**Jim’s PixInsight Cribsheet**

Rev33

**Image Processing Steps**:

1. Build Bias (including Superbias), Dark & Flat Master Calibration Frames
2. Select Light Frames using SubFrameSelector
3. Calibrate Image Stacks by Channel
4. Run Cosmetic Correction to further clean images
5. Separately Star Align Channel Stacks
6. Dynamic Crop each Stack
7. Integrate and, if undersampled, Drizzle combine each Aligned Channel Set
8. Dynamic Background Extraction for each Channel
9. Star Align RGB/NB Channel Stacks
10. Dynamic Crop R, G, & B/NB) Stacks
11. Apply Linear Fit to equalize R, G, & B/NB stack histograms
12. If shooting RGB only, create a synthetic Lum (if NB, use Ha image as Lum)
13. Create RGB/NB using ChannelCombination
14. If binned, Resample RGB Image
15. Neutralize Background in RGB Image (may not be needed if Step 10 used)
16. RGB Color Calibration (may not be needed if Step 10 used)
17. RGB/NB Linear Color Saturation with Curves Transformation\*
18. If necessary, run second Dynamic Background Extraction on RGB image\*
19. Deconvolve Lum to increase detail and tighten stars\*
20. MLT noise reduction of Lum and RGB/NB by removing first wavelet layer only\*
21. Consider very mild Unsharp mask to subtly sharpen features\*
22. If needed, Linear TGVD and/or MLT Noise reduction of RGB/NB and Lum\*
23. If separate Lum, use Channel Extraction to extract RGB L, a, and b channels
24. If separate Lum, ChannelCombine a and b channels with separate Lum
25. RGB and Ha and/or OIII Combine (if needed) w/NBRGBCombination or PixelMath
26. Histogram Transform Stretch RGB/NB & Lum Images
27. MaskedStretch for initial stretch of clone (used to create softer stars)
28. Increase Color Saturation (Saturation or Curves)\*
29. Use SHO-AIP Script or other techniques for NB color adjustments
30. Apply SCNR to remove excess green
31. Non-Linear Noise Reduction of RGB/NB and Lum (MMT and/or TGVD)\*
32. HT Touchup of RGB/NB and Lum (reset black point w/o clipping)
33. Star Align RGB/NB and Lum Images
34. LinearFit Lum & L component of RGB/NB images, then do LRGB/LNB Combine
35. TVGD noise reduction\*
36. HT Touch-up (black point reset)
37. If needed, for HDR image use HT to brighten target, w/o saturating\*\*
38. HDRMT to increase detail\*
39. Morphological Transformation to tighten stars\*
40. HT Touch-up (black point reset)
41. MLT and/or MMT for sharpening\*
42. Dark Structure Enhancement\*
43. LHE application\*
44. Saturation Curve
* = mask recommended; for Color Saturation consider ColorMask

