This is the difference from SIPS GUI, where the negative sign can be distinguished even if stated as negative zero degrees, because the SIPS code checks the text representation of the degrees for a presence of the “-“ character.

- **exp(r : real)**
  Returns e^r.
- **floor(r : real)**
  Returns nearest lower whole number.
- **frac(r : real)**
  Returns fraction part of floating-point number.
- **hms(hour, minute, second : real)**
  Converts angle expressed in hours, minutes, and seconds into radians.

Remark:

Refr to the **dms()** function description for details about converting negative angles less than one hour.

- **ln(r : real)**
  Returns natural logarithm.
- **log(r : real)**
  Return logarithm base 10.
- **max(r1, r2 : real)**
  Returns greater number of two arguments.
- **min(r1, r2 : real)**
  Returns a smaller number of two arguments.
- **norm(angle : real)**
  Returns angle in radians normalized to range 0..2π.
- **pow(base, power : real)**
  Returns $base^{power}$.
- **rad(r : real)**
  Converts degrees to radians.
- **rand()**
  Generates a pseudo-random number in the range 0 to 1.
- **real(i : integer)**
  Converts integer argument to real.
- **round2(r : real; num_places : integer)**
  Returns real value rounded to order num_places.
- **sin(r : real)**
  Returns sine of angle in radians.
- **sinh(r : real)**
  Returns hyperbolic sine of angle.
- **sqrt(r : real)**
  Returns square root.
- **tan(r : real)**
  Returns tangent of angle in radians.
- **tanh(r : real)**
  Returns hyperbolic tangent of angle.
- **val(s : string)**
  Converts string to real number. If the conversion fails, value 0.0 is returned.

String embedded functions:

- **caps(s : string)**
  Returns string with all letters replaced with capitals.
- **char(i : integer)**
  Returns string containing character with ordinal (ASCII table value) number i.
- **delete(source : string; position, length : integer)**
  Returns string **source**, with deleted **length** characters beginning at **position**. Positions are indexed from 0.
- **insert(source, target : string; position : integer)**
  Returns string **source**, with the **target** string inserted at **position**. Positions are indexed from 0.
- **item(source, delimiters : string; index : integer)**
  Returns N^{th} sub-string of the **source** string, separated from rests of source with any character(s) in **delimiters**. The number of sub-string is **index**, beginning from 0.
- **lows(s : string)**
  Returns string with all capitals replaced with small letters.
- **lpad(s : string; length : integer)**
  Returns string **s**, padded with spaces from left up to **length**. If string s length is greater or equal **length**, it is returned without changes.
- **ltrim(s : string)**

Returns string s with all left spaces cut (if any).

- **replace(source, target : string; position : integer)**
  Replaces a substring in **source** from **position** with **target**. Positions are indexed from 0.
- **rpad(s : string; length : integer)**
  Returns string **s**, padded with spaces from right up to **length**. If string s length is greater or equal **length**, it is returned without changes.
- **rtrim(s : string)**
  Returns string s with all right spaces cut (if any).
- **slice(s : string; position, length : integer)**
  Returns sub-string of **source** beginning at **position** with **length**. Positions are indexed from 0.
- **str(i : integer)**
  Converts integer to string.
- **str(r : real)**
  Converts real to string.
- **str(i : integer; base : integer)**
  Converts integer to string with a radix **base**.
- **str(r : real; format : string)**
  Converts real to string using **format** string.
- **subst(source, pattern, target : string)**
  Replaces the **pattern** string in the **source** with **target**.
- **trim(s : string)**
  Returns string s with all spaces on left and right cut (if any).

Formatting number conversions to string

The **str(r : real; format : string)** function uses s format string as a second parameter. Formatted conversion of numbers to string may be used if it is desirable to specify e.g. number of digits after decimal points etc.

Numbers are formatted according to a pattern of special characters within the format string. Special characters are:

- '#' is replaced with a number on the corresponding position. If there is no valid number at the position, a space ' ' is used.

```
str( 3.14, "###.###" ) returns "  3.14 "
```

- '@' is replaced with a number on the corresponding position. If there is no valid number at the position, '@' are replaced with zeros '0'.

```
str( 3.14, "@@@.@@@" ) returns "003.140"
```

- '*' is replaced with a number on the corresponding position. If there is no valid number at the position, no character is inserted into return string.

```
str( 3.14, "***.***" ) returns "3.14"
```

- All other characters are just copied to the resulting string.

Different formatting characters can be used for whole number part and decimal part of the number. For instance, the formatting string "###.@@@" fills the unused places before the decimal point with spaces, while the fraction part is padded with zeros.

If the number of valid numbers prior to decimal point exceeds number of formatting characters, the formatting characters are left in the string without replacing with a number.

If a dedicated formatting characters should be used in the output as is, do not place them into the formatting string, but use string concatenation. For instance, instead of using of formatting:

```
str( item, "Item # @@@@" )
```

use:

```
"Item # " + str( item, "@@@@" )
```

Procedures

As mentioned in the introduction to SPL, all code is contained in procedures (and functions). Procedure structure is as follows:

- Procedure starts with keyword **procedure**, followed by procedure name identifier.
    - Optional attribute **public** may follow the procedure name. This attribute means the procedure is visible from outside of current program, which means it can be called from other programs as well as used as program entry point by the user.

      Hint:

      The **Program Editor** tool marks all newly created procedures as **public** by default, not to confuse users with error messages when the try to call non-public, yet existing procedure. It is upon the user to decide which procedure is intended to be visible from outside and which is purely internal to the program.

- Optional **remark .. end_remark** section syntactically corresponds to the **remark** instruction within the code. Procedure remark is intended to provide comments related to the actual procedure.
- Optional section **constants** allows definition of local constants.
    - Each constant is defined as **identifier = constant_expression**
    - Initialization expression must be constants, the compiler must be able to evaluate id at compile time. So, it must not contain any variable or call to SPL defined functions. Only previously defined constants and embedded functions may be used.
    - Each constant has a type of its initialization expression.
- Optional section **variables** allows definition of local variables.
    - Each variable is declared as **identifier : type**
    - Every variable is initialized to default value according to declared type.
    - Optionally it is possible to append initialization expression, if the initial value should be different from default one **identifier : type = constant_expression**
    - Initialization expression must be constants, the compiler must be able to evaluate it at compile time. So, it must not contain any variable or call to SPL defined functions. Only previously defined constants and embedded functions may be used.
    - Optional attribute **static** indicates the variable is initialized only upon the first procedure call during program execution. Subsequent calls of the procedure do not initialize the variable, its value is maintained during the program execution.
    - Optional attribute **in** indicates the caller may set this variable prior to call of this procedure. If the procedure is used as program entry point, this variable can be set by the user. Setting of input variables is optional, if it is not set, defined initialization values are used.
    - Optional attribute **out** indicates the caller may get this variable after this procedure returns from the call.
- The procedure code itself begins after the keyword **instructions**.
- Procedure ends with keyword **end_procedure**.

Procedures can be called from other code using the instruction **call**. The **call** instruction is described later.

Procedure interface

SPL procedures do not declare parameters, intended to pass data from the caller and back to caller, and local variables in different ways. Local variables, input, and output parameters are declared the same way in the **variables** block.

Any variable marked with the attribute **in** is also part of the procedure input interface, allowing the caller to pass parameters into procedure.

Any variable marked with the **out** attribute is also part of the procedure output interface, allowing to return results from the procedure back to the caller.

A variable may be marked both **in** and **out**, in such case it is part of both input and output interface. Its value can be set by the caller prior to procedure call and also read back after the procedure returns.

The source code example below shows a procedure, which converts tangentially projected coordinates x, y to spherical coordinates RA and Dec:

```
procedure XYToRADec; public;
remark
  Convert Cartesian [x, y] to spherical [RA, Dec] coordinates using tangential projection
end_remark;
variables
  x       : real; in;
  y       : real; in;
  RA0     : real; in;
  Dec0    : real; in;
  RA      : real; out;
  Dec     : real; out;
  SinDec0 : real;
  CosDec0 : real;
instructions
  SinDec0 := sin( Dec0 );
  CosDec0 := cos( Dec0 );
  RA := RA0 + atan2( x, CosDec0 - y * SinDec0 );
  Dec := asin( (SinDec0 + y * CosDec0) / sqrt( 1 + x * x + y * y ) );
end_procedure;
```

The caller code needs not to define new values (assign expressions) to all input variables of the called procedure or function. Parameters are always identified by their names and any parameters can be left on their default values.

However, sometimes the procedure or function may need to know whether the caller assigned particular variable or not as the procedure or function behavior may differ. The embedded function **assigned(parameter_name)** returns **true** if the caller code assigned the parameter.

Hint:

It may seem that testing the parameter for default value may be used to distinguish if the parameter was assigned or not, but Note test for the default value is not enough, at the user may set the same value as the default one.

Passing entire arrays to/from procedure

Also array variables may be marked as input or output. In such cases, entire arrays are passed to/from procedure. If entire arrays are passed, the array identifier is used without indexing expression(s) in square brackets. Passing of entire arrays brings some important features:

- Arrays must be type compatible (which is of course valid for passing of scalar values). **Real** and **integer** types are compatible, and types are converted during assignment.
- Whole arrays must also have the same number of dimensions. For instance, it is not possible to assign vector into matrix etc.
  The motivation for this limitation is simple—the code handling the array already contains defined number of indexing expressions for respective array variable. For example, it is not possible to address an item in matrix with single index expression originally compiled for vector variable etc.
- However, number of items in every dimension may differ. The input array variable in the procedure may be declared as single-element array and the array with more elements may be assigned.
  SPL includes an embedded function high, which returns the highest index of the array in each dimension.

For instance, a procedure calculating the average value of all items in real vector may look like this:

```
procedure Average;
variables
  a   : real[1]; in;
  avg : real; out;
  i   : integer;
instructions
  avg := a[ 0 ];
  for i := 1 to high( a ) do
    avg := avg + a[ i ];
  end_for;
  avg := avg / (high( a ) + 1);
end_procedure;
```

The calling code may look like this:

```
…
variables
  values : real[10] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
  avg    : real;
instructions
  call Average set( a := values ) get( avg := avg );
  …
```

Noticed we assigned the 10-items long vector “values” into 1-item long vector “a” in the procedure. This caused the vector “a” in the procedure become a vector containing 10 values. To avoid a necessity to pass the actual length of the passed array, which would be very inconvenient and also would be a cause of many code bugs, the embedded function **high** is used to determine actual maximum index of the passed array.

Functions

Function structure is as follows:

- Function starts with keyword **function**, followed by procedure name identifier.
- Every function must return value of certain type. **Return type** is declared after function name and delimited by colon, for instance **function Calculate: real**
    - Optional attribute **public** may follow the type declaration. This attribute means the function is visible from outside of current program, which means it can be called from expressions in other programs.
- Optional **remark .. end_remark** section syntactically corresponds to the **remark** instruction within the code. Function remark is intended to provide comments related to the actual function.
- Optional section **constants** allows definition of local constants.
    - Each constant is defined as **identifier = constant_expression**
    - Initialization expression must be constants, the compiler must be able to evaluate id at compile time. So, it must not contain any variable or call to SPL defined functions. Only previously defined constants and embedded functions may be used.
    - Each constant has a type of its initialization expression.
- Optional section **variables** allows definition of local variables.
    - Each variable is declared as **identifier : type**
    - Every variable is initialized to default value according to declared type.
    - Optionally it is possible to append initialization expression, if the initial value should be different from default one **identifier : type = constant_expression**
    - Initialization expression must be constants, the compiler must be able to evaluate it at compile time. So, it must not contain any variable or call to SPL defined functions. Only previously defined constants and embedded functions may be used.
    - Optional attribute **static** indicates the variable is initialized only upon the first procedure call during program execution. Subsequent calls of the function do not initialize the variable, its value is maintained during the program execution.
    - Optional attribute **in** indicates the caller may set this variable prior to call of this function. Setting of input variables is optional, if it is not set, defined initialization values are used.
- The procedure code itself begins after the keyword **instructions**.
- The function return value is defined with the instruction **return**, followed by the expression. The expression must be of the same type as declared after function name. Every execution branch of the function must end with **return** instruction, function execution cannot end without returning value.
- Procedure ends with keyword **end_function**.

Function interface

Like in the case of procedures, any variable marked with the attribute **in** is also part of the function input interface, allowing the caller to pass parameters into function.

The **out** attribute is not allowed in functions, functions return just one value, used during evaluation of expression, which contains call to the function.

The source code example below shows a function, which calculates air mass from the height above horizon:

```
function AltToAirMass: real; public;
remark
  Calculates airmass from altitude using Pickering (2002) formula
end_remark;
variables
  Alt : real; in;
  hDeg : real;
  a : real;
instructions
  if (Alt >= 0) and (Alt <= pi / 2) then
    hDeg := deg( Alt );
    a := rad( hDeg + 244 / (165 + 47 * pow( hDeg, 1.1 )) );
    return 1 / sin( a );
  else
    return -1;
  end_if;
end_function;
```

Also function input parameter may be an array, and it is possible to pass entire array into it. The procedure calculating an average of a vector of real numbers above may be implemented as function like this:

```
function Average: real;
variables
  a   : real[1]; in;
  avg : real; out;
  i   : integer;
instructions
  avg := a[ 0 ];
  for i := 1 to high( a ) do
    avg := avg + a[ i ];
  end_for;
  return avg / (high( a ) + 1);
end_procedure;
```

The calling code may look like this:

```
…
variables
  values : real[10] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
  avg    : real;
instructions
  avg := Average( a := values );
  …
```

Instructions

SPL instructions are used to express algorithms. There are two kinds of instructions:

- **Control instructions** define a flow of the program. Control instructions are responsible for branching of the code execution according to defined conditions, for repeated execution of instruction blocks for a defined number of iterations or while a condition is true, or for transferring of program execution to another procedure etc.
- **Executive instructions** mostly evaluate expressions and modify variables. But SPL defines other instructions, which can be also classified as executive, like instruction **print**, which sends texts to the program output window.

By other words, control instructions decide what should be done and executive instructions do the actual work.

Some instructions are a part control and part executive. For instance, the **call** instruction redirects the program execution to the procedure, like control instructions, but at the same time assigns variables with values returned by the called procedure, like the executive instruction.

Instruction “if”

Instruction **if** allows branching of the code execution according to a boolean expression.

```
if boolean_expression then
  …
end_if;
```

If the boolean expression result is **true**, the code between keywords **then** and **end_if** is executed. Otherwise, it is skipped and code continues with an instruction after **end_if**.

A variant of the **if** instruction allows executing of the code also when the boolean expression result is false.

```
if boolean_expression then
  …
else
  …
end_if;
```

If the boolean expression result is **true**, the code between keywords **then** and **else** is executed, and then the code continues with an instruction after **end_if**.

If the boolean expression result is **false**, code after **then** is skipped up to **else**, and then the code between **else** and **end_if** is executed.

It is sometimes necessary to evaluate multiple expressions to divide code execution into multiple branches. For instance, the "signum" function should return:

- -1 for arguments less than 0
- 0 for arguments equal 0
- +1 for arguments greater then 0

The code implementing this function could be like:

```
if number < 0 then
  return -1;
else
  if number = 0 then
    return 0;
  else
    return 1;
  end_if;
end_if;
```

The **else** branch contains single instruction **if**, again containing **then** and **else** branches. Such cases occur quite often in various algorithms, which is why the SPL language **if** instruction allows to define any number of **elsif** branches. The above code can be written in a more concise way:

```
if number < 0 then
  return -1;
elsif number > 0 then
  return 1;
else
  return 0;
end_if;
```

As in the case of simple **if** instruction, the **elsif** conditions are evaluated only if the first **if** condition is **false**. The first **elsif** branch, which condition is evaluated **true**, is executed. Subsequent **elsif** conditions are not even evaluated and the code continues after **end_if**. The **else** branch is optional and its code is executed if none of the **if** … **elsif** conditions are met.

Instruction "for"

Instruction **for** performs a counted loop, it executes the code for a defined number of times. The **for** loop instruction needs a control variable, which is incremented (or decremented) with each pass through the loop and tested to determine if the desired number of loops was already executed.

```
for control_variable := init_expression to limit_expression do
  …
end_for;
```

The control_variable, as well as init_expression and limit_expression must be of type **integer** or **real**.

The control_variable is initiated to the result of init_expression. If the control_variable is less or equal to a result of limit_expression, the instruction up to the **end_for** are executed, the control_variable is then increased by 1 and again compared to the result of limit_expression. Otherwise, program continues with instruction after **end_for**.

If the init_expression result is greater than limit_expression result right from the beginning, instructions inside the **for** loop are not executed at all and program continues after **end_for**.

If it is desirable to update the control_variable by different value than 1 or possibly by negative value, the step is stated after the keyword **by**.

```
for control_variable := init_expression to limit_expression by step_expression do
  …
end_for;
```

If the step value is negative, the test for limit expression is of course inverted. The loop continues while the control_variable is greater or equal to the limit_expression result. For instance:

```
for i := 10 to 1 by -1 do
  …
end_for;
```

As the control variable can be of type real, also the step can be fractional number. For instance:

```
for angle := 0 to 2 * pi by pi / 4 do
  …
end_for;
```

Instruction “while”

The **while** instruction performs loop while the boolean_expression result is **true**. When the result is **false**, program continues after the **end_while**.

```
while boolean_expression do
  …
end_while;
```

If the condition if **false** right from the beginning, the block of instructions inside the **while** loop is entirely skipped and program continues after the **end_while**.

Instruction “repeat”

The **repeat** instruction performs loop until the boolean_expression result is **true** (this means while the condition is **false** and repeat loop exits when the condition becomes **true**). When the result is **false**, program continues after the **end_while**.

```
repeat
  …
until boolean_expression;
```

Regardless of the boolean_expression result, the block of instructions inside the **repeat** loop is always executed at last once.

Instruction “loop”

The **loop** instruction performs infinite loop. But such instruction would be worthless without a possibility to terminate the loop (infinite loop is a programming error). Instruction **exit**, anywhere inside the loop, terminates the loop and causes the program continues after the **end_loop**.

```
loop
  …
  if boolean_expression then
    exit;
  end_if;
  …
end_loop;
```

The **loop** instruction is a generalization of both **while** and **repeat** loops. With the condition at the loop begin, it acts as **while**. With the negative condition at the loop end, it acts as **repeat**. But the **exit** instruction can be anywhere inside the loop and there can be more than one **exit** inside the loop. Still, it is recommended to use **while** and **repeat** loops whenever possible to increase program clarity.

If there are more **loop** instructions nested (a **loop** inside another **loop**), the **exit** instruction always exits the nearest loop.

The **exit** instruction outside of the **loop** instruction block is not allowed.

Instruction "pause"

Because SPL is primarily intended for controlling of the observation sequences of robotic telescopes, programmers must be able to synchronize the program execution with real time.

The **pause** instruction delays the program execution for the defined number of seconds. This means the delay is relative, it is counted from the actual time of the **pause** instruction execution.

```
pause real_expression;
```

The **pause** instruction is implemented not to consume processor cycles during the wait. Instead, the program interpreter enters a sleep state and lets the operating system to wake it up after the desired time expires.

> Hint:
>
> The time parameter is a real number, so the pause length may be less than a second. For instance, **pause 0.01** instruction delays the program execution for 10 ms.

Instruction "wait"

The **wait** instruction implements waiting up to absolute time. Its argument is the Julian Date and **wait** instruction pauses program execution until the current Julian Date if greater or equal to the argument value.

```
wait real_expression;
```

SPL offers a set of native programs (more about importing other programs and available native programs is explained later). Native program "jd" exports functions, allowing to handle Julian Date calculations. The following example shows a **pause** instruction, which pauses program execution for 0.5 s, as well as the **wait** instruction, also pausing the program for another 0.5 s.

```
pause 0.5;
wait jd.Now() + 0.5 / 86400;
```

The call to function Now() from the imported program "jd" returns Julian Date at the instance of function call. The 0.5 / 86400 is half a second converted to days (**pause** instruction uses seconds, while the **wait** instruction uses days).

It is often desirable to wait until some specific local date and time, expressed in common calendar values (year, month, day, hour, minute, and second). The "jd" program offers function FromDate, which performs conversion of local date and time to Julian Date. The FromDate function uses the time zone information currently set in the operating system.

```
wait jd.FromDate( Year := 2023; Month := 10; Day := 30; Hour := 15; Minute := 35 );
```

Note the "FromDate" function is not an embedded function, but a function implemented in SPL (despite it is implemented in native program and its source code is not visible to thew users) and thus it uses name-based parameter passing instead of positional parameter passing.

Also, all parameters passed to SPL procedures are optional, so the example above misses the parameter **Second := 0**, as the default value is also 0 and it is not necessary to assign same values as the default ones.

Instruction "call"

The **call** instruction passes the program execution to the specified **procedure** (see the Procedures chapter). When the code in the specified procedure ends, program execution continues with the instruction following the **call** instruction.

```
call procedure_name;
```

> Remark:
>
> The current version of SPL does not support recursive procedure calls.

Each procedure can implement an interface, allowing the caller code to pass values into procedure before it is called and, after the procedure finishes, to read back results. The procedure interface consists of local variables marked with attributes **in** (for variables, which can be set by the caller prior the program execution enters the procedure) or **out** (for variables, which value the caller can read back after the execution of the procedure finishes). It is also possible to mark the variable both **in** and **out**.

The keyword **set** may follow the procedure name. The **set** keyword is followed by a list of assignments in parentheses, which sets values of input parameters. These assignments are like general assignment instructions (see the description of the instruction **let**)—remote **in** variable name is followed by assign operator := and an expression of the compatible type.

Then a keyword **get** may follow. It defines section, in which procedure results can be copied back to the caller code. The syntax is similar to the **set** section, but the remote and local sides are swapped—local variable sits on the left and a remote variable on the right is assigned to it. Also, while in the **set** section the right side can be an expression, in the **get** section the right side is a single remote variable.

```
call procedure_name set( remote_in_varaible := local_expression; … )
                    get( local_varaible := remote_out_variable; … );
```

For instance, suppose we have a procedure, which calculates equatorial position of the Sun. Beside other local variables necessary to perform calculations, the procedure defines input variable JD and output variables RA and Dec.

```
procedure SunPosition;
variables
  JD : real = 0; in;
  RA : real = 0; out;
  Dec : real = 0; out;
  …
instructions
  …
end_procedure;
```

The caller code can call the procedure:

```
variables
  alpha : real;
  delta : real;
  …
instructions
  …
  call SunPosition set( JD := jd.Now() )
                  get( alpha := RA; delta := Dec );
  …
```

The procedure input parameter named JD is assigned with a result of a function Now(), which returns current Julian Date, in the **set** section. The caller code also declares two real variables alpha and delta for the Sun's equatorial coordinates. Then it calls the "SunPosition" procedure and after the procedure finishes, it copies back resulting coordinates to the local variables alpha and delta in the **get** section.

Both input and output parameters are passed using their names. This means the order of parameters is arbitrary. Also, any parameter, both input and output, may be skipped. Non-assigned input parameters are then initialized with their default value. Input parameters are also local variables and every local variable may have initialization expression. Even if the initialization expression is missing, the default value is used. If the output parameter is skipped, the caller code simply does not read the value of the variable.

Hint:

The embedded boolean function **assigned(paremeter_name)** may inform the called procedure code if the particular input variable was assigned by the caller or not.

Like individual parameter, the whole **set** and **get** sections may be omitted, if the caller code does not want to change parameter default values or is not interested in procedure results.

Hint:

The **call** keyword is optional; the SPL syntax allows using it but does not require it. Because the **Program Editor** tool is capable to generate text form from a graph representation of the program, it needs to know if the user prefers to use the **call** keyword or not. See the Code generation from Graph mode for details.

Also, the **let** keyword (described later) is optional and the dialog allows to select generation of both keywords independently. By default, the **let** keyword is skipped, but the **call** keyword is generated to help especially novice SPL users navigate the program source code.

Passing of whole arrays

If the input or output variable is an array, entire array must be passed. It is not possible to read e.g. single element of the output array into scalar variable.

Let us have for instance matrix multiplication procedure:

```
procedure Mul;
remark
  Matrix multiplication M = A . B
end_remark
variables
  A     : real[1, 1]; in;
  B     : real[1, 1]; in;
  M     : real[1, 1]; out;
  i     : integer;
  j     : integer;
  k     : integer;
  Error : boolean; out;
instructions
  if high( A, 1 ) <> high( B, 0 ) then
    Error := true;
  else
    resize M[high( A, 0 ) + 1, high( B, 1 ) + 1];
    for i := 0 to high( A, 0 ) do
      for j := 0 to high( B, 1 ) do
        for k := 0 to high( A, 1 ) do
          M[i, j] := M[i, j] + A[i, k] * B[k, j];
        end_for;
      end_for;
    end_for;
  end_if;
end_procedure;
```

Then we can pass entire matrices:

```
…
variables
  M1 : real[3, 3] = [[1,2,3],[4,5,6],[7,8,9]];
  M2 : real[3, 3] = [[9,8,7],[6,5,4],[3,2,1]];
  M3 : real[1, 1];
instructions
  call Mul set(A := M1; B := M2)
           get(M3 := M);
  …
```

Hint:

Notice the matrices A, B, and M in the “Mul” procedure are declared as [1,1] only and their real dimensions after assignment are determined using the **high** function. However, the resulting matrix M is not assigned with a larger matrix and it must be resized inside the procedure to fit the result using the **resize** instruction. The **resize** instruction is described later.

Shortened syntax of parameter passing

The SPL compiler supports shortened syntax of parameter passing. Such syntax can be used if both left and right sides of the parameter assignment are expressed with the same single identifier. This means the expression on the right side of the assign contains just one local or global constant or variable, and this variable is name equals to the name of the called procedure input parameter name. Let us consider the following example:

```
variables
  RA : real;
  Dec : real;
  JD : real;
  …
```

```
instructions
  …
  JD := jd.Now();
  call SunPosition set( JD := JD )
                   get( RA := RA; Dec := Dec );
  …
```

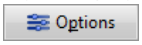
The call can be written in a shortened way like this:

```
  call SunPosition set( JD )
                   get( RA; Dec );
```

The short syntax may be freely interweaved with the full assignment syntax in the source code.

Shortened syntax cannot be used if an array element is assigned into input scalar variable of the procedure or function, despite the source array and target scalar variable having the same name. This is because the source array must contain index expressions in square brackets while the target scalar variable has no indexes.

```
…
variables
  A : real[3, 3] = [[1,2,3],[4,5,6],[7,8,9]];
  B : real[3, 3] = [[9,8,7],[6,5,4],[3,2,1]];
  M : real[1, 1];
instructions
  call Mul set(A; B)
           get(M);
  …
```

Usage of the shortened parameter passing syntax when the **Program Editor** generates source text from the **Graph** representation is controlled in the 

dialog box, mentioned above.

Instruction “return”

The **procedure** code ends and the program execution returns to the caller when the last instruction of the procedure **instructions** block is executed. But sometimes it is desirable to end the procedure code execution prematurely, for instance is some condition is not met. In this case the instruction **return** terminates procedure code and returns control to the caller immediately.

The **return** function is somewhat different when used in a **function**:

- The **return** in function must be followed by an expression of the same type like the function return value. This expression is evaluated upon return from the function and used as the function value.

```
  return expression;
```

- The **return** statement if mandatory in functions—a function must always return some value. If the function code branches, without joining the branches at the end of the function, every branch must terminate with a **return** statement.

Instruction “stop”

The **stop** instruction terminates execution of the whole program. It could be used e.g. to handle critical errors. Programs can be stopped anywhere; this means in the main procedure, but also in any other procedure or function.

Instruction “let” (assign instruction)

The assign instruction evaluates an expression, and the result assigns to the variable on the left side. Both variable and expression types must be compatible (boolean to boolean, numeric to numeric, string to string).

```
let variable := expression;
```

The assign instruction can be introduced by the keyword **let**, but because the usage of the **let** keyword is optional and it is not generated by default (see the **call** instruction description for details about optionally generated keywords), the **let** instruction typically looks just like this:

```
variable := expression;
```

The assign instruction uses the operator **:=** to distinguish it from the comparison (assignment is not an equation like comparison, it modifies a left-side variable with a right-side expression value).

Remark:

The SPL syntax allows usage simple equation symbol = in the place of the assign operator, but the **Program Editor** tool always uses proper assign operator := when generating program source text from graphic representation.

Remember that assignment is not limited to numerical values, any value type can be assigned. For instance, the following code:

```
if a = b then
  res := true;
else
  res := false;
end_if;
```

is correct, but the same functionality can be achieved by single line:

```
res := a = b;
```

If there is an array element on the left side of the assignment, the indexing expression(s) must follow the array variable name:

```
for i := 0 to high( a ) do
  a[ i ] := i;
end_for;
```

Assigning whole arrays

Whole arrays can be assigned in the SPL program identically to passing of whole arrays to and from procedures. The same limitations apply (arrays must be type-compatible and number of dimension of both arrays must be the same).

Note while scalar value assignment evaluates an expression on the right side and the resulting is assigned to the value on the left side, in the case of whole arrays, arrays identified just by their respective names are on both left and right sides of the assignment; there are no whole-array expressions avaliable.

Instruction "resize"

As mentioned above, SPL code is able to change the dimension sizes of an array, despite the array type and number of dimensions must be always preserved, because the code already contains a defined number of indexing expressions.

In some cases, changing dimension sizes is performed implicitly through whole array assignment. When a whole array is passed to or from a procedure or assigned to another array using the **let** instruction, the dimension sizes of the target array is set to the sizes of source array.

However, some algorithms require explicit resizing of array dimensions, typically to fit the sizes of source arrays. For instance, the result of multiplication of matrices A and B is a matrix with depth of the matrix A and width of the matrix B. So, the matrix multiplication code starts with resizing the resulting matrix M with **resize** instruction (remember the A and B input matrices are resized upon assigning input matrices to them):

```
procedure Mul;
variables
  A : real[1, 1]; in;
  B : real[1, 1]; in;
  M : real[1, 1]; out;
  …
instructions
  resize M[ high( A, 0 ) + 1, high( B, 1 ) + 1 ];
  …
```

Warning:

Any attempt to access an array element beyond the defined size of any dimension causes runtime error and program termination. This protects SPL code from memory corruption and buffer overruns, often causing system crashes.

So, as good practice, the SPL code should never rely on some dimension range if the entire array is assigned or resized. Always use the embedded function **high** to get the actual maximum indexes of all array dimensions.

Zero-length arrays

No dimension of an array can be declared less or equal than zero. This means, every declared array has at last one item.

However, sometimes arrays are used to pass e.g. list of connected cameras or a list of filters available in the camera etc. Then it is necessary to handle a situation that no camera is currently available or a camera has no filters. An instruction or procedure can indicate such situation by returning **zero length array**. The code handling the returned array can figure out the array is zero-length through a **high** function call, which returns -1 is this case.

> Hint:
>
> As mentioned earlier, the dimension size of an array is a result of **high** function + 1. So, -1 + 1 = 0 items.

Naturally, any attempt to access any element of zero-length array leads to runtime error, even the index 0 is invalid. But if the array is always accessed up to index returned by **high** function, the code should be safe. The body of the **for** loop:

```
for i := 0 to high( V ) do
  sum := sum + V[i];
end_for;
```

is not performed as the loop counter going from 0 to -1 means the **for** loop body is fully skipped.

The zero-length arrays are allowed for one-dimension arrays (vectors) only. Two and more dimensional arrays cannot be zero length.

Also, zero-length array cannot be declared. It may be created as a result of **resize** instruction or may be returned e.g. from **invoke** instruction.

Instruction "invoke"

The **invoke** instruction is somewhat similar to the instruction **call**. Only the **invoke** instruction does not call a procedure implemented in SPL, but invokes one of the SIPS REST API endpoints. Explaining the SIPS REST API in detail is beyond the scope of this guide; the https://restfulapi.net/ site contains detailed explanation.

> Remark:
>
> The SIPS REST API is primarily designed to allow access to SIPS devices like camera, focuser, telescope mount etc. from web-based applications (see the **Remote Access Web Server** chapter for details). SLP reuses this API for the same purpose—to access all hardware devices controlled by SIPS from the program code.

Also, the syntax of the **invoke** instruction resembles the syntax of procedure **call**. Instead of procedure name, which must be an identifier in SPL, the target REST API endpoint to be invoked is identified by an URL in quotes, as the URL string may contain characters not allowed in identifiers (typically slashes / delimiting folders).

```
invoke "endpoint URL"
```

The REST API uses a more complex parameter passing compared to simple **in** and **out** variables, used for procedure call in SPL. Client (SPL code in our case) and server implementing the REST API (SIPS core) exchange structured texts, formatted using the JSON standard. Again, this guide cannot describe the JSON format, visit the https://www.json.org/json-en.html site for details.

Like the **call** instruction, also the **invoke** instruction use **set** section to pass parameters to REST endpoint and **get** section to retrieve results back to SPL code. The content of the **set** and **get** sections is a structure similar to JSON format, but slightly streamlined:

- As opposed to JSON, where record items can be named by general strings (including spaces) and thus also item names are stated in quotes, SIPS REST API always uses **identifiers** to name record items. So, item names in the **set** and **get** blocks are identifiers without quotes.
- Like in the **call set** block, the right sides of the **invoke set** block are SPL expressions of the proper type.
- And as in the **call get** block, the right sides of the **invoke get** block are SPL variables, to which the returned values are copied.
- JSON records (structures) are marked using curly brackets {} like in JSON.

```
invoke "endpoint URL"
  set {
    keyword1: expression;
    keyword2: {
      keyword3: expression;
      keyword4: expression;
      …
    }
    …
  }
  get {
    keyword1: variable1;
    keyword2: {
      keyword3: variable3;
      keyword4: variable4;
      …
    }
    …
  };
```

- JSON arrays are marked using the same delimiters as in JSON by square brackets [] but one important limitation. Because SPL currently does not support structures (data records), only arrays of simple data type are supported. The SIPS REST API is designed to comply with this limitation—instead of array of records, SIPS REST API uses record of arrays.
  When arrays are passed to/from the **invoke** command, it is always necessary to use whole arrays. The **invoke** command **set** JSON array is created according to the number of items in the passed array and similarly, whole array read from **get** JSON are resized according to number of items in the JSON array.

```
invoke "endpoint URL"
  set {
    keyword1: [
      array1
    ]
    keyword2: [
      array2
    ]
    …
  }
  get {
    keyword1: [
      array1
    ]
    keyword2: [
      array2
    ]
    …
  };
```

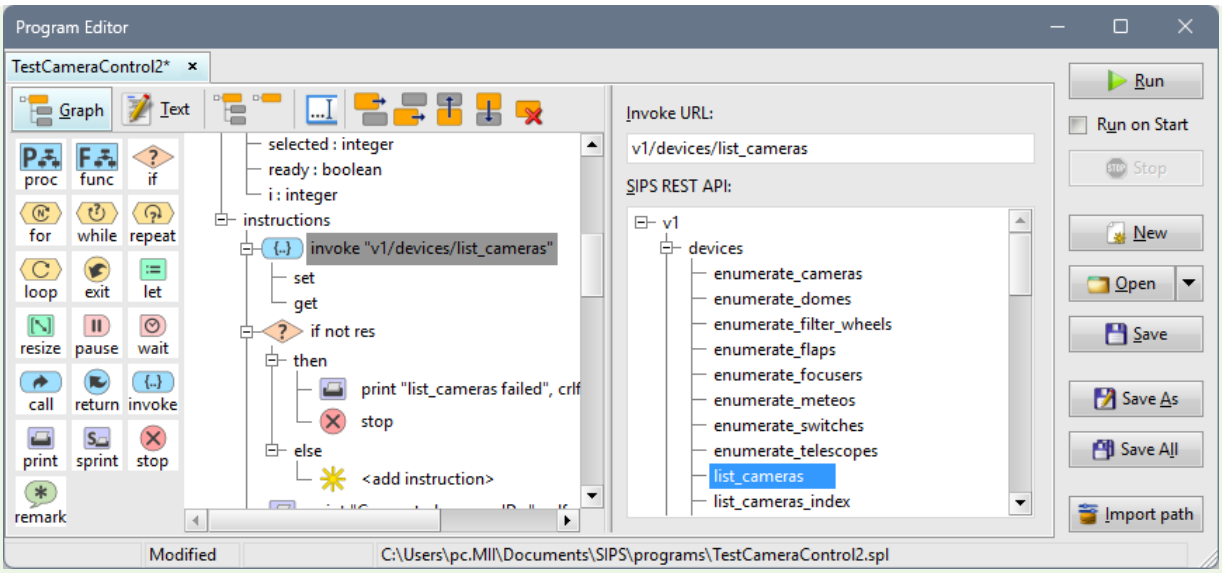
Warning:

The **invoke** command can return zero-length array if the received JSON did not contain any array item. An attempt to access any item of such array would lead to runtime error and program termination. Always use the embedded function **high** to determine the size of arrays returned from **invoke**.

If the invoked endpoint does not define input or output or if the SPL program does not set or get any item, the **set** and/or **get** sections may be completely omitted in the source code (like in the case of the **call** instruction, where both setting and getting parameters are optional).

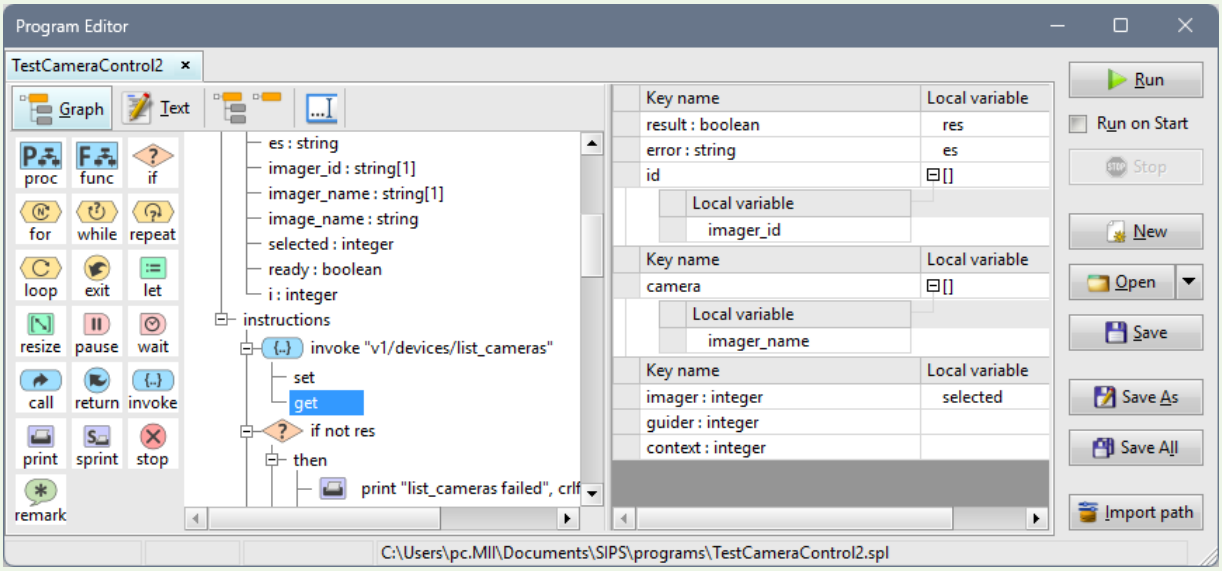
Hint:

The **Program Editor** tool in the **Graph** mode offers all available REST API endpoint URLs when the **invoke** instruction is selected.

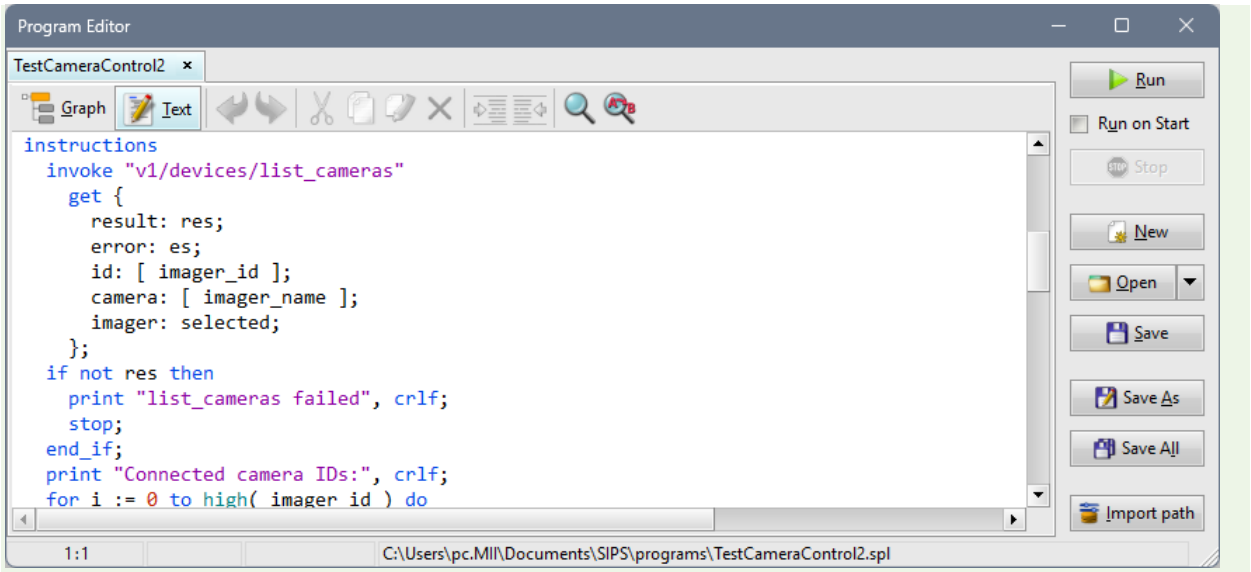


So, it is possible to pick the URL from a tree of all available endpoints, displayed below the URL edit box, on the right-side instruction parameter pane.

Also, the **set** and **get** blocks offer a JSON-like structure, into which it is possible to write expressions (for **set** block) and pick variables (for the **get** block).



The invoke instruction, selected, and parametrized in the Graph mode, is then properly generated to program source code when the Program Editor is switched to the Text mode.



Instruction "print"

The **Program Editor** tool offers **Graph** and **Text** modes to create the programs (see the following chapter for details). When the program is run, the **Program Editor** tool switches to the third mode—**Program Output**.

By default, the **Program Editor** writes the "Program Started" and Program Stopped" messages to the output text. But the programmer is free to write any text to the program output window using the **print** instruction.

```
print expression, expression, …, expression;
```

The **print** instruction is followed by a list of expressions, delimited by commas. Each expression is evaluated and the result is inserted into program output text.

- Boolean expression values are represented by keywords "false" and "true".
- Numeric expression values are converted to string representation. However, there are several options how to change the number formatting.
  - Integer numbers can be printed using any radix, using the **str** function with a second numeric argument.
  - Both integer and real numbers may be arbitrary formatted using a **str** function with a second, string argument, which defines formatting string. See the **str** function description for details.
- String expression results are just put to the output text.

There are three reserved keywords, which can be used in the comma-separated list of expressions following the **print** instruction.

- The **crlf** keyword causes the following output will start on the next line.
- The **cls** (clear screen) keyword clears the output text window.
- The **cll** (clear line) keyword clears the output text last line.

Instruction "sprint"

The **print** instruction is unique as it accepts virtually unlimited sequence of expressions, which evaluated values are written into program output window. The variant of the instruction named **sprint** works similarly, only the result is not directed to the program output window, but to the string variable, defined prior to the list of expressions.

```
sprint string_variable, expression, expression, …, expression;
```

Instruction "remark"

The **remark** instructions somewhat overcome the fact, that the **Program Editor** tool cannot preserve comments when switching between Graph and Text modes. The Graph mode allows for any code modification, adding, moving, any deleting instructions etc. Any possible comments, typically bound to some specific location in the source code, would appear in different and possibly meaningless part of the program.

So, the **remark** instruction is capable to hold any text up to the **end_remark** keyword. At the same time, it is handled as any other instruction, which means it is displayed also in the Graph mode.

```
remark
  Any text…
  …
end_remark;
```

Importing programs

Every global **constant**, global **variable**, **procedure**, and **function** may be marked **public**. In such case, these objects become visible for other programs and can be used in their code. This feature allows to reuse once written and debugged code, which is a very powerful capability (and virtually essential feature) in program development.

If any program wants to use any public objects from another program, then it must import it first. The optional **import** block must be defined prior to **constants** or **variables** sections, as well as any **procedure** or **function**.

```
program program_name;

version 1;

import
  program_name,
  program_name,
  …,
  program_name;

…
```

Public objects from imported programs must by referred using a **fully qualified identifiers**, consisting of the imported program identifier and the object identifier, delimitated by a dot. Usage of the fully qualified identifiers eliminates any name conflicts, which would occur if two imported programs contain objects using the same identifier. The following example demonstrates usage of fully qualified identifiers:

```
program prog1;
…
procedure Calculate;
…
end_program.
```

```
program prog2;
…
procedure Calculate;
…
end_program.
```

```
program calc;
…
import
  prog1,
  prog2;
…
procedure main;
instructions
  …
  call prog1.Calculate;
  call prog2.Calculate;
  …
end_procedure;
…
end_program.
```

Because imported programs are identified only by their name in the **import** section, the program file name (without the ".spl" extension) and program name, stated after the keyword **program** in the program source code, must be equal.

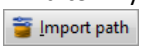
Remark:

Because SPL code is case-sensitive and the Windows file system is case-insensitive, the case of the file name may differ from the name in the program source code. However, when the program is imported, its name in the **import** section (later used in fully qualified identifiers) must be cased as in the program name in the source code.

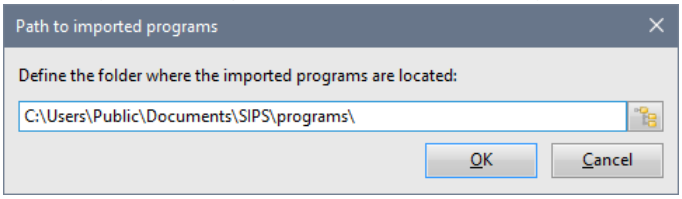
Programs must not be imported cyclically. Before the actual program is compiled, the compiler tries to compile all imported programs first. It is not unusual that the imported programs import other programs and so on. In such case the compilation is again paused and the compiler starts compilation of programs imported into imported programs etc. If the compiler, descending through the import tree, reaches a program, which compilation is in progress and only is paused waiting for compilation of imported programs (this means, program import relations create a loop), the whole process is terminated with error.

Imported programs folders

Because the imported programs are defined only by their name in the source code, the SPL compiler must know here to look for files, containing the programs. The compiler currently checks two locations.

- First, the compiler checks if the imported program file exists in the same folder like the program containing the import.
- Second, the compiler tries to find the file in the folder, defined as an import folder in the **Program Editor**. By default, the common shader folder "\Users\Public\Documents\SIPS\programs" is used, but the 

button opens a dialog, which allows do define any other folder.



SIPS native programs

The SPL runtime implements number of **native programs**. These programs are not implemented in SPL (there is no SPL source code for them), they use native code similarly to other SIPS components. But every SPL program can import any native program and use procedures and function offered.

The main purpose of native programs is to provide interface to SIPS functions, like handling of images (loading and saving FITS files, etc.), lists (adding images to list, enumerating images in list, etc.), astrometry and other areas.

The set of native programs is expanding with new SIPS versions released, but the existing programs and their public procedures and functions are kept fixed to maintain backward compatibility with existing SPL code.

Native programs available for SPL code are:

- **system** program allows file handling like coping files and enumerating files in directories etc. Also, the **system** native program allows the code to shutdown the host PC;
- **jd** program offers procedures and functions for Julian Date and normal local date as well as geocentric JD and heliocentric JD conversions and offers functions returning actual Julian Date.
- **text** program allows the SPL program to parse or create common text files of any format.
- **ini** program helps handling of text files, following .INI file conventions.
- **json** program implements procedures for parsing as well as creating of structured text files corresponding to JSON format.
- **sips** program contains functionality implemented by various parts of the SIPS itself, for instance, definition of star search parameters, plate solving of images (astrometry), handling of alarm sounds and also parsing and formatting of common Right Ascension, Declination, Latitude and Longitude coordinate formats.
- **image** program handles SIPS images (FITS files).
- **list** program handles image lists.
- **serial** program provides interface for SPL code needing to communicate over serial lines with devices, not handled by device drivers in SIPS.

Hint:

When importing programs to SPL code in the **Graph** mode, the combo-box offering available programs always contains all available native programs.

When calling procedure, the combo box offering available procedures lists all procedure from the selected imported program, of course including native programs.

When using the **Expression Editor**, all functions offered by all imported programs (including native programs) are listed.

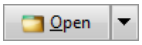
Example programs

Every SIPS installation also creates a directory "C:\Users\Public\Documents\SIPS\programs" and copies some example programs to it. Example programs often offer useful functionality (for instance, Equatorial to Azimuthal coordinate conversions, algorithms to calculate Sun position or air mass in the certain altitude etc.).

Program Editor tool

The **Program Editor** tool window uses a tabbed interface, which allows the user to load multiple programs at once. The command pane, to the right of the tabbed panes, contains commands controlling tool configuration and loading, saving, and running of individual programs.

- 

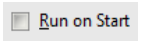
Create a tab with a new empty program skeleton, containing just one procedure with generic name. User should rename the procedure as well as the program itself.
- 

Open program file to a new tab. The drop-down button on the right side opens a list of the least recently used programs.
- 

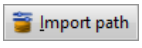
Save program in the actual tab.
- 

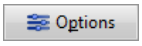
Save program in the actual tab using a new file name. As the file name and program name should be identical (if the program public objects are to be used by another program, same file and program names are mandatory), it is usually necessary to rename also the program itself.
- 

Save all modified programs open in individual tabs.
- 

Run program in the actual tab.
- 

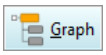
Define program, which will be loaded and run upon SIPS program start.
- 

Stop running program.
- 

Define path, on which the imported programs are to be searched.
- 

Define **Program Editor** tool options.

Each program in a tab can be edited using one of two modes:

- 

Graph mode
- 

Text mode

The **Program Editor** can switch between modes by simply clicking the respective button. While it is always possible to switch to the text mode, switching back to the graph mode involves compilation of the program and thus it requires the program to be syntactically correct. Any syntax error causes displaying of an error message. The tool stays in the text mode and sets the cursor to the position of the error. It is upon the user to fix all syntax errors to enable the switch to graph mode.

Hint:

Switching between modes can be done from keyboard using respective hotkeys:

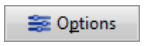
- <Ctrl>+<G> switches the editor to the **Graph** mode
- <Ctrl>+<T> switches the editor to the **Text** mode.

Code generation from Graph mode

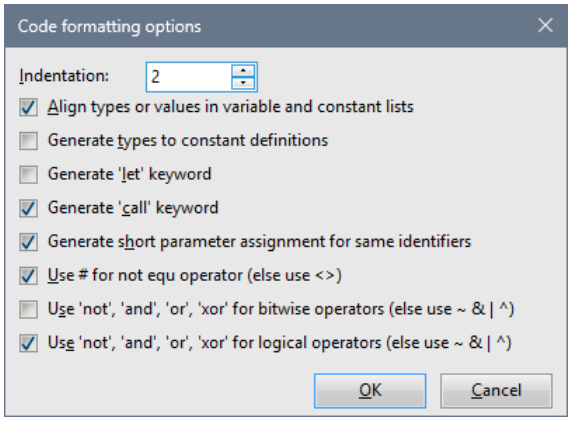
The program source text is not maintained when the **Program Editor** is in the **Graph** mode. Instead, it is deleted once the program is successfully compiled and the editor finishes switching to the **Graph** mode. Then it is generated from the program graphics representation when switching back from **Graph** to **Text** mode.

> Hint:
>
> So, switching from text mode to graph mode and back also works as code refactoring—the source text is generated with proper indentation, spacing etc.

The same is valid for saving the program from the **Graph** mode. As programs are always saved in the source text form, saving program requires the text source is generated from the program graphics (tree) representation and then saved.

The 

button in the command pane opens the **Code formatting options** dialog box, which allows the user to modify the source code formatting preferences.

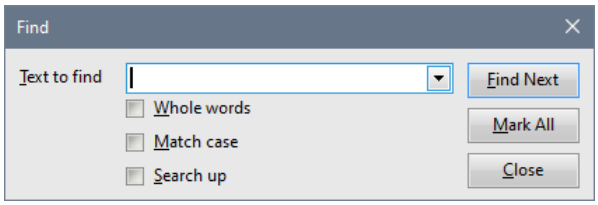


Text mode

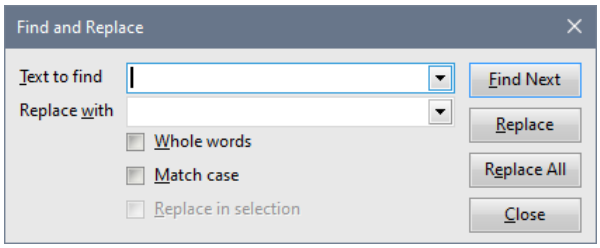
The program can be created and modified using standard text editor in the text mode. The used editor highlights the SPL keywords and literals using different colors to make orientation in the source code easier. The tool-bar contains standard text editing functions.

-  Undo last editing change, hotkey <Ctrl>+<Z>.
-  Redo the previously undo-ed change, hotkey <Ctrl>+<Y>.
-  Cut (copy to clipboard and delete) the selected block of text, hotkey <Ctrl>+<X>.
-  Copy the selected block of text to clipboard, hotkey <Ctrl>+<C>.
-  Paste text from clipboard, hotkey <Ctrl>+<V>.
-  Delete the selected block of text to clipboard, hotkey <Del> or <Backspace>.
-  Indent the selected block of text.
-  Un- indent the selected block of text.
-  Open **Find** text dialog box, hotkey <Ctrl>+<F>.
-  Open **Find and Replace** text dialog box, hotkey <Ctrl>+<H>.

Note the **Find** window, opened using the  tool or the <Ctrl>+<F> hotkey,



or **Find and Replace** window opened using the  tool or the <Ctrl>+<H> hotkey,



are not modal. This means these windows may stay opened while editing text. While these windows remain on top of the SIPS main window, the do not stay above the **Program Editor** tool window as well as above other SIPS tool windows. However, closing them and opening again, either using the tool-bar button or hot-key, and opening them again causes they again appear above the **Program Editor** tool window.

Graph mode

The **Graph** mode of the **Program Editor** is intended to help especially the beginning programmers with creation of SPL programs. It offers an easy way to choose instruction, to properly define instruction parameters etc.

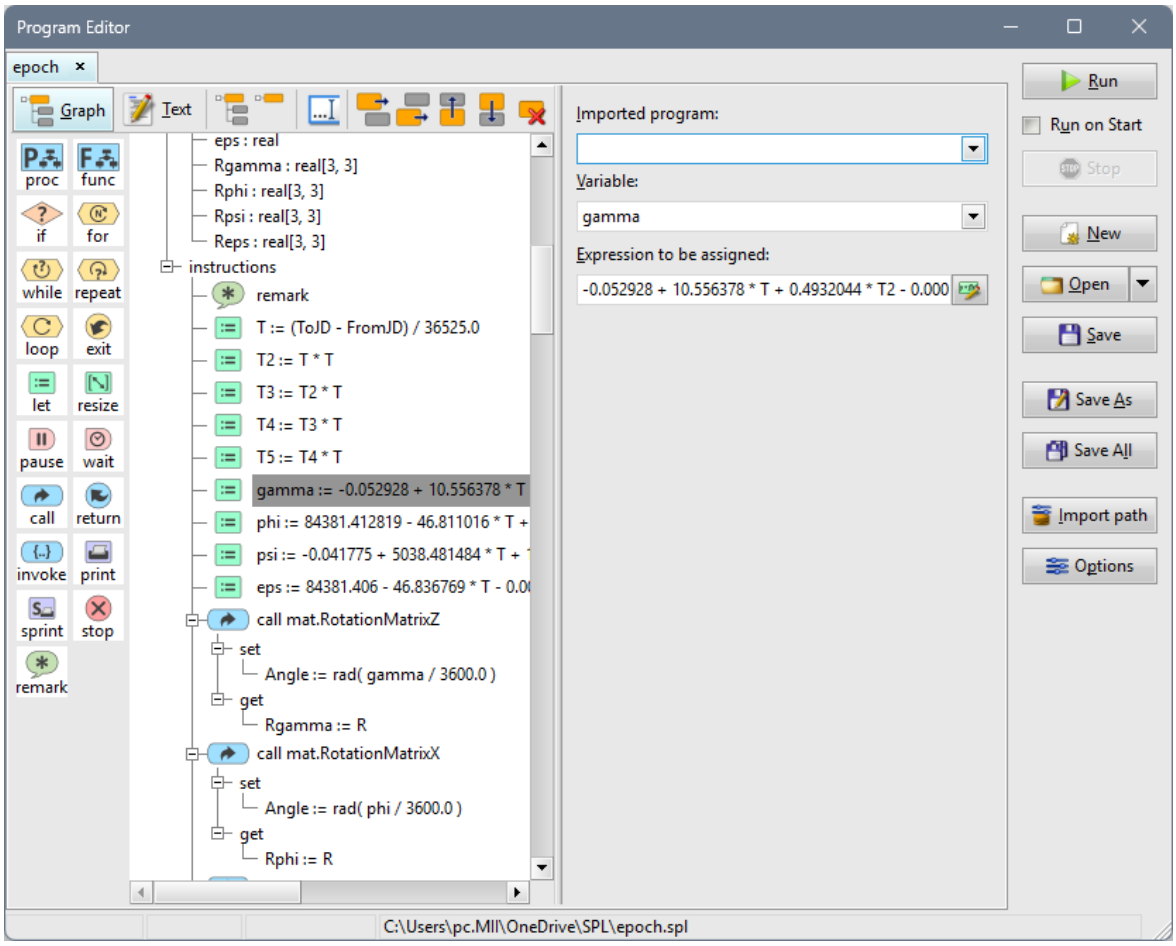
However, as there are many editing tools available in the **Graph** mode, like **Expression editor** of instruction editing panes, even experienced users may find the **Graph** mode useful when searching for some embedded function, imported program variable or procedure, for the **invoke** instruction parameter structures etc.

The **Graph** mode relies on the fact, that every SPL program is a tree-like structure (and not only SPL program, this is true for all modern programming languages):

- The single root of the tree is the program itself.
- Then the tree extends to nodes—block of global constants and variables, procedures, and functions.
- Global constants and variables nodes contain directly leaves—definition of constants and declaration of variables.
- Procedure and function nodes may also contain local constant and variable nodes as well is directly leaves of executive instructions (e.g., assignments, invokes, …) or nodes of control instructions (conditional instructions, loop instructions, …).
- Control instruction nodes then again contain nested nodes of other control instructions (r.g., **for** loop inside the **then** branch of the **if** statement) or leaves (assignments, …).

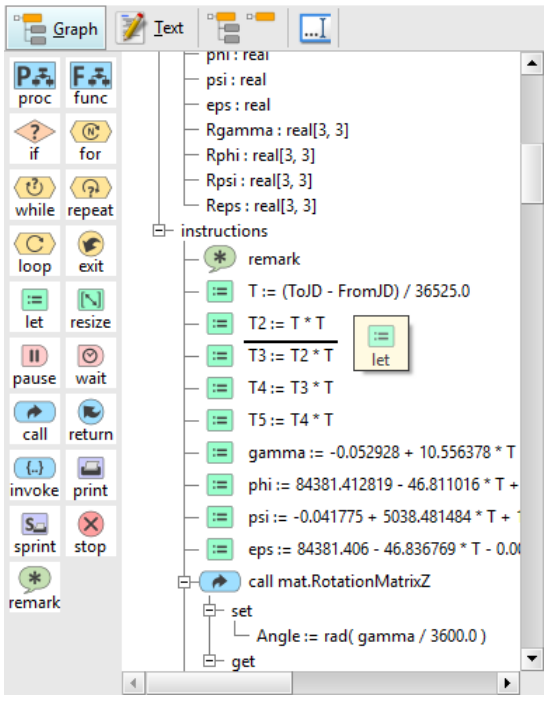
The **Program Editor** tool is in the **Graph** mode divided into three panes (with the command pane on the very right, but the command pane is the same in both **Graph** and **Text** modes):

- **Instruction palette** pane on the left.
- **Program tree** pane in the middle.
- **Instruction properties** pane on the right.



Instruction palette pane

The **Instruction palette** pane lists all available instructions. It acts as a source for the mouse drag-and-drop operations, which insert new instructions into the **Program tree** pane. Just press the left mouse button above some instruction in the palette and while holding the mouse button pressed, drag the mouse above the tree. The small semi-transparent pop-up box with the instruction icon and keyword appears under the mouse cursor. The tree highlights the position, on which the instruction will be inserted if dropped if the mouse button is released, using the black horizontal line.



Hint:

The editor does not allow to drop an instruction where it does not belong. For instance, instruction cannot be dropped into constants. Also, the **exit** instruction cannot be dropped outside the **loop** etc.

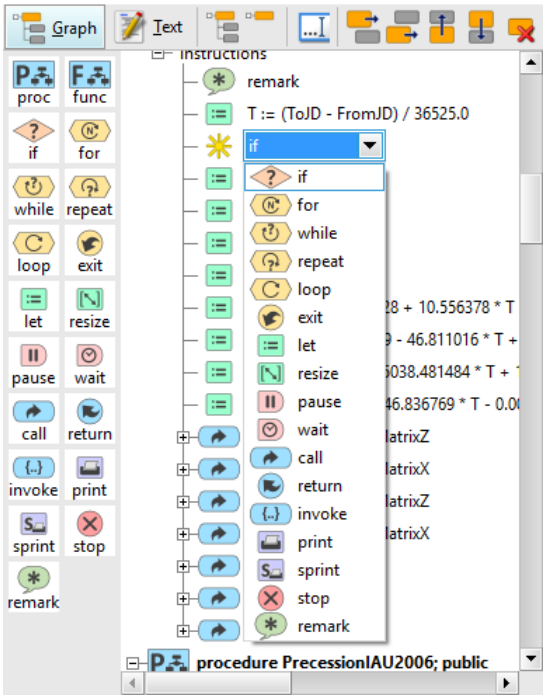
Program tree pane

The **Program tree** pane shows the program in the graphical representation of a tree structure (this gives the **Graph** mode its name). Each tree node is preceded with a small box with [+] or [-] signs. Clicking on the box expands or collapses the particular tree node.

Expanding and collapsing of the selected tree node can be performed also using tool-bat commands  (expand) and  (collapse). However, while the [+] sign expands only single tree level, the corresponding tool-bar button expands whole sub-tree up to leaves.

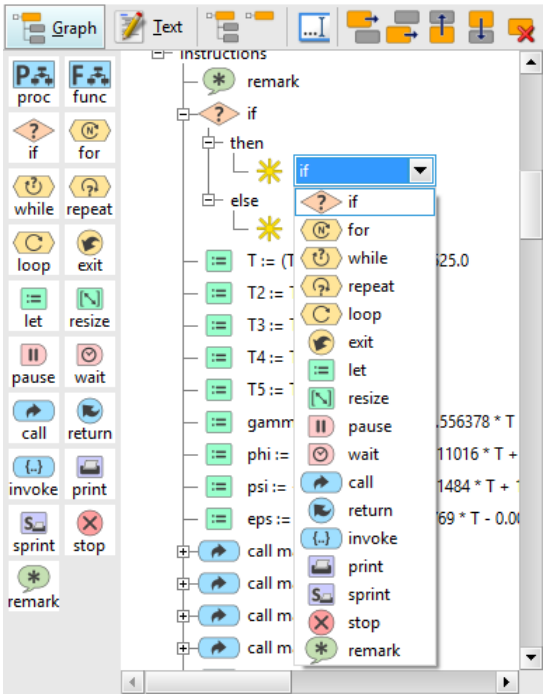
New instruction can be inserted several ways:

- Instruction can be dragged-and-dropped using the mouse from the **Instruction palette** (see the previous sub-chapter).
- Tool-bar buttons  and  insert new instruction before or after the currently selected instruction.



A combo-box box with all available instruction appears on the place of new instruction, allowing the user to select one. If no instruction is chosen and the combo-box is just deselected, it only disappears without inserting any instruction.

- If a **procedure** or **function** is selected in the tree, the buttons inserting general instructions are removed from the tool-bar, and special buttons directly inserting either **procedure** ( and ) or **function** ( and ) are added.
- Every empty tree node (typically a control instruction like for instance **if** or **for**, but without nested instructions), the editor places a placeholder named **<add instruction>** in the position of nested instruction sequence. When the user selects this placeholder, a combo-box box with all available instruction appears.



Once at last one instruction is inserted into an instruction block, the **<add instruction>** placeholder is no longer displayed, other instructions must be added by some of above-mentioned ways.

Also, an instruction order can be changed by several ways:

- Instructions may be dragged-and-dropped over the entire procedure or function. In such case the three behaves similarly like if a new instruction is dragged from the palette. The limitation to one procedure or function is applied to keep the instruction scope the same. By other words, every instruction can contain local constant and variable references, not available in other procedures or functions, which is why the drag-and-drop crossing the scope boundary is not allowed.
- Instructions, as well as procedures and functions, may be moved up and down the current block using the  (move up) and  (mode down) tool-bar buttons.

A selected instruction, procedure or function may be deleted using the  button.

Elsif branches handling in the Graph mode

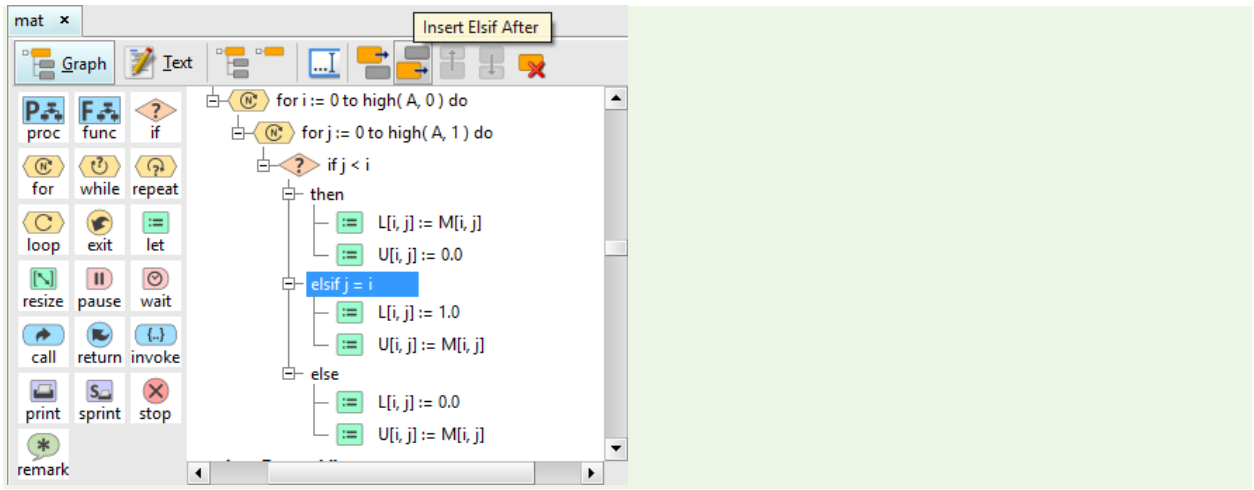
The **if** instruction is special compared to other control instruction in its ability to contain virtually unlimited branches (sequences of instruction, executed upon fulfilling of the defined condition expression). While for instance the loop control instructions (**for**, **loop**, **while**, and **repeat**) contain exactly one instruction sequence, the **if** instruction contains **then** sequence, any number of optional **elsif** sequence, and optional **else** sequence.

While both **then** and **else** branches are always present in the **Graph** mode, the **elsif** branches are present only if defined. The **Program Editor** handles the list of the **elsif** branches though the toolbar, displayed when the **then**, **else**, or **elsif** branch is selected.

Hint:

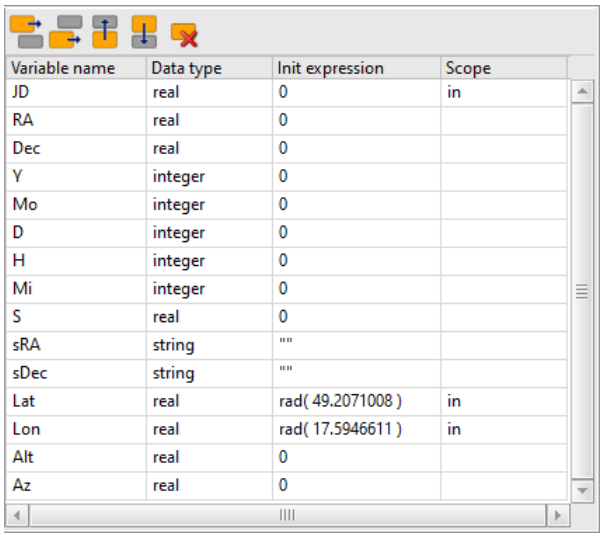
The **if** instruction semantics define the **elsif** conditions are evaluated sequentially in the order in which they appear in the program source code, and the first branch, which condition evaluates **true**, is executed. So, the order if **elsif** branches is important.

Individual branches can be moved forward and backward using respective toolbar buttons, but the **Program Editor** in the **Graph** also allows to drag-and-drop each **elsif** branch to different position. However, dragging is always limited to the current **if** instruction only.



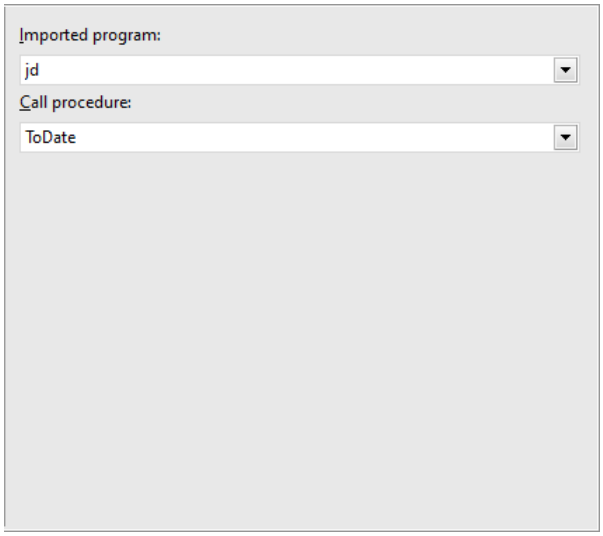
Instruction properties pane

The content of the **Instruction properties** pane depends on the node, currently selected in the program tree. If, for instance, a **variables** or **constants** block is selected, the properties pane displays a table with defined variables or constants. It allows modification of a list (adding, moving, and removing), definition of types, initialization expressions, attributes etc.

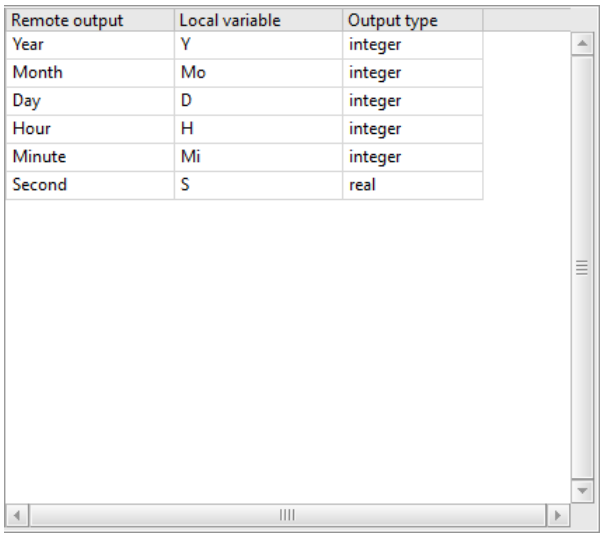


Variable name	Data type	Init expression	Scope
JD	real	0	in
RA	real	0	
Dec	real	0	
Y	integer	0	
Mo	integer	0	
D	integer	0	
H	integer	0	
Mi	integer	0	
S	real	0	
sRA	string	""	
sDec	string	""	
Lat	real	rad(49.2071008)	in
Lon	real	rad(17.5946611)	in
Alt	real	0	
Az	real	0	

A selected **call** instruction shows combo-boxes allowing to choose which **procedure** is to be called,

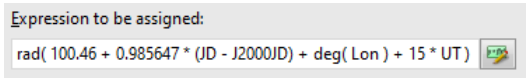


while the **call set** or **call get** tree nodes selected cause displaying of tables with the chosen procedure input and output parameters etc.

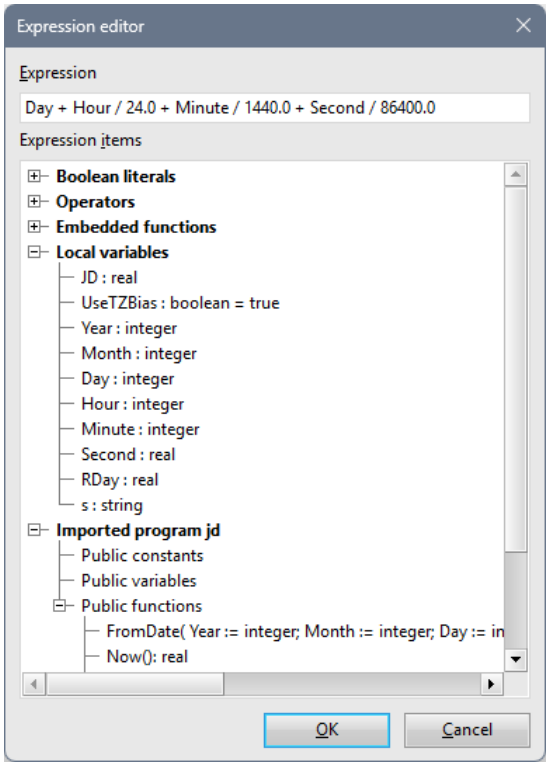


Remote output	Local variable	Output type
Year	Y	integer
Month	Mo	integer
Day	D	integer
Hour	H	integer
Minute	Mi	integer
Second	S	real

The GUI control elements, displayed in the **Properties panes**, are designed to be as helping to the user as possible. This means if a variable of some type is expected, the combo-box with all available variables of the desired type is used, so the user can just choose. If an expression is to be entered, the edit box contains a button opening **Expression Editor**.



The **Expression Editor** dialog box shows all possible expression items available, including all operators, constants, variables, embedded or imported functions etc. So, it is not necessary to browse this manual to get the proper name desired object; double-clicking on the object in the tree inserts its name into expression.



SPL versions

The **version** keyword, automatically generated by the **Program Editor**, indicates the actual SPL compiler version used to create each program. If the version of loaded program is greater than the version supported by the used SPL compiler, the loaded program may contain constructs, not supported by actually used compiler. It is recommended to update the SIPS to the latest version prior to running or editing of such program.

Version history

SPL version 1 introduced in SIPS v4.0

- Initial SPL release.

SPL version 2 introduced in SIPS v4.1.1

- Added REST API “v1/flaps” and “v1/switches” endpoints.
- Added native program “serial”, intended to control COM ports.

SPL version 3 introduced in SIPS v4.1.9

- Added embedded function **assigned()**.

SPL version 4 introduced in SIPS v4.1.10

- Added interfaces for the second and third switches and weather station devices.

SPL version 5 introduced in SIPS v4.2

- Added embedded function **norm()**.
- The **pause** and **wait** instructions are newly allowed also within functions.
- Added new **print** instruction keyword **cll** (clear line).
- The **invoke** instruction no longer causes Runtime Error when the REST Endpoint returns Result = false.

SPL version 6 introduced in SIPS v4.2.1

- Added the possibility to define **remark .. end_remark** section directly following the procedure or function header, as well as the entire program. The syntax is identical to the **remark** instruction within the program code, but technically remarks in program, procedure and function headings are not instructions.
- Added embedded functions **dms()** and **hms()**.

SPL version 7 introduced in SIPS v4.2.4

- The **if** instruction was extended with a possibility to define **elsif** branches.

SPL version 8 introduced in SIPS v4.2.5

- The **set** and **get** blocks of the **call** instruction, as well as the set block of **functions** invoked within expressions, now supports shortened parameter passing syntax.
- The REST API “v1/telescope” extended with **find_object** command
- The REST API “v1/imager” extended with autofocus-related commands:
  - **focus_state**
  - **autofocus_start**
  - **autofocus_active_frame**
  - **autofocus_stop**
- The SPL native program **jd** added functions:
  - **ToDateString**
  - **DateToDateString**
- The SPL native program **sips** was extended with:
  - constants for undefined values (undefined RA, undefined magnitude, …)
  - **GetObservatoryParameters** procedure
- The SPL native program **image** added procedures:
  - **FindStars**
  - **EnumerateStars**

SPL version 9 introduced in SIPS v4.2.7

- The **image** native program procedures **Rotate90**, **Rotate180**, **Rotate270**, **Rotate**, **FlipHorizontal**, **FlipVertical**, **Bin**, **Resample**, and **Crop** were extended with **CreateNew** boolean input parameter and **NewImageName** string

output parameter, allowing to optionally create new image instead of performing of the image transformation on the existing image.

SPL version 10 introduced in SIPS v4.2.8

- New procedures were added to the SPL native program **image**:
  - **Sum** creates sum of two images.
  - **Difference** creates difference between two images.
  - **Add** increases all pixels of an image by a defined integer value.
  - **Subtract** decreases all pixels of an image by a defined integer value.
  - **Multiply** multiplies all pixels of an image by a defined real value.
  - **Divide** divides all pixels of an image by a defined real value.

SPL version 11 introduced in SIPS v4.3.2

- Constant definition allows explicit definition of type, similarly to initialized variables. Constant value then must correspond to defined type.

SPL version 12 introduced in SIPS v4.4.0

- Added support for arrays:
  - Definition of array constants
  - Declaration of array variables, including optional individual items initialization
  - Assignment of whole arrays using **let** instructions
  - Passing whole arrays to/from procedures
  - Passing whole arrays to/from **invoke** instruction (JSON arrays of simple type are mapped to SPL arrays)
  - New embedded function **high** returns highest index of selected array dimension
  - New instruction **resize** for changing of dimension sizes during runtime

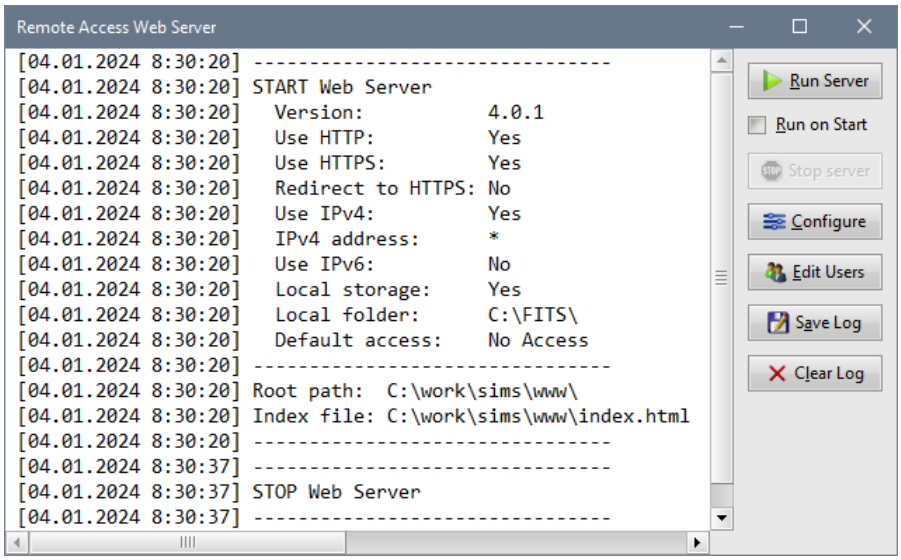
SPL version 13 introduced in SIPS v4.4.2

- New embedded functions for bit shifting **shl** and **shr** were added.
- New embedded functions returning the greater or less number **max** and **min** were added.
- It is possible to use multiple **cll** keywords in sequence in the **print** command to delete more lines.

Remote Access and Alpaca Web Server

The **Remote Access and Alpaca Web Server** can operate in two modes:

1. In the **Remote Access** mode, the server publishes a SIPS REST API (sometimes denoted as RESTful API)[13], which can be used to control SIPS from various clients. Such clients can run on the same PC or on any other device connected to the PC running SIPS over local TCP/IP network or over Internet from any place on Earth.
2. In the **Alpaca** mode, SIPS also runs the HTTP(S) server, but the REST API published follows the ASCOM Alpaca[14] standard intended to remotely access devices (cameras, filter wheels, telescope mounts, etc.), which means all devices connected to SIPS can be accessed by remote clients through standard protocol over a network.



SIPS installation contains a Remote Access client application, called **Web SIPS**, implemented in JavaScript, intended to run in any web browser, regardless if it runs on a full-fledged desktop PC, laptop, tablet, or on a mobile phone. The **Web SIPS** connects to the SIPS server using the REST API and allows remote control of SIPS observing sessions. In fact, this client is another version of SIPS, implemented in JavaScript language and thus able to run in web browsers. Still, there are some significant differences between standard **Desktop SIPS** and **Web SIPS**:

- The Web SIPS does not contain any device drivers, it relies on the Desktop SIPS to handle all devices. The Web SIPS needs to communicate with a Desktop SIPS through its web access server.
- The Web SIPS functionality is limited to observing session control. Any subsequent image processing, from simple calibration up to astrometry and photometry processing, must be performed using Desktop SIPS.
- The Web SIPS GUI is streamlined and simplified. It typically shows one task a time and does not allow opening different tools in different windows and distributing them on the desktop etc. However, it is of course possible to open multiple web browser tabs or web browser windows to use more instances of Web SIPS at once.

Remark:

Every internet server must be accessible to clients. This means each server must be identified within the TCP/IP network by its unique IP address. In vast majority of cases clients do not use the IP address directly, but ask the DNS (Domain Name Service) to translate the server's name into IP address (for instance, the www.gxccd.com server name currently translates into unique IP address 217.115.241.22), and only then the resolved IP address is used to communicate with the server.

Keep this on mind when planning the Web SIPS usage—SIPS must be run on a computer, accessible by the clients, and either with fixed IP address assigned or with a valid name, resolvable to IP address by DNS. Unfortunately, this is often not the case on many home or office networks, hidden beyond network routers running DHCP (Dynamic

[13] See the https://restfulapi.net/ site for details.
[14] See the https://www.ascom-standards.org/ site for details.

Host Configuration Protocol) services and performing NAT (Network Address Translation) when accessing the outer world etc.

Also, it is a big difference if the SIPS Web Server is to be accessible to users on a local network only or from any place over the world using the Internet. Virtually any router allows some form of accessing of the computer on the local network from outside world, like for instance port forwarding etc. But it is beyond scope of this guide to deal with TCP/IP network devices configuration.

mDNS in SIPS

The **Multicast Domain Name System** (mDNS) allows computers (or mobile phones and other devices), connected to the local network, to resolve a host name to IP address without any manual DNS system configuration. So, it is possible to access such devices simply by typing "device.local" (where "device" should be replaced with actual device name) into address line and the mDNS system handles translation of this name into corresponding IP address.

Remark:

It is quite common that devices in the local network are not configured with fixed IP address but instead use the DHCP (Dynamic Host Configuration Protocol). A device using DHCP just asks the DHCP server (typically a local network router/gateway) to assign it with an IP address. Also, other network properties, necessary for proper network functioning (network mask, gateway address, DNS server address, …), are assigned by DHCP server, too. So, it is common that the actual IP addresses of individual devices may change over time, which makes using them directly even more difficult.

In the case of SIPS Web Server, the mDNS name is predefined as "sips.local", but it can be set to any other name (see the following chapter). If, for instance, multiple SIPS programs are running on multiple computers to control different observing setups, it is desirable to define the mDNS name individually on each computer to allow users to select particular SIPS instance/computer.

Hint:

It is also possible to run multiple SIPS instances on a single computer and access them individually.

The first way is to manually assign multiple IP addresses to one computer and then to configure each instance with unique IP address. In such case DHCP cannot be used.

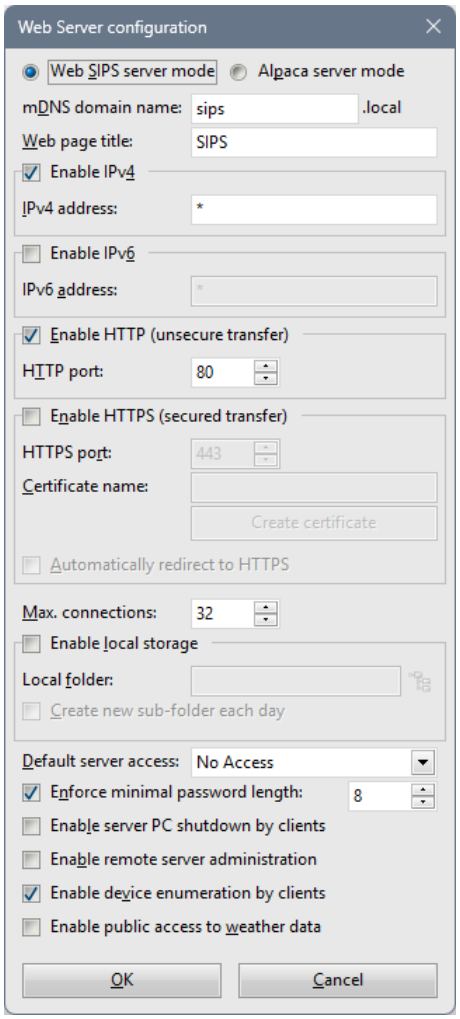
Or it is possible to keep single IP address for a computer and distinguish individual instances by different IP port numbers. However, if other IP port than default one is used, such servers must be accessed with a domain name followed by IP port number.

It is worth noting that such setup should run individual SIPS instances with unique configuration files, so the Web Server configuration is preserved. See the **Configuration handling command line parameters** for details.

Web Server configuration

The SIPS Web Server is designed to be easy to use even by users without WWW server and HTTP protocol background. Still, the server can be configured to fit various network configurations and users' needs. Server configuration is opened using the 

button.



The **Web Server configuration** dialog box offers to define:

- **Web SIPS server mode** and **Alpaca server mode** distinguish if the server handles SIPS REST API or Alpaca API.
- **mDNS** name of the server.
- **Web page title** defines a name, which will be displayed by the WWW browser as page title. The ability to define different titles for different servers could be useful when controlling multiple SIPS instances remotely using multiple WWW browser windows.
- **Enable IPv4** option and **IPv4 address**. Use the * to tun the server on all IPv4 addresses, assigned to the computer.
- **Enable IPv6** option and **IPv6 address**. Use the * to tun the server on all IPv6 addresses, assigned to the computer.
- **Enable HTTP** and define **IP port** on which the web server listens to client HTTP requests.

  Warning:

  HTTP protocol transfers everything using plain (open) text, including usernames and passwords. This may be no issue if the traffic goes directly between client and server only. But especially in public networks (on airports, hotels etc.) this may not be the case—it is not that difficult for malign users to listen to the network traffic and capture login credentials.

  This is one of the main purposes of using of HTTPS protocol—to encrypt transfer between client and user, so even if anybody listens to it, the content remains unreadable. So, if the SIPS Web Server is accessible from public Internet, usage of HTTPS only is strongly recommended.

- **Enable HTTPS** (Secure HTTP) and define **IP port** on which the web server listens to client HTTPS requests.
- **Max. connections** allow us to make sure server is not overwhelmed with too many users at once.

- **Enable local storage** and **Local folder** allows the users to store captured images on the computer, on which the SIPS server runs. If local storage is not allowed, the user must capture images one by one and download each image to local computer prior to starting next exposure. Such mode of operation is suitable for some demonstration or simple educational applications, but any serious usage of Web SIPS requires local storage enabled. But keep in mind modern cameras with hundreds of megapixels may fill the drive space quite fast.
    - **Create new sub-folder each day** option causes the web server creates a folder named with the current date YYYY-MM-DD in the **Local folder** for each observing day. So, newly acquired images will not overwrite data from previous nights.

      Remark:

      The new folder is created upon SIPS startup. But if the web server runs continuously for multiple days (and nights), new sub-folder for another day observations is not created automatically, because it is not possible to clearly define at which time the new folder should be created – typically one night data should be stored in one folder, but the date changes at midnight. So, the Web SIPS application contains a command button allowing manual creation of new folder.

- The **Default server access** option defines access rights for non-logged users. Denying access for such users is probably the best practice for majority of installations. Allowing non-logged users to view the site or even to control the observation could be allowed only on isolated intranets, where the user access is limited by other means.
- The **Enforce minimal password length** checkbox allows to prohibit the users from using passwords shorter than the number of characters defined in the count box, as too short passwords are security risks.
- The **Enable server PC shutdown by clients** option decides if the User's Menu in the Web SIPS application contains the command for remote shutdown of the host PC running SIPS.
- The **Enable remote server administration** option allows users to change their passwords using the command in the User's Menu in the Web SIPS application. Also, users with Administration privileges, can manage users (add, remove, and edit individual user entries, reset their passwords, …) remotely from the web browser.