NB = NarrowBand imaging with Ha, OIII, and SII filters

***Philosophical Considerations (What PixInsight has taught me, I think)***

1. *When shooting RGB images, based on discussions threads in the PI Forum with PI developer Juan Conejero and others, I now only image RGB frames (i.e., I shoot no luminance frames) using 1x1 binning and use the R, G, & B master integrated images to create a synthetic Luminance frame as per step 9, above. If you are interested in the discussion, see the attached:* [*http://pixinsight.com/forum/index.php?topic=6042.15*](http://pixinsight.com/forum/index.php?topic=6042.15)
2. *PixInsight has taught me that SNR quality definitely trumps target quantity. PI rewards taking the time to maximize SNR for a single target versus padding your target list. When I was just starting, I would shoot up to 6 targets a night. Now I shoot one target over two nights if possible. That typically gives me over 25 ten minute subframes with each of my R, G, and B filters (or my NB filters).*
3. *Take the time to do the little things that add real value both in the field and when building images. At the top of that list is building high quality bias (and superbias), dark and flat master frames. I have a number of notes below on how I build each so I won’t repeat that here, but it is worth mentioning that I try to shoot as many calibration subframes as possible. That means a minimum of 64 (currently using 100) dark frames for each dark master iteration (I shoot my darks to match the exact time of my lights), 25 or 36 frames for each flat in the field, and 400 frames to build my bias. And, if you have a cooled CCD, you can shoot those darks and bias frames at home during full moons or cloudy weather so it’s not like you are wasting imaging time to build quality sets. And remember to reshoot those darks and bias frames periodically (e.g., at least every 6 months)*
4. *I know several of the noise reduction tools can be used in the linear stage, but be gentle to avoid stretching artifacts that are hardly noticeable in the linear stage that become real processing problems in the non-linear stage. I have learned that unless you have really noisy images (and that probably means you are not doing enough to maximize SNR; see point 2), there is no reason to do heavy noise reduction before you permanently stretch your images. A light touch here goes a long way. That said, I have been really impressed with how well MLT works for noise reduction in the linear stage so if you are using that tool, you are likely going to be fine.*
5. *Experiment, experiment, experiment! Anything in this cribsheet re settings are only intended to be starting points or settings that work for my rig. You owe it to yourself to make tweaks. The beauty of PixInsight is the combination of its logic with an infinite ability to tweak the tools to do what you need. It really is amazing software and by playing with it you start to get a sense of what all those settings do. I have yet to process an image where I don’t learn something new or discover a better way of doing it, often due to the great suggestions from the pros in the Forum.*
6. *Please let me know if you find this helpful, but also where I can make it better. This is especially true if I just have something flat out wrong. It does no one any good for me to keep repeating the same error in revision after revision. It’s also why this cribsheet is up to Rev31, and I am certain there will be at least another 20 to replace this one. You can find me on the Forum on a regular basis, but also feel free to drop me an email at* *jkmorse57@gmail.com**.*

**Setting up your Workspace:**

PI is a very complicated set of tools and you will want to make your life easier by keeping the tools you use most handy and easily accessible. While you should experiment with what works best for you, here is a screenshot of the set up that I have adopted as my ‘base’.



Let me explain each detail in turn.

1. Note the changes in the stack of ‘Explorers’ that are located in the upper left corner in the default startup. I like to arrange the explorers on the side of the work screen as follows: In the lower left panel I stack the History Explorer, Process Explorer, and Process Console since those are the ones I use most. In the upper right panel I stack Format Explorer, View Explorer, and File Explorer. I close all the others but you will want to check if you want easy access to any of the others for your needs. This arrangement helps me to not open menus when I don’t mean to (which happened all the time when I left things tagged in the upper left corner) and keeps what I do use handy.
2. You can see I moved the toolbars around a bit too. The default is a bit crowded for my taste, making finding things that much slower. As I have it setup, I move the image manipulation tools (zoom in, zoom out, etc.) to the far left, and use the extra space to move the STF and Mask panels to the right, keeping things a bit more separated and easier to find.
3. For the tools themselves, I have found that the system that works best for me is to organize my workspace by opening and minimizing all of my “go-to” tools and then stacking them in likely order of use to the far right of the screen. [Important note: you cannot do this for ‘dynamic’ tools. They must be opened on a one off basis so for them, the best solution is to keep them in your ‘favorite processes’ folder.] I also load all of my favorite PixelMath formulae (they are the ones mentioned elsewhere in this cribsheet) into the ProcessContainer so I have those available with just a mouse click or two. Note: I have now added the “Convolution” and “CosmeticCorrection” tools since I am using those processes now. The above screen capture reflects those additions, as well as LHE, which should have been there all along. I dropped the “Image Container” since I rarely use it in practice.
4. Once you have everything set up the way you like, just save it all as a “Project” under the ‘File’ tab. I just label mine as ‘Base Workspace’ and then open it whenever I start a PixInsight session and it all loads up just the way I like it.
5. The project area gets a lot of tweaking in my setup since I am always experimenting with my workflow and saving the new fix is as easy as overwriting the old project file with a new save. Or, you may want to have different project files for different types of work. Use this convenient process to make your life easier and keep trying new things to maximize your benefit.
6. Obviously, you can also save projects while you are working them as well. One thing I have not used, nor found a real use for yet, is the ability to use multiple workspaces at the same time. I guess you could use it to have several projects going on at once, but I have not seen the benefit of multiple work spaces on one project. If you are using those extra work spaces and find them handy, please drop me a line to explain why you like using them and I will include here.

**Saving Files**

If you have been doing image processing for a while, you will have learned that it’s easy to forget where you left off from one session to the next, or even what you did when in a long processing session. That is compounded with a program like PI where you have so many tools at your fingertips and each one has lots of individual settings.