  Warning:

  Server administration should be allowed on HTTPS connection only, which ensures data payload is encrypted. Open text HTTP transfer may exhibit login names and corresponding passwords to any malign user, capable to physically monitor network traffic.

- The **Enable public access to weather data** option allows the server to provide the data from **Weather station** drivers without any access restrictions. So, if this option is checked, the weather data are visible even for users without login credentials.

HTTPS and web certificates

HTTPS requires a security certificate to operate properly. The certificate performs two functions:

1. Encrypts communication.
2. Ensures the opposite site of communication is authentic, which means the traffic was not redirected to some imposter malign site.

While every certificate is capable to encrypt communication, only certificates derived from publicly recognized root certificates can be used to authenticate the remote server. Such certificates are either bought (and regularly renewed) from a certificate authority or it is possible to get a free certificate, but at the cost of installing a software package performing regular communication and renewing of the certificate with a certificate authority.

Still, even if we give up on the server authentication, encrypted communication using HTTPS remains very important. This is why SIPS allows to create self-signed certificate using the **Crete certificate** button.

Hint:

All modern WWW browsers recognize that the self-signed certificate, used for HTTPS transfer, is not derived from recognized authority, and displays more-or-less prominent warning (sometimes the warning is so prominent, that it is difficult for the user even to find a way to accept risks and proceed to the server). But it is usually possible to

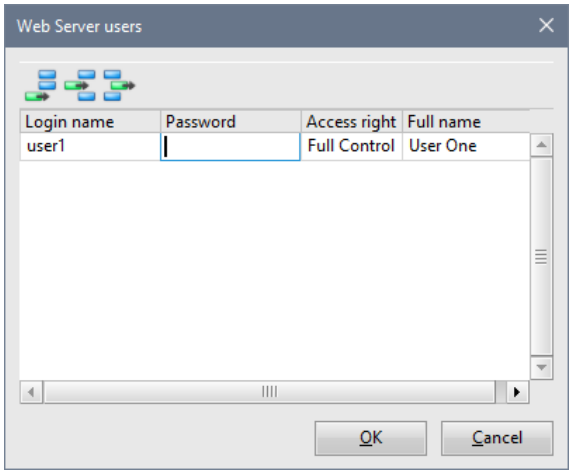
confirm such security exception (we know the server we access is a SIPS Web Server using self-signed certificate) and utilize secure HTTPS transfers such certificate allows.

User management and access rights

As the SIPS Web Server may be exposed to the hostile world of Internet, full of attacks and attempts to compromise every publicly accessible server, the users' access is controlled by strict rules.

Users are managed using a dialog box, opened by the 

button.



The server defines 4 levels of access rights:

1. **Administrator**—complete control of the server, administrators can revoke control from non-administrator users.
2. **Full Control**—users can acquire SIPS observing control.
3. **View Only**—users can monitor (view) the observing session and download observed data but cannot acquire observation control.
4. **No Access**—cannot login (used for default server access).

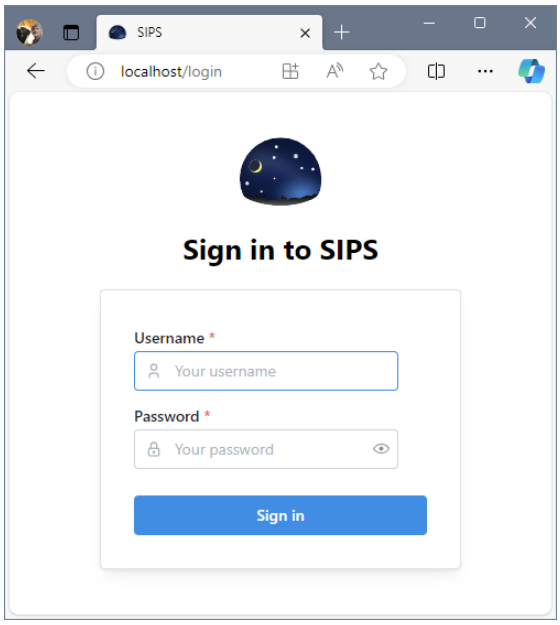
Remark:

User **Login name** and **Password** are required for every user. If either of these field (or both) are empty for any user, a notification dialog box appears, requiring either filling respective fields or removing the affected user from the list.

The **Web Server configuration** dialog box allows the administrator to enforce minimum password length. In such case, the password must not be shorter than the defined number of characters.

Also, Login names must be unique. If two or more users share the same Login name, similar notification dialog box appears, demanding usage of unique login names.

If the server's default access is set to **No Access** (a recommended value), every user is asked to provide login credentials upon connecting to the server.



Running the web server

SIPS is accessible from the WWW browser only when its Web Server is running. The web server runs (accepts the requests over a HTTP/HTTPS connection from remote web browsers) only after the user clicks the Run Server command button. The Stop server command button stops the server and SIPS is then no longer accessible from the web.

Running the web server on SIPS start

Checking of the **Run on Start** check box in the **Web Server** command pane causes SIPS to run the server on startup. SIPS is then immediately accessible from the web browsers, without the necessity to click the **Run Server** button by a local user.

> Remark:
>
> SIPS can run either the REST API **Web Server** or a procedure or on startup, but not both. When the **Run on Start** check bock in the **Web Server** window is checked, the same check box in the **Program Editor** tool is unchecked.

Remote observation control

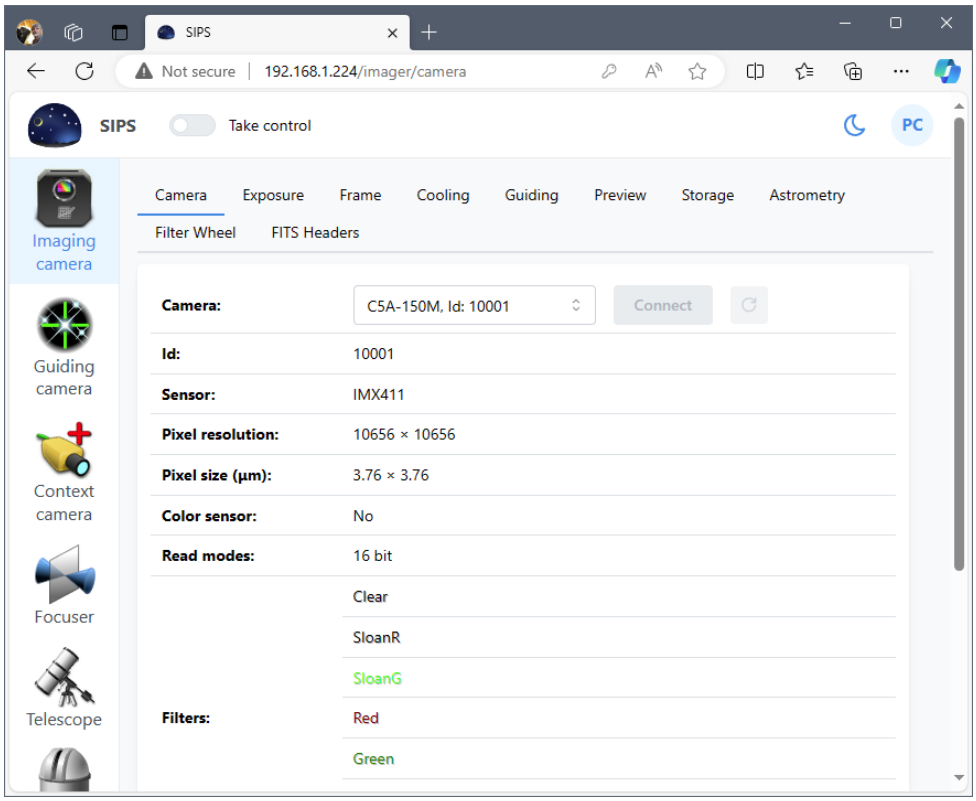
The SIPS remote access model is **one controller—many viewers**.

Users with **View Only** access rights can monitor the SIPS operations but cannot acquire control. Such user can also download acquired images.

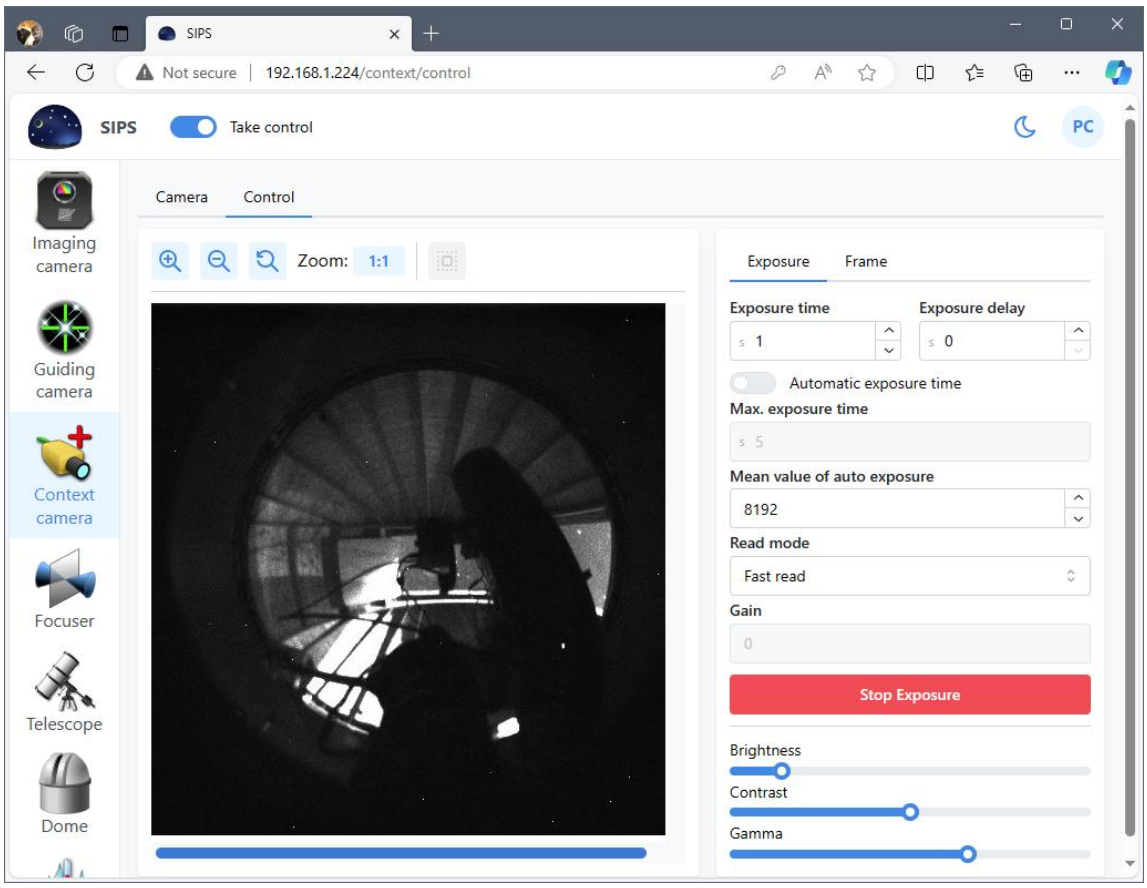
Users with at last **Full Control** access rights can acquire control of the observing session, providing control is not owned by another user. Controller can open and close roll-off roof or observatory dome slit, point the telescope, configure camera, start/stop guiding, acquire images etc.

But only one controller can be active at any time. Users with **Administrator** rights may acquire control any time, regardless of some other user already owns controller right or not.

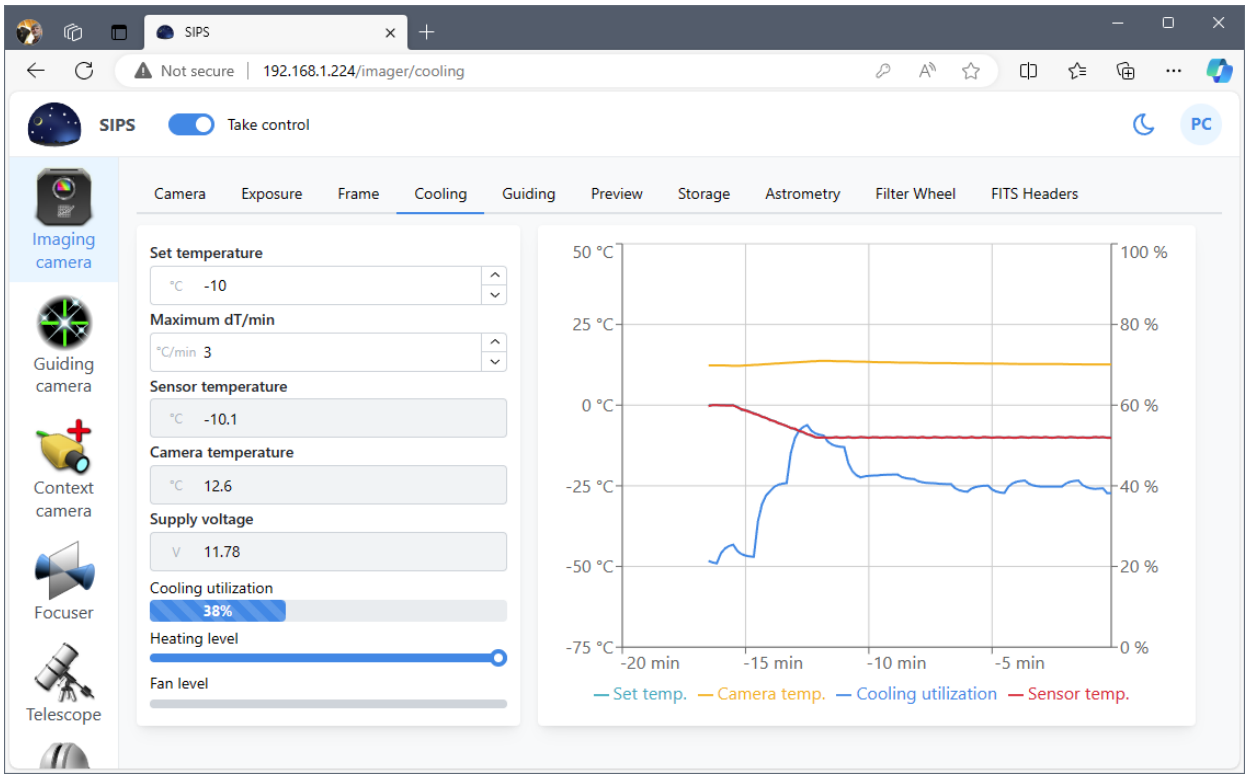
Despite a user owns **Full Control** or **Administrator** rights, no controlling action can be performed immediately after login. It is necessary to take control first.



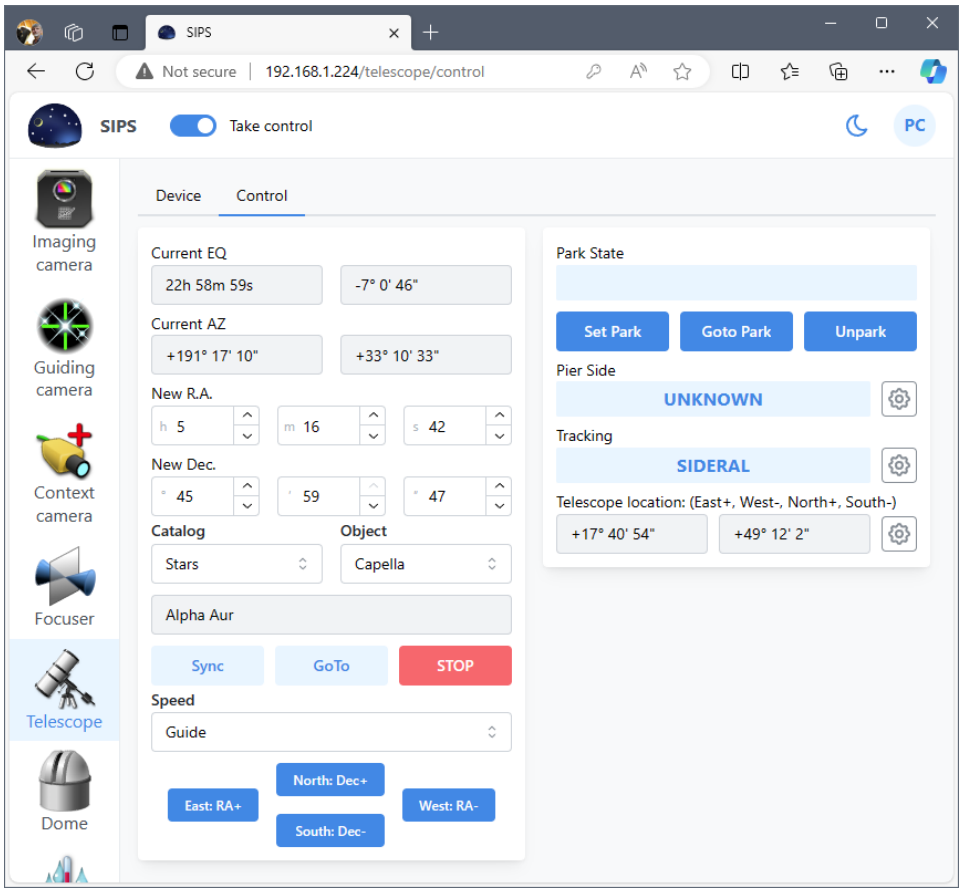
Only after the control is taken, it is possible to perform any tasks beside viewing the observatory state, like choosing cameras for individual roles, activating the image acquisition by all cameras, …



… setting sensor temperature …



… or pointing the telescope etc.

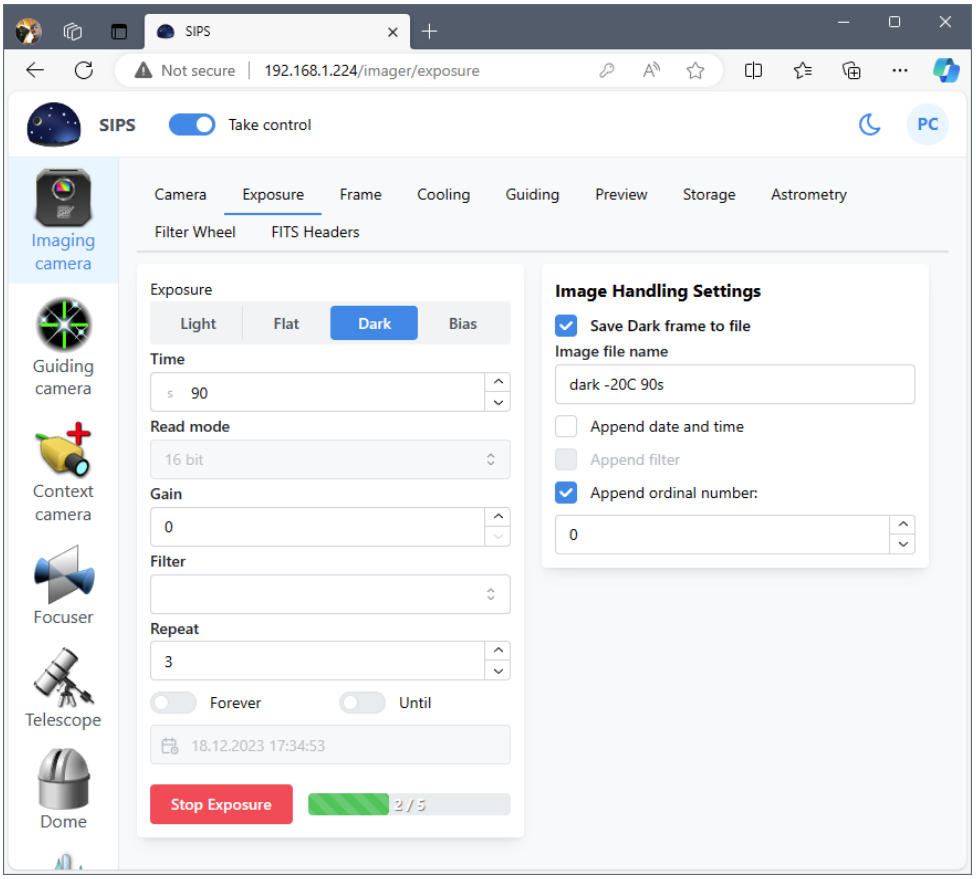


Working with local storage

Images, acquired with modern cameras with tens or even hundreds of megapixel sensors, need tens or hundreds of megabytes of drive space. Depending on imaging cadence, one night observing session can generate gigabytes or even tens of gigabytes of data. Despite the capacity of computer storage steadily increases, not all computers, dedicated to control the observatory, are equipped with latest terabyte-class drivers.

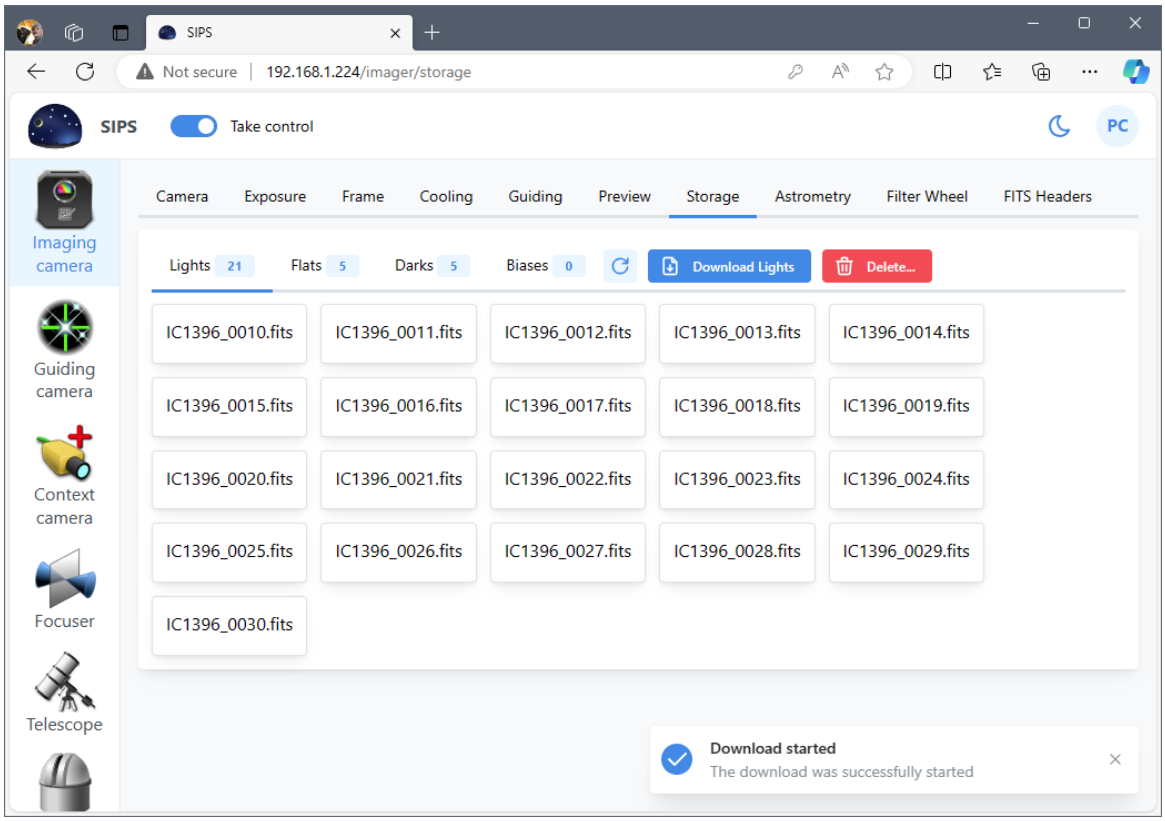
So, the SIPS **Remote Access Web Server** allows for two modes of operations, determined by the state of the **Enable local storage** checkbox in the **Web Server configuration** dialog box:

- Without local storage used by remote users. In such mode, each image captured by the main imaging camera must be downloaded by the user prior it is overwritten by the next exposure. While such operation mode ensures no user can fill the computer drive with data, the necessity to download each image prior another exposure is finished can be hardly used for long run observing sessions. Such mode is suitable for setups used for demonstration purposes, public viewing etc.
- When the usage of the local storage is allowed, it is possible to let the SIPS store acquired images locally. It is then the responsibility to the SIPS administrator and users with Full Access rights to consider the hardware capabilities of the used PC.
  The **Image Handling Settings** allows to define if the acquired images will be saved to local storage and how their names are created. The available options correspond to the same options used by the desktop SIPS.

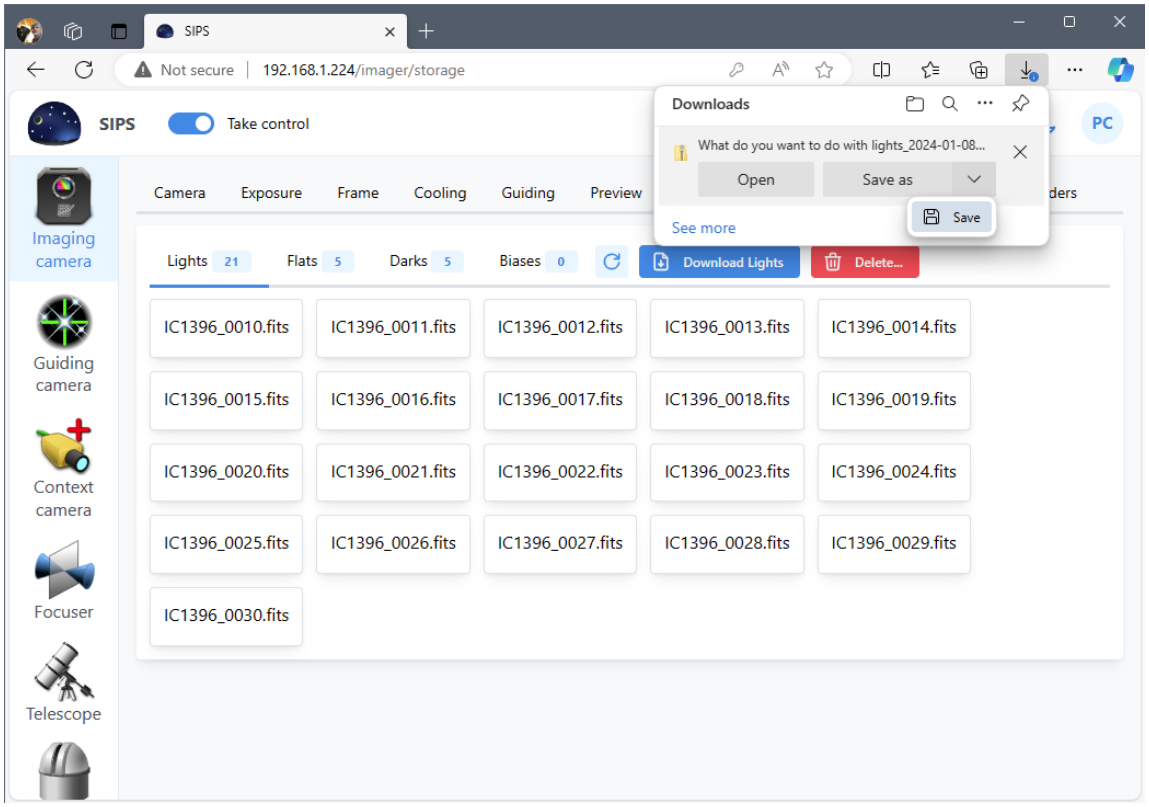


The images stored on the computer running the **Remote Access Web Server** are accessible through the **Storage** tab of the **Imaging camera** page.

The user interface allows to download individual FITS files (by clicking to respective file) or to download all images of the certain type (light, dark, flat, or bias) as a single compressed ZIP file.



The Web SIPS informs the user that the download of a ZIP file with all images was initiated, but the SIPS running on the server must compress all files first and this needs some time. Only after the ZIP file is created, the web browser starts the download. The exact behavior depends on the browser settings—either the ZIP file is saved to the default folder for all downloads or the browser asks the user where the ZIPZ file should be stored.



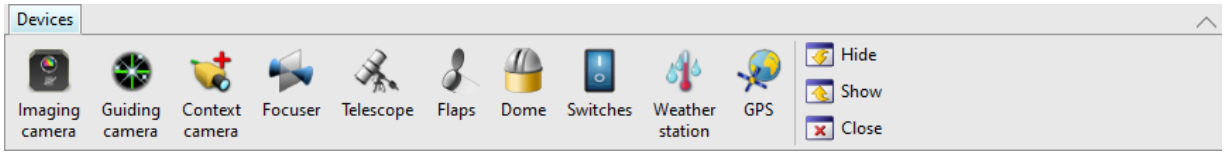
Also, the user interface allows to delete all images of the selected type (light, dark, flat, or bias).

Remark:

The SIPS WWW server does not allow to list folder contents or to access files on the host PC based on the path passed in the URL for security reasons. The FITS file download function is limited only to FITS files, located in the respective image type sub-folders (named “light”, “dark”, “flat”, and “bias”) of the **Local folder**, defined in the **Web Server configuration** dialog box.

Device Handling Tools

This chapter focuses to the group of tools intended for handling hardware devices.



Imaging camera tool controls main imaging camera (exposures, exposure series, binning and sub-frames, sensor cooling, …). Depending on the connected camera features, this tool also controls the filter wheel, either controlled by the camera or a standalone one, camera GPS receiver etc.

Guiding camera tool allows advanced guiding, including plate-solve based guiding using all acquired stars. The tool implements Automatic calibration, moving to different declination or GEM swapping without a necessity to repeat calibration, image dithering etc.

Context camera shows telescope surroundings using the context camera. Context camera image can be also recorded on video.

Focuser enables connection to motorized focusers. Automatic focusing is handled by **Imaging camera** tool.

Telescope tool controls the telescope. Advanced functionality like high precision pointing (plate solving of acquired images and synchronization of coordinates for up to sup-pixel GoTo precision is implemented in the **Astrometry** tool.

Flaps tool allows control of telescope covers (flaps), as well as flat field calibration panels, possibly with adjustable brightness, also moving in front of the telescope.

Dome tool controls either observatory dome or roll-off roof. Advanced functions like synchronization of the dome slit azimuth with the telescopes mounted of GEM are also implemented.

Switches tool is a generalized tool controlling various switches (both binary and analog)

Weather station tool connects to various meteorological and environmental sensors (thermometers, rain detectors, cloud cover sensors, seeing or sky brightness sensors etc.).

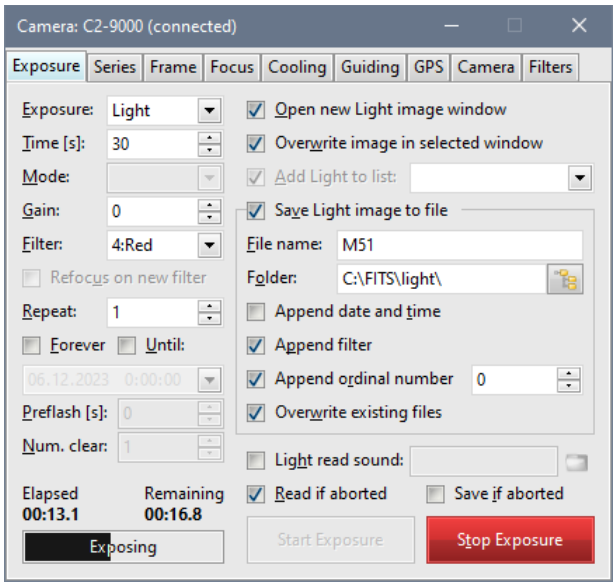
GPS tool allows connection of external GPS receiver, used for computer time synchronization and as a source of geographic location.

> Remark
>
> Every **Device** handling tool can operate with various devices of the certain class (cameras, telescope mounts, …). So, every device tool window contains GUI controls, allowing the user to select respective driver for the used device. Which drivers are offered to the user to choose from depends on the SIPS global configuration. See the **Global configuration file** sub-chapter of the first introductory **SIPS Basics** chapter for details.

Imaging camera and Filter wheel

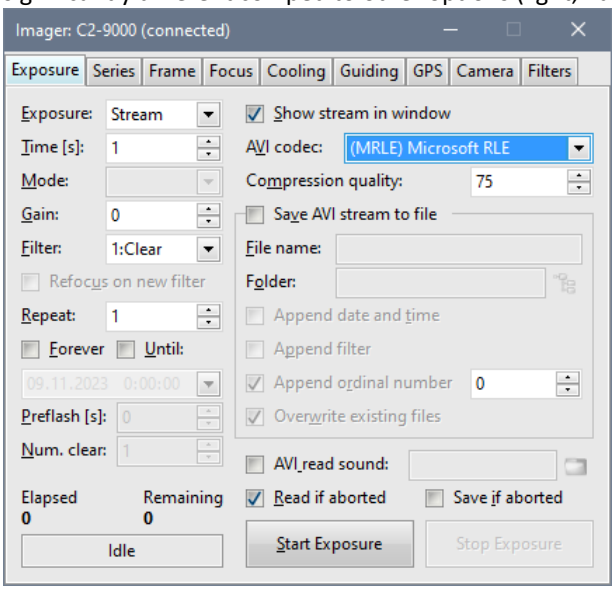
The **Imaging camera and Filter wheel** tool window contains multiple tabs; each one containing GUI controls related to a specific group of camera functions. The window title shows selected imaging camera name and connection status (connected or unplugged).

Exposure tab



The topmost **Exposure** combo-box allows selection of type of exposure performed upon **Start Exposure** button click:

- **Light**—normal exposure. The selected filter (if available) is moved into position and the shutter opens for defined time.
- **Flat**—image exposed as normal light image (filter is used, shutter opens), but intended for flat field calibration.
- **Dark**—dark frame exposure. It is not possible to choose the filter, the shutter remains closed during the exposure.
- **Bias**—takes camera bias frame (a dark frame with zero exposure time). Bias fames are superfluous for proper image calibration despite the fact, that some image processing software packages require them (see the **Calibration** chapter for detailed explanation), but SIPS allows acquiring them for testing purposes.
- **Stream**—creates AVI stream from images read from camera. Because of significant difference between saving of individual images into FITS files and saving single AVI file, also controls on the right side of this tab are significantly different comped to other options (light, flat, dark, and bias).



SIPS offers all codecs, installed in the particular operating system, and also allows definition of compression quality. It is also possible to store uncompressed images into AVI stream. From the image processing point of view is the last option the best one (images are not distorted by lossy compression), but resulting AVI files are then significantly longer.
Images are stored in AVI stream with 8-bit per color even if the camera can provide 16 bits per pixel. The same algorithms are applied to the frames like when they are exported to JPG or PNG files—stored images are in principle the ones displayed on monitor. Image brightness scale is "stretched" according to low and high stretch limit and gamma curve, color palette can be applied to monochrome images etc.
Please note not all compression algorithms can handle monochrome frames (8 bits per pixel), created by monochrome cameras. It is always necessary to make sure that the particular codec and camera works together.

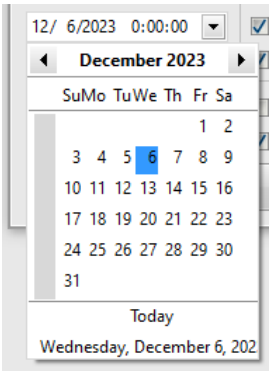
The following set of controls define image acquisition parameters:

- The **Time** count-box defines exposure time in seconds. The count-up and count-down step is not fixed, but always have a value of the most significant decimal place of the exposure time.
- The **Mode** combo-box allows selection of camera read mode, providing the selected camera offers multiple modes.
- The **Gain** count-box allows definition of camera gain. Not all cameras are able to adjust gain, but especially modern CMOS cameras allow setting of gain.

Remark:

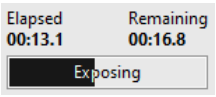
SIPS does not allow to set camera offset. Offset setting is superfluous and brings not benefits. It is only a possible source of problems when combining images taken with different offsets etc.

- The **Filter** combo-box offers all filters available. This combo-box offers filters either controlled by the camera (this means without standalone filter wheel driver) or filters offered by independently controlled filter wheel. See the **Filter** tab description for details.
- **Refocus on new filter** checkbox instructs SIPS to change focus when using another filter, compensating for focal plane shifts caused by different filter thickness. Some prerequisites are necessary to allow this function:
  - The filter description must contain number of steps necessary to refocus.
  - Focuser driver must be selected in SIPS and the focuser must be online.
- The next group of controls allow repeated exposures. The are three exclusive ways how to define repetition (only one can be active at a time):
  - The **Repeat** count box defines number of images to be taken upon start exposure.
  - The **Forever** check box causes new exposure starts immediately after a previous one (infinite exposures). To stop the loop, either uncheck the Forever checkbox (currently running exposure will be finished, but new one is not started) or click the **Stop Exposure** button.
  - The **Until** checkbox allows repeating of exposures up to a defined date and time. The date and time are defined in local time, using the date-time control:



- The **Preflash** and **Num. clear** count-boxes are related to setting preflashing of the sensor with NIR light to even RBI (Residual Bulk Image) conditions. Especially full-frame architecture CCD sensors suffer from RBI, despite even some modern large-pixels CMOS sensors also show RBI to some extent. If the selected camera does not support RBI preflash, these controls remain disabled.

- A progress bar indicates the camera state (exposure step just active). If the current step is exposure itself, progress bar graphically shows the past and remaining time to complete the exposure.



  The displayed states are:
  - Idle
  - Setting Filter
  - Clearing Sensor
  - Exposing
  - Downloading
  - Guiding
  - Recording

Options affecting handling of images, downloaded from the camera, are located on the right side.

Remark:

It is worth emphasizing that **image handling options are defined and stored separately for individual exposure types**. This means options defined for handling dark frames (for instance, if they are saved, file names, folders, …) are not used for light images etc. It is necessary to define handling for all acquired exposure types—**light**, **dark**, **flat**, and **bias** frames. **Stream** exposures have completely different image handling options, allowing to select codec, compression ratio etc.

- **Open new Light/Dark/Flat/Bias image window** causes downloaded image will be opened in window.
- **Overwrite image in selected window** will cause the currently selected window will be reused for new image.

  Warning:

  If the content of the selected window is modified and not saved, it will be lost.

- **Add Light/Dark/Flat/Bias to list** causes the newly download image will be added to selected list.
- **Save Light/Dark/Flat/Bias image** saves the image as a FITS file to defined folder under defined name.
  - **File name** defines the new FITS image prefix.
  - **Folder** defines the folder where new files will be created.
  - **Append date and time** causes the name will be extended with current local date and time.
  - **Append filter** causes the name will be extended with the used filter name.
  - **Append ordinal number** extends the file name with ordinal number, beginning with the value of count-box.
  - **Overwrite existing files** checkbox causes possibly existing files will be overwritten without asking.

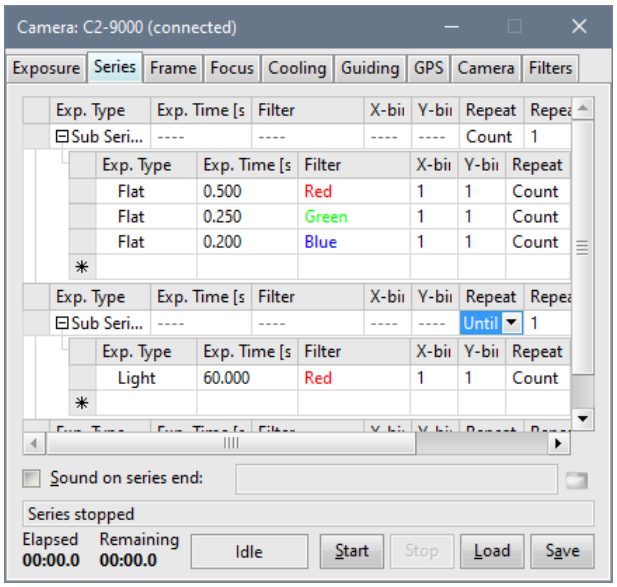
Hint:

It a series of images are exposed and saved to files distinguished by ordinal number, and the last exposure was corrupted (e.g. the telescope was shaken by accidental bump etc.), it is simply possible to decrement the ordinal number in the count-box by one—if the **Overwrite existing files** is checked, the next exposure will be saved under the same file name like the current (corrupted) exposure. The corrupted file will be overwritten and the series will be uninterrupted.

The last group of controls affects SIPS signaling of exposure end and a behavior when the exposure is interrupted.

- **Light/Dark/Flat/Bias sound** allows to define any sound file to be played when the exposure ends and image is downloaded. A sound file can be any .WAV file as well as any other file, which can be played using the operating system codecs (e.g. MP3).
- **Read if aborted** checkbox indicates if the image is read from camera when the exposure is prematurely terminated by the **Stop Exposure** button.
- If the **Read if aborted** option is checked, then the second checkbox **Save if aborted** indicates id the prematurely downloaded image is saved in the case the Save option is checked.

Series tab



The **Series** tab allows definition of more sophisticated exposure sequences than simple repeating of one exposure with the same exposure type, time, and filters.

The exposure sequence described in the **Series** tab allows combination of various exposure types (light, dark, flat) with various exposure times, binning, and filters. The sequence is edited in the hierarchical sheet, where every exposure (including certain repeat count) is represented by single line with columns:

Exp. type combo-box allows choosing of three basic exposure types: Light Image, Dark Frame and Flat Field (plus Sub Series, described later).

- **Exp. type** combo-box allows choosing of three basic exposure types: **Light/Dark/Flat** (plus **Sub Series**, described later).
- **Exp. time** defines exposure time in seconds.
- **Filter** combo-box allows specification of required filter.
- **X-bin** and **Y-bin** count-boxes define binning.
- **Repeat** combo-box defines if the exposure is repeated for a certain **Count** of images or **Until** specified date and time.
- **Repeat limit** column then changes according to previous column value to simple count-box or date-and-time control.

The fourth possible value of the **Exp. type** combo-box is **Sub Series**. If this value is selected, a whole new sub-table is created instead of single line. Parameters **Exp. time**, **Filter**, and **Binning** are meaningless (they are disabled) for **Sub Series** line, but **Repeat** and **Repeat limit** still can be defined for the whole sub-series. Inserting of sub-table enables repeated exposures through different filters etc.

The **Series** tab shows an information line, showing the state of currently running series, below the tree pane. The “Series Stopped” text indicates the series is not running. When the series is started, the information line shows the state of series:

- **Item N of M** indicates the current item of the series. Every item in the list can be either (repeated) exposure or (repeated) sub-series.
- **Exposure N of M** indicates exposure. Exposure parameters (exposure type, filter, exposure time) are displayed in parentheses:

```
Item 3 of 5 [ Exposure 1 of 10 ( Light, Red, 10 )]
```

- **Series N of M** indicates sub-series. Inside sub-series are in square brackets individual items:

```
Item 3 of 5 [ Series 1 of 2 [ Item 3 of 10 [ Exposure 1 of 10 ( Light, Red, 10 )]]]
```

Remark:

It is possible to modify series even while the series is running, but such modifications do not affect currently running series. The whole series description is copied upon series start and the exposure sequence is controlled by the invisible copy.

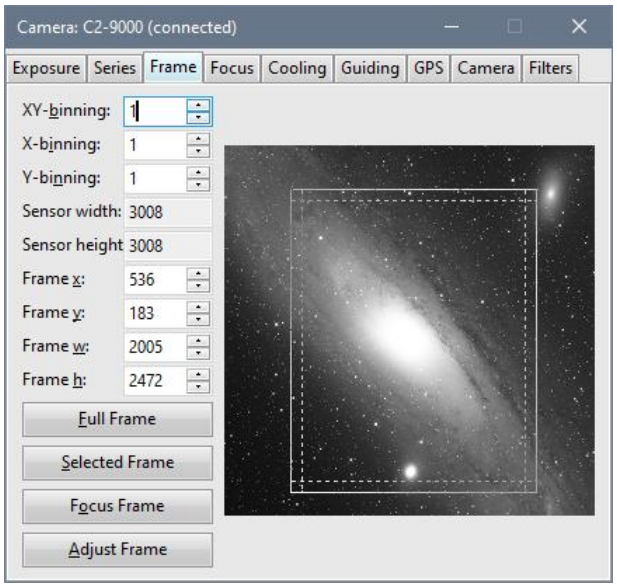
The indicator of elapsed and remaining exposure time of the current exposure as well as the progress bar indicating the exposure progress graphically is located below the information line. Both controls behave the same as in the **Exposure** tab, which controls single exposures.

Sound on series end allows selection of sound file, which will be played when the whole series finishes. A sound file can be any .WAV file as well as any other file, which can be played using the operating system codecs (e.g. MP3).

Once defined series can be saved into files and load again. Series description is a structured text file with keywords defining individual parameters. For instance, the series displayed on the image above is represented by text description (the text is shortened to save space):

```
SIPS Exposure Series File
list
  repeat_num = 1
  list
    repeat_num = 1
    item
      exp_type = flat
      exp_time = 5E-1
      filter_index = 4
      binning_x = 1
      binning_y = 1
      repeat_num = 5
    end_item
    …
  end_list
  list
    repeat_num = 20
    item
      exp_type = light
      exp_time = 6E+1
      filter_index = 4
      binning_x = 1
      binning_y = 1
      repeat_num = 1
    end_item
    …
  end_list
end_list
```

Frame tab



The **Frame** tab allows control of sensor binning and sub-frame readout.

The image displayed as the background of the frame selection box is the currently active image. But not every selected image is used—only images of exactly the same resolution as is the resolution of currently selected camera are used. Note that binning alters actual camera resolution. This is because the frame selection box represents the whole area of the detector, within which the frame is defined.

X-binning and **Y-binning** count-boxes defines horizontal and vertical binning. The maximal values of horizontal and vertical binning depend on the camera used. The **XY-binning** count box affects both horizontal and vertical binning simultaneously, so it is not necessary to define the same binning twice.

Remark:

The **X-binning** and **Y-binning** count-boxes operate independently only is the connected camera supports independent binning in both axes. If the connected camera cannot set binning independently, changing either of the value changes also the complementary one, the same way like the **XY-binning** count box.

The Frame x, Frame y, Frame w and Frame d count-boxes allow definition of sub-frame to be read. Sub-frame can be defined not only by these count-boxes, but also by mouse using the frame selection control on the right side. Frame can be moved or zoomed by mouse dragging.

Remark:

Note that the frame selection control is always smaller than is the resolution of the sensor. Moving by one pixel in the frame selection control means moving by many pixels in the sensor frame. If it is necessary to define frame with single pixel resolution, frame definition count-boxes must be used.

The **Full Frame** button expands read frame to its maximum size.

The **Selected Frame** button allows alternative definition of sub-frame. This button can be used only if the image, selected as the background of the frame selection box, already has selection frame active. In such case the readout frame can be set according to image selection frame. This is probably the most intuitive way to define the read frame—just select a part of image by mouse drag, switch to the **Frame** tab of the **Imaging camera** tool and click the **Selected Frame** button.

The **Focus Frame** button sets the read frame to dimensions of frame displayed in the **Focus** tab. Such frame can be moved over a star desired to use for focusing.

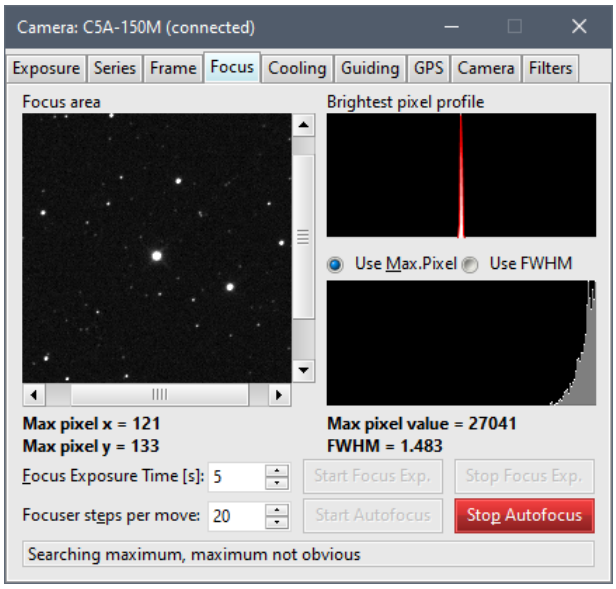
Hint:

Frame used for focusing can be much higher and even the full-size frame can be used. The **Focus** tab then shows scroll bars around the displayed portion of the image, so we can choose any part of the image on which we want to focus. Using of sub-frame typically has an advantage of faster downloads.

Many sensors (especially CMOS ones) impose certain limitations to the position and/or size of the sub-frame, which can be read by the sensor hardware. If such limits exist, the camera driver is forced to read bigger portions of the sensor than the user defined and crop the frame to the desired values in software. Sometimes it is even necessary to read a whole sensor.

It is recommended to click the **Adjust Frame** button, adjusts the frame coordinates according to sensor limitations. Adjusted frame is then read from the sensor, without a necessity to read a bigger portions or even whole sensor and crop image in firmware.

Focus tab



The **Focus** tab displays small portion of the sensor. Actual size of the subframe can be set in the **Frame** tab by clicking the **Focus Frame** button, but there is no limit on subframe size used for focusing. It is possible to focus on entire sensor area, only the download is slower. However, the two panes on the right side (Brightest pixel profile and Brightest pixel history) are always related only to the visible portion of the image. So, if the visible portion of the Focus area is moved and some brighter star becomes visible (or is moved out of the area), the history charts show it as brightest pixel changes, but these changes are not caused by changing of the focus.

Pressing the **Start Focus Exp.** button starts continuous exposures of the length defined in the **Focus Exposure Time** count-box. The **Stop Focus Exp.** button terminates the focusing sequence. Each start of the sequence clears the **Brightest pixel profile** and **Brightest pixel history** charts.

There are two chars aiding the interactive focusing on the right side of the focusing frame. The upper-right **Brightest pixel profile** chart shows profiles of the brightest pixel within the displayed area for three last exposures. The most recent exposure profile is shown in white, previous exposure profile in light grey and the oldest exposure profile in dark grey. The content of the bottom-right chart depends on which of the two radio-buttons is selected:

- **Use Max. Pixel** option causes the history of the brightest pixel value is displayed. Despite the simplicity of just using the brightest pixel to judge focusing, it is very powerful indicator of focus quality. Especially very close to the optimal focus, the value of the brightest pixel changes very significantly.
  On the other side, the brightest pixel value is strongly influenced by the seeing. Longer exposure times, used for focusing, eliminate seeing influence to some extent, but also slow down the focusing process.
  Also, high differences among brightest pixel values of individual focusing exposures, caused by seeing, indicate the focus is close to optimum and image sharpness is more limited by seeing than by focus.

- **Use FWHM** option depends on the Gaussian fit of the profile, from which the FWHM (Full Width Half Maximum) value is derived. The FWHM is a robust indicator of the star image quality, but as opposed to brightest pixel, changes of FWHM are not that significant very close to optimal focus. Also, the Gaussian fit precision strongly depends on the camera sampling (pixel angular size). If stars are projected to disks only a few pixels in diameter, the fit is significantly less precise than if the star images span tens of pixels.

Automatic focusing

Automatic focusing requires a motorized focuser. Also, the focuser must have proper driver installed in SIPS and must be online else the automatic focusing feature cannot be used.

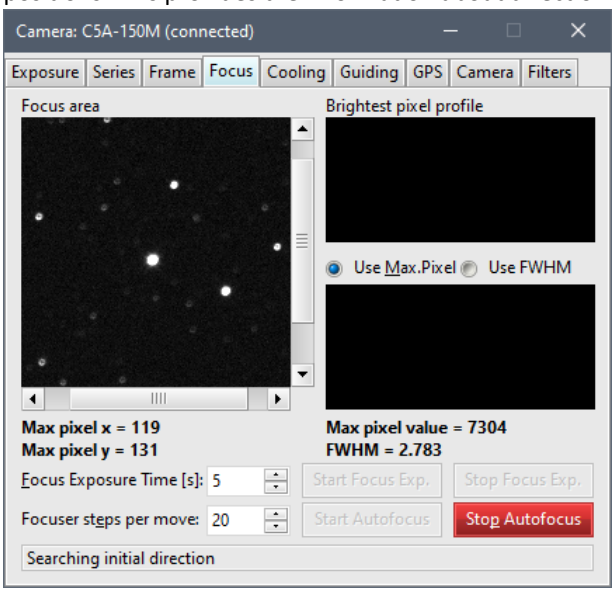
Hint:

Motorized focuser interface allows definition of the number of steps, performed per single **Steps In** or **Steps Out** button click. Single step (if the focuser driver offers its physical length, the Focuser tool also displays actual size in millimeters) is often too short to cause noticeable image change.

Automatic focusing does not use the value defined in the **Focuser** tool window, but it uses the value **Focuser steps per move** value instead. It is very important to choose value big enough to significantly change the image. It is impossible to find trends in a few images if the shift is so small that the change is hidden in the noise. As explained later, large step does not negatively impact focusing precision.

Automatic focusing assumes that the best focused image of a star creates the highest pixel value(s). So, the task of the autofocus procedure is to find a focuser position in which the brightest star displayed in the Focus area pane achieves highest pixel values. Autofocus is performed in three steps:

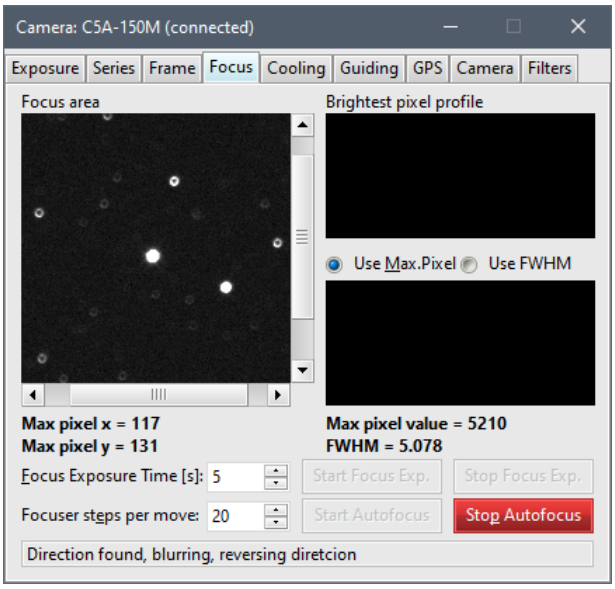
- SIPS tries to find the “slope” of the maximal pixel values on subsequent images acquired with different focuser positions. This provides the information about direction the focuser must be moved to improve focus.



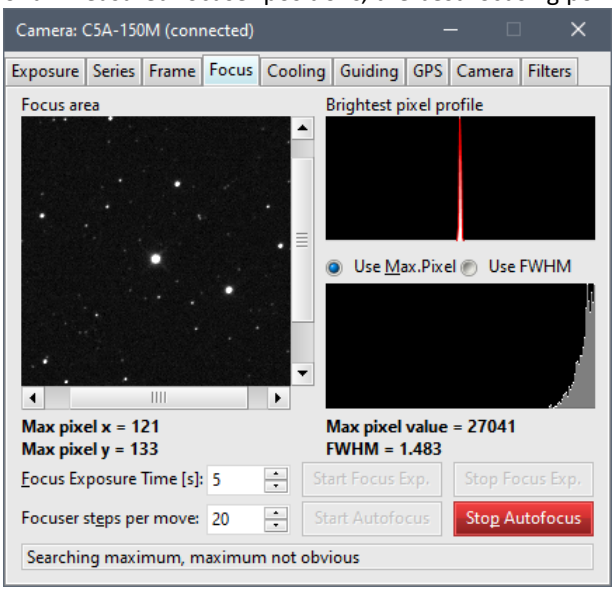
Remark:

Autofocus must start from a focuser position, which ensures a suitable star image is acquired and the S/N ratio is high enough. This means the brightest pixel value must be well above the background noise level. The current algorithm is not intended for finding proper focus starting from fully defocused images, on which only uniform noise is detected.

SIPS performs on at last 12 subsequent images and checks for the brightest star pixel. If the brightest pixel goes down, SIPS reverses the direction and repeats the sequence.



- Then the software performs series of exposures while moving focuser among them until distinct peak of maximal pixel values can be clearly detected. SIPS tries to fit a Gaussian curve to the set brightest pixel history values (or vertically inverted series of FWHM values) and if the function maximum lies within the central third of all measured focuser positions, the best focusing point is considered found.



Remark:

Both steps 1 and 2 moves focuser for defined number of steps (**Focuser steps per move** parameter) between exposures. Using small movement does not automatically lead to best focus. In fact, the single move should be relatively large to cause significant changes in the image. Brightest pixel value is affected by seeing, telescope mount tracking irregularities, by star image position relative sensor pixels etc. These irregularities affect maximal pixel value around proper focus much more than in the case of images out of focus. It is desirable that the sampled values span wide range of maximal pixel values, from clearly defocused up to properly focused.

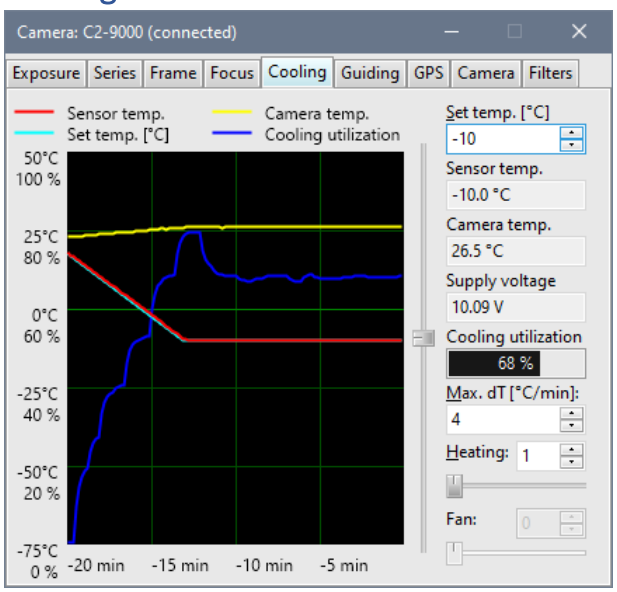
- The focuser is then moved to the point of maximum value of the fitted function (Gaussian mean). Let us note that while the distance of focuser positions among individual exposures, used for focusing, is defined by the **Focuser steps per move** parameter, the best focus position is always calculated to the maximal precision allowed by the motorized focuser hardware.

Hint:

The image displayed when autofocus stops is not the best focused image. In fact, it is the last one, acquired during the autofocus sequence, typically quite defocused. Keep on mind SIPS needs to pass the best position to be able to calculate its location and return to it.

It is always possible to click the **Start Focus Exp.** button after the automatic focusing finishes to show current image and to make sure autofocus worked well.

Cooling tab



The **Cooling** tab displays the sensor temperature, camera internal temperature, and some other information like power supply voltage and cooling utilization. It also contains controls allowing selection of the sensor target temperature and the speed (ramp) of temperature change. Also, this tab is used to control camera cold chamber optical window heating as well as camera fans.

Remark:

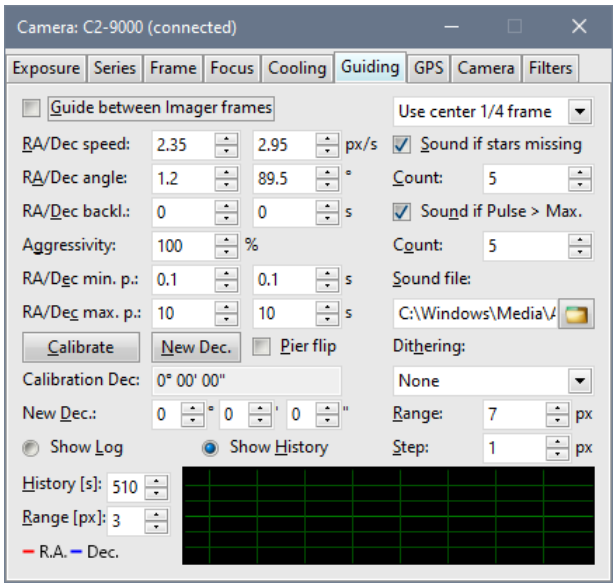
The target temperature is not immediately propagated to the camera, but the required temperature changes only at the speed defined by **Max. dT** count-box. This feature brings several advantages. First, rapid temperature changes are not good for electronic devices. Some sensor manufacturers define the maximum speed of cooling (or heating), some manufacturers do not. But changing of the sensor temperature only at the speed of few degrees per minute prevents building of hot/cold pixels and other defects on the chip with time.

Availability of shown values as well as ability to control cooling, window heating and fans depend on the capabilities of the connected camera.

Controls for **Heating** and **Fan** allow regulation of sensor cold chamber front window heating (to prevent fogging or even frosting in wet conditions) and Peltier hot side cooling fans. Please note different cameras offer settings of these parameters in different number of steps or even do not allow to control such function at all. If the connected camera does not allow to control specific function, the GUI controls are disabled (greyed). If only basic on-off switching is available, just two values 0 (off) and 1 (on) are offered. If the camera allows more precision regulation, more steps are available to the user.

The left pane shows the graph of four values (Sensor temperature, Set temperature, Camera temperature and Cooling utilization) over last 20 minutes.

Guiding tab



The **Guiding** tab is intended for inter-image guiding. This feature is designed for modern mounts, which are precise enough to keep tracking with sub-pixel precision through the single exposure, and irregularities only appear on the multiple-exposure time-span. Such irregularities are caused by many sources (polar misalignment, differences in refraction calculations, mount and telescope mechanical deformation etc.) and cannot be completely avoided. Inter-image guiding then performs slight mount position fixes between individual exposures of the main camera, which eliminates “traveling” of the observed objects through the detector area during observing session.

Because this guiding method uses main imaging camera, it does not use another guiding camera and naturally does not need neither OAG nor separate guiding telescope to feed the light into it. This also brings another consequences:

- The **Guide using: Guiding camera/Telescope** option is missing from inter-image guiding setup, as the inter-image guiding always use **Telescope Pulse-Guide** interface. If the particular telescope driver is not equipped with such interface, inter-image guiding cannot be used. Especially older mounts often rely on the “Autoguider port” only and thus require the opposite port, typically offered by the guider camera. But such mounts are often not capable to keep tracking for any reasonable long exposure without regular guiding, so this limitation is not that important.
- The **Position detection: Brightest star/Plate match** is also missing as the guiding method is always **Plate match**. This means stars are searched within the whole image and matched with the stars on the reference frame (reference frame is the first frame taken after the inter-image guiding is turned on). This method results into more reliable guiding compared to brightest star method, as the mutual offset between frames is calculated as an average of offsets of all detected and matched stars. Brightest star method also requires frame cropping around the chosen star, which is naturally not possible in the case of the main camera.

It is necessary to detect at last 3 stars within the image to be able to use **Plate match**. If less stars are detected, this method of guiding cannot be used.

Searching for stars is thus crucial for proper function of the inter-image guiding. One algorithm as well as one set of parameters for star searching is used within all SIPS tools (there is a second set of star search parameters, intended for guiding camera, but this is not relevant in this case).

Remark:

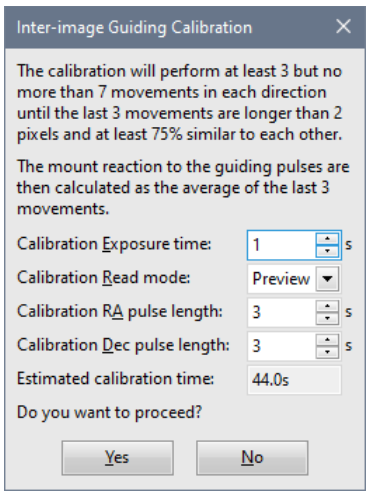
However, there is a difference in searching for stars for the purpose of inter-image guiding compared to for instance astrometry. The astrometry and photometry often use two apertures—the smaller one not to miss weak (small) stars close to bright ones, and the greater not to miss very bright (big) stars within the image. But if two apertures are defined, only the greater aperture (the second one) is used if stars are searched for guiding. The plate matching algorithm uses only the brightest stars either way, so finding all weak stars only consumes CPU time and memory.

It is recommended to use the **Astrometry** tool to check whether stars within the image can be properly detected and possibly to modify star searching parameters. This procedure as well as the searching parameters are thoroughly described in the **Astrometry** tool chapter.

Hint:

Especially if large-sensor cameras are used on wide-field telescopes, the number of detected stars can be huge and astrometry needs a lot of computing time. But pointing precision does not increase if there are thousands of stars found and matched instead of just tens or hundreds. So, the **Guiding** tab offers a combo-box, which allows cropping of the image, used for plate-matching of exposures, to 1/2, 1/4 or 1/8 of the image (the fraction is in linear dimension, so the image area, on which stars are searched, is 1/4, 1/16 and 1/64, significantly reducing the needed computation time).

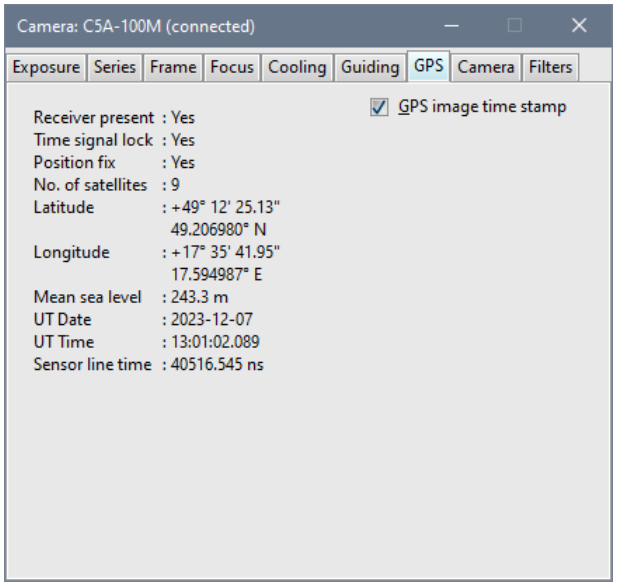
Similarly to regular guiding, also the inter-image guiding requires calibration (refer to the **Guiding camera** tool description for calibration details). The only difference is the inter-image guiding calibration does not use predefined exposure time and read mode, but offers entering of exposure time for calibration only, as the normal imaging camera exposure time could be hundreds of seconds and the calibration would take very long time.



Hint:

The “missing stars” alarm can be very effectively used also for weather monitoring. If some clouds appear and the stars cannot be found on the image, SIPS can alert (wake-up) the observer.

GPS tab



If the used camera is equipped with a GPS receiver, the **GPS** tab shows all related information.

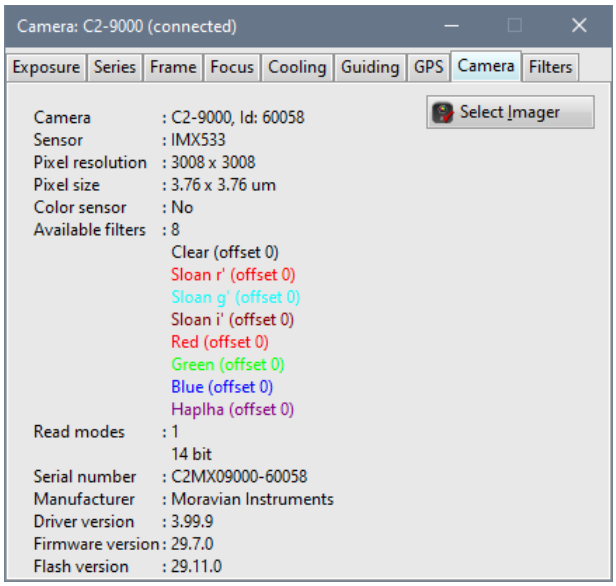
The only control in this tab is the **GPS image time stamp** checkbox, which determines whether the images acquired from camera will be time-stamped with PC system time or with a precision time read from the GPS receiver (ability to determine the exposure origin with the µs precision is the main purpose of the GPS receiver presence in astronomical camera).

Hint:

CMOS sensors with rolling shutter do not start exposure of the whole image at once, but the exposure is started line by line with certain delay. So, to determine the time of exposure start for each line, it is necessary to add the line y-coordinate multiplied by time needed to handle each line.

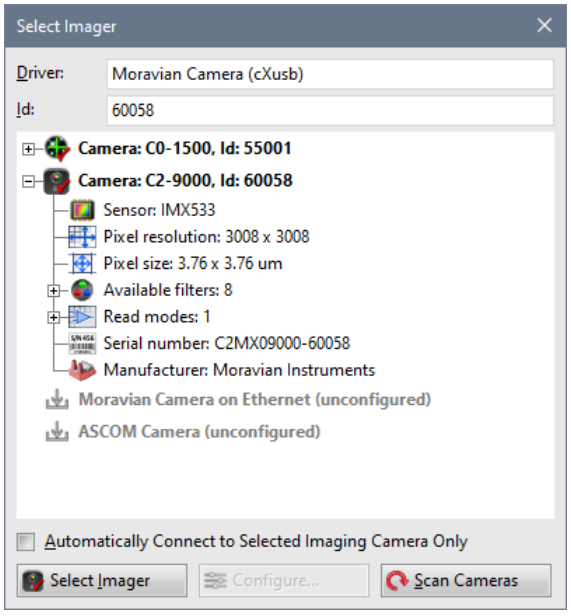
The line delay can be calculated from theoretical values, but there can be some differences caused by used electronic components manufacturing tolerances etc. Moravian Instruments camera drivers measure the real value of the line time using the GPS signal (the used GPS receivers provide one second pulses with a precision in the order of ns) and provide it to the user. SIPS GPS tab shows this value as **Sensor line time**.

Camera tab



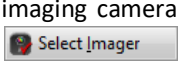
SIPS is designed to work with any camera, providing appropriate camera driver exists and is loaded (camera drivers are listed in the global SIPS configuration file). The **Camera** tab summarizes properties of the selected imaging camera and allows selection of any connected camera as an imaging one.

The **Select Imager** button allows choosing of a camera from all connected cameras.



All available connected cameras are displayed in the tree. Individual tree branches can be expanded to show camera details (sensor type, resolution, etc.).

SIPS allows for three active cameras—one for imaging, another for guiding and yet another for context.

- Currently active imaging camera name is displayed in the **Imaging camera** tool window title bar. Active Imaging camera is also indicated by the corresponding icon in the displayed tree. Another imaging camera can be selected by highlighting desired camera in the list of available cameras and clicking the 

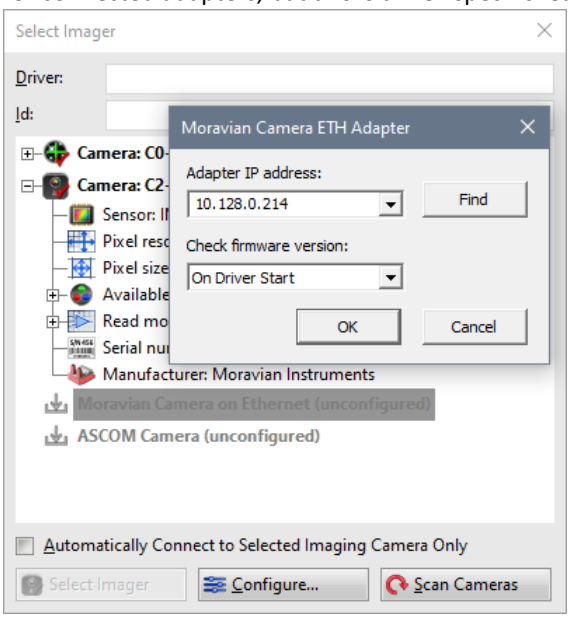
button.
- Similarly, selected guiding camera (if any) is marked by a respective icon.
- The selected context camera (if any) is also marked by its icon.

SIPS camera drivers can behave in two ways:

- Plug-and-play drivers automatically show all connected cameras, there is no need to manually configure them. Typically, cameras connected through interface like USB, which natively supports plug-and-play operation, act this way.
  If a camera is newly connected to the computer while the SIPS is already running, it is necessary to scan for available cameras to show the attached camera in the tree using the 

button. But when the already registered camera is disconnected, it remains in the tree and can be selected.
  If no camera, handled by the particular Plug-and-play driver, is connected to the PC, then the camera driver remains “invisible” in the camera tree.
- Some camera drivers require configuration. Typical example is a camera connected over TCP/IP (it is necessary to choose the IP address of the camera) or ASCOM compatible camera (ASCOM is general interface, it is necessary to select which camera should be connected through the ASCOM). In such cases the driver always displays one line in grey color, representing the driver itself (but no specific camera). Such grayed line cannot be selected as camera, but the 

button is enabled and driver-specific configuration dialog box appears.
  For example, cameras connected through the Gx Camera Ethernet Adapter are enumerated and offered for selection only upon configuration defines IP address of the adapter (there is a possibility to scan the network for connected adapters, but this is driver-specific feature).



The camera connection status is displayed in the brackets in the tool window title. When the camera is not connected, it is of course impossible to start the exposure etc. and controls commencing such actions are disabled.

If there are more cameras of the same type connected to one computer, it is not possible to distinguish them from the camera name or chip description. This is why every camera has unique identifier, either number or string, in SIPS. It is a task of the camera driver to provide the identifier—some cameras have unique identifier already stored in their permanent memory, otherwise the driver itself must generate the unique identification. The identifier is always

displayed in brackets following the camera name, so it is always possible to choose the desired camera among all connected ones.

The **Automatically Connect to Selected Imaging Camera Only** checkbox modifies SIPS handling of camera connection.

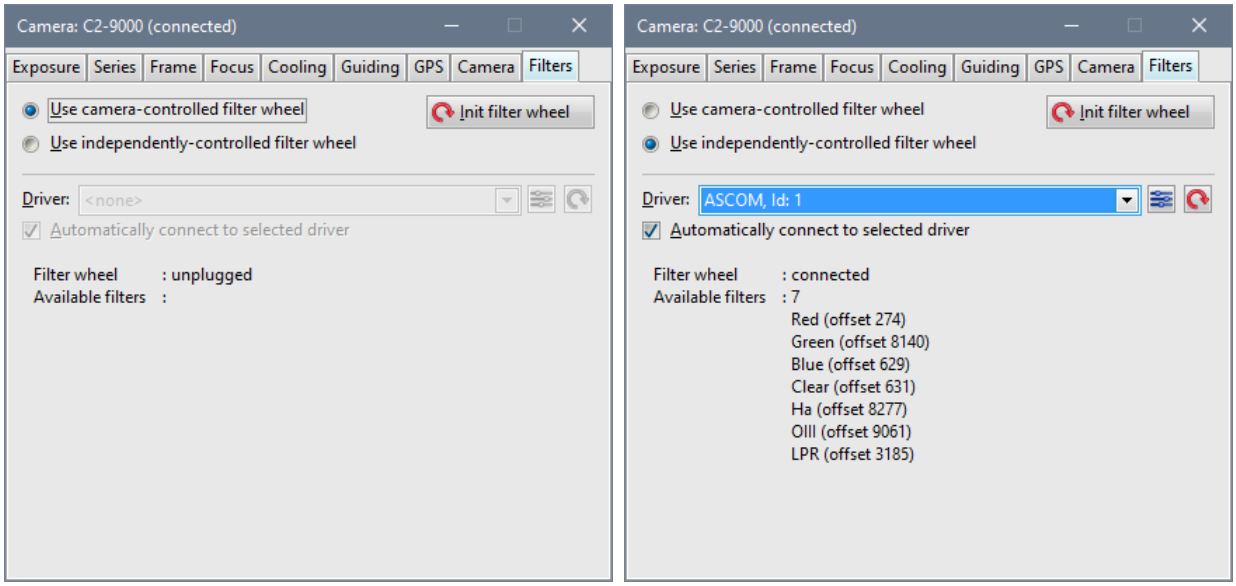
If the checkbox remains un-checked, SIPS scans all connected cameras on startup. The first enumerated camera is then selected as imaging one (this behavior is valid for imaging camera only, guiding and context cameras remain unselected, providing the corresponding checkboxes of these cameras are also unchecked).

Connecting to the first found camera may be helpful e.g. in the lab environment, where many cameras are used, but such behavior is not convenient if SIPS is used to control an observatory and a fixed set of cameras is connected to the host PC. In such case, an observer has to manually select each camera prior start of the observing session.

Also, when SIPS is run prior the camera is powered on, no camera is selected. After the camera is switched on and connects to the PC, it is again necessary to open the camera selection dialog box, scan all cameras and selected it.

Checking of the **Automatically Connect to Selected Imaging Camera Only** checkbox causes SIPS always tries to find the desired camera and select it as imaging one even if it is not enumerated on the first place. When the SIPS is started and camera is attached to the PC only later, SIPS detects the camera is plugged in and selects it as imaging camera without any user's action.

Filters tab



SIPS implements two ways how to control filter wheel:

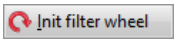
- Filter wheels controlled (and powered) by the camera electronics and thus also by the **camera driver**. Every camera driver can implement API calls for filter wheel control.
- Standalone filter wheels, controlled and powered independently and offering own driver.

The **Use camera-controlled filter wheel** and **Use independently-controlled filter wheel** radio-buttons distinguish which driver SIPS uses to set filters (camera driver of standalone filter wheel driver).

If the standalone filter wheel driver is to be used, its driver selection and configuration behaves the same way like other SIPS controlled devices (telescope, focuser, …).

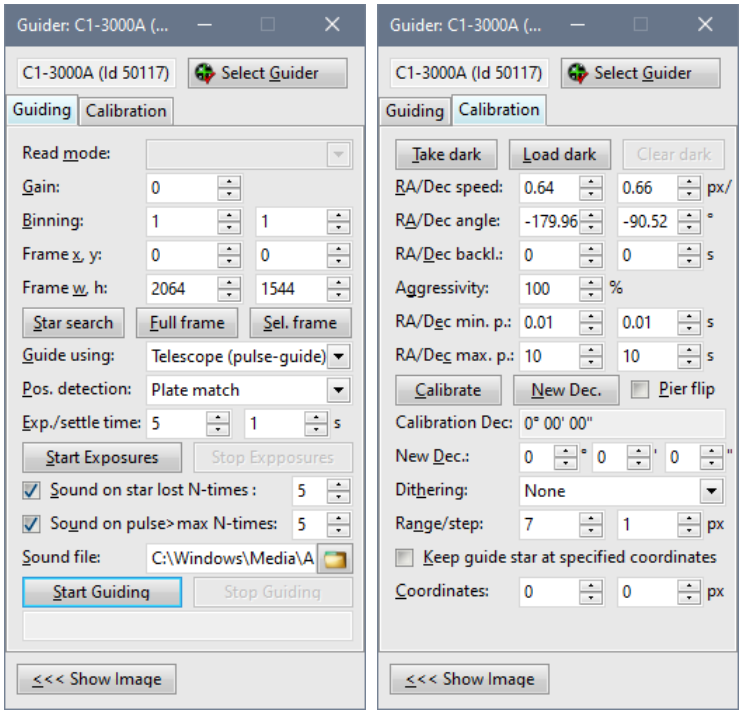
- The **Driver** combo-box shows all available drivers.
- If the driver needs configuration (for instance, ASCOM needs to select which ASCOM controlled filter wheel is to be used), the (Configure Filter Wheel driver) button next to the combo-box opens the configuration dialog box.
- And the (Scan Filter Wheel drivers) button next to the Configure button enumerates all connected drivers for the case the filter wheel was connected when the SIPS was started.

The **Automatically connect to selected driver** checkbox function is the same like the similar option of camera driver. It ensures SIPS will select the desired filter wheel driver even if it is not enumerated as a first one and also connects to the driver even if the plug-and-play device is connected to the PC after SIPS is run.

The 

button re-initializes the filter wheel. This typically means the filter wheel performs the same operation like when it is powered up and finds the first filter position. However, not all filter wheels offer this function, for instance the ASCOM standard does not allow to perform such operation.

Guiding camera



Astronomical mounts, capable to keep the star images perfectly round during long exposures, are more exceptions than a norm. Modern cameras and telescopes allow for perfectly sharp and high-resolution images, so even a small irregularity in mount tracking appears as star image deformations.

If the mount is capable to keep the irregularities below the limit defined by star shape, the inter-image guiding, using the main imaging camera, can be used to correct for very slow drifts (see the **Imaging Camera, Guiding tab** sub-chapter). A standalone guiding camera and a software performing mount corrections in much shorter intervals than is the exposure time is needed in all other cases.

SIPS provides support for automatic guiding with standalone guiding camera through the **Guiding camera** tool. Guiding corrections are not calculated in the guiding camera itself, camera only sends acquired images to the PC. SIPS running on the PC calculates the difference from required state and sends appropriate corrections to the telescope mount. The plus side of using a host PC CPU to process images is the overwhelming computational power of current PCs compared to any embedded processor inside the guiding camera. Guiding algorithms running on PC can determine star position with sub-pixel precision, they can match multiple stars to calculate average difference, which limits the effects of seeing, etc.

Calculated corrections can be sent back to mount using either PC-to-mount link or using the so-called “autoguider” port of the guiding camera.

- If the mount controller is a modern one and supports so-called “pulse-guide interface”, using it is the most straightforward way to guide the mount. Of course, mount driver must be installed in SIPS and the mount must be connected and online for such functionality. Having mount interface connected to the host PC, which controls the whole observing session, brings numerous advantages, and makes observing much more comfortable either way. Another plus is the possibility to use any camera for guiding, it is not necessary to use specialized guiding camera with autoguider port.
- If both the guiding camera and the telescope mount are equipped with an autoguider port, it is enough to connect the camera (or standalone guiding interface) and the mount using 6-wire cable and guide the mount directly.

  Remark:

  The autoguider port originated before the “intelligent” mount controller become common—it allows guiding the mount by simply grounding (connecting a pin to the common or “ground” pin) of one of four pins,

representing four guiding directions. While the particular pin is grounded, mount keeps moving by guide speed in the corresponding direction. With the introduction of mount controllers, equipped with own embedded computers and capable to perform guiding corrections themselves, sending the pulse-guide command to the mount is much simpler. And no special control ports on both camera and mount and another cable between mount and camera is necessary.

The **Guiding camera** tool window contains all necessary controls to perform telescope mount guiding. Because guiding is rather complex task and requires many parameters on the one side and provides the user with information about mount performance, guiding reliability etc. on the other side, the controls of the **Guiding camera** tool window are divided into two separate tabs—**Guiding** and **Calibration**.

It is highly desirable to view the images provided by the guiding camera when searching guiding star and during guider calibration. On the other side it is not necessary to continuously monitor these images during regular guiding. This is why the **Guiding camera** tool window can be displayed in two modes:

- Shrunk mode, showing all parameters and only a single status line, which is updated with the actual status.
- Extended mode with panes displaying current guiding camera image, graph showing history of guiding corrections and log of all performed operations.

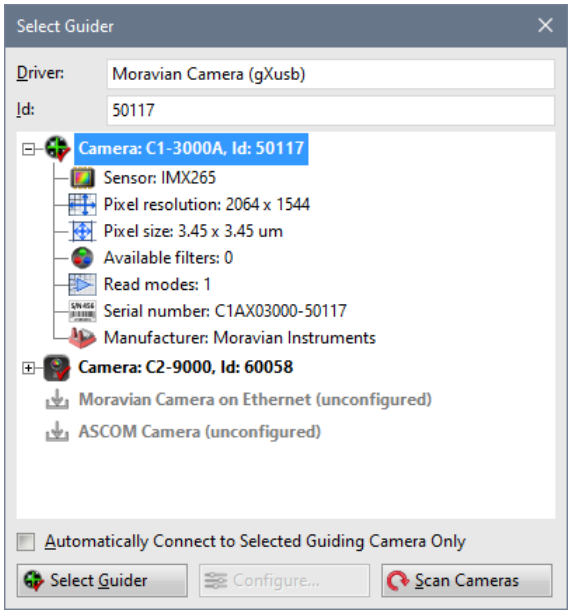
Remark:

The status line on the bottom of the **Guiding** tab shows the last line written to the log pane, visible in the extended mode.

Switching between modes is performed by lower-left buttons <<< Show Image and >>> Hide Image.

Guiding camera and guiding interface setup

Guiding camera is selected by clicking the Select Guider button the same way as the imaging camera (described in chapter Imaging camera and Filter wheel).



There are no limitations in which camera can be selected as guider. If the camera can be used in SIPS (there is a driver available), it can be selected as guiding camera. The only restriction is a necessity to be able to guide through telescope driver pulse-guide interface if the selected camera is not equipped with an autoguider port (see the description above).

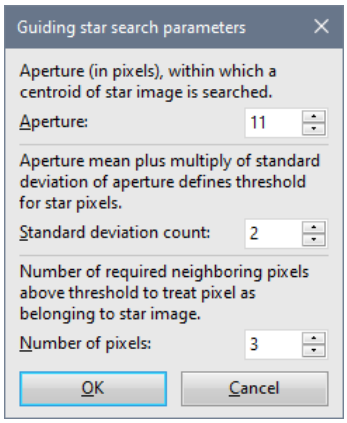
Numerous guiding camera parameters can be set:

- **Read mode** combo-box allows choosing of the required mode, if the camera supports multiple modes (e.g. fast or preview mode and slow low-noise mode etc.).

- **Gain** count-box allows setting of gain, if the camera supports it. High gain should be beneficial for weak guiding star, but if the guiding star saturates, guiding precision drops as it is not possible to properly determine star centroid. So, lower gain and/or shorter exposure time can be used.
- **Binning** count-boxes defines horizontal and vertical binning. The maximal values of horizontal and vertical binning depend on the camera used. Binning can be used to gain S/N for the long focal length and small guiding camera pixels.
- **Frame x, y, w, h** allows definition of a sub-frame. If the **Brightest star** is chosen as the **Position detection** method, cropping of the image read from guiding camera to a subframe around selected guiding star is essential to successful guiding.
  Subframe can by defined either manually using count-boxes or it is possible to drag a mouse over image on the left pane and click the **Sel. frame** button. Selected area coordinates will be copied to **Frame** definition count boxes. The **Full frame** button again sets the full image to be read.
- **Guide using** allows selection how to move the mount.
  - **Camera Autoguider port**. If both the guiding camera and the telescope mount are equipped with the autoguider port (and they are connected with 6-wire cable), use the guider camera to perform mount tracking adjustments.
  - **Telescope (pulse-guide)**. If the mount controller supports so-called "pulse-guide interface", use it to guide the mount. Of course, the mount driver must be active in SIPS and the mount must be connected to the host PC and online.
- **Pos. detection** selects how SIPS determines tracking irregularities.
  - **Plate match**—SIPS performs basically the same operation like in the case of sub-pixel matching of multiple exposures or astrometry. Number of triangles are created from the brightest stars and they are matched to triangles on reference frame.
    Although plate matching requires at last three stars on the guiding image and thus it is suitable either for short focal length guider telescopes or for rich star fields, the image shift is calculated from multiple star positions and is less sensitive to random errors like seeing, radiation spikes etc. On the other side, it is much more CPU demanding.
  - **Brightest star**—SIPS calculates the centroid of the brightest star on the image acquired by the guider. The centroid position is calculated to the fraction of pixel precision, so the guiding can be very precise even when the guider is used on the telescope with a short focal length.
- **Exp./settle time** selects exposure time for guiding camera and a settle time, for which SIPS waits after guiding correction, before next guiding camera exposure starts. While the later parameter (settle time) is often 0 (mount, which requires time for stabilization after guiding correction, is hardly usable for serious work), the first parameter defines exposure time and it is important. The time must be long enough to capture guiding star with good S/N and at the same time short enough to allow fixing of the mount tracking irregularity before the error is too big. Exposure time depends on many parameters—how big is the guiding camera field of view and how bright star is available within the field of view, how big is the guider telescope and what is the guiding camera sensitivity, how good is the mount and how long it can track without a necessity to correct it, etc.

Searching stars

Regardless of the **Brightest star** or **Plate match** position detection method is used, it is necessary to find a star or multiple stars on guiding camera image. SIPS shares one set of star search parameters among multiple tools (**Blink Images**, **Astrometry**, **Photometry**, …), as it is supposed all these tools work with images from the same camera and telescope. But the guiding camera is an exception—it often uses another telescope, intended for guiding, and even if the guiding camera shares the telescope with the main imaging camera through OAG, the guiding camera often differs in pixel size. Thus, the guiding camera use separate set of star search parameters.



Because the purpose of the star searching algorithm for guiding is only to find brightest stars within the image, numerus parameters are omitted—it is not possible to use two apertures, no star mask is iterated and star profiles are not fit with Gaussian curves.

- **Aperture** is the diameter (box size) of pixel area, in which a star centroid is searched. General rule is the star image should well fit in the box else the star will be not recognized. Too big aperture on the other side covers more pixels than necessary.
- **Standard deviation count** defines the threshold, above which a pixel is considered to belong to a star image.
- **Number of pixels** defines how many neighboring pixels above threshold must be detected to consider them star pixels. This condition eliminates single bright pixels above threshold like traces of sensor hot pixels etc.

Hint:

Note all images downloaded from guider camera are automatically processed and all found stars are ringed with green circles. This feature helps to determine whether the star search parameters are properly defined. If for instance the brightest star on the image, intended for usage as guiding star, is bigger that the green circles around weaker stars and the brightest star itself is not in green circle, increase the **Aperture** parameter.