In order to help manage all of that information, I try to give myself ‘tells’ in the save name itself. Therefore, when I am fairly early in the processing sequence, I may have a file that looks something like this: M51\_Red\_WCS40-26\_DCrop\_DBE. That is shorthand for my M51 red master, integrated using Winsorized Sigma Clipping with a low setting of 4.0 and a high setting of 2.6, and I have Dynamic Cropped and applied the Dynamic Background Extraction routine. There is no magic, just an effort to make my life a little easier when I want to go back and tweak something and need to know exactly what sequence I used in my processing routine so I can get to the right spot.

Another thing to consider is saving files in PI’s native format, which is platform independent. To do so:

- Select EDIT > Global Preferences from the main menu.
- On the Preferences tool window, select the File I/O Settings item on the left side.
- Uncheck the Use native file dialogs option.
- Click the Apply Global button (blue sphere) or press F6.

To quote Juan, “Now when you select File > Open, File > Save As, or make a double click on the workspace, PixInsight will use its own dialog windows instead of the native platform dialogs. Tip: On platform independent file open/save dialogs, you can drag a folder and drop it on the left side to create a shortcut that will be remembered across sessions.”

**FITS Incompatibilities**

One area that may be a source of frustration (at times it has driven me absolutely crazy) is the fact that FITS is not a standard across platforms and therefore you may experience problems in PI due to the fact that it treats FITS files differently than your capture software. I have run into this problem using MaximDL, but I know others have had the issue with other capture software as well. Note that the PI developers try to be extremely disciplined in this area but they can only do so much to solve the problem automatically. If you have not experienced this issue, I am jealous as hell. If, on the other hand, you too run into this problem, usually manifesting itself in weird looking images in PI that, in the case of lights, makes them look blown out (grey before stretching and blown out after) or in the case of normal lights, images that end up overly dark after ImageCalibration, it is resulting from data truncation, most often because the capture software saves images using signed integer FITS when it is brought into PI, you lose all the data represented by negative numbers.

It is critical if you experience this problem, NOT to give up on PI! Instead, look at my new section after the Image Integration Hints. I have solved the problems using one of those methods and I am confident you can as well (particularly using the input hints). If not, write me at jkmorse57@gmail.com or post the problem on the PixInsight forum and we will try and find a fix together.

Note that the PI developers have just introduced a new .XISF file format for saving your image files that, in the long term, may finally put these issues to bed (at least I remain hopeful). The developers have stated that .xisf should be fully compatible with FITS. Also, they have left this an option for now so if you want to use this new format, you will need to go into global preferences and change the default file format from .fits to .xisf. I encourage you to give it a try.

**My Website**

I have finally gotten around to updating my website, including posting the images I have processed in PixInsight**.** I also include lots of other things there, including some tutorials that you might find helpful, particularly if you are relatively new to astroimaging. For example, I try to take the mystery out of creating flat frames. You can find me at [www.jimmorse-astronomy.com](http://www.jimmorse-astronomy.com). Check it out and let me know what you think.

**Tool Notes:**

*Linear vs. Non-Linear Processing Steps:*

**Tools working on linear and non-linear images equally well:**

   AutoHistogram
   CurvesTransform
   DynamicAlignment
   HistogramTransformation
   IntegerResample
   Invert

**MaskedStretch**

MultiscaleLinearTransform Noise Reduction
   RangeSelection
   RGBWorkingSpace assignment
   Rotation

SCNR
   StarAlignment

 **TGVD Noise Reduction**   Statistics

   processes that only alter the physical shape of the image

**Tools best used on linear images:**

   AssistedColorCalibration
   AutomaticBackgroundExtraction
   Color Calibration

CosmeticCorrection
   Fast Rotation
   Deconvolution
   DeconvolutionPreview
   DynamicBackgroundExtraction
   Dynamic Crop
   Dynamic PSF
   GradientHDRComposition
   GradientHDRCompression
   GradientMergeMosaic
   GREYCstoration
   HDRComposition
   ImageCalibration
   ImageIntegration