If we decided to use the **Brightest star** to calculate guiding corrections instead of **Plate match**, and there are more similarly bright stars in the field of view, it is possible that some other star becomes brighter from time to time due to random errors and the algorithm would be misled. So, it is important to limit the image frame to area around the chosen guiding star for the **Brightest star** position detection.

Guiding Calibration

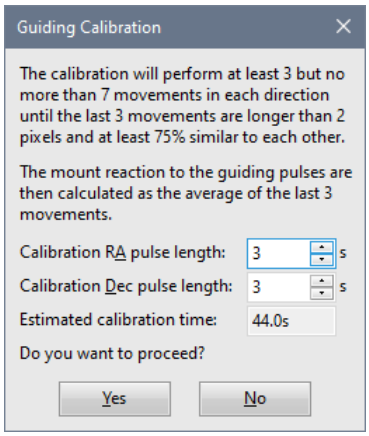
Every automatic guiding system must be calibrated before use. When the software detects that the reference star moved on the image, it must move the mount in the opposite direction to correct the tracking error. The mount movement is controlled by the length of the correction pulse. Proper pulse length is calculated from the known distance on the image (expressed in pixels) and the mount guiding speed (expressed in pixels per second).

Four parameters are necessary to calculate correction pulse lengths and orientations:

- R.A. speed
- Dec. speed
- R.A. angle
- Dec. angle

While it is in theory possible to calculate these parameters from mount guiding speed, object equatorial coordinates, guider telescope focal length and guiding camera pixel size, in reality these parameters are always measured “experimentally” during the calibration. This process is initiated by clicking the **Calibrate** button.

The calibration process starts with a dialog, defining the **Calibration RA pulse length** and **Calibration Dec pulse length** parameters. Pulses with defined length in seconds are used to shift the star (move the mount) in both directions to measure how the star position reacts to mount movement. The four guiding parameters, mentioned above, are then calculated from the measured offsets.



As usual, suitable pulse lengths depend on many parameters, and it is highly recommended to make some experiments with each observing set (mount, camera, telescope) to determine proper values.

- In general, longer pulse causes longer star movement, which results into more precise calculation of guiding parameters.
- But too long pulse can cause the star is moved out of selected frame and calibration fails.
  Another possibility for calibration failure is that too long pulse moves another (brighter) star into frame and calibration results will be wrong.
- Also tracking irregularities themselves speak against too long calibration pulses. If the mount requires guiding pulses every few seconds, irregularities influence the star position and limit calibration precision.

Hint:

SIPS introduced a new, robust guiding calibration algorithm beginning in version 4.2. The calibration need longer time, but it is resistant to mount mechanical backlashes in both axes.

The calibration performs at least 3 but no more than 7 movements in each direction until the last 3 movements are longer than 2 pixels and at least 75% similar to each other. The mount reaction to the guiding pulses is then calculated as the average of the last 3 movements.

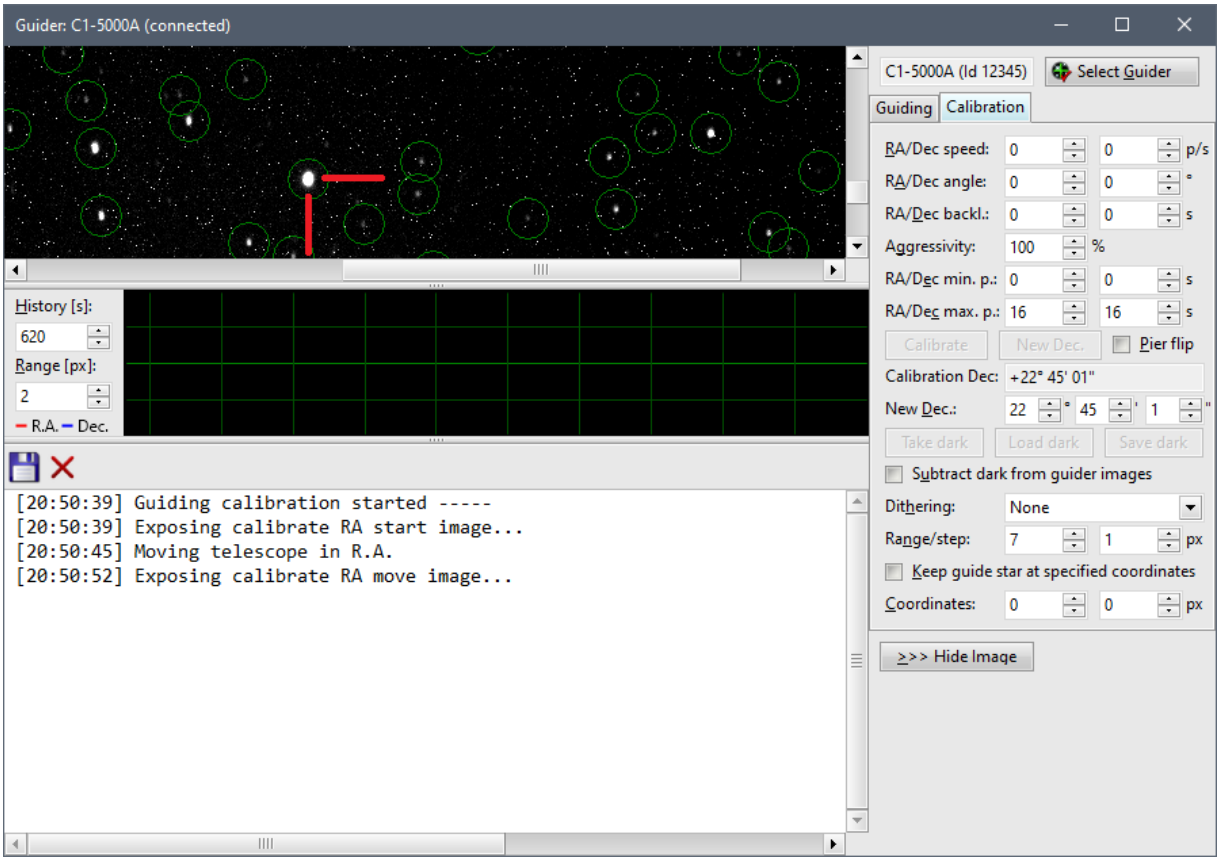
SIPS performs guiding calibration in several steps, illustrated below.

Remark:

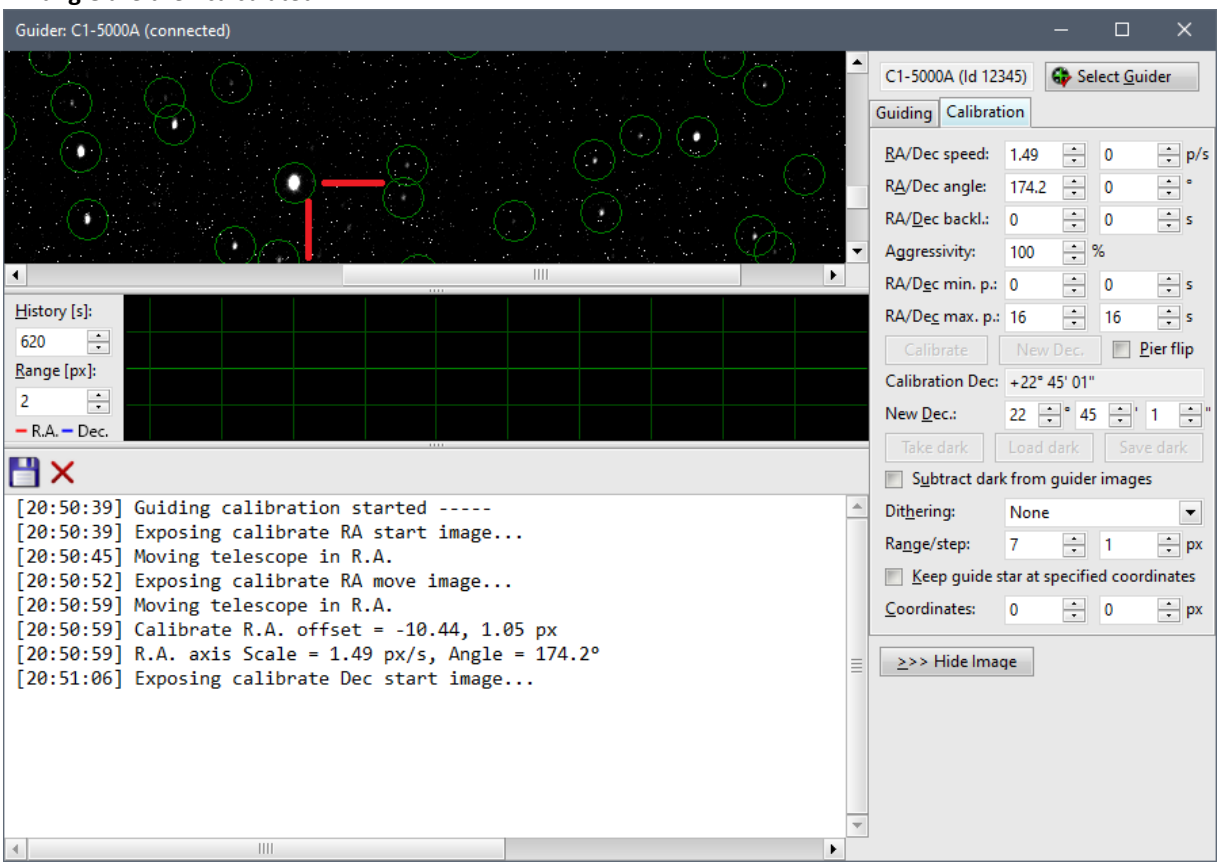
Please note that the following calibration example is very close to ideal situation. We used precise telescope mount without any measurable backlash and periodic error. The mount itself could track minutes without guiding and its reaction to guiding pulses is immediate and accurate within a fraction of arc-second.

Typical mass-marketed mount, on the other hand, requires guiding corrections every few seconds because of irregularities. Periodic error in the order of tens of arcseconds and significant backlash in both axes may cause various problems. It is OK to manually alter the determined guiding parameters (e.g. round the angles and make them perpendicular, adjusting the Dec. speed to compensate for the backlash during calibration etc.) once the user gains experience with the used mount. Note that while R.A. speed varies with declination, Dec. speed is constant and when this parameter is correctly determined and proven during several observing sessions, it can be entered manually even if the calibration result is different due to backlash and various random events.

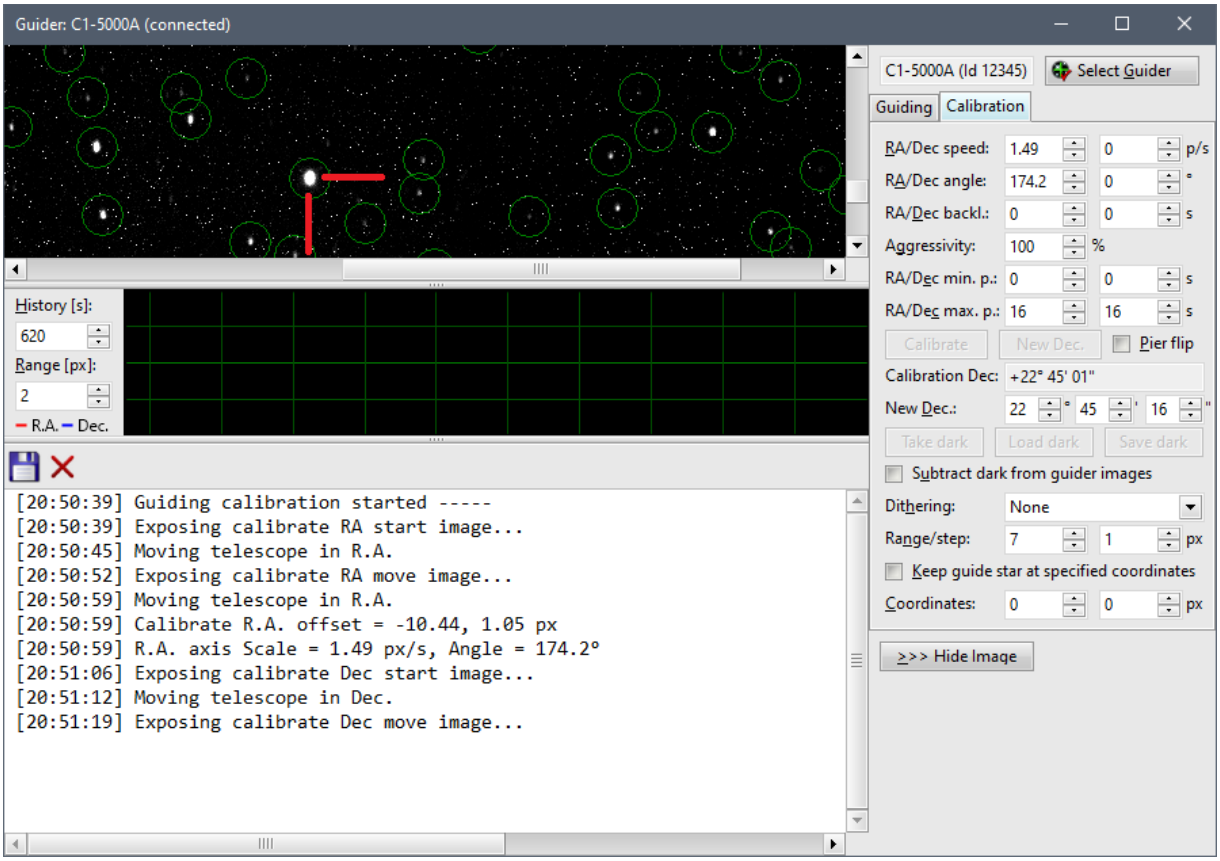
1. SIPS takes the first reference frame.



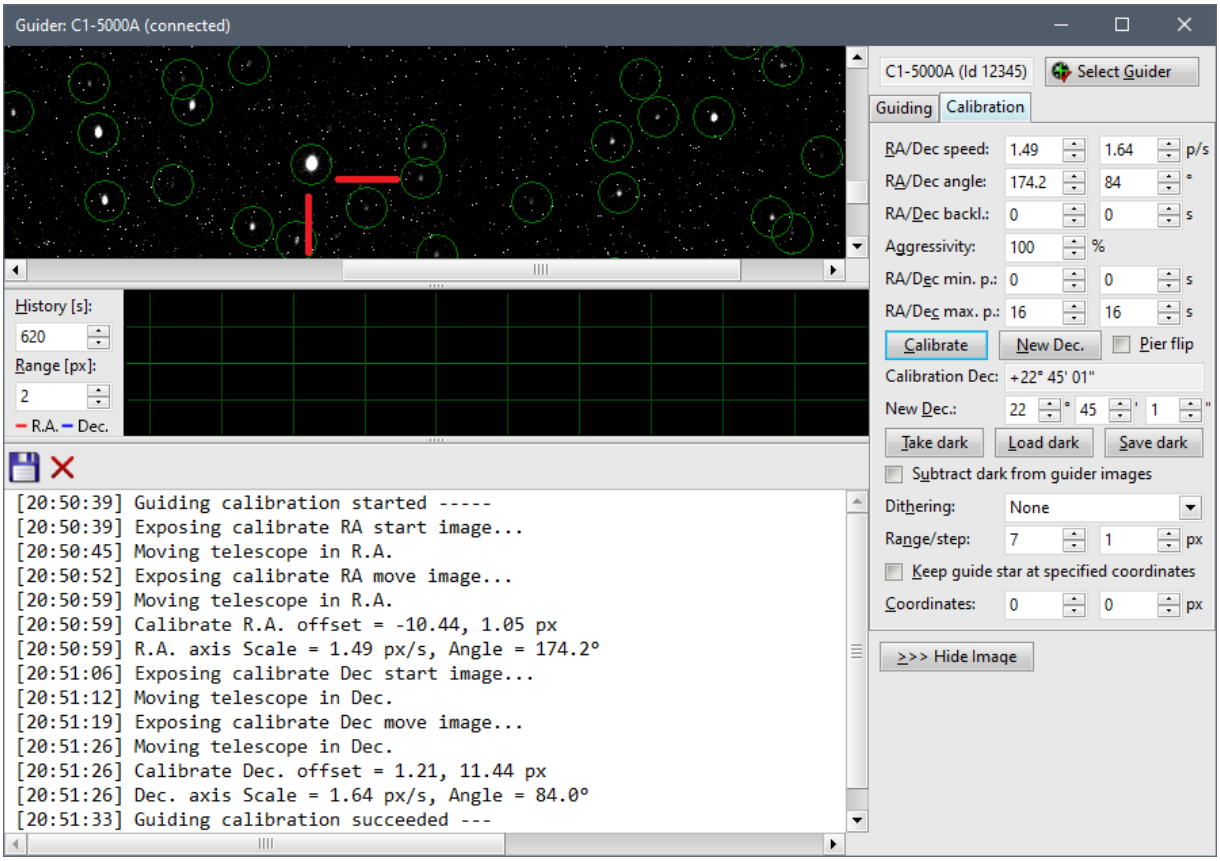
2. The mount is moved in R.A. for the period defined as **Calibration RA pulse length**.
3. SIPS then takes another frame and calculates the difference between star centroids on reference and on actual frame. The difference represents the mount reaction on the movement in Right Ascension. The **RA speed** and **RA angle** are then calculated.



4. The mount is returned to the original position by moving it for the same time in reverse direction and another reference frame is taken.



5. The mount is moved in Declination for the period defined as **Calibration Dec pulse length**.
6. SIPS then takes another frame and calculates the difference between centroids of stars on these two frames. The difference represents the mount reaction on the movement in Declination. The **Dec speed** and **Dec angle** are then calculated.



7. The software then moves the mount back to the original position.

Hint:

A very good indication of proper calibration is a difference between determined angles close to 90°.

Reusing calibration parameters on different fields

Telescopes on the German Equatorial Mounts typically cannot track over a meridian. It is necessary to flip the telescope tube relative to the mount pier within some (small) angle after crossing the meridian. Images are then rotated 180° relative to the orientation prior to tube swapping.

While it is just possible to re-do calibration after pier flop, the **Pier flip** checkbox just rotates the calibration vectors 180°, so it is no necessary to perform calibration again.

Also, moving the telescope to another field typically causes re-doing of the calibration, because the speed in R.A. increases when the new position is closer to the pole. However, the **Guiding tool** can store declination, at which the calibration was performed (if the telescope mount is capable to return proper declination, it is stored automatically, else the user can enter declination manually). Then a new **RA speed** parameter can be calculated after declination changes without a necessity to re-calibrate guiding.

Warning:

Because both R.A. and Dec. speed and angle are calculated independently, SIPS does not require any particular guiding camera orientation. Camera in any position on any telescope (mirroring the image or not) can be properly calibrated. However, for proper adjustment of RA speed parameters after moving to different location on the sky, R.A. and Dec. axes on the guiding camera images should correspond to actual coordinates on the sky.

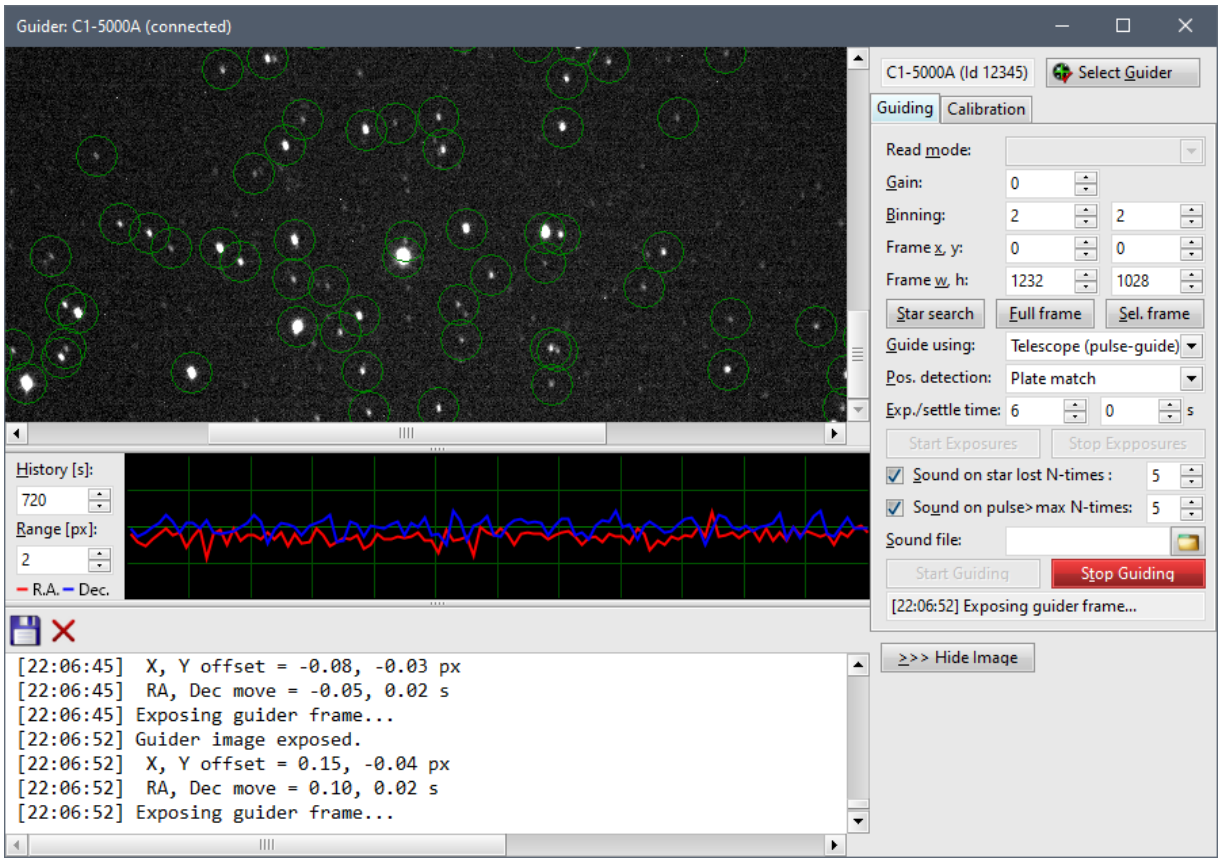
Regular Guiding, Charts and Logs

After successful finishing of the above steps, the **Start Guiding** button starts with first guiding frame, taken as reference one.

Then SIPS continues with periodic reading of the guiding camera, searching the star(s), calculating the difference, calculation the correction pulse lengths (based on the parameters determined during calibration) and invoking mount correction movements.

If the **Guiding camera** tool window is kept in the mode showing the image, charts and logs, it is possible to observe not only guider image, but also chart with history of guiding corrections as well as a text log showing history of guider actions.

Guiding corrections history chart shows both R.A. and Dec. offsets, expressed in guiding camera pixels. Two count boxes **History** and **Range** allow adjusting of the chart according the user's needs. It is useful to set History to the period of the mount worm gear (or to any arbitrary value if a friction mount is used). Range is naturally set according to mount tracking precision. Both parameters can be arbitrary adjusted to view either longer history or to “zoom into” some detail.



The pane with text log is a very useful tool for finding possible guiding problems. Log file saves history of calibration including the determined parameters, every frame read and offset determined from the frame as well as corresponding pulse lengths. If anything went wrong (guiding star is lost etc.), log keeps the record about it.

Hint:

The **Save Log File** button allows saving of the complete list into text file. This allows keeping of the record as well as sending it to some experienced user, which can help with finding source of guiding problems etc.

Guiding Reliability and Alarms

Number of things can go wrong while guiding—the mount can perform step movement due to some mechanical glitch or wind blast or clouds can cover the field of view to name a few. Failure of guiding automatically results into failure of imaging and this is why users need to be alarmed to either fix the problem or to finish the whole session.

As mentioned earlier, very basic precondition to successful guiding is reliable detection of guiding star. Star search may fail if:

- The Aperture is too small to cover entire star. If there is no contrast in brightness between background pixels and pixels belonging to star image, SIPS has no way how to distinguish the star. Focusing the guider telescope or enlarging of the Aperture can solve this problem.
- Star is too weak. Pixels belonging to very dim star may stay below the threshold limit (N-times the standard deviation of the Aperture area) and may not be recognized as star pixels. Using of such dim star close to detection threshold is not recommended for guiding either way, because guiding precision suffers significantly in such case. Prolonging the guider camera exposure time or choosing different (brighter) guide star is the only solution.

Another possibility of guiding star non-detection is cloud cover. SIPS can alarm the user if the guide star is lost. And because this event can be only temporary, the alarm may not be raised upon first occurrence of this event.

Check the **Sound on star loss N-times** checkbox and define the number of subsequent events to raise the alarm. Then SIPS starts to play the chosen sound file (wave file, MP3 file or any other sound file—pick up your favorite for alarm) if the **Guiding camera** tool reads the defined number of guider camera frames and cannot find guiding star on it.

The number of failures must be subsequent. If for instance the threshold is se to 5 occurrences and a star is lost 4-times, then it is found and again is lost 4-times, no alarm will be raised.

Hint:

The “guiding star lost” alarm is also a very good way how to detect clouds.

Sudden shift in motion due to some random event (mount gear damage or manufacturing fault, wind blow etc.) can occur and it is desirable to skip such frames, where the guiding star is way off the reference position, and not to try to correct the mount to this temporary spike. Unfortunately, there is no way how to distinguish offsets caused by temporary conditions, which should be skipped, from permanent offsets, which should be eliminated by commanding the mount to move star back to reference position.

As a compromise, SIPS introduces parameters R.A. and Dec. maximal pulse length. If either R.A. or Dec. (or both) calculated pulse length is longer that defined values (recognized star is too far from the reference position), entire frame is skipped. Note skipping of frame is logged in the log panel of the **Guiding camera** tool.

Similarly to losing a star, skipping a frame due to R.A. or Dec. guiding pulse exceeding defined limits can cause an alarm (a sound is played). And also in this case an alarm is raised only if this event repeats by defined number of times. E.g. radiation spike, causing false brightest star, occurs only once, so such frame is skipped and no alarm is raised.

Hint:

When the sound starts, it is possible to mute it by the <Esc> key or by clicking the 

tool in the SIPS main window **Settings** ribbon.

Dithering

Dithering is used mainly by astrophotographers and its main purpose is to spread hot pixels, possible bad columns and other artifacts around defined area of the acquired images instead of keeping them on constant position. This allows removing of artifacts during final stacking of individual exposures by some statistical method, rejecting extreme values (e.g. sigma clipping).

When dithering is activated, each single exposure of the main imaging camera is guided as usual. But when the imaging camera exposure finishes, SIPS calculates offset to the reference star, based on chosen method, and moves the position of the guiding star to newly calculated position. Then the next exposure of the main imaging camera begins. Moving of the desired guiding star position to different coordinates among imaging camera exposures results into shift of the images relative to main camera detector, which is desired from the reasons mentioned above.

Remark:

The **Guiding camera** tool needs to cooperate with the **Imaging camera** tool, because dithering changes guiding star location between each main imaging camera exposures. So, naturally dithering while guiding works only if the **Imagin camera** tool performs exposures.

Dithering can be activated using **Dithering** combo-box. Also, two more parameters **Range** and **Step**, expressed in guider camera pixels, define dithering area. These parameters delimit number of pixels, by which the offsets will be spread around the reference star position. **Range** defines half of the square dimension, surrounding the reference star (reference position will be moved up to +/-Range in both x and y axes). **Step** allows to increase the position increment from 1 pixel to larger distance.

Dithering can be activated using **Dithering** combo box. Default value is **None** (no dithering), but three more values are available:

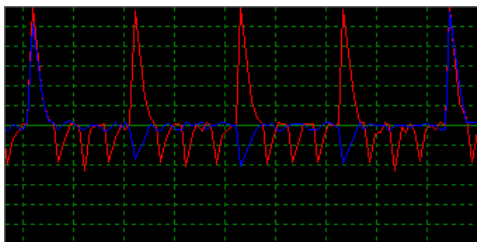
- **Rows then Columns**: reference position is moved around the square with +/-Range side. First position will be in the upper left corner, then the location will be moved by Step pixels to right and so on until x location will reach +Range offset. Then the position will be moved Step pixels down and back to left to begin the next line.

- **Columns then Rows**: reference position is moved around the square with +/-Range side. First position will be in the upper left corner, then the location will be moved by Step to bottom and so on until y location will reach +Range offset. Then the position will be moved Step pixels right and back to top to the begin next column.
- **Random**: location is chosen by pseudo-random generation of offsets in the +/-Range interval. Please note the Step parameter is ignored in this mode, any pixel location within the defined square may be chosen.

Hint:

Be careful with regular dithering modes (**Rows then Columns**, **Columns then Rows**) when using Series of exposures, repeating exposures through different filters. There is a possibility that dithering will move the guide star to the same location when exposing through the same filter. Because only images acquired through the same are stacked, the effect of dithering could be efficiently eliminated—mutual offset will be among images taken through different filters, not among multiple images taken through the same filter, which are later stacked. If you are not sure, use Random mode.

The Dithering in action, when the star is moved over a square with **Range** set to 3 pixels and **Step** set to 2 pixels, is very well illustrated on the **Guiding camera** tool graph pane:

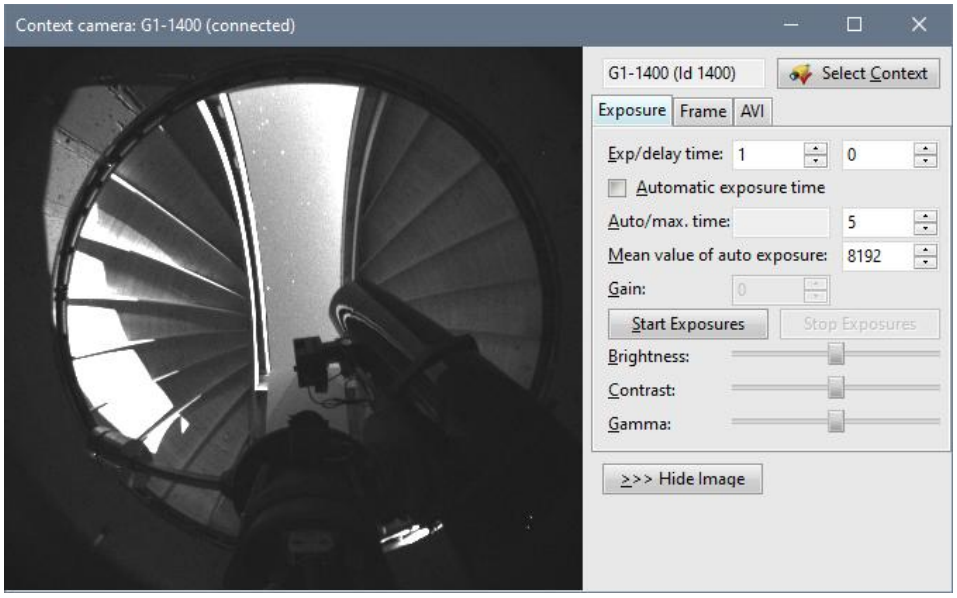


The corresponding log is:

```
[22:47:06] Exposing guider frame...
[22:47:12] Guider image exposed.
[22:47:12] X, Y offset = -6.10, -5.81 px
[22:47:12] RA, Dec move = -7.15, -6.66 s
[22:47:12] Moving star to next dithering position
[22:47:12] Guiding pulse limit inactive
[22:47:19] Exposing guider frame...
[22:47:24] Guider image exposed.
[22:47:24] X, Y offset = -1.22, -2.16 px
[22:47:24] RA, Dec move = -1.43, -2.46 s
[22:47:27] Exposing guider frame...
[22:47:32] Guider image exposed.
[22:47:32] X, Y offset = -0.41, -0.22 px
[22:47:32] RA, Dec move = -0.48, -0.26 s
[22:47:32] Exposing guider frame...
[22:47:38] Guider image exposed.
[22:47:38] X, Y offset = 1.86, -0.22 px
[22:47:38] RA, Dec move = 2.17, -0.23 s
[22:47:38] Moving star to next dithering position
[22:47:38] Guiding pulse limit inactive
[22:47:40] Exposing guider frame...
[22:47:45] Guider image exposed.
[22:47:45] X, Y offset = 0.27, -0.21 px
[22:47:45] RA, Dec move = 0.32, -0.24 s
[22:47:46] Exposing guider frame...
```

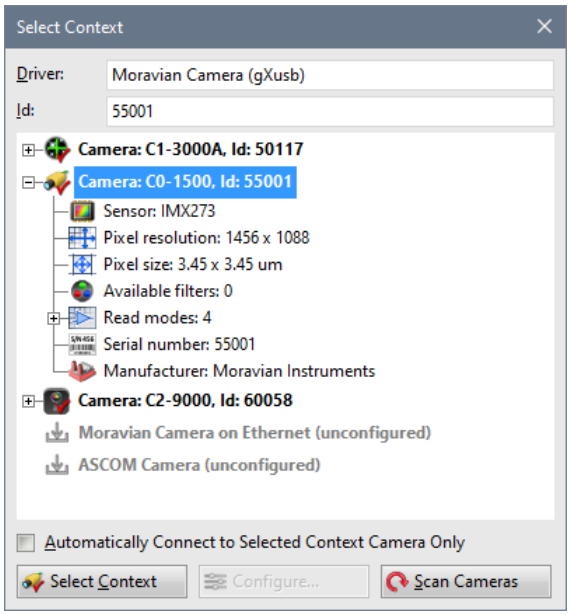
The resulting images, stacked without any mutual offsets (left) and properly aligned (right), clearly indicates mount movements during dithering:

Context camera



The **Context camera** tool shows an image of a camera, dedicated to providing an overview of the observatory.

Selection of the context camera works the same way as in the case of main imaging or guiding cameras. Context camera is selected using a Select Context button, which opens a pick list of all available cameras. And it is also possible to instruct SIPS to always work with one specific camera as a context one, despite it is not online when SIPS runs etc.



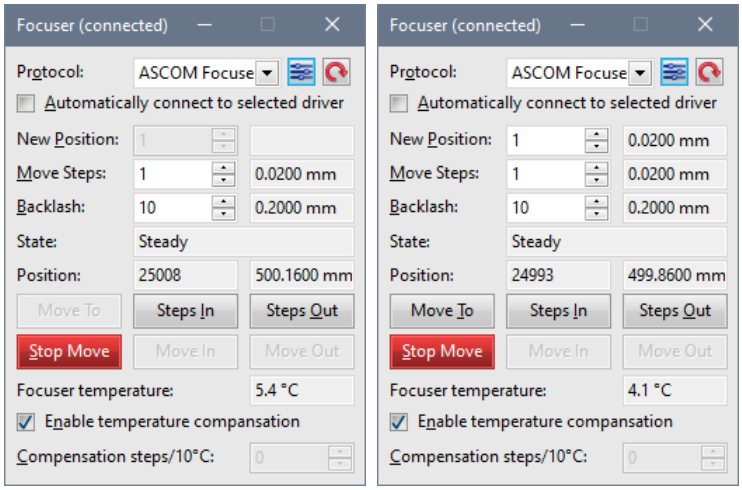
Context camera often provides images with dynamic range exceeding the 8-bit color depth of computer monitor. So, similarly to images taken with main imaging camera, also the image from context camera must be stretched when displayed on screen. The **Context camera** tool offers different stretching adjustment controls than the ones used to define stretching of normal images. Instead of directly manipulating low and high limits (see the Histogram and Stretch tool description for details), three sliders **Brightness**, **Contrast** and **Gamma** define the look of context camera images.

The **Context camera** tool is the only one in SIPS, capable to automatically adjust exposure time. Checking the **Automatic exposure time** option causes the exposure time is adjusted to achieve the **Mean value of auto exposure** of downloaded images. The **Auto** non-editable edit box shows the exposure time, estimated by the algorithm to get images with defined mean value. The **max. time** defines the upper limit of automatic exposures.

All other controls, defining **Read mode**, **Binning** and **Frame** works the same way as for guiding or main imaging camera.

The **Context camera** tool is capable to save images from context camera to AVI video file, for possible later inspection etc. Video controls are the same as in the case of **Stream** exposure type of the main imaging camera (see the **Imaging camera and Filter wheel** chapter).

Focuser



The **Focuser** tool allows controlling of motorized focuser. As with all devices controlled from SIPS, proper driver must be present to allow SIPS to control the focuser.

- A **Protocol** combo-box shows all available drivers.
- If the driver need configuration (for instance, ASCOM needs to select which ASCOM controlled focuser is to be used), the (Configure Focuser driver) button next to the combo-box opens the configuration dialog box.
- And the (Scan Focuser drivers) button next to the Configure button enumerates all connected drivers for the case the PnP focuser was connected when the SIPS was started.

The **Automatically connect to selected driver** checkbox function is the same as the similar option of other devices. It ensures SIPS will select the desired focuser driver even if it is not enumerated as a first one and also connects to the driver even if the plug-and-play device is connected to the PC after SIPS is run.

The focuser driver may work in two modes—absolute and relative, and the **Focuser** tool adjusts its GUI according to the mode of actually selected driver:

- In the **Relative** mode (the left image on the top of the page) the user commands the focuser to move relative to the current position. This means the focuser moves by the positive or negative value, defined in the **Move Steps** count-box. The “in” direction means closer to the tube, “out” direction means farther from the tube. Movement can be performed by clicking the **Steps In** and **Steps Out** buttons.
  The **Backlash** count-box allows definition of the number of steps, which will be added to the requested number of steps when the movement direction is changed to compensate for motorized focuser gear backlash.
- In the **Absolute** mode (the right image on the top of the page) the focuser maintains also absolute position and is able to refer it to the user. Also, the focuser drive is capable to move to absolute position defined in the **New Position** count-box using the **Move To** command button.
  Naturally, also relative movement can be performed using the **Steps In** and **Steps Out** command buttons, as new absolute position can be calculated from current **Position** and value of relative movement in **Move Steps**.

Some drivers allow to initiate continuous movement until the certain button is kept pressed (buttons **Move In** and **Move Out**). The ability to perform continuous movement is optional and not all drivers support this function. If the function is not supported by the selected river, respective buttons remain disabled.

If the driver provides information necessary to convert focuser steps to distance measured in millimeters, providing the diver provides the necessary information about the actual step size. The **Focuser** tool then shows also distance in millimeters right to the distance in steps. This helps to see how big the actual focusing step is reality.

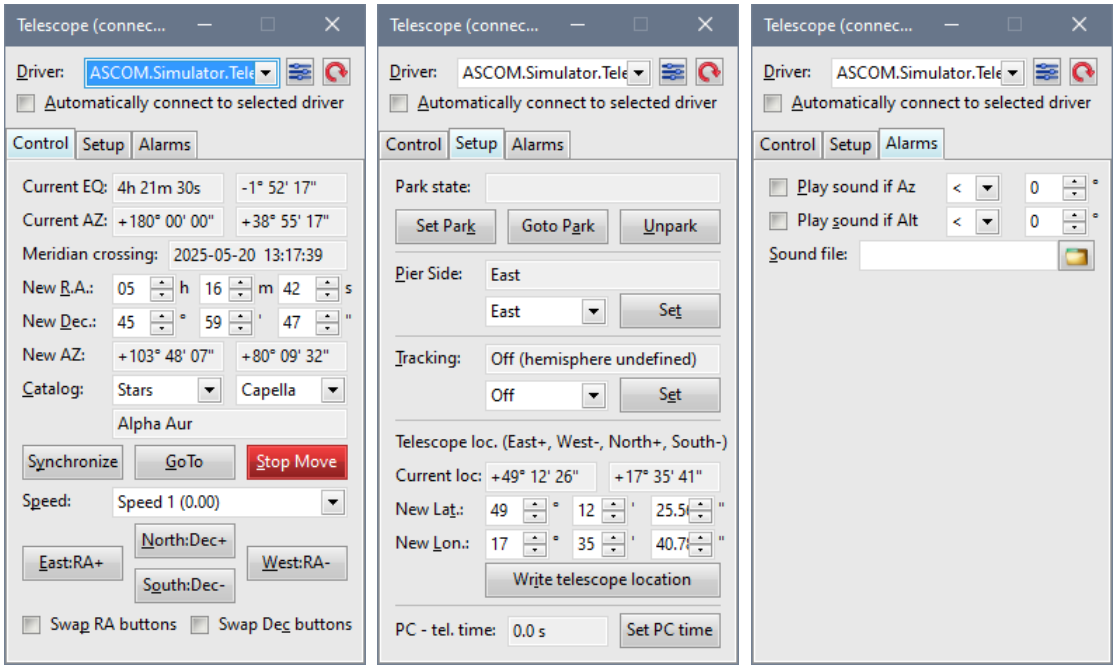
Some focusers may implement a capability to compensate for the focus plane movements caused by the thermal expansion of the telescope tube and/or optics (mirrors). If the focuser driver offers this functionality, the Focuser tool allow turning it on and off by the **Enable temperature compensation** checkbox.

The **Focuser** tool (and SIPS focuser drivers) also allows to define the amount of compensation in the **Compensation steps/10°C** count-box.

Remark:

The ASCOM standard do not implement this function and the compensation value must be defined elsewhere. So, if the ASCOM focuser driver is connected to SIPS, the **Compensation steps/10°C** count-box remains always disabled.

Telescope



The **Telescope** tool allows controlling of telescope mount, equipped with computer interface, from the SIPS environment.

- The **Driver** combo-box shows all available drivers.
- If the driver needs configuration (for instance, ASCOM needs to select which ASCOM controlled telescope mount is to be used), the (Configure Telescope driver) button next to the combo-box opens the configuration dialog box.
- And the (Scan Telescope drivers) button next to the Configure button enumerates all connected drivers for the case the PnP telescope was connected when the SIPS was started.

The **Automatically connect to selected driver** checkbox function is the same as the similar option of other devices. It ensures SIPS will select the desired telescope driver even if it is not enumerated as a first one and also connects to the driver even if the plug-and-play device is connected to the PC after SIPS is run.

The telescope is rather complex device, requiring many controls. So, the GUI is divided into three tabs, which allows to shrink the tool window size and to preserve precious screen space while observing.

Control tab

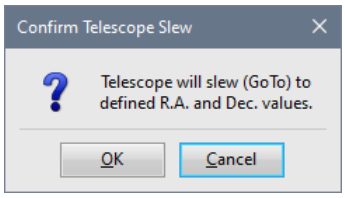
The **Current EQ** and **Current AZ** boxes show the equatorial and azimuthal coordinates, to which the telescope currently points.

Remark:

The Altitude/Azimuth coordinates are not read from the telescope controller, but they are calculated by SIPS. Converting Equatorial coordinates to Azimuthal ones requires evaluation of some goniometric functions and other calculations, which can be performed very quickly on a PC with huge computational power. Mount controllers on the other side are usually equipped with embedded microprocessors, very often without hardware support for floating point calculations and emulating floating point operation in software takes several orders of magnitude more time.

But converting Equatorial coordinates to Azimuthal ones requires knowledge of universal time and telescope location (geographical coordinates). If the time and/or location on the PC and the mount differ, then the Azimuthal coordinates will be calculated differently. So always make sure the PC and mount time as well as location is set to the same values. The **Telescope** tool helps with such synchronization, as described later.

The **New R.A**. and **New Dec.** count-boxes allow settings of new equatorial coordinates. These coordinates can be used to synchronize the telescope position (new position will become telescope current position—command button **Synchronize**) or to go to the new position (command button **GoTo**). Because accidental click on **GoTo** or **Synchronize** buttons can harm observing run, the **Telescope** tool always require confirmation this action was invoked intentionally:



Hint:

Sometimes coordinates are not provided in hours or degrees, minutes, and seconds, but in degrees in the form of decimal number (fraction of degrees). While conversion from decimal degrees to minutes and seconds is simple, it involves a few divisions and subtractions and majority of people cannot do such math without calculator or at last pen and paper.

This is why the equatorial coordinates controls are capable to perform "smart paste" operation. If a number with a fraction part (containing decimal point) is pasted to the **New Dec.** degrees count-box, SIPS interprets it as a declination expressed in decimal degrees. It transforms the value to whole degrees, minutes, and second and fills all three count-boxes accordingly.

In the case of right ascension, the value, containing decimal point, pasted to **New R.A.** hours count-box, is also interpreted as decimal degrees as it is customary to use degrees not hours also for right ascension when equatorial coordinates are expressed in the fraction form. So, the value is first divided by 15 to convert it to hours and only then it is transformed to whole hours, minutes, and second.

The **Telescope** tool offers the user a simple catalog function. The function of catalog is simple—every time the user chooses a **Catalog** entry, coordinates of the entry are copied to the **New R.A.** and **New Dec.** controls. It is then possible to **GoTo** or **Synchronize** the telescope with new coordinates.

Command button **Stop Move** halts any telescope movement (e.g. **GoTo** command).

The **Speed** combo-box allows choosing of the telescope speed used by the direction buttons below. The available speeds depend on the protocol used, so the combo-box is always filled with values reported by the telescope driver.

The four motion buttons allow direct moving of the telescope in R.A. and Dec. Both R.A. and Dec. directions can be swapped using the **Swap R.A. buttons** and **Swap Dec. buttons** checkboxes.

Hint:

While all telescope mounts are required to support basic command like reporting coordinates or go to new position, not all mounts support all functions described below. If some function is not supported, the controls allowing its control will remain disabled (grayed).

Setup tab

The **Park state** shows whether the telescope is currently parked or not. The command buttons **Set Park**, **Goto Park** and **Unpark** allow definition of parking position, moving the telescope to the defined position, and switching to parking state (typically when observing finishes) as well as unparking of the telescope (typically on observing start).

The **Pier Side** informs about the current side of the telescope tube relative to the German Equatorial Mount (GEM) cross. Associated combo-box and the **Set** command button allow swapping of the tube explicitly if desired.

Remark:

The pier side is important for German Equatorial Mounts only, which are limited in accessing of the same hemisphere as is the side of the tube. This means if the tube is east of the pier, western hemisphere can be accessed, but access to eastern hemisphere is limited, depending on the mount design and OTA dimensions. Opposite limitations are valid for western hemisphere.

If the mount is type is fork, no limitations apply and pier side is not defined.

The **Tracking** shows the current tracking state. Associated combo box and the **Set** button allow setting of tracking, depending on the used mount capabilities. Tracking can be set to lunar or solar beside the default sideral, but it can be also stopped.

The **Telescope loc.** define the geographical coordinates of the observing site. It is possible to write location, determined by various ways, e.g. using mobile device GPS, map application etc. Such location can be written to the mount controller (**Write to Telescope** button). The **Current loc.** boxes show which location is currently set in the mount controller.

The **New Lat.** and **New Lon.** Values are set to the location defined in the global **Observatory Setup** dialog box and updated each time this dialog box is used to define new observatory location. The **Observatory Setup** dialog may be also used to read location from connected GPS receiver, either standalone **GPS device** or the **GPS module** of the connected imaging camera.

Hint:

Also, the **New Lat.** and **New Lon.** count-boxes are capable to perform the "smart paste" operation. See the Hint below the **New R.A.** and **New Dec.** count-box description.

The **PC - tel. time** shows the current time difference between the PC clock and mount controller clock. Because PCs have wide range of possibilities how to set exact time (through Internet Time Protocol or connected GPS receiver, ...), the **Telescope** tool allows only one way time update, the **Set PC time** button writes the time on PC (considered the valid one) to the mount controller.

Alarms tab

The **Play sound** … check boxes allow playing sound alarms on two conditions. One condition is related to azimuth, another to altitude. It is possible to play alarm when a telescope coordinate is lower or greater than a predefined limit value. These alarms can be used e.g. to swap German Equatorial Mount if azimuth passes local meridian or to terminate the observation if the altitude of observed objects is lower that defined minimal altitude etc.

Creating of own catalogs

A default "catalog.ini" file, containing the Messier catalog and subsets of NGC and IC catalogs, is installed with SIPS. This is simple text file, so every user can modify it or create own catalog files.

Catalogs and objects, offered in the **Telescope** tool GUI, are read from files named "catalog.ini", which can be located at three folders:

1. "\Users\%current_user%\Documents\SIPS\ini\catalog.ini" is a catalog proprietary for currently logged user. There is no "catalog.ini" installed here, but users may create own catalog files here.
2. "\Users\Public\Documents\SIPS\ini\catalog.ini" is a catalog available for all users. The default "catalog.ini" file is installed here.
3. A "catalog.ini" file located on the same path as the "sips.exe" program itself. SIPS installation does not create any "catalog.ini" file in this folder.

If the "catalog.ini" is found in multiple folders, the **Telescope** tool merges individual catalogs contained in all found files.

Structure of "catalog.ini" file is as follows. Individual sections are used as individual catalogs, displayed in the first combo box. Section names are in square brackets in the file. Following lines represent individual catalog entries and they are displayed in the second combo-box.

Every catalog entry line consists of four strings in quotes, delimited by spaces, tabs, commas or any other delimiters.

```
[M]
"1" "05:34:30" "22:01:00" "Tau 8.4m Pl (Crab Nb)"
"2" "21:33:30" "-00:49:00" "Aqr 6.5m Gb"
"3" "13 42 12" "28 23 00" "CVn 6.4m Gb"
…

[NGC]
"1" "00h07m18s" "27d43m00s" "Peg Gx 12.9m"
"6" "00h09m30s" "33d19m00s" "And Gx 13m"
```

```
"14" "00 08 48" "15 49 00" "Peg Gx 12.1m"
…
```

The first string is an object name. The two following strings are R.A. and Dec. coordinates of the object. And the fourth string is an optional note, displayed in the **Telescope** tool below the **Catalog** boxes.

The equatorial coordinates are defined as three numbers (hours/degrees, minutes, and seconds), delimited by any non-numeric character, except + and - signs. The last value (seconds) can be entirely skipped if no such precision is needed. The example above shows different conventions in the Messier and NGC catalogs for demonstration purposes.

SIPS also accepts the catalog entry lines in the older format, used in SIPS version 3 and earlier.

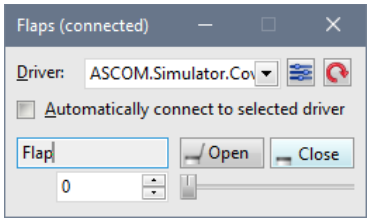
```
[M]
1 05 34 30 22 01 00 Tau 8.4m Pl (Crab Nb)
2 21 33 30 -00 49 00 Aqr 6.5m Gb
…
```

Individual strings, representing object name and coordinates, are not enclosed in quotes and there must be at last 7 strings delimited by spaces or tabs.

- The first string is interpreted as an object name.
- The following three strings represent object right ascension hours, minutes, and seconds.
- The next three strings represent declination degrees, minutes, and seconds.
- Any possible trailing string up to the end of line is used as a note.

Such format imposed some restrictions—the object name must not contain spaces and there must be exactly three strings for each equatorial coordinate.

Flaps



Especially remotely operated telescopes are often equipped with motorized, computer-controlled cover, which protect the optics against dust when the telescope is not operating.

- The **Driver** combo-box shows all available drivers.
- If the driver needs configuration (for instance, ASCOM needs to select which ASCOM controlled flap is to be used), the (Configure Flaps driver) button next to the **Driver** combo-box opens the configuration dialog box.
- And the (Scan Flaps drivers) button next to the Configure button enumerates all connected drivers for the case the PnP flaps driver was connected when the SIPS was started.

The **Automatically connect to selected driver** checkbox function is the same as the similar option of other devices. It ensures SIPS will select the desired flaps driver even if it is not enumerated as a first one and also connects to the driver even if the plug-and-play device is connected to the PC after SIPS is run.

Sometimes, the cover also offers a flat light source for acquiring calibration flat frames. If the particular flap does not offer brightness settings, the value count-box and slider, located below the flap name and **Open** and **Close** buttons, are omitted.

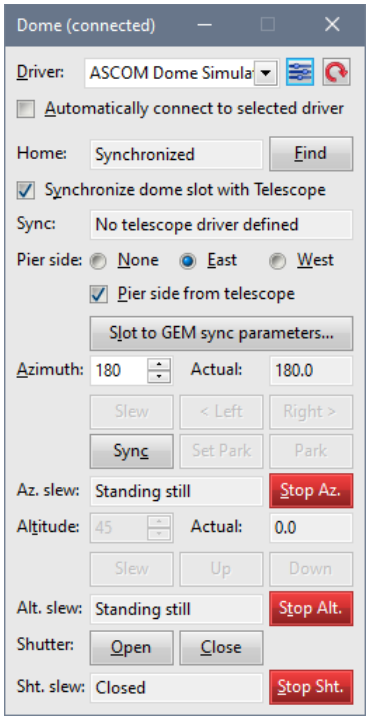
The **Flaps** tool provides a GUI allowing to open and close flap, as well as to set light source brightness, providing the calibration cover driver allows to do so.

The flap state is indicated by the **Open** and **Close** buttons highlight. If the flap is open, the **Open** button is highlighted and vice versa.

Remark:

The **Flaps** tool and the SIPS flaps driver API allows for multiple flaps to be controlled. Controls, related to individual flaps, are arranged below each other and the tool window height adopts according to actual number of flaps, supported by the selected driver.

Dome



Constructions of observatory housings are very different. If the housing is motorized and controlled with a computer capable to communicate with a host PC, it is possible to control the observatory dome or roll-off roof operations from the SIPS software package through the **Dome** tool.

- The **Driver** combo-box shows all available drivers.
- If the driver needs configuration (for instance, ASCOM needs to select which ASCOM controlled dome is to be used), the (Configure Dome driver) button next to the **Driver** combo-box opens the configuration dialog box.
- And the (Scan Dome drivers) button next to the Configure button enumerates all connected drivers for the case the PnP dome was connected when the SIPS was started.

The **Automatically connect to selected driver** checkbox function is the same as the similar option of other devices. It ensures SIPS will select the desired dome driver even if it is not enumerated as a first one and also connects to the driver even if the plug-and-play device is connected to the PC after SIPS is run.

Remark:

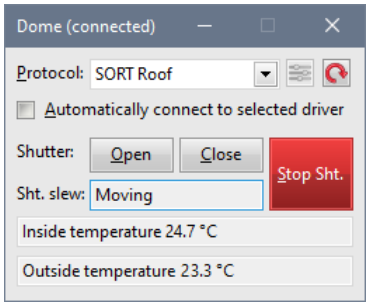
If the telescope is located under roll-off roof, its opening and closing are one-time actions, performed upon the beginning and ending of the observing session, so controlling of these actions from the whole observatory system brings little benefits. Exceptions are remote observatory sites, where it is not possible to open the roof manually, and the robotic setups performing fully automatic, unattended observations.

But in the case of rotating observatory dome it is necessary to keep the dome slit in proper azimuth to allow the telescope to see the sky. If the German Equatorial Mount is used, dome azimuth differs from telescope azimuth (only rarely is the dome slit so wide that the telescope has unobstructed view of the sky regardless of the tube position relative to telescope pier) and computation of proper dome azimuth is a bit complex. A PC running SIPS is capable to continuously calculate optimal azimuth and rotate the dome.

The Dome tool GUI adopts itself to the capabilities of the connected driver. The image above shows the **Dome** tool connected to the ASCOM dome driver simulator, which implements complete functionality contained in the driver interface:

- Dome **azimuth** (rotates the dome).
- Dome opening **altitude** (typically an observatory dome slit is opened from 0° to more than 90° and setting altitude is not necessary).
- **Shutter**, be it dome slit shutter or roll-of-roof.

So, for instance, if the dome slit opens as a whole and does not need altitude settings, the **Altitude** related controls will be omitted and the **Dome** tool window will be shrunk. Also, if the **Dome** tool is used to control a roll-of-roof observatory, the **Azimuth** and **Altitude** controls are not displayed and the tool window show the **Shutter** related controls only:



Remark:

The dome interface also allows measurement of up to four temperatures. If any temperature is returned by the dome driver, the SIPS **Dome** tool shows the measured values at the bottom of the tool window.

Rotating observatory dome must be synchronized to know the actual azimuth. It depends on the dome construction whether the rotation mechanism knows its absolute position (it is continuously in “synchronized” state and does not need to **Find** home) or it is necessary to **Find** some origin first and only then it is possible to ask the driver to rotate to certain azimuth.

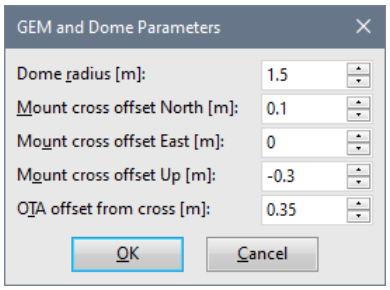
Synchronization of the dome slit with telescope position

The **Synchronize dome slit with Telescope** checkbox causes the dome is rotated to ensure that the telescope sees the sky through the slit. Of course, this function requires the rotating dome is properly synchronized as well as the telescope driver is working (the driver is installed, selected and telescope online). The **Pier side** radio-buttons affect how the dome azimuth is calculated:

- If **No offset** is selected, dome azimuth is simply set to the same value as is the telescope azimuth. This option is suitable for fork mounted telescopes.
- If the GEM is used, the telescope tube is always on the side of the dome (mount) center—on the eastern side when the telescope looks to the western hemisphere or on the western side if the telescope looks to eastern hemisphere. Only rarely is the dome slit so wide that it does not matter if the tube is off the dome center. So, it is almost every time necessary to slightly adjust dome slit azimuth, so the telescope is not obstructed by the dome itself.

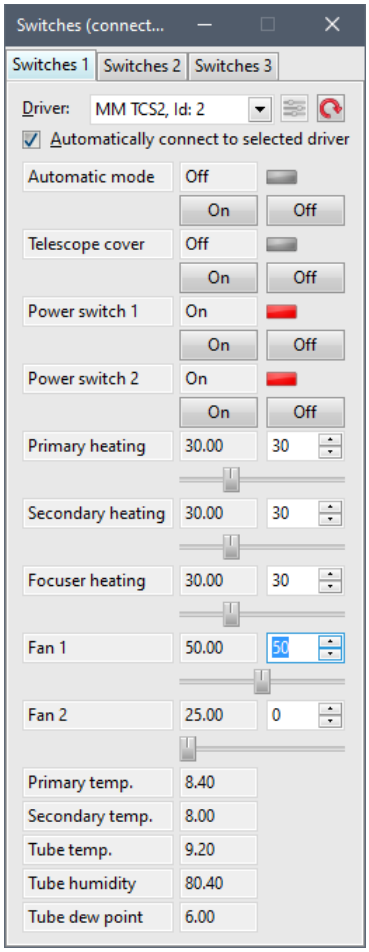
To be able to calculate optimal dome slit position, the **Dome** tool needs to know whether the tube is one the eastern or western side of the pier. This can be selected manually (**East** and **West** radio-buttons) or if the telescope driver provides such information, the **Dome** tool can read it from the telescope (**Pier side from telescope** checkbox). Let us note not many telescope controllers provide such information.

The actual azimuth correction is strongly non-linear function and it cannot be easily approximated. So, the SIPS calculates the exact intersection of the telescope tube axis with the dome. To be able to perform such calculation, several dimensions of the dome and GEM must be entered:



These parameters allow calculation of the coordinates of cross-section tube and mount axes.

Switches



An observatory is often a more complex set of instruments than just a telescope and camera. Some devices are supported in SIPS through the specialized drivers and associated GUI tool windows (e.g., focuser or observatory dome), others need not special treatment beside turning on and off (dew heaters, telescope fans, calibration light sources, etc.). Such devices are typically attached to some remotely controllable switches, possibly allowing also setting of analog value (e.g., the voltage). The **Switches** tool in SIPS then allows controlling of such array of individual switches.

- The **Driver** combo-box shows all available drivers.
- If the driver needs configuration (for instance, ASCOM needs to select which ASCOM controlled switches are to be used), the (Configure Switches driver) button next to the **Driver** combo-box opens the configuration dialog box.
- And the (Scan Switches drivers) button next to the Configure button enumerates all connected drivers for the case the PnP switches driver was connected when the SIPS was started.

The **Automatically connect to selected driver** checkbox function is the same like the similar option of other devices. It ensures SIPS will select the desired switches driver even if it is not enumerated as a first one and also connects to the driver even if the plug-and-play device is connected to the PC after SIPS is run.

The **Switches** tool window allows selection of up to three different drivers simultaneously. Each driver is handled independently in separate tabs.

The **Switches** tool is designed to show controls corresponding up to ten switches. Controls, related to individual switches, are arranged below each other and the tool window height adopts according to actual number of switches, supported by the selected driver.

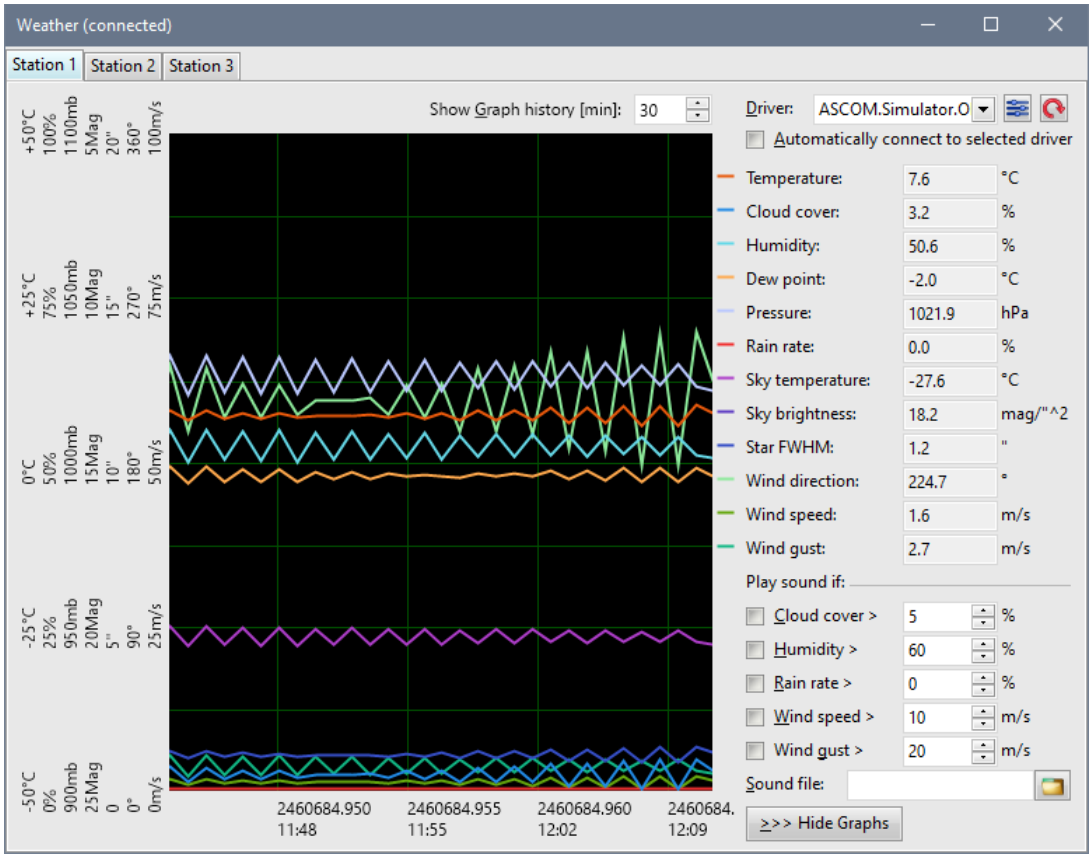
The “switch” can be either just on/off switch or a more complex device allowing setting of output value in defined range. For instance, a fan controller or dew heater controller may allow to regulate fan speed or heating power using output voltage.

- Binary switches are controlled by the **On** and **Off** buttons. Actual switch state is indicated by an icon, either glowing red to indicate “on” state or just grey for “off” state.
- Analog switches show current output value as a number, and new value can be set either by count-box or slider.

Remark:

The switches driver API allows also for read-only “switches”, only displaying current value, be it on/off state or analog value. Such switches then lack GUI controls allowing to modify the state.

Weather station



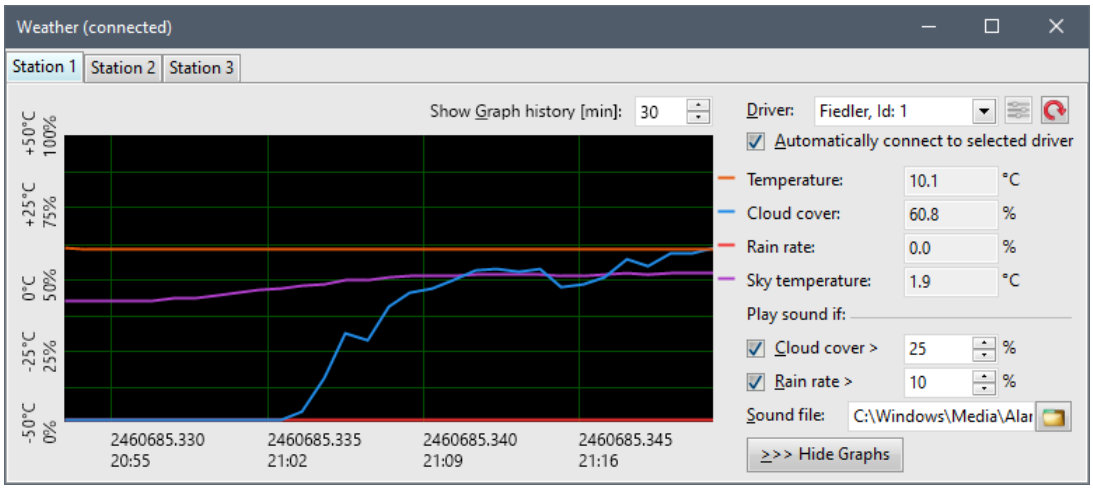
The **Weather station** tool shows a set of weather and sky related values.

- The **Driver** combo-box shows all available weather station drivers.
- If the driver needs configuration (for instance, ASCOM needs to select which ASCOM controlled weather station is to be used), the  (Configure Weather Station driver) button next to the **Driver** combo-box opens the configuration dialog box.
- The  (Scan Weather Station drivers) button next to the **Driver** combo-box enumerates all connected drivers for the case the PnP weather station was connected when the SIPS was started.

The **Automatically connect to selected driver** checkbox function is the same like the similar option of other devices. It ensures SIPS will select the desired weather station driver even if it is not enumerated as a first one and also connects to the driver even if the plug-and-play device is connected to the PC after SIPS is run.

The **Weather station** tool window allows selection of up to three different drivers simultaneously. Each driver is handled independently in separate tabs. There are no limitations on values measured by selected drivers, all drivers may measure same values, e.g. on different locations, or different drivers may measure different values (for instance, star FWHM is likely measured by different device/driver than air temperature and humidity).

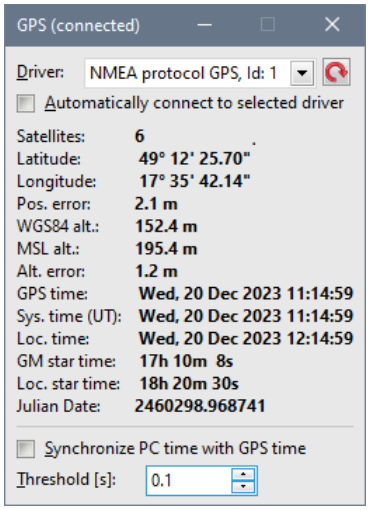
Which values are actually displayed depends on the used driver, as different drivers may provide different set of measured values. The image above shows the **Weather station** tool window with ASCOM Observing conditions simulator driver, providing artificially generated data on all available channels. This is why the window shows all values as well as all history graphs. In reality, the majority of weather station devices and their drivers provide only a subset of all possible channels. The **Weather station** tool window then shows only a reduced number of values and graphs. Also, the sound alarm checkboxes only for supported channels are provided.



The graph pane on the left can be shown or hidden using the **Show Graphs** and **Hide Graphs** buttons on the bottom of the tool window. The **Show Graph history** count-box, located above the graph pane, determines how long history (in minutes) is displayed. The longest recorded history is 1440 minutes (24 hours).

The alarm section, below the list of displayed values, allows playing of selected sound file when some 5 defined values exceed specified limit. The **Sound file** can be any .WAV file as well as any other file, which can be played using the operating system codecs (e.g. MP3). So, the observer can be informed when the cloud cover, humidity, rain rate, wind speed or wind gusts exceed acceptable limit.

GPS



The GPS (Global Positioning System) network of orbiting satellites provide geographic coordinates (longitude, latitude, and altitude) as well as precise time to any client device, capable to receive and process signals broadcasted by these satellites. This information is important for scientific astronomical observations.

The GPS tool allows handling of GPS receivers from the SIPS environment.

- The **Driver** combo-box shows all available GPS receiver drivers.
- The (Scan GPS drivers) button next to the **Driver** combo-box enumerates all connected drivers for the case the GPS was connected when the SIPS was started.

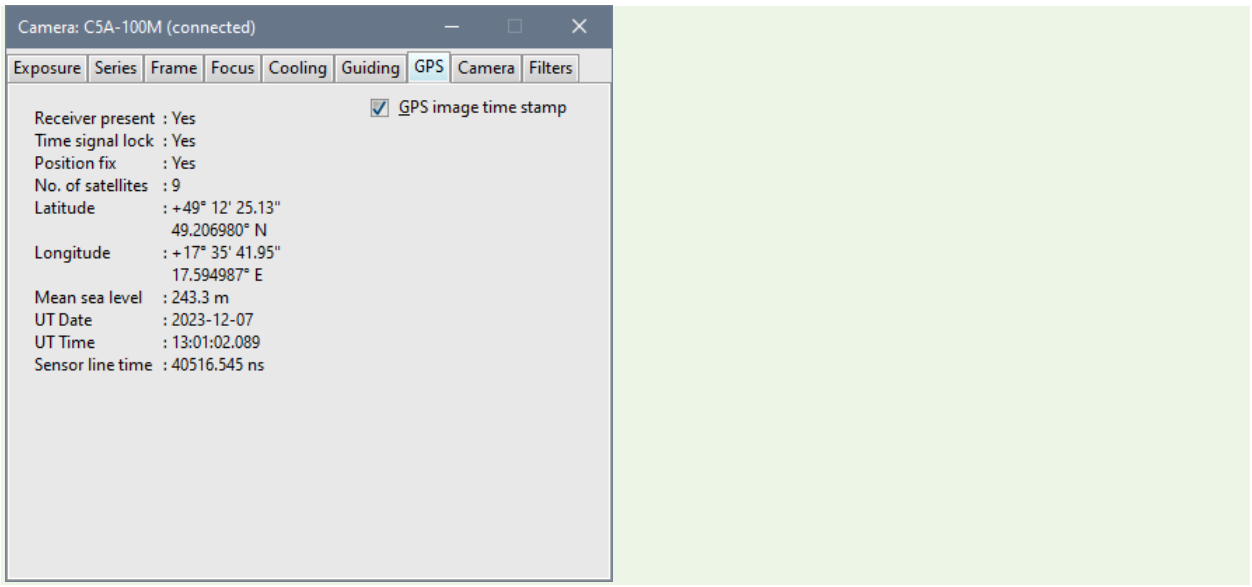
While every PC provides information about real time, the absolute precision of this time varies. Some computers can keep the difference between their clocks and the real time in the order of seconds for a month, but on the other side, differences of many seconds per day is not an exception.

All modern PCs, connected to the Internet, synchronize real time using the Network Time Protocol (NTP). The precision of the real time is the often better than 1 second, which is enough for vast majority of scientific observations.

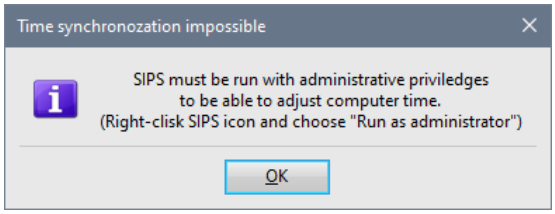
Hint:

Modern astronomical cameras can be equipped with own GPS receiver. Because of a tight integration of the receiver with camera hardware, the precision of exposure timing information can achieve an order of microseconds. Such cameras are intended for precise astrometry of fast-moving targets (Earth artificial satellites etc.).

Also, the **Imaging camera and Filter wheel** tool displays geographic location and time read from the camera GPS receiver.



The **GPS** tool allows synchronization of computer time with GPS-provided time when the difference exceeds defined limit (**Threshold** count-box). However, modern Windows operating systems prohibit applications from updating the system time until they run in the administrative mode. From historical reasons, it is not enough to be logged as a user with Administrator rights (member of Administrators system group), it is necessary to explicitly mark SIPS "Run as Administrator" in the shortcut properties. It is also possible to right click the SIPS icon and choose the "Run as Administrator" item from the pop-up menu. If the user tries to check **Synchronize PC time with GPS time** checkbox and administrator rights to alter system time are not granted to SIPS process, user is notified:

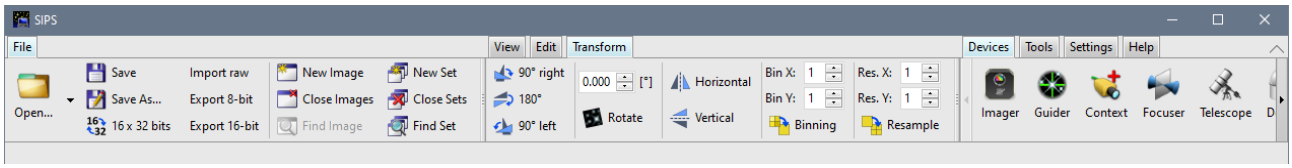


The **GPS** tool displays all information obtained from the GPS device (number of satellites above horizon, geographic coordinates, altitude, coordinates precision, etc.). It also calculates some other time information (Julian date, star time, etc.).

This tool can be also used without any GPS device connected. It then displays local and universal time, Julian date, and Greenwich mean star time. Local star time cannot be calculated without the knowledge of the current longitude.

Appendix A: SIPS user interface

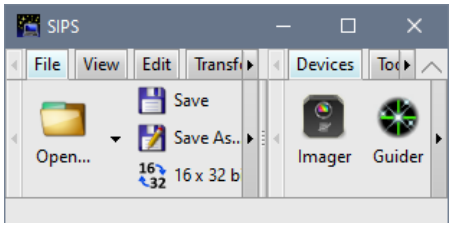
Beginning with version 4, SIPS abandoned classical and commonly known user interface based on menus, with a few most frequently used commands offered as icon-buttons, arranged in the underlying tool-bar. The new user interface is based on **ribbons**, containing both command buttons and basic controls (count-boxes, combo-boxes, …), necessary to perform respective commands and actions.



The basic idea of the new interface is to offer the user as many actions as possible directly, without the need to search through nested menus and to reduce the need to open dialog windows. However, the common implementation of the ribbon interface, introduced in other programs, is limited to a single ribbon visible at once. So, choosing a command button, located in a ribbon not currently visible, requires to show particular ribbon first with a click on its tab and only then it is possible to click respective command button. SIPS ribbons are on the other side designed to be highly configurable—multiple ribbons can be visible at once, ribbons can be grouped or located around all four sides of the SIPS main window etc.

Command ribbons are arranged in **groups** and each group is located in a **container**. Container can contain just single group of ribbons or can be divided into more groups. Each group can contain one or multiple ribbons.

If the particular ribbon does not fit the given space, it can be rolled with arrows, which appear on the left and right sides (or top and bottom sides, if the container is oriented vertically). The same applies if there is so many tabs in a single group that it is impossible to display them all.



Hint:

Beside the rolling with arrows, ribbons as well as tabs can be rolled with mouse wheel. That is usually much faster and much more comfortable.

The system works with two types of ribbon containers:

- Containers, which are a part of the SIPS main window. These ribbon containers are of four types according to the window edge to which they are snapped—**top**, **bottom**, **left** and **right**.
  Tabs, allowing choosing of ribbons located in one group, are oriented according to container location.
- Containers can be located separate pop-up windows, floating above the main SIPS window.

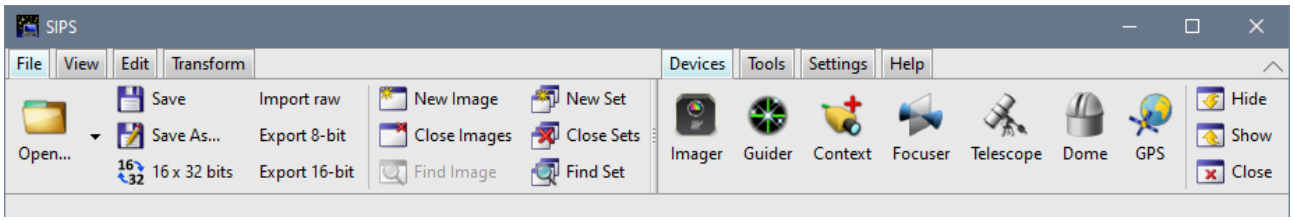
Ribbon arrangement

The location of ribbons can be freely adjusted by grabbing appropriate tab with mouse (click on the tab with the left mouse button and hold) and dragging. The system hints the position, where the dragged ribbon will be located using semi-transparent rectangles. This way, it is possible to change the order of ribbons in a group, move them between groups or into newly created group, move it between containers etc.

Hint:

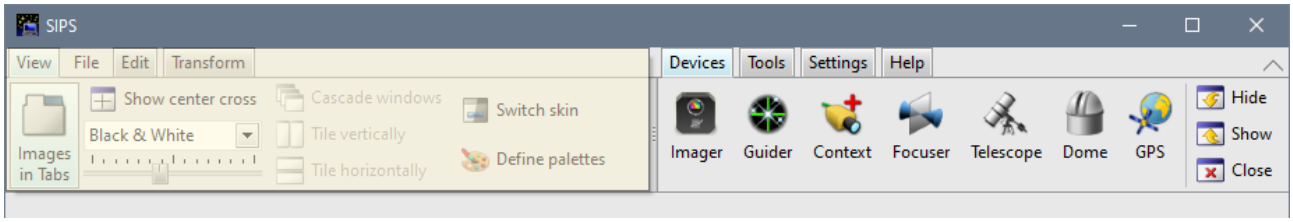
The arrangement (but also ribbon visibility etc.) is saved when SIPS is closed and restored again when SIPS is started.

Let us show the basic possibilities for handling of individual ribbons. Let us start with two groups of ribbons in the top container.

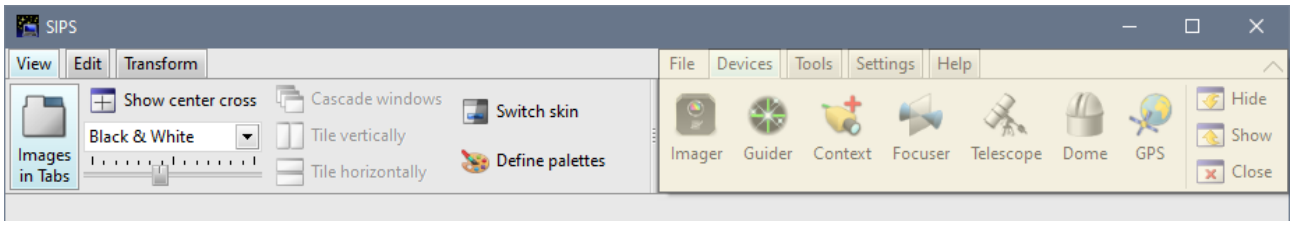


By dragging the **File** tab, we can:

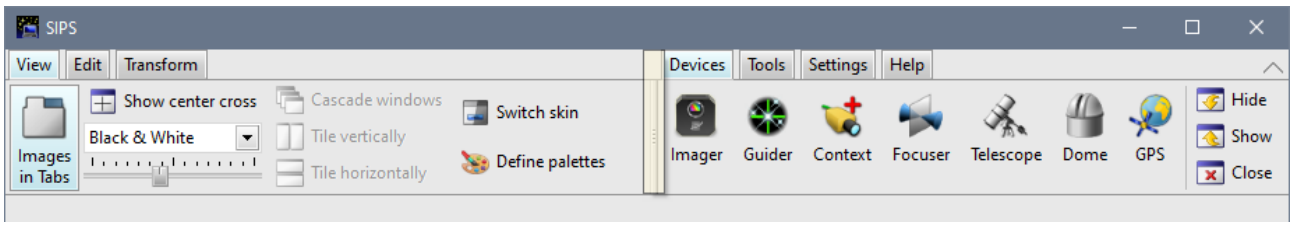
Change the order of ribbons in the left group—the **File** ribbon is now the second on in the first group, after the **View** ribbon:



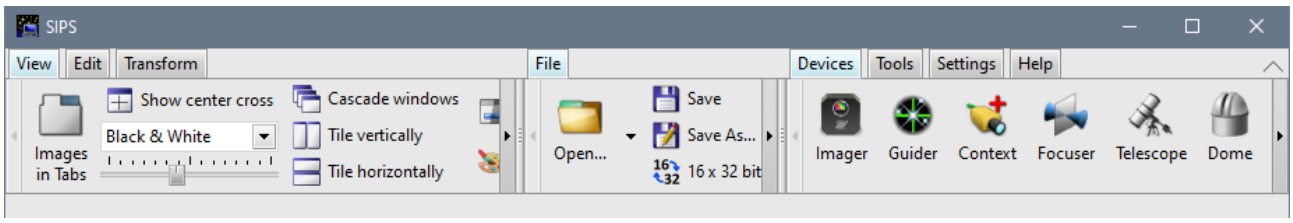
Drag the ribbon into another existing group—the **File** ribbon is moved to another group of ribbons:



Create a new group for the dragged ribbon by dragging it into the space between groups:

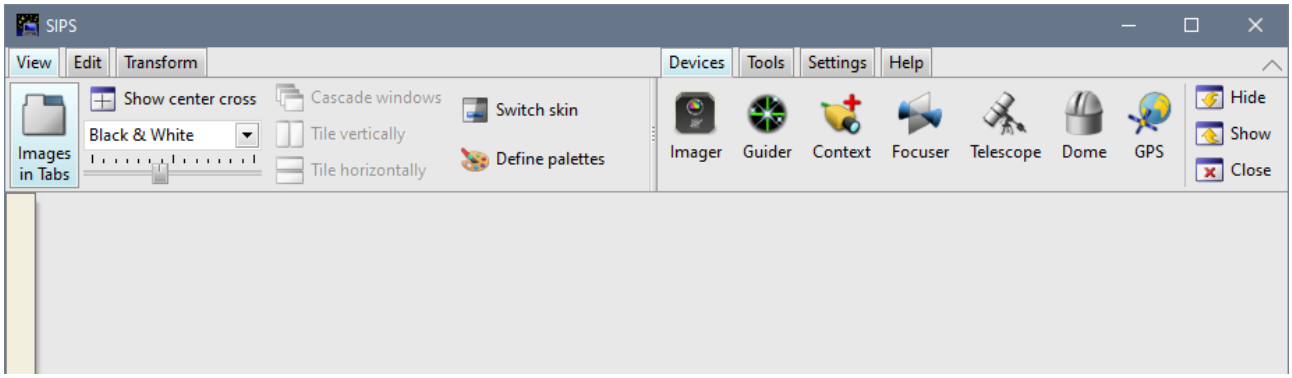


The **File** ribbon will appear in the newly created group between the existing ones:

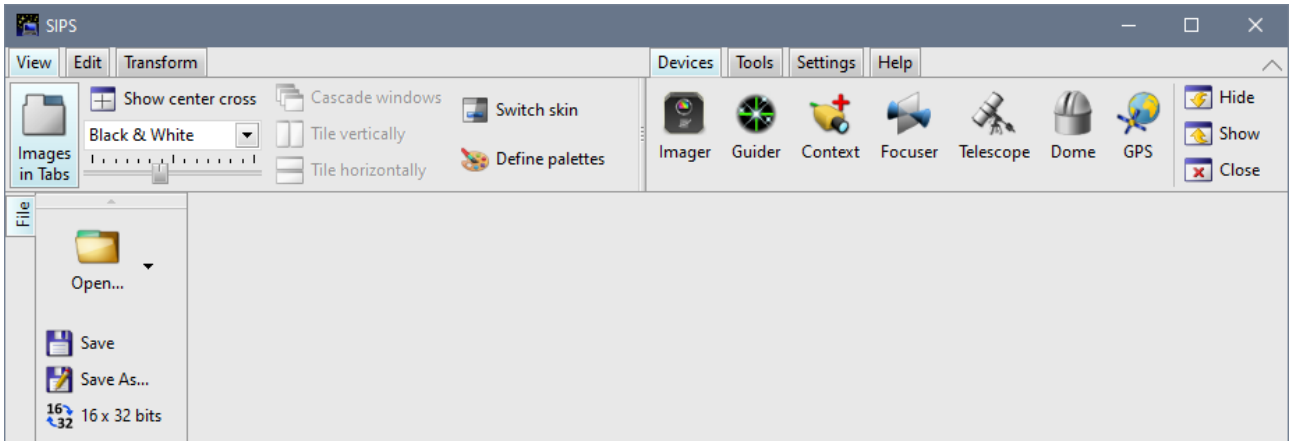


Let us note it is possible to create a new group also on the left side, before the current first group and also on the right side, after the current second group.

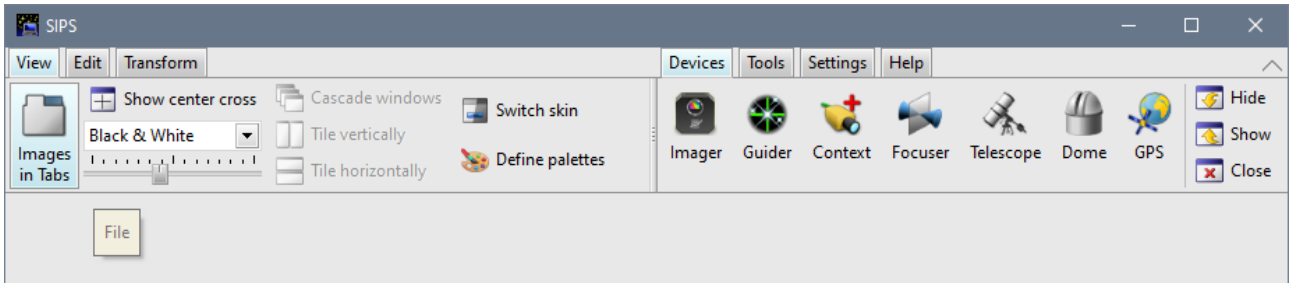
It is possible to create a completely new container for the dragged ribbon:



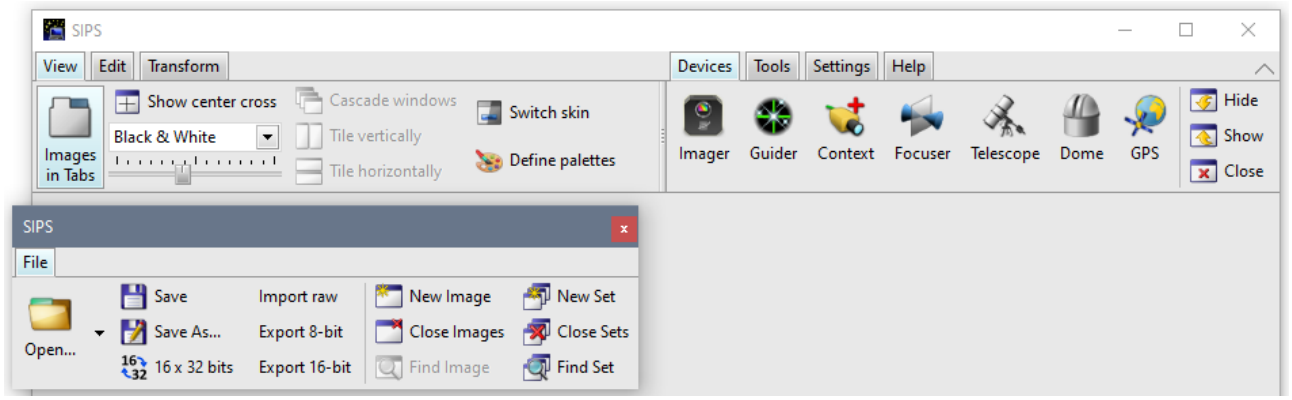
The **File** ribbon is then moved to the newly created **left container** with just one group. Note the individual tools are re-arranged to reflect different container orientation:



If the ribbon is dragged outside all existing containers or fixed container place-holders, SIPS will create an individual floating window for it.



SIPS with **File** ribbon in a stand-alone pop-up window:



Remark:

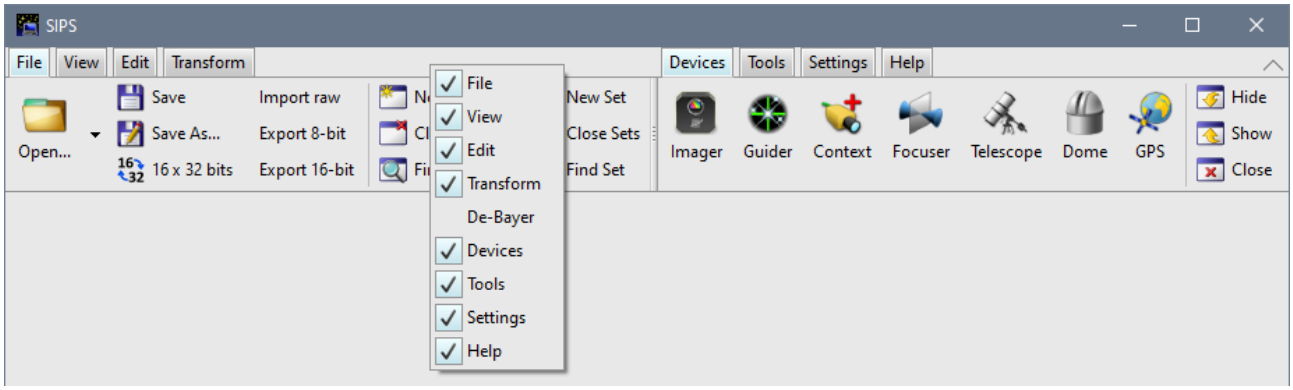
Pop-up ribbon windows can accommodate multiple ribbons, but always contain only one group. If multiple ribbons in pop-up window must be visible at once, it is necessary to create new window for each ribbon.