 LinearFit

MLT Noise Reduction
   ScreenTransferFunction

Superbias

**Tools best used on non-linear:**

   ACDNR

CurvesTransformation - for saturation

 DarkStructureEnhancement
   HDRMultiscaleTransform
   LocalHistogramEqualization
   MorphologicalTransform
   ATWT Noise Reduction

 MMT

 DarkStructureEnhancement

 LHE

**That leaves the following uncategorized:**

**Processes:**

   ChannelCombination
   ChannelExtraction
   ChannelMatch
   CloneStamp
   ColorSaturation
   Convolution
   DefectMap
   DigitalDevelopment
   FourierTransform
   ICCProfileTransformation
   InverseFourierTransform
   LarsonSekanina
   NoiseGenerator
   RestorationFilter
   SimplexNoise
   StarMask
   UnsharpMask

**Scripts:**
Image Analysis:

   ExtractWaveletLayers
   ImageSolver
   NoiseEvaluation

Instrumentation:

   BasicCCDParameters
   CalculateSkyLimitedExposure

Utilities:

   BackgroundEnhance
   CanonBandingReduction
   CosmeticCorrection
   FFTRegistration
   StarHaloReducer

***Image Selection***

SubframeSelector Script to identify best original FITs

a. Expressions:

 i. Approval: FWHMSigma < 2 & SNRWeightSigma > -2

 ii. Weighting: FWHMSigma

***Image Calibration and Integration***

*Calibration Frames:*

*[Note that there is a lot of detail regarding the settings in the ImageCalibration tool that you should familiarize yourself with. Bitli produced some great “unofficial” documentation for this tool at:* [*http://pixinsight.com/forum/index.php?topic=7149.msg48225#msg48225*](http://pixinsight.com/forum/index.php?topic=7149.msg48225#msg48225) *I encourage you to check it out. Areas to explore include whether your set up requires you to check the ‘calibrate’ and/or ‘optimize’ buttons and how to use ‘input hints’ to address particular issues]*

*Bias & Darks:*

Open Image Integration tool

Load relevant Raw Frames

Settings:

Combination: Average

Normalization: No Normalization

Weights: Don’t Care (all weights =1)

Scale estimator: Median Absolute Deviation from the median (MAD)

Pixel Rejection (1):

Winsorized Sigma Clipping

No Normalization

Check all boxes

Pixel Rejection (II):

Sigma Low 4.0

Sigma High 3.0

Range Low 0.0

Range High 0.98

Global apply

*Master Dark Calibration: (Optional: see note below)*

[*Note: I added this step in Rev 29 but is should be considered optional only since it is not the way PI recommends creating a calibrated Master Dark. The stated method in the PI documentation is to**calibrate your Master Dark in the ImageCalibration tool by using both a master bias frame and a master dark, then checking the “calibrate” button in the Master Dark section.*]

After building the master dark, calibrate the master using only your master bias frame, thereby creating a clean, bias-free Master dark for later use.

*Superbias:*

Build a high quality Master bias frame based on above (you can use as few as 20 to 40 subframes, but why would you? It takes no time to get a really solid set. I shoot 400 bias subframes to build my master bias frame. Some argue that with that many subframes you aren’t gaining much by building a superbias, but every little bit helps)

Open Superbias tool, use a setting of 6 or 7 layers (7 if you are working from a weak set of 40 or less bias subframes to build your master bias).

*The one caveat to using a superbias is if your master bias shows banding that is removed by the superbias process. In those circumstances you are better off using the master bias since that will better calibrate your light frames. Test by building a superbias and doing a careful comparison with your masterbias.*

*Flat Frames:*

Step I - Calibration:

Open Image Calibration Tool

Load Raw Flat Frames

Load Appropriate Master Bias or Superbias

[Load Appropriate Master Dark]

[*Some recommend using only a Master Bias, particularly for low noise CCDs, since it creates a cleaner Master Flat in PI; I have adopted this method*]

Global Apply

Step II – Integration:

Open Image Integration Tool

Load Calibrated Flat Frames

Settings:

Combination: Average

Normalization: Multiplicative

Weights: Don’t Care (all weights = 1)

Scale estimator: Iterative k-sigma / biweight midvariance (IKSS)

Pixel Rejection (1):

 Rejection Algorithm: Winsorized Sigma Clipping

 Normalization: Equalize Fluxes

 Check all boxes

Pixel Rejection (2):