The pop-up ribbon window orientation (horizontal or vertical) is defined by the orientation of container, from which the ribbon was dragged. Dragging a ribbon from vertical (left or right) container creates also vertically oriented pop-up ribbon window.

Showing and hiding ribbons

Even though the set of ribbons and contained command buttons and other controls is firmly given, it is possible not only to rearrange individual ribbons, but also to hide and show ribbons according to user's preferences.

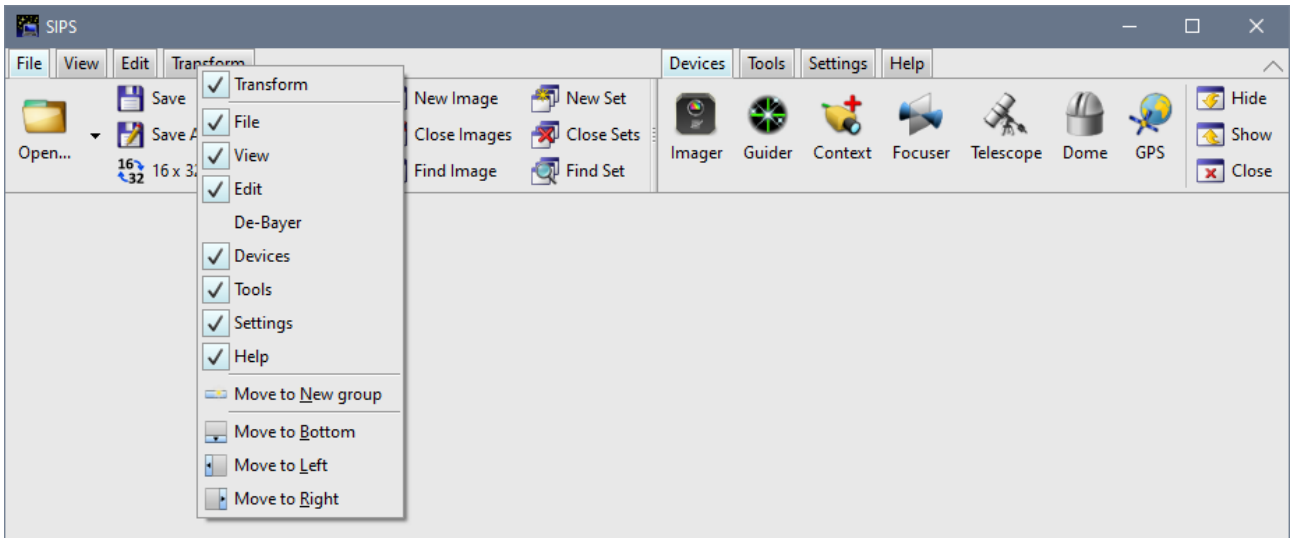
Showing and hiding of individual ribbons is performed through the context menu, which can be opened by clicking right mouse button above the tabs space in any container. This menu indicates, which ribbons are opened and which are hidden. If a ribbon is hidden, it is possible to show it within a group of a container, on which the menu was opened.



Hint:

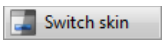
The ribbon containing tools for De-Bayer processing of images from single-shot-color cameras is a good candidate for hidden ribbon when the SIPS is used to process images from monochrome cameras only.

If the menu is opened by right-clicking not above blank space, but right above a tab, the tab is shown as the first above the separating line. This feature is intended for quick hiding of selected ribbon.

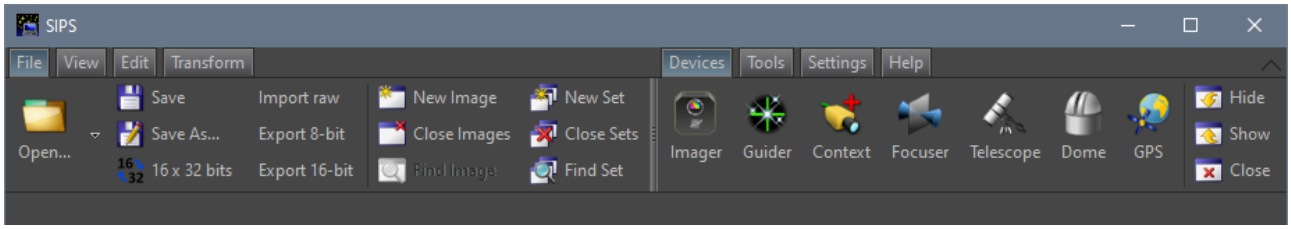


If a menu is opened above a certain tab, other actions, like moving into a new group (this is possible also by mouse dragging, see previous sub-chapter), moving into a newly created container (if there is an empty space for such container in the main window) are also offered.

GUI color skins

SIPS allows user to switch between light and dark GUI colors using the 

tool in the View ribbon.



The dark mode may be used for instance when SIPS is used to control observing at night etc.

> Remark:
>
> Some parts of the GUI are not controlled by the application, but by the operating system itself (for instance the widow title bars). Such GUI components are not affected by SIPS skin color.

Appendix B: Device drivers supplied with SIPS

Numerous devices are supported in SIPS natively. Depending on the interface between the device and the host PC, drivers can operate several ways:

- Plug-and-play drivers (typically USB connected devices) ensure the controlled devices are useable immediately after they are plugged into PC without any configuration.

  Remark:

  The exception from this rule are devices using USB only to tunnel legacy COM interface traffic, because the RS-232C serial lines are often physically missing from modern PCs—such drivers need to be configured like the legacy drivers.

- Some drivers need to be configured using a configuration dialog box, opened by the **Configure** button in the respective SIPS tool window. Typical examples are drivers handling multiple devices through some general interface (for instance, ASCOM drivers need to select which ASCOM-connected device is to be used) or drivers utilizing non-plug-and-play connection media (for instance, cameras connected through the Moravian Camera Ethernet Adapter need to define IP address of the adapter).
- Drivers using legacy communication lines (typically serial COM lines, regardless if the physical layer is hardware RS-232C, RS-485, or COM line tunneled through USB connection) need a configuration file. Configuration files are text files, following .INI format rules. So, they can be modified using for instance Windows **notepad** or any other text editor.

Hint:

Even some plug-and-play driver may use .INI configuration file to define some rarely used configuration options. For instance, Moravian Camera drivers allow setting parameters like overscan region read or binning options (hardware or software, average or sum, …) in the .INI file. See respective driver description for available parameters.

SIPS device drivers, relying on .INI configuration files, use the .INI files, named as the driver DLL, only with different extension. So, for instance, the “nmea.dll” GPS driver use the “nmea.ini” configuration file. These configuration files are searched sequentially in three folders in the following order:

1. User specific folder “Documents\SIPS\ini\”
2. Shared public folder “\Users\Public\Documents\SIPS\ini\”
3. A folder, where a driver .DLL file is located. This typically means “\Program Files\Moravian Instruments\SIPS4 EN 64-bit”, or possibly “\Program Files (x86)\Moravian Instruments\SIPS4 EN 32-bit”

Warning:

Windows Explorer shows the “\Users\Public\Documents\” folder as “\Users\Public\Public Documents\”, probably in a (failed?) attempt to make things easier to find by general users?

If the configuration file is not found in the first folder, each driver tests if the file exists in the second folder and only if still not found, the third option is tested.

Warning:

Device drivers, included in the previous SIPS versions, tried to load configuration files only from the 3rd path (the location of the driver DLL itself). But SIPS v4 installation process copies the default configuration files to both 2nd path (public documents) and 3rd path (location of SIPS EXE and DLL files) as the fallback option. So, if the configuration file on the 3rd path is edited, it is in fact not used, because a copy on the 2nd path is found and thus used.

So, if for instance a COM port number is modified in “\Program Files\Moravian Instruments\SIPS4 EN 64-bit\nmea.ini”, file, the NMEA driver still cannot connect to the GPS because a configuration file in “\Users\Public\Documents\SIPS\ini\nmea.ini” is found and thus used.

The above-mentioned hierarchy of configuration file search order offers a very high flexibility of handling various configuration, especially if multiple users want the device behave differently.

Typically, configuration files in the public documents are modified. As the public document folder is shared among all users of the particular PC, such change affects all users. This is a desired behavior, as for instance a weather station device, connected to the PC, uses the same COM port regardless of which user is logged in and uses the device.

But in certain cases, different users may want to use different configuration. Say, for example, one user of the C5 camera wants to average binned pixels and respect saturation, because he uses the camera for photometry. Another user uses the camera for aesthetic photography and prefers adding pixels when binning and ignore saturation. Every user then can create a “cxusb.ini” file in own “Documents\SIPS\ini” folder and the camera driver will use it in precedence.

Remark:

While the SIPS v4 installation procedure creates the default configuration files in the Program Files and Public Documents folders, files in the user-specific Documents folder are not created. If they are to be used, they must be copied there by the user.

Drivers using legacy COM serial interface

Numerous devices, supported by SIPS, still use legacy serial line interface. The RS-232C hardware port if often missing from modern PCs and thus the USB-to-RS232C (or USB-to-RS485 etc.) converters must be used, but such converters use software layers, emulating legacy COM port. So, from the software point of view, handling of such devices is the same regardless if they are connected to physical COM port or to virtual COM port, tunneled through USB.

COM ports can communicate at wide range of speeds (typically from 2.4 kbps up to 115.2 kbps) and using different data formats (with or without parity bits, with one or two stop-bits). As COM port is not plug-and-play, there are no means available for the driver to discover these parameters, it is necessary to provide them to the driver to work properly. These parameters are typically defined in the **[comm]** section on the driver configuration file, for example:

```
[comm]
port = COM1
baudrate = 9600
parity = none
databits = 8
stopbits = 1
timeout = 1000
```

The **port** parameter identifies the COM port number, to which the device is connected.

The **baudrate** parameter defines the speed of the line, used by the device.

The **parity** parameter defines if the transferred bytes are extended with parity bit. The possible values are **none** for no parity bit and **even** or **odd** for respective parity bit.

The **databits** parameter defines number of bits in each transferred byte.

The **stopbits** parameter defines if each byte is followed by one or two stop bits. The value of this parameter can be either number (**1** or **2**) or keyword (**one** or **two**).

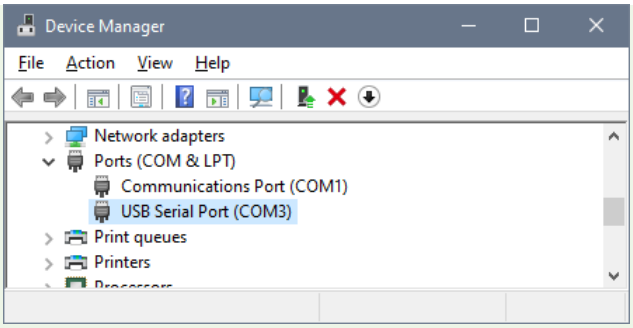
The **timeout** parameter defines the response timeout in milliseconds, after which the device is considered disconnected.

Hint:

The number of virtual COM port, created by USB-to-COM converter devices, can be seen in the Windows Device Manager.

The Device Manager can be opened by several ways, maybe the most straightforward way is to right-click the Windows icon in the task-bar and selected the Device Manager from the opened pop-up menu.

Then unfold the “Ports (COM & LPT)” line to see all available COM ports. Note the virtual COM port is displayed only if the converter device is actually connected to the PC.



Camera drivers

Camera drivers require special handling in SIPS, as cameras are the only type of devices, which can be selected multiple times (main imaging camera, guiding camera, context camera).

> Remark:
>
> As mentioned above, USB camera drivers are plug-and-play and require no configuration to work. However, there are a set of functions whose default behavior can be modified using a configuration file, although most users will stick with the default values.

Because different cameras, handled by single driver, may require different parameters, sharing the same configuration file may cause conflicts. This is why the “gxusb”, “gxeth” and “cxusb” camera drivers extend the previously mentioned convention of naming the configuration files.

By default, the configuration file of the same name like the DLL, only with the .INI extension, is used. This remains valid also for camera drivers. But every Moravian Instruments camera has unique identifier number (this is particularly useful if multiple cameras of the same type are connected to the host PC and the user need to assign them to desired roles). So, the driver prefers to use the configuration file, which name is extended with the camera identifier number, expressed in decimal without any leading zeros. For instance, let us have the C5A camera with identifier number 10001. Then the configuration file named “cxusd.10001.ini” will be searched on all three paths, mentioned above. Only after no such file exists on all three paths, the configuration file “cxusb.ini” is tested. This allows us to create configuration file for every used camera independently.

gxusb

The “gxusb.dll” driver handles all CCD based Gx class cameras as well as the global-shutter CMOS Cx cameras:

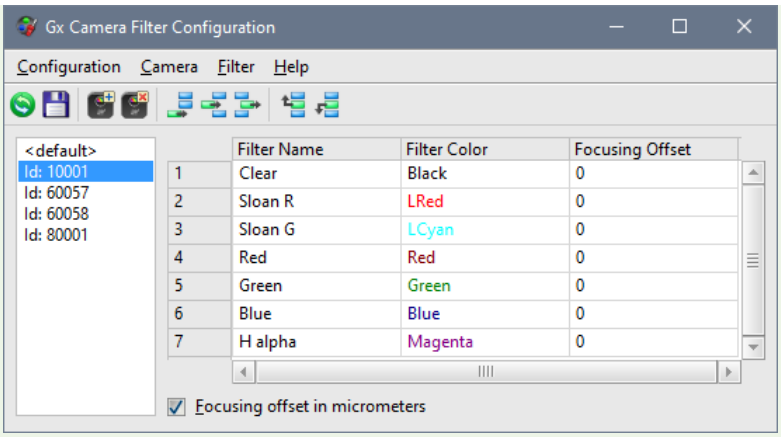
- All C0, C1, and CG models.
- Global-shutter C1+ and C2 series 3000, 5000, 7000, and 12000, including the 12-bit only „A“ models.

The “gxusb.ini” can contain three sections. The **[driver]** section may contain single boolean parameter **MicrometerFilterOffsets**. It defines if the offsets of individual filters, possibly defined in .INI file, are expressed in µm or in focuser device-dependent steps.

The **[filters]** section allows definition of filers in the camera attached filter wheel. Individual lines of this section do not stick with the .INI file typical line structure “key = value”, but just contain three values, delimited by commas “Filter_name, Filter_color, Filter_offset”. The filter color defines a color, with which the filter name is listed in the filter picking combo box in SIPS GUI, it is not the actual filter color. It can be expressed using common keyword for 8 basic colors and their light variants (Black, Blue, Cyan, Green, Yellow, Red, Magenta, LGray, DGray, LBlue, LCyan, LGreen, LYellow, LRed, LMagenta). Also, a number can directly represent the RGB value in individual bytes of the 32-bit unsigned integer.

Please note the filter description does not affect the filter wheel function. These values are only reported by the driver API function **EnumerateFilters** and user software typically use the enumerated values to put filter name into FITS header, to refocus the camera upon filter change etc.

> Hint:
>
> Both the **MicrometerFilterOffsets** and the **[filters]** section of the “gxusb.ini” file is a legacy feature, as there is a GUI tool to define filter names, colors and refocusing offsets available:



Filter parameters defined using this tool takes precedence to values defined in the “gxusb.ini” configuration file. However, the ability to define filters in the .INI file may be a fallback solution in some special cases.

The **[crop]** section with four numeric parameters **x**, **y**, **w**, and **h** can be used to limit a connected camera resolution to defined sub-frame. It could be used for instance to cut the non-illuminated portions of the sensor or sensor edges, on which the image quality is below acceptable distortions etc.

The sub-frame, defined in the **[crop]** section, is always defined in camera pixels, regardless of the used binning.

Remark:

The **[crop]** section causes the camera to report limited resolution to all software packages, using the gxusd.dll drivers. Technically, the same effect can be achieved by defining the sub-frame for instance in the **Frame** tab of the SIPS **Imaging camera** tool. But sub-frame must be redefined on every SIPS run. This can be a source of problems if for instance calibration frames (dark frames and or flat field frames) are reused among individual observing runs—the sub-frame must be defined exactly the same each time.

If the cropping is performed by the driver, the client software uses full resolution, and no sub-frame must be defined. On the negative side, canceling of the cropping requires modification of the “gxusb.ini” file and re-launching SIPS, so the parameters take effect.

cxusb

The “cxusb.dll” driver handles all rolling-shutter CMOS cameras, as well as global-shutter sensor camera models with onboard memory.

- Global-shutter C2 cameras with onboard RAM (C2-7000A and IR based C2-1300IR and C2-5000IR models).
- Rolling-shutter C1+ (C1+9000 and C1+46000)
- Rolling-shutter C2 (C2-9000 and C2-46000) models.
- All other rolling-shutter sensor-based cameras of C1×, C3, C4 and C5 series).

The “cxusb.ini” configuration file can define all sections, used in the “gxusb.ini” files (**[driver]**, **[filters]**, and **[crop]**).

Hint:

The GUI tool, allowing description of filters in the wheel, works the same way also for cameras controlled by CxUSB driver.

The **[driver]** section of the “cxusb.ini” file can contain several more parameters, influencing behavior of individual camera models.

Parameters affecting C4 cameras

The boolean parameter **C4Overscan** set to **true** causes the image returned by the C4-16000 camera will include the 64 pixels wide black-level overscan area to the right of the image.

The boolean parameter **C4BothGains** set to **true** causes the C4-16000 camera no longer allows changing of read modes, only single 12-bit read mode is available. Also, the camera resolution changes from 4096×4096 to 4096×8192 pixels.

Returned images then contain high-gain frame in the lower portion of the image and high-gain frame in the upper portion of the image.

Parameters affecting cameras with Sony IMX rolling-shutter sensors

The following parameters modify behavior of C1+ and C2 cameras with rolling shutter as well as all C1×, C3, and C5 camera binning.

The boolean parameter **HWBinning** set to **true** causes the in-camera hardware binning is used and software binning is no longer available. This mode brings faster download time, but also introduces several restrictions:

1. Maximal binning is limited to 2×2, higher binning modes are not available.
2. Asymmetrical binning modes (1×2, 2×1, …) are not allowed.

The boolean parameter **BinningSum** defines if the binned pixels are added or averaged. The default value, used if this parameter is missing from the .INI file, varies depending on the camera models:

- Cameras offering 16-bit digitization (C1×, C3, and C5) use **BinningSum = false**, as adding pixels may cause pixel saturation and thus invalidate photometric and astrometric measurement.
- Cameras offering 14-bit digitization (C2-9000) use **BinningSum = true**, as adding 4 pixels in with 14-bit resolution (2×2 binning mode) utilizes full 16-bit pixel dynamic range. This is not valid for higher binning modes, so it is on the user to set this parameter according to used binning.

The boolean parameter **BinningSaturate** set to **true** causes the resulting binned pixel is set to saturation value if any of the source pixels is saturated. For aesthetic astro-photography, keeping this parameter false (the default value for all cameras) could result into slightly better representation of bright star images, but for research applications, this parameter should always be set to true.

gxeth

The “gxeth.dll” driver provides interface for all cameras of all types (Gx and Cx), connected using the Moravian Camera Ethernet Adapter. To handle all possible configuration of all connected cameras, the “gxeth.ini” configuration file may contain all sections and parameters, defined for both “gxusb.ini” and “cxusb.ini” configuration files.

The only exception is the **[crop]** section, which is not supported for cameras connected using the Ethernet adapter.

The **[driver]** section of the “gxeth.ini” allows also definition of the driver specific parameters:

The numeric parameter **ConnectionTimeout** (or **ConnectTimeout**, the driver accepts both variants) defines timeout for TCP/IP connection in seconds. The default value is 1 second.

The numeric parameter **SendTimeout** defines timeout for TCP/IP send operation in seconds. The default value is 3 seconds.

The numeric parameter **ReceiveTimeout** defines timeout for TCP/IP receive operation in seconds. The default value is 30 seconds.

ascom_camera

The “ascom_camera.dll” driver handles cameras connected through ASCOM interface. The driver does not use any configuration file.

Standalone filter wheel drivers

sfw

The “sfw.dll” driver controls Moravian Instruments **Standalone Filter Wheels** (SFW), connected directly to the host PC using the USB line.

As opposite to External Filter Wheels (EFW), which are controlled as well as powered from the camera, standalone wheels require own power and control (USB) lines and thus require also own driver. The **Imaging camera** tool allows to define if a camera-controlled (be it internal wheel inside the camera head or external wheel, just attached to camera head but controlled by the camera) or standalone filter wheel is to be used in the **Filters** tab.

The “sfw.ini” configuration file can define filter-related section **[filters]**, used in the “gxusb.ini” and “cxusb.ini” files. Filters can be described there the same way.

> Hint:
>
> The GUI tool, allowing description of filters in the wheel, works the same way also for standalone filter wheels, controlled by the SFW driver. Also, the SFWs have unique identifier, like the cameras, allowing to configure filters for any particular standalone wheel.

sfweth

Like the “gxeth.dll” driver provides interface to all Moravian cameras, connected through the Moravian Camera Ethernet Adapter device, the “sfweth.dll” driver allows connection to the Moravian Instruments **Standalone Filter Wheels** (SFW), attached to the Ethernet adapter.

The “sfweth.ini” configuration file may contain the same parameters like the “sfw.ini” configuration file for filter wheels connected directly using the USB line. Also, the GUI tool configuring the filters in the wheel may be used the same way also for SFW connected through Ethernet.

ascom_filters

The “ascom_filters.dll” driver handles filter wheels connected through ASCOM interface. The driver does not use any configuration file.

GPS receiver drivers

gps18

The “gps18.dll” driver is intended for GPS receivers, using the Gramin USB Protocol. The driver does not use any configuration file.

nmea

The “nmea.dll” driver handles all GPS receivers, connected through COM serial line and using the NMEA (National Marine Electronics Association) standardized protocol. This driver uses a “nmea.ini” configuration file, containing parameters for COM interface (see the **Drivers using legacy COM serial interface** sub-chapter).

Telescope mount drivers

nexstar

The “nexstar.dll” driver handles telescope mounts using the Celestron (Synta) NexStar controller, communicating through COM serial line. This driver uses a “nexstar.ini” configuration file, containing parameters for COM interface (see the **Drivers using legacy COM serial interface** sub-chapter).

meade

The “meade.dll” driver handles telescope mounts using the Meade LX200 and derived protocols, communicating through COM serial line. This driver uses a “meade.ini” configuration file, containing parameters for COM interface (see the **Drivers using legacy COM serial interface** sub-chapter).

In addition to the **[comm]** section, the “meade.ini” configuration file can contain also **[driver]** section, containing one parameter **protocol**. This parameter allows modification of the used communication protocol with functions, implemented in the later hardware:

- **LX200** is the basic version of the protocol. This value is used if the **protocol** parameter or the entire **[driver]** section are missing.
- **LX200_16** protocol contains enhancements introduced in the 16” version of the LX200 telescope.
- **AUTOSTAR** protocol contains enhancements introduced in Meade Autostar controller.
- **AUTOSTARII** protocol contains enhancements introduced in Meade Autostar II controller.
- **PULSAR2** protocol contains extensions introduced in the Pulsar 2 mount controller.

ascom_tele

The “ascom_tele.dll” driver handles telescope mounts connected through ASCOM interface. The driver does not use any configuration file.

Focuser drivers

mmfocuser

The “mmfocuser.dll” driver handles the MM4 Focuser, manufactured by the 2EL company.

While this driver uses a “mmfocuser.ini” configuration file, containing parameters for COM interface (see the **Drivers using legacy COM serial interface** sub-chapter), beginning with SIPS v4.4 the configuration file is not required at all. Serial line parameters like communication speed, parity, etc. are set by default and the serial line COM port is determined automatically based on the device serial number.

By default, any device serial number, corresponding to the MM4 Focuser, is accepted, but for the case multiple devices are connected to the host PC, the new keyword “serial” is introduced in the **[comm]** section of the .ini file:

```
[comm]
…
serial = MM4-1234
```

Of course, if the specific serial number is to be specified, the .ini file must be properly defined.

ascom_focuser

The “ascom_focuser.dll” driver handles motorized focusers connected through ASCOM interface. The driver does not use any configuration file.

Flaps drivers

ascom_flaps

The “ascom_flaps.dll” driver handles telescope covers and calibration flaps connected through ASCOM interface. The driver does not use any configuration file.

Dome drivers

ascom_dome

The “ascom_dome.dll” driver handles observatory domes and roll-off roofs connected through ASCOM interface. The driver does not use any configuration file.

Switches drivers

mmswitches

The “mmswitches.dll” driver handles the Telescope Control System version 2 or 3 (**TCS v2/v3**) switch-box and autonomous heater and fan controller manufactured by the 2EL company.

While this driver uses a “mmswitches.ini” configuration file, containing parameters for COM interface (see the **Drivers using legacy COM serial interface** sub-chapter), beginning with SIPS v4.4 the configuration file is not required at all. Serial line parameters like communication speed, parity, etc. are set by default and the serial line COM port is determined automatically based on the device serial number.

By default, any device serial number, corresponding to the TCS2 or TCS3 device, is accepted, but for the case multiple devices are connected to the host PC, the new keyword “serial” is introduced in the **[comm]** section of the .ini file:

```
[comm]
…
serial = TCS2-1234
```

Of course, if the specific serial number is to be specified, the .ini file must be properly defined.

ascom_switches

The “ascom_switches.dll” driver handles array of controllable switches connected through ASCOM interface. The driver does not use any configuration file.

Weather station drivers

fiedlermeteo

The “fiedlermeteo.dll” driver handles the Fiedler RDH11-T (rain, environment temperature and sky temperature sensor) and RVT80/85 (humidity and air temperature sensors), communicating through RS-485 serial line. Both devices must be switched to use the MODBUS communication protocol.

> Remark:
>
> To handle both sensors by a single driver, they must be connected to single RS-485 bus, which means they must be using one serial COM PC interface. If each sensor uses own COM port, it is necessary to use two instances of the driver, selected to different tabs of the **Weather station** tool.
>
> Using of RVT8x together with RDH11 on single bus (handled by single driver) not only adds humidity channel to the set of measured values, but also allows for more precise air temperature measurements. Unfortunately, the temperature returned by the RDH11 device is often higher than the real air temperature, because the RDH11 temperature sensor is affected by the heating of the rain sensor. As the air temperature is also used to calculate cloud cover from the difference between air and sky temperatures, cloud cover returned by the RDH11 device is affected by error.
>
> Because the air temperature returned by RVT80 is much more precise (approx. ±1 °C, compared to temperature up to 7 or 8 °C higher), the driver prefers it also for cloud cover calculations. If the RVT85 device, equipped with Pt100 thermometer, is used, even better air temperature is used by the driver. Only if no RVT8x device is connected, driver fallbacks to air temperature returned by RDH11.

This driver uses a “fiedlermeteo.ini” configuration file, containing parameters for COM interface (see the **Drivers using legacy COM serial interface** sub-chapter).

In addition to the **[comm]** section, the “fiedlermeteo.ini” configuration file can contain the sections **[sensor_RDH11]** and **[sensor_RVT8x]** containing parameters related to the sensor itself and the used MODBUS protocol, used on top of the COM interface.

> Remark:
>
> The **[sensor]** section is used for RDH11 parameters when the **[sensor_RDH11]** is missing for backward compatibility.

- The **trace** boolean parameter allows turning on of the debugging out, directed to the Windows standard debug output.
- The **address** numeric parameters defined the MODBUS protocol address. The default value 8 is used by the RDH-11 devices and 11 by the RVT80/85 ones, providing the address was not changed by a Fiedler utility.
- The **temperature_offset** and **sky_temperature_offset** numeric parameters allow calibration of the absolute temperature returned by the sensor, other than modifying coefficients in the device itself.
- Other parameters, specific to **[sensor_RDH11]** section only, directly reflect registers in the RDH11-T sensor. Refer to the Fiedler manual for explanation:
  - rain_coeff_A0
  - rain_coeff_A1
  - rain_coeff_A2
  - rain_coeff_A3
  - temperature_coeff_A0
  - temperature_coeff_A1
  - rain_mean_value
  - rain_hysteresis
  - max_heat_temperature
  - heat_off_dt
- Two more parameters, also specific to **[sensor_RDH11]** section only, define how the driver calculates the cloud cover from the air and sky temperatures:
  - cloudy_sky_dt
  - clear_sky_dt

When the difference between air and sky temperature is lower than **cloudy_sky_dt**, the driver returns 100% cloud cover. When the air and sky temperature difference is greater than **clear_sky_dt**, returned cloud cover is 0%. Temperature difference between these two limits is interpolated as 1 to 99% cloud cover.

ascom_meteo

The “ascom_meteo.dll” driver handles weather stations connected through ASCOM interface. The driver does not use any configuration file.

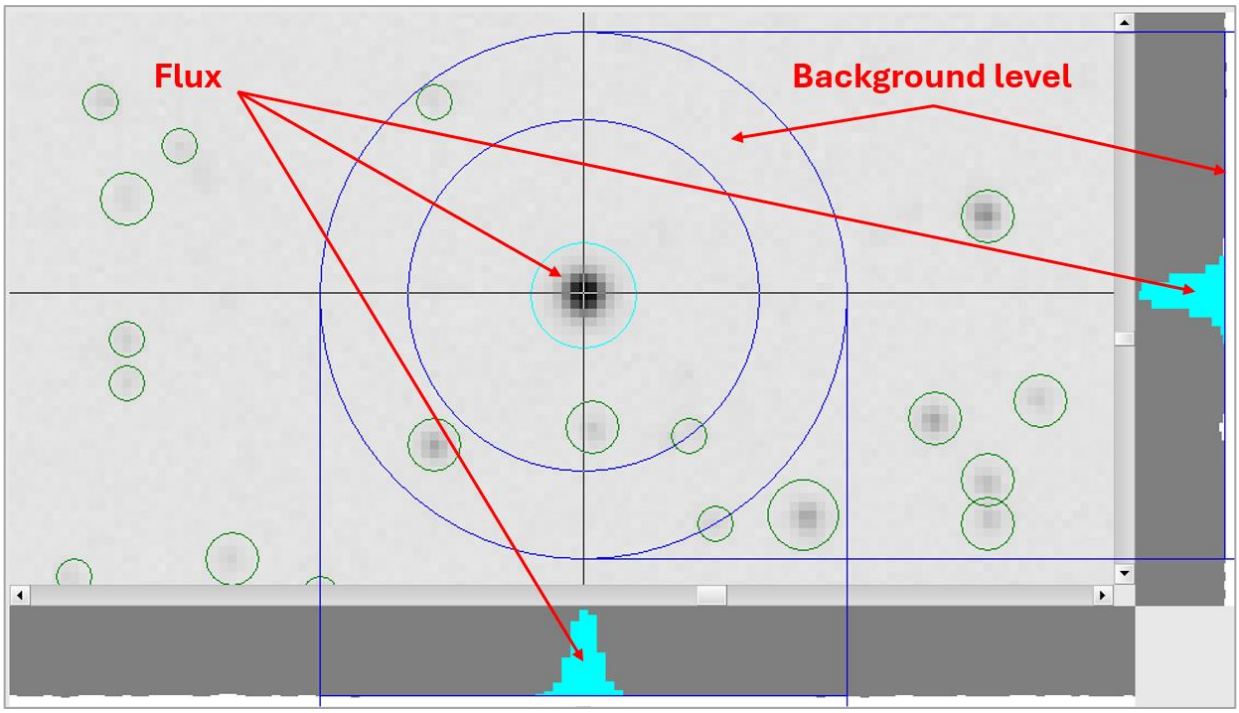
Appendix C: A short introduction to photometry

The main task of photometry is to measure **flux** of astronomical objects and its changes in time. It represents amount of light captured from each object (star, galaxy, asteroid, comet, etc.) within the image and it's calculated as a sum of values (ADU[15] counts) of all pixels belonging to the particular object. This means pixels positioned withing a defined circle (**photometric aperture**) centered around the star's centroid.

Remark:

For simplicity, we will call the measured object **variable star** or just a **variable** from now on, as the recording of brightness changes of variable stars is the most typical application of photometry. Also, such stars are often marked with an abbreviation **VAR**. However, let's keep in mind that everything explained here is also valid for other astronomical objects like asteroids, novae, supernovae, etc.

However, ADU values of each pixel do represent not only the amount of light captured from the variable star, but also the light from the sky background (natural background as well as light pollution) and by sensor bias. It is the task for photometric software to determine the background value and subtract it from each pixel, so the measured flux corresponds only to light coming directly from the object.



To achieve the best possible estimation of the background value, it is not determined globally for the image, but individually for each measured object. Notice the three circles around the star in the figure above. The smallest, cyan, one represents the photometric aperture. The two larger blue circles define the area, known as **background ring**, which is used for calculating the background value.

Hint:

It is no exception that there are no other stars in the background ring, especially in the star-poor sky regions. On the other hand, it is virtually impossible to define a background ring, so it does not contain any other star in star-rich regions within the Milky Way. SIPS of course excludes such stars from the background value calculations, but sometimes a dim star remains undetected and thus its light affects the background value. Also, some other artifacts, like for instance bright star diffraction spikes, may affect the background value. So, especially for dense star fields, it is highly recommended to set the **Use robust mean for background** option in the **Photometry parameters** dialog box (see the Photometry calculation for details).

[15] The abbreviation ADU means Analog-to-Digital converter Unit. It is a number returned from A/D Converter when the charge accumulated in each pixel s digitized.

Since human's perception of brightness is not linear (we are able to see a really huge range of brightness), astronomers start using a nonlinear unit as well. This logarithmic brightness unit is called **Magnitude** and it is defined in such a way that a **100-times brighter** object (with 100-times greater flux) differs **by 5 magnitudes**. The magnitude scale is also defined as negative, i.e., a lower magnitude value means brighter object. So, the object with Magnitude 1 less is $\sqrt[5]{100} = 2.5119$ times brighter.

Instrumental magnitude

Based on the definition above, we can calculate the **instrumental** magnitude Mag_I from flux F using the following equation:

$$Mag_I = -2.5 \cdot \log_{10}(F)$$

It strongly depends on the instruments used (telescope and camera; thus the name), exposure time, and other parameters. Naturally, the flux accumulated by a small lens and low sensitivity camera will be much smaller than a flux from large telescope and premium high sensitivity camera.

Unfortunately, the flux also varies with the weather conditions, object altitude above horizon, and other conditions. So, instrumental magnitude alone is not useful as the influence of these effects often overshadows the intrinsics brightness changes of the variable star, which we want to measure.

Remark:

This is why neither SIPS **Photometry** nor **Astrometry** tools work directly with instrumental magnitudes.

Relative magnitude

To eliminate flux variations with other causes than the real object brightness changes, another **comparison** star is used as a brightness reference. It is assumed that all factors influencing the variable star flux, also affect the comparison star in the same way. Thanks to logarithmic magnitude scale, the **comparison star instrumental magnitude** is just subtracted from **variable star instrumental magnitude** to obtain variable star relative magnitude Mag_R.

$$Mag_R = -2.5 \cdot \log_{10}(F_V) - (-2.5 \cdot \log_{10}(F_C))$$

This can be simplified to the **Pogson equation**,

$$Mag_R = -2.5 \cdot \log_{10}\left(\frac{F_V}{F_C}\right),$$

which bypasses the need to calculate the intermediate instrumental magnitudes and uses directly the fluxes.

The uncertainty of relative magnitude can be defined as follows:

$$\sigma_{Mag} = \frac{2.5}{\ln(10)} \cdot \sqrt{\left(\frac{\sigma_V}{F_V}\right)^2 + \left(\frac{\sigma_C}{F_C}\right)^2}$$

Unfortunately, using the variable and comparison star flux ratio does not eliminate all unwanted effects. Different colors are absorbed differently by our atmosphere; red light is less absorbed than blue one (this is why the sky is blue as well as why Sun or Moon close to the horizon appear redder). So, if the variable and comparison star colors are different, their fluxes vary with changing air-mass over the observing session differently, and thus their ratio changes as well. This effect (different extinction ratio) significantly affects the relative magnitude.

The very effective way to eliminate (or at least suppress) this effect is choosing variable and comparison stars with the same color. Extinction is then the same for both stars.

Using a filter to limit the range of wavelengths also limits this effect. Unfortunately, it would be necessary to use a very narrow-band filter to eliminate it completely and such filters are not used in research applications for various reasons. Standard photometric filter passbands widths are tens, or even more than a hundred nanometers, and different star colors still affect the flux ratio.

Absolute magnitude

The relative magnitude gives no idea how the object is "really" bright, as it depends on the brightness of the used comparison star. Thanks to the logarithmic nature of the magnitudes, there is no natural zero point like "zero brightness". A reference zero point (a star with brightness 0 Mag) is then chosen as a part of some absolute magnitude standard.

So, we can think about absolute magnitude[16] as a relative magnitude but related to a chosen reference point (zero-magnitude star). While it is practically impossible to always select a zero-magnitude comparison star, it is also not necessary. If we know the absolute magnitude of the comparison star, we can get the absolute magnitude of the variable star simply by adding the comparison star absolute magnitude to the measured relative magnitude of the variable star. However, there are some caveats in this thinking.

In general, star flux varies with wavelength depending on the star temperature (stars mostly follow the Planck's law of black body radiation). Also, sensor sensitivity in different wavelengths differs, as well as optics and atmosphere transparency etc. This is why the term "absolute magnitude" without specification of the color (photometric system—wavelength interval defined by a standard photometric filter) is only a vague term. It is useful to give a general idea about celestial object brightness, e.g. during public lectures, but it is not used in research at all.

Remark:

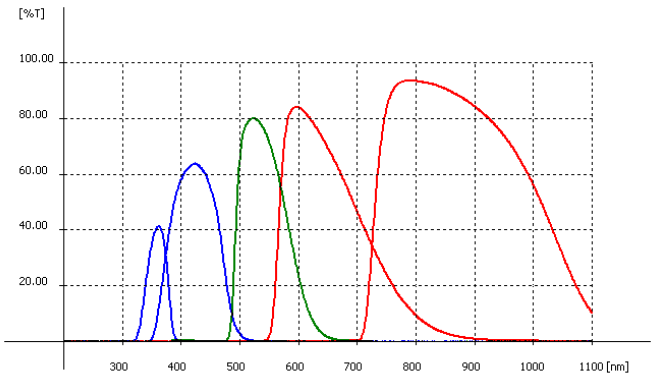
While the dimmest stars visible by naked eye brightness is approx. 6 Mag, Vega is 0 Mag, Sirius (brightest star except Sun) -1.5 Mag, Jupiter can reach around -2.5 Mag, Venus -4.5 Mag, full Moon -12.5, and Sun even -26.7 Mag.

Otherwise, color is important when talking about absolute magnitudes. For instance, cold red stars are several times brighter in red light compared to blue light. And similarly, a white dwarf can be many times brighter in blue compared to a brightness in red color.

Astronomers defined several photometric standards over time. Such standards are always associated with a set of filters, defining a passband (a range of wavelengths).

Johnson-Cousins standard

Johnson-Cousins is one of the oldest and still widely used photometric systems. It consists of 5 filters denoted U (ultra-violet), B (blue), V (visual), R (red), and I (infra-red). The passbands are illustrated on the image below:



Remark:

The UBV filters were introduced by Jonhson and Morgan in the 1950s, but their filters for red and infra-red colors were not consistent and poorly reproducible. Only later Cousins (in the 1970s and 1980s) redefined these filters, which is why the last two filters are sometimes denoted R_C and I_C.

These filters are created by adding metal-oxide dopants directly into the glass, as the technology of manufacturing filters using thin interference layers was not available these times. This caused several disadvantages of the UBVRI set:

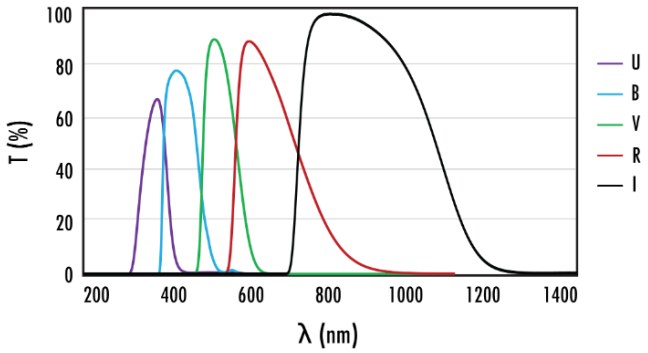
[16] The "absolute magnitude" here means magnitude related to some standard photometric system, not a magnitude expressing object brightness in the 10 parsecs distance.

- Peak transparency is well below 100%, it reaches only ~40% in the case of U filter. So, using these filters wastes precious light.
- The passband edges are not very sharply defined and individual passbands significantly overlap, which negatively affects physical interpretation of observations.

Bessel (Johnson-Bessel) standard

The low peak transparency of Johnson filters, particularly in the U and B passbands, lead to an effort to modernize the UBVRI filter set. The modification, known as Bessel or Johnson/Bessel filter set, primarily focused on increasing the peak transparency.

While the Bessel approximation of the UBVRI passbands can in theory reach 100% peak transparency, in reality such filters are virtually impossible to manufacture, and the peak transparency is lower. But still, the Bessel filters offer significant gains in transparency, which results in lower exposure times and/or better S/N of captured images.

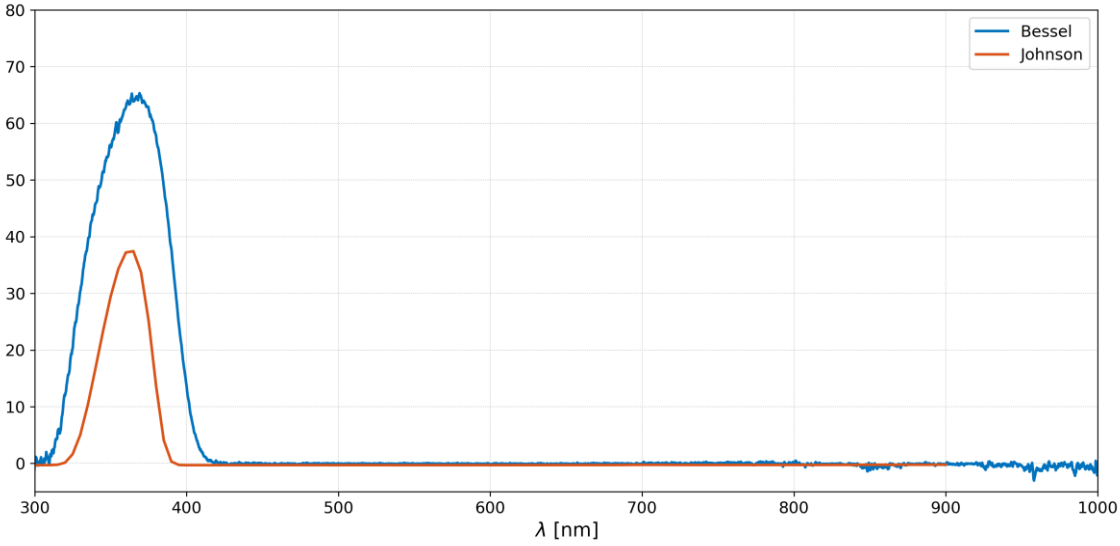


Unfortunately, the Bessel filter variants do not differ only by peak transparency, but also by passband ranges. So, Johnson-Cousins and Bessel magnitudes are not directly exchangeable, it is necessary to perform magnitude transformations between these two systems. E.g., the Landolt fields UBVRI magnitudes are Johnson-Cousins, not Bessel ones.

Warning:

Many filter manufacturers now offer Bessel filter variants instead of traditional Johnson-Cousins filters. But only some of these manufacturers properly mark these filters as Bessel ones, sometimes the filters are named just “UBVRI photometric filters” or even “Johnson-Cousins UBVRI filters” without mentioning they are the Bessel variants.

For example, consider the difference between the Johnson and Bessel U filters:



The integral signal passing through the Bessel U is several times greater compared to Johnson U, but also the spectral range of the Bessel U is significantly wider.

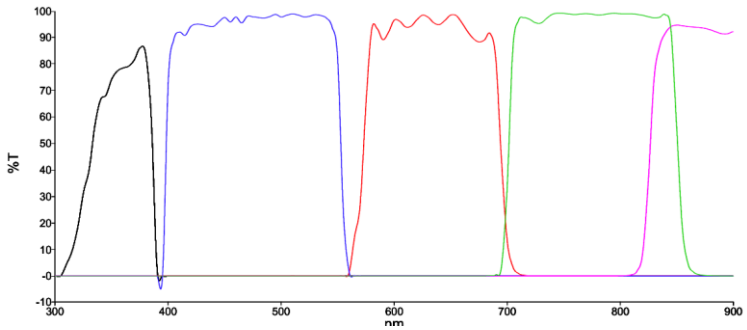
Sloan standard

The Sloan Foundation Telescope at the Apache Point Observatory, used to perform the Sloan Digital Sky Survey (SDSS), rejected the aging UBVRI filters and introduced modern set of filters with well-defined passband edges and high transparency within the passband. The filters, abbreviated u, g, r, i, and z, are made from colored glass substrates and interference layers.

> Remark:
>
> It is an interesting fact, that the SDSS "z" passband was not defined by a real filter, but just by the decreasing sensitivity of the used detectors.

However, for use with other telescopes and cameras than the original Sloan Digital Sky Survey camera, the slightly modified standard was created. To distinguish the modified standard from the original one, letters marking individual filters are suffixed with an apostrophe (the modified Sloan standard filters are denoted u', g', r', i', and z'). Filters following this standard are now commercially available.

The modified Sloan passbands are illustrated on the image below:



It is worth noting that virtually entire visible light 400-700 nm is covered with just two Sloan g' and r' filters. Also, peak transparency of these filters approaches 100% so using them does not unnecessarily waste precious light.

Glass standard

The Glass[17] JHK filters cover infra-red part of the electromagnetic spectrum. Individual filters are centered around 1.2 µm, 1.6 µm, and 2.2 µm. However, except for the recently introduced (but still prohibitively expensive) SWIR sensor-based cameras, these wavelengths are not reachable by amateurs and SIPS supports them only for completeness.

> Remark:
>
> As usual, there are many modifications of this standard and many sky surveys used slightly modified filters (e.g., 2MASS) and it is usually necessary to apply transformations to convert magnitudes among them.

Absolute magnitude and Landolt fields

Since comparison stars with precisely defined magnitudes and appropriate color indices are not guaranteed in the field of interest, a bit more complex but also more general procedure leading to absolute magnitudes was introduced. It is based on flux-to-magnitude transformation, calibrated on so-called **Landolt fields**—areas uniformly distributed around celestial equator, so some Landolt fields are always available for observation. Stars within Landolt fields have very precisely measured absolute magnitudes. Originally, UBVRI magnitudes were available, now also u'g'r'i'z' magnitudes are added. The purpose of these fields is to calibrate the observing setup (telescope, camera, filters), which means determining of transformation function terms, which then allows to calculate absolute magnitude just from observed fluxes.

The transformation function could be like this:

$$Mag = -2.5 \cdot \log_{10}(F_V) + q + c_1 \cdot CI + e_1 \cdot AM$$

Where:

- F_V: measured object flux

[17] Refers to an infrared astronomer Ian Stewart Glass, not a material from which the filters are made.

- q: offset
- c_1: the first-order color coefficient
- CI: the star color index (e.g., Johnson B – V magnitudes)
- e_1: the first-order extinction coefficient
- AM: air mass

Free parameters q, c_1, and e_1 are typically determined by fitting the above function on fluxes from images of one Landolt standard field.

As mentioned above, the measured flux strongly varies with air-mass (object height above horizon) as well as with weather conditions. So, best results require calibrating of the transformation function each night (or even several times during night if the weather conditions significantly change).

More complex transformation functions may be used to better eliminate unwanted effects and enhance precision, like for instance higher-order polynomials representing color dependence:

$$Mag = -2.5 \cdot \log_{10}(F_V) + q + c_1 \cdot CI + c_2 \cdot CI^2 + e_1 \cdot AM$$

Also, a different color system may be used (Sloan g'–r' can be used instead if B–V if Sloan filters are used), multiple color indexes can be used at once (Sloan g'–r' and r'–i' in one equation), etc.

Ensemble photometry

Relative magnitudes suit many scientific tasks well: determining of the minimum or transit time, period analysis, identification and characterization of the multiple star systems, … This is why the SIPS Photometry tool was originally designed with only relative magnitudes in mind. Also, converting magnitudes to some absolute photometric system is time consuming procedure; it requires capturing some Landolt field in addition to target object, data must be excessively processed, at least if the process is not automated and only general tools (like Excel) are used.

Here comes the idea of **Ensemble photometry**[18]—obtaining absolute magnitudes using a transformation function **calibrated on every processed image**. No Landolt fields are used, but information about color index and magnitude in the used passband for many stars within each image must be known to allow calibration.

What's more, with enough calibration stars, the Ensemble photometry allows using of star position within the frame to enhance precision. Calculating with flux-to-magnitude transformation variations over a field of view helps compensate for missing or low-quality flat fielding, background gradients, etc.

Because Ensemble photometry is calibrated for each image, the extinction coefficient is not used:

$$Mag = -2.5 \cdot \log_{10}(F_V) + q + c_1 \cdot CI$$

- F_V: measured flux
- q: offset
- c_1: the first-order color coefficient
- CI: the star color index

The quality of transformation (absolute magnitude precision) depends on the reliability of star magnitudes used for calibration. Precision increases with number of stars used for calibration, many hundred or even thousand stars with known magnitudes lead to good results.

The Ensemble photometry in SIPS can use several transformation functions, differing in **color** and **spatial term orders**. Color term order can be in the range [1..2], spatial term order can be in the range [0..2] (spatial term order = 0 means star position within the image does not affect the transformation). The transformation with the higher order terms available looks like:

$$Mag = -2.5 \cdot \log_{10}(F_V) + q + c_1 \cdot CI + c_2 \cdot CI^2 + x_1 \cdot X + y_1 \cdot Y + x_2 \cdot X^2 + y_2 \cdot Y^2 + xy \cdot XY$$

[18] Sometimes the term "Ensemble photometry" is used for relative photometry using more than one comparison star (e.g., AAVSO protocol uses the term "ensemble" this way), do not confuse it with SIPS Ensemble photometry.

Again, the F_V is the variable star flux and *CI* is its color index; but the transformation above uses also an object position within image *X* and *Y*. Free parameters are:

- q: offset
- c_1: first order color coefficient (mandatory)
- c_2: second order color coefficient (optional)
- x_1, y_1: first order spatial coefficients (optional)
- x_2, y_2, xy: second order spatial coefficients (optional)

SIPS currently supports three color indexes:

- Johnson B – V
- Sloan g' – r'
- Glass J – K

Color passband can be chosen from:

- Johnson-Cousins U, B, V, Rc, Ic
- Sloan u', g', r', i', z'
- Glass J, H, K

As explained earlier, an astrometry catalog, used for astrometry solution, must contain magnitudes for selected color index and passband. USNO catalogs, created from "red" and "blue" Palomar plates, do not contain standard magnitudes. UCAC5 catalog contains Gaia G magnitude, which unfortunately does not correspond to any standard passband, and Cousins R magnitude, which is not enough for ensemble photometry. So, UCAC4 is currently the only catalog containing magnitudes in many Johnson-Cousins and Sloan passbands available in SIPS.

As mentioned above, the ensemble photometry precision strongly depends on the number of available calibration stars within each image:

- If there are less than a few tens of stars, precision is very low.
- About a hundred calibration stars lead to moderate precision.
- Many hundreds or thousands of stars ensure that the precision is often better or equal to very best relative photometry.

In general, the ensemble photometry brings several pluses:

+ Measured object brightness is obtained directly in absolute magnitudes.
+ No comparison star(s) used; no need to carefully find non-variable, uncontaminated comparison stars with similar color.
+ Eliminates outliers caused by a comparison star (e.g., a satellite crossing the comparison star aperture etc.).
+ Same or better precision compared to instrumental magnitudes (with enough calibration stars).
+ Spatial dependence suppresses errors caused by VAR/CMP distance within image.

But it also brings some downsides:

− Astrometry reduction is obligatory.
− UCAC4 must be currently used as the primary astrometric catalog.
− Even in UCAC4, magnitudes in some color passbands are missing in some sky areas.
− A standard photometric filter, included in the UCAC4 catalog, must be used to capture images (unfiltered observations cannot be reliably processed).
− Enough stars within the field of view are necessary for proper calibration (a wide-field telescope/camera is a big advantage).

Appendix D: SIPS version updates

Version 4.0

Version 4.0.0 released on December 11th, 2023.

Initial SIPS 4 release.

Version 4.0.1 released on January 17th, 2024.

Updates:

- Driver configuration .INI files are now searched on the user's "Documents\SIPS\ini" path first, and only if not found there, the "\Users\Public\Documents\SIPS\ini" and SIPS exe file paths are tested.
- The "catalog.ini" file, used in the **Telescope** tool, is also searched on three paths—user's "Documents\SIPS\ini", "\Users\Public\Documents\SIPS\ini" and SIPS exe file path. Catalogs offered by the **Telescope** tool GUI are merged from the "catalog.ini" files, found on all three paths.
- Also, the "catalog.ini" file format was updated; individual catalog entries use strings in quotes.
- The **Photometry task** dialog box tries to automatically fill remaining values like file mask, dark, flat, and reference files etc. upon entering of a folder, containing files to be processed.
- GUI controls, intended for entering of coordinates in the form of degrees or hours, minutes, and seconds (for instance, in the **Telescope** tool or **Plate parameters** dialog box), were extended with a "smart paste" feature. This new function causes a coordinate, pasted from clipboard in the form of fraction degrees, is automatically converted to whole degrees, minutes, and seconds and subsequent count-boxes are also updated.
- The SIPS User's Guide was extended with the description of the **Focuser**, **Telescope**, **Dome**, **Weather Station**, and **GPS receiver** devices.
- The SIPS User's Guide was extended with the **Appendix B: Device drivers supplied with SIPS**.

Bug fixes:

- The SPL instruction **pause** with > 10 s delay continued 5 s sooner.
- The SPL **Program Editor** window can no longer be minimized while the program is running, as the window runs modally.

Version 4.0.2 released on January 25th, 2024.

Bug fixes:

- If the first filter in the wheel was selected, the **Imaging Camera** tool did not append the used filter name to the saved FITS file name when the **Append filter** option was checked.
- The **Weather station** tool history graph was distorted if a tool window was closed and opened again.
- The Fiedler RDH11-T weather station driver communication line could stop responding.

Version 4.0.3 released on March 8th, 2024.

Updates:

- The **Focuser** tool was redesigned to reflect the behavior of ASCOM focuser drivers:
  - The tool distinguishes between absolute and relative position settings. Depending if the focuser is absolute or not, the **Focuser** tool enables the newly introduced **New Position** count box and **Move To** command button.
  - The **Enable temperature compensation** checkbox functionality is newly separated from **Compensation steps /10°C** count-box, as the ASCOM focuser interface does not support this functionality (ASCOM only allows turning compensation on and off, but actual compensation value must be defined elsewhere).
- The newly introduced **READMODE** FITS header contains string description of the used read mode.
- Added native support for Moravian Instruments **Standalone Filter Wheels** (SFW), connected either directly to the host PC using USB line or through the Moravian Camera Ethernet Adapter device.

- Added support for C2-9000, C1×26000, C1×61000, C3-26000, C3-61000, C5A-100M, C5A-150M, C5S-100M, and C5S-150M cameras with **firmware version 8.x** and higher. This firmware updates the handling of sub-frames and eliminates some of restrictions to sub-frame coordinates of the previous firmware versions.
- Added support for **C0** and **C1 version 3** cameras.

Bug fixes:

- The **Photometry** tool combo-boxes with individual apertures doubled the list of apertures when a photometry was run for the second time on the same set of data. Selecting aperture from the second occurrence of each aperture (a second half of the list) then had no effect.
- The **Append filter** option in the **Imaging camera** tool, used to create file names of images newly read from camera, now check if the filter name contains characters not allowed in file names (e.g., the '<' or '>' characters if the filter is stated as "<undefined>") and replaces them with underscore '_' character.
- The **SPL Program Editor** started the **call** instruction in the program tree with random string if generating of the **call** keyword was turned off.
- SIPS appearance on high-DPI screens (above 100%) was fixed:
    - Windows, narrower than certain limit (for instance, the **Image Info** tool, but also **Photometry parameters** dialog box etc.), were displayed wider than necessary with a blank space on the right side.
    - Some dialog boxes, containing paragraph texts (for instance, the **New image transformation** or **Sensor Parameters** dialog boxes etc.) were displayed with empty vertical space below each paragraph.

Version 4.0.4 released on April 15th, 2024.

Updates:

- The **Program Editor** tool newly allows to define a program, which will be run upon SIPS start, without a necessity to load and run the program by the user.
- The **Filter** combo-box in the **Exposure tab** of the **Imaging camera** tool is no longer disabled when acquiring dark or bias frames. This allows usage of opaque filter to capture dark frames with cameras not equipped with mechanical shutter.

Bug fixes:

- If the **Use independently-controlled filter wheel** radio button in the **Filters tab** of the **Imaging camera** tool was selected, the **Init Filter Wheel** command button did not cause refreshing of the **Filter** combo-box in the **Exposure tab** and resetting the list of filters to the first position.

Version 4.1

Version 4.1 released on June 26th, 2024.

Updates:

- Added support for **Flaps** devices (motorized telescope covers) and flat field calibration panels, moved in front of telescope, including the brightness control.
- Added support for **Switches** devices, allowing binary and analog control of heating strips, fans etc.
- The **Dome** tool GUI was redesigned to adopt itself to controlled dome or roll-of-roof capabilities. Only the controls corresponding to available functions are shown. So, for instance, only the **Shutter** interface is shown if roll-of-roof is controlled, or the **Altitude** interface is hidden if a dome with a slit is attached and no dome opening altitude is controlled etc.
- Similar dynamic GUI modifications are implemented in the **Weather station** tool. Only the values, supported by the attached driver, are shown to preserve display space.
- Added support for C1+ cameras with rolling-shutter sensors, like the C1+9000.
- Added support for C2 cameras with global shutter sensors, equipped with GPS receiver used for precision image timing.
- The "sips.ini" driver configuration file is read not only from the path where the SIPS executable is installed, but also from the "\Users\Public\Documents\SIPS\ini" directory. Adding a driver to SIPS no longer needs a modification of the configuration file in "Program Files" directory, but additional configuration file may be created in public directory. SIPS then uses all drivers defined in both configuration files.

- New native program **serial**, allowing the SPL code to communicate over the COM lines, was added.

Bug fixes:

- The **Program Editor** tool in the **Graph mode** fixed several issues:
  - The **let** and **sprint** instructions optional program name of the assigned variable, imported from another program, was not properly initialized to empty string.
  - The **let** instruction custom view combo box, offering variables to be assigned, contained local variables only if these variables were marked "public".
- The SPL compiler wrongly reported error, if the index or position expression in the embedded functions **slice**, **delete**, **insert**, **item**, **replace**, **subst**, and **char** were integer, real type was required, despite results are always converted to integer.
- The **Weather station** tool could display distorted history charts if the used computer timing mechanism activated the tool a few milliseconds prior to programmed time. Now the tool is resistant to such possible timer activation uncertainties and history charts should be displayed correctly.
- The **Imaging camera** tool used the exposure time, defined in the **Exposure** tab, also for inter-image guiding calibration, despite the calibration dialog box allows to define different exposure time for calibration only.
- When some new object was selected from the catalog of the **Telescope** tool, coordinates were properly shown in the **New R.A.** and **New Dec.** count boxes, but actual numerical R.A. and Dec. values were not updated until each count box was manually confirmed (selected and then either deselected again or the <Enter> key was pressed). So, if the **Synchronize** or **Go To** command was performed without manual confirmation of each count box, coordinate values originally present prior to catalog object selection were used instead of the new visible ones.
- The **Automatically connect to selected driver** option for drivers/tools other than camera worked only for non-PnP drivers or PnP drivers connected when SIPS started. If a non-camera PnP driver (e.g. USB connected dome controller) was not present (enumerated) upon SIPS startup and was added only later, SIPS did not dynamically connect to such device, despite the option for automatic connection was checked.

  Remark:

  Please note the USB-to-COM (RS-232C) converters are considered connected when the converter itself is plugged into USB port, despite the device itself is not connected to RS-232C or TTL-COM line.

Version 4.1.1 released on July 8th, 2024.

Updates:

- The **Plate astrometry information** dialog box was extended with the **Defined pixel size includes binning** option to settle the dispute if the pixel size, stated in the FITS header, should pixel dimension before binning or after binning.
- **Web SIPS** application was slightly redesigned to show the currently handled device icon on narrow displays (e.g. on mobile phones), when the left device list, on the page left side, is hidden.

Bug fixes:

- The **Weather Station** ASCOM driver retuned the sky brightness channel in expressed Luxes, not in Magnitude per square arc-second. Now the proper unit of sky brightness is returned also by ASCOM driver.
- The **Web SIPS** application showed uninitialized values when the **Automatically Connect to Selected Imager/Guider/Context Only** option was checked and the chosen camera was not yet connected to the host PC.
- Some of the SIPS v4.1.0 ASCOM drivers (e.g. ascom_telescope.dll) did not initialize correctly when the v4.1 was installed over the existing v4.0.x installation. The cause was "sips.ini" driver definition file, wrongly copied also to the "\Users\Public\Documents\SIPS\ini" folder. The "sips.ini" driver configuration file in the Public Documents folder is intended for integration of the independently installed the 3rd party drivers, not for duplicating of drivers installed with the SIPS itself.
  The workaround was either to uninstall the v4.0.x prior to v4.1 installation or to delete the "sips.ini" from the Public Documents folder. Either way, the v4.1.1 properly handles even drivers, duplicated in the "sips.ini" located in Public Documents.

Version 4.1.2 released on August 1st, 2024.

Updates:

- New driver for the **MM4** motorized focuser from the 2EL was added to SIPS distribution.
- New driver for the Telescope Control System version 2 (**TCS v2**) switch-box and autonomous heater and fan controller from the 2EL was added to SIPS distribution.
- The **Web Server Configuration** dialog box now allows to determine if the users would be able to shutdown the host PC or not. If allowed, the **Web SIPS** application contains the **Shutdown Server PC** option in the user's menu.
- In addition to already present optional check if the newer version is available upon SIPS start, explicit query for the latest available version of SIPS was added to the **Help** tab.
- The **Dome** tool detects if the dome controller is used to control roll-ff roof only and if yes, it asks for confirmation prior to opening and closing of the observatory roof (shutter). This precaution was implemented because most roll-off roof observatories require a telescope to be parked in horizontal position prior to roof movement, else the roof and the telescope may collide.

Bug fixes:

- The redesigned GUI of the **Dome** tool, adopting itself to the capabilities of the used drivers, lead to SIPS crash when a dome controller, supporting the roll-off roof (this means without the ability to set dome azimuth) was connected and the **Web Server** was used.
- The **Read Location from Camera GPS** button in the **New FITS Headers** dialog box converted the latitude and longitude, already expressed in degrees, once again from radians to degrees, which lead to 180/π-times greater values.

Version 4.1.3 released on September 10th, 2024.

Updates:

- The **Web SIPS** web-browser based application was significantly updated, enhanced, and polished.
- The **Web SIPS** astrometry is newly asynchronous, not causing timeout error messages.
- The **Web SIPS** ZIP file creation and download is newly asynchronous; the progress bar is displayed for the user.
- The **Web SIPS** application recognizes if new images were created after the image ZIP file was created and asks the user if the existing ZIP file is to be downloaded or new ZIP file should be created.
- The **Remote Access Web Server** newly allows definition of the title string, displayed as the WWW browser page title.
- The **Remote Access Web Server** with enabled local storage now allows creation of new folder for acquired images each observing day. The **Web SIPS** application can browse individual folders and selectively download or erase individual image types.
- The star filtering option for the **Star Sheet** lines in the **Photometry** tool is now kept for all selected images, the filter is not reset when another image is selected. Also, the sheet responses are is much faster when filtering is active.
- The **Telescope** tool newly displays the time of meridian crossing.
- The **Imaging camera** tool opens a warning dialog box when series of images are to be saved without file name distinguishing option (ordinal number or date and time) and with file overwrite enabled.
- The **Guiding camera** tool newly contains checkbox, allowing to control if the dark frame should be used for guiding camera images.
- The **Guiding camera** tool allows saving of the acquired dark frame.
- The **Field curvature** can be newly fitted with Legendre polynomials, in addition to backward compatible monomial polynomials.
- The **Field curvature** numerical least square fitting method was reworked to be much more robust and reliable. Also, stars with saturated pixels are automatically rejected from fitting as their centroids cannot be precisely determined.

Bug fixes:

- The **Guiding camera** tool could stop guiding then the minimal guiding pulse length was defined and the calculated pulse was below the defined minimal length. Guiding then should be resumed using the **Start Guiding** command button.
- The **GEM Swap** check box did not invert the declination axis direction, so guiding calibration should be performed again after swapping the tube side on GEM.
- The **Guiding camera** tool properly opens message boxes asking the user to cover or uncover the telescope when dark frame is exposed with camera without mechanical shutter.
- The dark frame is properly applied on the **Guiding camera** images when the binning is used.
- The **Web SIPS** application displayed timeout error message when astrometry calculation took too long on the server side.
- When the user entered filtering for the **Star Sheet** lines in the **Photometry** tool, clicking to individual lines highlighted different lines in the sheet.
- The CxUSB camera driver did not work properly with the C5-100M camera in hardware 2×2 binning mode and full frame size.
- ASCOM telescope driver did not enumerate available speeds correctly and thus did not allow to select speed.
- The Garmin GPS USB driver did not always connect to the GSP receiver in 64-bit SIPS version.

Version 4.1.4 released on October 30th, 2024.

Updates:

- The **Web SIPS** web-browser based interface was extended with Focusing tab, allowing to invoke automatic focusing as well as to focus imaging camera manually.
- The **Web SIPS** application added a possibility to download preview images in the loss-less PNG format instead of default JPEG.
- The **Remote Access Web Server** with enabled local storage now utilizes all available CPU cores to create ZIP files.
- The **Photometry** **Field Description** pane newly contains a tool to automatically assign comparison stars to defined variable stars according to either B-V or J-K color indexes.
- When guider camera or main imaging camera use the telescope pulse-guide interface for guiding corrections and the telescope driver becomes temporarily offline, guiding is not entirely stopped, but only currently calculated correction is skipped and guiding continues.

Bug fixes:

- The **Web SIPS** web-browser based interface fixed issue with Imaging camera cooling charts interruptions when the browser was offline.
- The **Web SIPS** image preview function did not respect maximal allowed zoom, affecting especially images from high-resolution cameras.
- The **Web SIPS** fixed issues with guider calibration.
- The **Web SIPS** download ZIP was limited to 4 GB file size.
- Fixed issue in **Photometry** **Star Sheet** with star filter active. Star names updates (adding, modification or deleting of name) were not preserved when the filter was turned off or new filter was applied.
- The image soft-binning function updates also FITS headers containing pixel angular size, which is necessary for proper astrometry solution.
- The CxUSB driver properly orients image of the C1+9000 camera.

Version 4.1.5 released on October 31st, 2024.

Bug fixes:

- SIPS could wrongly determine the maximal allowed zoom on some images, not allowing to enlarge them beyond current size neither by mouse wheel nor by the zoom slider in the **View** ribbon.

Version 4.1.7 released on November 14th, 2024.

Updates:

- The **Web Server configuration** dialog box newly allows the administrator to enforce minimum password length.
- The **Web SIPS** web-browser based interface now includes Image information, showing whole image or selected frame statistics, in the Imager Preview page.
- The Imager Preview page of the **Web SIPS** application allows direct download of the current image in the FITS format.
- The **Web SIPS** newly allows settings of one of the predefined image stretching types.
- The Weather station data displayed by the **Web SIPS** now includes also history graphs of individual values.
- The **Web SIPS** application now supports remote administration (user management, password changes, …).

Bug fixes:

- The **Web SIPS** did not allow downloading of individual FITS files from the server local storage.
- The **Web SIPS** was not able to create and download a ZIP file of FITS images, if there were non-ASCII characters (typically some non-English language-specific character) used in the SIPS Web Server root path.
- The **Web SIPS** Telescope page fixes an issue with direct control of the mount movement using the direction buttons.

Version 4.1.8 released on November 20th, 2024.

Updates:

- The **Web SIPS** web-browser based interface **Image information** updates to the sub-frame selected in the **Preview** page.
- If the **Automatically connect to selected driver** checkbox function is active for any device, the web-based application shows it as connected device (the **Connect** button is disabled), but with an hour-glass symbol, indicating that SIPS waits for actual device plug in and the device will be connected as soon as possible.

Bug fixes:

- SIPS appearance on high-DPI screens (above 100%) was fixed:
  - The slider controls adjust sliding box size according to the used DPI.
  - All sheets (for instance, list of stars in the **Astrometry** tool or the **Star Sheet** in the **Photometry** tool etc.) adopt default column width according to defined DPI.
  - Minor fixes of the appearance of the **Web Server Configuration** and **User management** dialog boxes.

Version 4.1.9 released on November 29th, 2024

Updates:

- The SPL language was extended with new embedded function **assigned()**, the SPL version advanced to 3.
- The **Web SIPS** Telescope page shows azimuthal coordinates of the newly entered equatorial coordinates, so the user is informed where the telescope will be pointing after GOTO command.

Bug fixes:

- Fixed the sub-frame coordinates in the **Web SIPS** web-browser based interface **Image information** pane.
- Configuration information is properly saved prior to the host PC shutdown from the **Web SIPS** web-based application. Prior to this update, all user management changes (changed passwords, added or removed users, …) were discarded when SIPS was not properly stopped, but the host PC was remotely shut down.

Version 4.1.10 released on January 10th, 2025

Updates:

- The **Switches** tool was extended to allow to use up to 3 different drivers of switching devices simultaneously.
- The **Weather station** tool was extended to allow to use up to 3 different drivers of weather station devices simultaneously.
- The interfaces for the second and third Switches and Weather station devices was added to the **SPL** device interface, the SPL version advanced to 4.

- The **Program Editor** window status bar was extended with more text-editing related information, like cursor line and columns, editor insert/overwrite or read-only status etc.
- The **Photometry** tool was extended with optional outlier filtering.
- SIPS newly recognizes non-standard FITS header 'EXP_TIME' for image exposure time, in addition to standard 'EXPTIME'.

Bug fixes:

- Selecting a camera to the particular role (imager, guider, or context), which was already selected to different role, cause the original role is canceled and new role is used. However, when the camera in its original role just performed an exposure, SIPS ended with error. New version properly terminates possible camera exposure in progress.
- The **Switches**, **Flaps**, **Weather station**, **Focuser**, and **GPS** tool windows could crash when the **Automatically connect to selected driver** checkbox was checked when no driver were selected.
- When the **Combine monochrome images** tool was used to create either sum or median of stacked frames, SIPS tried to get the image center coordinates, pixels scale etc. from the connected devices, providing such option was chosen for newly acquired images in the **New FITS Headers** dialog box. Now the **Combine monochrome images** tool takes this information from the stacked images, regardless of actually connected devices.
- The **Imaging camera** start exposure did now wait for the **Guiding camera** when the mount slewed to the new position during image dithering.

Version 4.1.11 released on January 13th, 2025

Bug fixes:

- Some ASCOM camera drivers were not visible in the SIPS camera selection dialog.

Version 4.1.12 released on January 17th, 2025

Updates:

- The **Guiding camera** tool uses new, much more robust algorithm for calibration, designed to tolerate mechanically imperfect mounts with significant backlashes in both axes.
- The **Context camera** tool parameter controls were rearranged into three tabs.
- The **Context camera** tool newly allows to transform the camera image (rotation and flipping).
- The Fiedler weather station driver was extended with support of RVT80 and RVT85 humidity and temperature sensors.
- The Fiedler weather station driver supports usage of multiple .ini configuration files with names corresponding to the tab index of the **Weather station** tool. So, multiple instances of the driver can be used simultaneously, each using its own .ini file and thus own serial interface port number.
- The TCS2 switches driver supports usage of multiple .ini configuration files with names corresponding to the tab index of the **Switches** tool. So, multiple instances of the driver can be used simultaneously, each using its own .ini file and thus own serial interface port number.

Bug fixes:

- SIPS properly generates the new FITS image ROWORDER header to inform other software about the bottom-up image vertical axis orientation, as well as reads this header when loading FITS files generated with top-down orientation. However, the row order was not set to bottom-up when an image, created by other software with top-down row order, was loaded and then saved by SIPS.

Version 4.2

Version 4.2 released on February 28th, 2025

Updates:

- The SPL language was extended with new embedded function **norm()**, the SPL version advanced to 5.
- The SPL language newly allows the delay instruction (**pause** and **wait**) within the functions.
- The SPL instruction **print** newly accepts new keyword **cll** (clear line).

- The SIPS REST API, accessible using the SPL language **invoke** instruction, was extended with indexed enumeration commands for devices, filters, and camera read modes.
- New **Program Editor** tool hotkeys were added: <Ctrl>+<T> for switching to Text mode, <Ctrl>+<G> for switching to Graph mode, and <Ctrl>+<R> for running the program.
- The robust guiding calibration, introduced for the **Guiding camera** tool in version 4.1.12, was implemented also for the inter-image guiding with the main imaging camera.
- The **Web SIPS** application allows explicit command that the telescope tube was swapped to different side of the pier than the side on which the guiding was calibrated.

Bug fixes:

- Guider calibration should return the telescope to the point in which the guiding started.
- The SPL language compiler does not generate false errors that a function does not return a value.

Version 4.2.1 released on April 2nd, 2025

Updates:

- Added native driver support for the new Moravian Instruments **C2-46000** CMOS camera.
- The **Image Math and Filters** tool was extended with the ability to perform **image and scalar** operations as well as **image filter** functions on the whole image list at once. Destination is the not a single image, but another list, into which the newly created processed images are added.
- The SPL language allows to define **remark .. end_remark** section directly following the procedure or function header, as well as the entire program.
- The SPL language was extended with new embedded functions **dms()** and **hms()**, converting angle in degrees/hours, minutes, and seconds into radians. The SPL version advanced to 6.
- The **Web Server** configuration newly allows enabling of public access to weather station data for any user, even without login credentials.
- The **Web SIPS** application was extended with audible alarm sounds, like the desktop SIPS version. This allows the user to be notified on certain situations, like for instance loss of guiding stars, telescope crossing of specified azimuth or altitude etc.

Bug fixes:

- The **Photometry task** dialog box wrongly interpreted negative declination with zero degrees, when the negative sign was placed in the degrees count box (negative zero was not interpreted as negative).
- Fixed the automatic scaling of the R.A. speed when the telescope moved to new declination. This issue affected both **Guiding camera** and **Inter-image guiding** with main imaging camera.
- The minimal movement distance, required by guiding calibration procedure, is scaled in R.A direction according to the current declination. This allows proper calibration also when the telescope points close to the pole and the actual image shift after R.A. movement is very small.
- The SPL language compiler properly interprets the **for** loop step value, stated after the **by** keyword.
- Fixed an issue with displaying of the number of already acquired images in the **Web SIPS** application, which occurred when the context was switched to another device and back to imaging camera during counter exposure series.

Version 4.2.2 released on April 23rd, 2025

Updates:

- The new **Find Star** ribbon pane was added to the **Astrometry** tool.
- The **VSX Catalog Lookup** function in the **Astrometry** tool copies the object primary name in the VSX catalog to the **Name** item of the star sheet.

Bug fixes:

- The C2-46000 camera driver fixes the problem with sub-frame (ROI) image read.

Version 4.2.3 released on May 5th, 2025

Updates:

- The **Astrometry** tool introduced the online light curve pane, controlled from the **Brightness Monitor** ribbon pane.

Bug fixes:

- The function of pasting of the decimal degrees value into the Degrees or Hours count box of the DMS/HMS controls, used for e.g. equatorial coordinates or geographic location, wrongly calculated the minutes and seconds if the decimal degree value was negative.
- The astrometry solution could fail when the brightest star in the image appeared very close to the image edge.

Version 4.2.4 released on June 2nd, 2025

Updates:

- The SPL Language instruction **if** was extended with **elsif** branches.
  - The SPL version was advanced to 7.
- The **Settings** ribbon pane introduced the **Observatory Setup** dialog box, which allows to define observatory name, location, telescope parameters etc.
- The **New FITS Headers** dialog box is newly capable to use information defined in the **Observatory Setup** dialog box to automatically fill FITS headers (FOCALLEN, OBSERVAT, INSTRUME, LAT—OBS, LONG-OBS, ELEV-OBS). The ability to read location from connected devices was removed from the **New FITS Headers** dialog box and moved to the **Observatory Setup** dialog, which newly functions as the single universal place to define observatory location.
- The Telescope tool uses the location defined in the **Observatory Setup** dialog box as the default new value for updating telescope location.
- The GUI controls in the **Telescope** tool window was re-arranged into three tabs (**Control**, **Settings**, and **Alarms**), which allowed to significantly shrunk the window depth, preserving the screen area.

Version 4.2.5 released on August 1st, 2025

Updates:

- The **set** and **get** blocks of the SPL **call** instruction, as well as the set block of **functions** invoked within expressions, now support shortened parameter passing syntax. This means that if both left and right sides of the assignments are the same, it is enough to write just single identifier without **:=** delimiter and once more repeated identifier.
  - The **Program Editor Options** dialog box allows to define if the SPL source code, generated from the Graph representation, will use shortened assignment syntax or not.
- The REST API "v1/telescope" was extended with **find_object** command.
- The REST API "v1/imager" was extended with autofocus-related commands:
  - **focus_state**
  - **autofocus_start**
  - **autofocus_active_frame**
  - **autofocus_stop**
- The SPL native program **jd** added functions:
  - **ToDateString**
  - **DateToDateString**
- The SPL native program **sips** was extended with:
  - constants for undefined values (undefined RA, undefined magnitude, …)
  - **GetObservatoryParameters** procedure
- The SPL native program **image** added procedures:
  - **FindStars**
  - **EnumerateStars**
- The SPL version was advanced to 8.
- The CxUSB camera driver reads the sensor line time on every time-stamp read, not only on the very first time one. This ensures the line time value is always kept on actual value for the case the more precise time is

available over time (e.g. when more satellites are acquired and the 1PPS time is provided with even higher precision).

Bug fixes:

- The Program Editor tool ensures the number literals of type **real** are always generated with decimal point (even if the number has no fraction part, the **.0** is still added to its representation in the source text), so its type is properly maintained during compilation.
- The **Program Editor** properly opens imported program with error and places the cursor on the error location instead of creating of a new tab with empty program.
- The Color picker dialog box is displayed properly on Hi-DPI displays (> 100%).

Version 4.2.6 released on August 13th, 2025

Updates:

- The Photometry tool can display the **g'-r'** color index in the Star Sheet, based on the Sloan color system. Of course, a catalog containing the Sloan magnitudes must be used for astrometry processing (e.g. the UCAC4 catalog).

Bug fixes:

- The Program Editor tool in the **Text** mode properly colors the remark .. end_remark source text section when the cursor appears in the middle of the **remark** keyword while rolling the text.
- The SPL compiler fixed evaluation of the logical operators (**and**, **or**, **xor**, **not**) applied on integer numbers.
- The SPL compiler fixed termination of the program upon the **Stop Program** button clicks. Previously, the current instruction block (e.g. one pass of the **loop** or the **then** or **else** block of the **if** instruction etc.) was finished prior to breaking of the program.
- The **GetObservatoryParameters** procedure of the SPL native program **sips** returned swapped values of the longitude and latitude geographic coordinates.

Version 4.2.7 released on September 9th, 2025

Updates:

- The **image** native program procedures **Rotate90**, **Rotate180**, **Rotate270**, **Rotate**, **FlipHorizontal**, **FlipVertical**, **Bin**, **Resample**, and **Crop** were extended with **CreateNew** boolean input parameter and **NewImageName** string output parameter, allowing to optionally create new image instead of performing of the image transformation on the existing image.
- The SPL version was advanced to 9.
- The Context camera window remains visible (despite its settings cannot be modified) even if the SPL script is running.
- Also the Astrometry tool can display the **g'-r'** color index, based on the Sloan color system.
- The behavior of the Tools and Devices ribbon buttons was slightly updated:
    - In addition to current behavior of clicking on the ribbon button, which:
        - Creates a tool window if it does not exist.
        - Restores (shows) a tool window if it is hidden.
        - Moves a tool window to top (above other tool windows) if it is behind other tools.
    - A new behavior was added:
        - Click the button, which window is shown and on top, minimizes (hides) the tool window.
- The Switches tool sets analog input controls to the actual values read from the device upon startup.
- Controls displayed in the tabs (e.g. Imaging camera, Telescope, etc.) now respond to the <Tab> and <Shift><Tab> keys by selecting of the following or previous control, the same way like in the non-tabbed dialog boxes.

Bug fixes:

- The text editor in the Program Editor tool fixed bugs:

    - The text is properly colored (color-highlighted keywords, different color for number and string literals etc.) even if the currently displayed portion if the text contains selected text block. This was the case after the Find command, which scrolled text to the found string as well as selected (and highlighted) the found string to the text block.
    - The **Find** and **Replace** dialog windows adjust themselves properly to the Hi-DPI displays.
- Running of the **SPL** program no longer creates a new image folders for each day, even if the option is selected in the **Web Server Configuration**.
- Device drivers based on serial-line (COM port) connection (mmfocuser, meade, mmswitches, and nexstar) could skip sending of some commands to the respective device, providing the command was issued prior to finishing of the currently executed command (this bug was introduced in the SIPS v4.2.1).

Version 4.2.8 released on September 15th, 2025

Updates:

- New procedures were added to the SPL native program **image**:
    - **Sum** creates sum of two images.
    - **Difference** creates difference between two images.
    - **Add** increases all pixels of an image by a defined integer value.
    - **Subtract** decreases all pixels of an image by a defined integer value.
    - **Multiply** multiplies all pixels of an image by a defined real value.
    - **Divide** divides all pixels of an image by a defined real value.
- The SPL version was advanced to 10.
- A new button, which sets the telescope mount **New R.A.** and **New Dec.** coordinates the with the coordinates of the selected star, was added to the **Astrometry** tool.

Version 4.3

Version 4.3.0 released on January 27th, 2026

Updates:

- Both **Astrometry** and **Photometry** tools newly support Ensemble photometry (see **Appendix C: A short introduction to photometry** for details).
    - The **Astrometry** tool contains new ribbon pane **Ensemble Photometry**.
    - Usage of the Ensemble photometry is also supported in the **Brightness Monitor**, which is capable to show light curves of three independent stars, instead of VAR/CMP/CHK charts shown in Relative photometry mode.
    - The **Photometry** tool contains new ribbon pane **Ensemble Photometry**.
    - Ensemble photometry may be included into **Photometry tasks**.
    - Ensemble photometry mode can be toggled by a new tool button in the **Mark Stars** ribbon pane. Marking stars functionality changes in the Ensemble photometry mode, as there are no comparison stars.
    - Also, the **Field description** is updated to support the Ensemble photometry.
    - **Photometry report** containing extended attributes can be created for ensemble light curves.
- The **Remote Access and Alpaca Web Server** was extended with **ASCOM Alpaca** mode, in which the SIPS works as the Alpaca-compatible server offering all connected devices to the remote clients.
- The SIPS web server newly handles the mDNS standard in both Web SIPS and Alpaca modes. So, SIPS web server can be accessed from the local network with “sips.local” name (the name in the front of the “.local” domain is user configurable, “sips” is just the default value) without any DNS configuration.
- The star **g’-r’** color index can now be included into CSV files exported by the **Photometry** tool.

Bug fixes:

- If the option “Automatically connect to the selected Imaging/Guiding/Context camera only” is checked and the camera is unplugged during the SIPS startup, the driver handling the camera could cause SIPS crashes.
- Filter names, as well as read-mode names, read from FITS headers, are trimmed from leading and trailing spaces. So, if for instance a filter name is used to create FITS file name, the superfluous spaces do not appear in the file name.

- **Context camera** did not continue with exposures after unplug/plug event, even if the **Continue Exposures on connect** option was checked in the **Settings** ribbon pane.
- The **Dome** driver continues with following of the telescope azimuth even after the driver disconnect/connect event, providing the **Synchronize dome slit with telescope** option was checked.

Version 4.3.1 released on February 11th, 2026

Updates:

- New **Remove Gradient** tool was added.
- The **Create Color image** tool analyzes the FILTER header as well as the image file name itself (if the image is saved) for occurrence of the "red", "green", and "blue" strings for the newly added images and automatically assigns the corresponding color to them.
- Stars found in images are no longer highlighted using one aperture used for search, but with the automatically determined aperture instead, reflecting the star flux (brightness).

Bug fixes:

- General transformation (arbitrary angle rotation) of images with 32-bit pixels could damage bright image areas due to integer overflow. This transformation is used e.g. when stacking images, not only for explicit rotation. While stacking of 16-bit images into a 32-bit one was unaffected, a combination of already stacked 32-bit sums into RGB image could lead to artifacts within bright stars.
- The maximal allowed zoom of very large images was already limited not to exceed the 16-bit integer size limit of Windows bitmaps. Another limitation was newly added to protect SIPS from an attempt to allocate zoomed-out Windows Device Independent Bitmap (DIB) with total size exceeding maximum allowed limit. This could happen with color DIBs only, with each pixel consuming 3 bytes.

Version 4.3.2 released on April 1st, 2026

Updates:

- **Constants** defined in **SPL** may optionally define explicit type. The **Code formatting options** dialog box allows to define if types are to be included to constant definition when the program source text is generated from **Program Editor** Graph mode.
- The SPL version was advanced to 11.

Bug fixes:

- The ASCOM guiding camera driver, used to control ASCOM-compatible cameras from SIPS, could send wrong declination pulse length when the mount is guided through the camera "Autoguider" port.
- When image was saved under new name (and thus renamed) in **Tabbed workspace**, the original name remained in the image window title, visible after the workspace was switched back to the windowed mode.
- An attempt to close the **Program Editor** window while program was running resulted in repeated opening of confirmation dialog.
- The **Program Editor** in the **Graph** mode could report false error message "Unknown identifier" when compiling expressions containing local constants or variables immediately after saving program.

Version 4.4

Version 4.4.0 released on May 25th, 2026

Updates:

- The **SPL** compiler was extended with arrays:
  - Definition of array **constants**
  - Declaration of array **variables**, including optional individual items initialization
  - Assignment of whole arrays using **let** instructions
  - Passing whole arrays to/from **procedures** and **functions**
  - Passing whole arrays to/from **invoke** instruction (JSON arrays of simple type are mapped to SPL arrays)
  - New embedded function **high** returns highest index of selected array dimension

    - New instruction **resize** for changing dimension sizes during runtime
- The SPL version was advanced to 12.
- The **Astrometry** tool newly displays also field rotation relatively to north direction.
- The focusing **History** pane, displayed in the **Focus** tab of the **Imaging camera** control tool, shows a fitted Gaussian profile, a mean value of which is used as optimal focus position.
- The MM4 Focuser driver (mmfocuser.dll) and TCS2/TCS3 telescope control unit driver (mmswitches.dll) no longer need a COMx serial port name properly defined in the .ini file. These drivers now automatically connect using proper COM port, on which a particular device is connected.

Bug fixes:

- SPL did not allow definition of local constant or variable with the same name as the global constant or variable.

Version 4.4.1 released on May 28th, 2026

Updates:

- When there is an image with calculated astrometric solution selected in SIPS workspace, the **Astrometry** tool new hotkey <Ctlr>+<K> copies a string formatted “RA: HHh MMm SS.sss Dec: +DD° MM' SS.ss" Rot: DDD.dd°” with image center coordinates and rotation to clipboard.

Version 4.4.2 released on June 19th, 2026

Updates:

- The SPL language was extended with new embedded functions:
    - **shl** shifts bis the integer number to left
    - **shr** shifts bis the integer number to right
    - **max** returns greater of the two numeric arguments
    - **min** returns smaller of the two numeric arguments
- Multiple occurrences of the **cll** keyword in the **print** instruction list of values deletes multiple lines backwards in the program output.
- The SPL version was advanced to 13.

Bug fixes:

- The SPL **invoke** endpoint “v1/imager/set_exposure_parameters” did not set the “read_mode” parameter correctly.
- SPL native procedure image.EnumerateStars() failed when the star index exceeded total number of detected stars.
- SPL native procedure list.New() caused the newly created list name was duplicated in the global list of image lists, which prohibited proper list selection.