Sigma Low 4.0

 Sigma High 3.0

 Range low 0.0

 Range High 0.98

*Light Frames:*

Step I – Image Calibration

Open Image Calibration Tool

Load Master Flat Frame

Load appropriate iteration matching, Master Dark

Load Master bias frame

If using Master Bias, check “calibrate” button in Master Dark section

[*Note 1: I prefer matching Master Dark iteration time to iteration time for light frames over dark scaling and this sequence reflects that preference*]

[*Note 2: I had previously advocated using only an uncalibrated master dark since it already included the bias, but I have been testing and get much better results using a calibrated master dark with bias removed and a separate master bias.*]

Global Apply

Step II - Cosmetic Correction.

Open CosmeticCorrection Tool

Check both the Use Master Dark and Use Auto Detect boxes.

Load a high quality Master Dark frame (not calibrated)

Usually the default values of 3 work best for both (they need to be checked).

Set the path to save the images and run the process.

Check some of the images to be sure it did a good job.

(*Note: This is a new process in my workflow based upon a discussion I had in the PI Forum. When you combine the images after CosmeticCorrection, you should see a much cleaner result. Dithering really helps get the cleanest stacked image. Between CosmeticCorrection and combining the images, you should get a very clean master light frame*. *It is critical, however, to use a calibrated master dark, with bias removed, otherwise you risk adding in artifacts removed in the calibration process.*)

Step III – Star Alignment

Open StarAlignment
Select relevant images
Set best image as Reference Image
Set Output Directory

Check Distortion Correction to correct minor defects

Unless major defects, use standard settings

* Distortion Residual: 0.005
* Iterations 20

In Star Detection, set "Hot pixel removal" to 1
Global Apply

If undersampled data, check “Generate drizzle data”

[Note, it addition to helping with image alignment, I like to use Distortion Correction as another quality check on my subs. When you select a quality reference image and PI cannot find a correction solution after 20 iterations, then you know the image in question may be flawed. For me PI typically finds a solution in 5 or fewer iterations. I take a close look at and generally delete any image from my stack if it takes more than 12-15 iterations for PI to find a solution or where no solution is found. Cross-referencing with the SubFrameSelector script usually confirms that these subs are outliers]

 *ImageIntegration*

Open ImageIntegration
Select registered images

Select best image as Reference Image

Set Image Integration to "Average"
Set Normalization to "Additive" or “Additive w/scaling”
Set Weights to "Noise evaluation"

Scale estimator: Iterative k-sigma / biweight midvariance (IKSS)
Check "Evaluate noise"

[*Note: As for Combination method, never use Median for production work. You will lose around 20% of signal as compared to Average integration. Always use Average and the appropriate pixel rejection algorithm.*]

Pixel Rejection (1) settings:

≤ 5 images: Percentile or Averaged Sigma Clipping

6 – 9 images: Sigma or Averaged Sigma Clipping

10 – 14 images: Sigma Clipping or WSC

15 – 24 images: Winsorized Sigma Clipping (WSC)

≥25 images: Linear Fit Clipping or WSC

With Linear Fit I have been advised that you have to crank the values as high as possible, as long as you get rid of "all" the outliers (hot and cold pixels, sat trails, etc.). Use 7 sigma low and 5 sigma high, as a starting point.

[*Note: My experience has been that I get better rejection results with WCS than I do with Linear Fit, at least with data sets of around 30 subframes; but feel free to try both if you have a large data set and compare the results.*]

Rejection Normalization: Scale + Zero Offset

Do Not Check “Clip High Range” or “Clip Low Range”

Critical to compare results of clipping choices when ≤ 9 images (though I recommend always comparing different settings in the base case anyway)

Pixel Rejection (2) settings:

For all rejection algorithms listed above, start with defaults except for Sigma clipping, in which case start with 4.0/4.0 for Lums, 3.0/2.0 for color

[e.g., f*or m*y *system, a setting in WSC of 4.0 for Sigma low and 2.6 for Sigma high is an effective starting point*]

Adjust each setting separately by small increments until “best fit” (I find it’s more important to adjust Sigma high than Sigma low)

Test using SubFrameSelector (can use ROI to speed processing)

Once best fit achieved, repeat with other setting

[Note: lower number = more rejected pixels]

If using undersampled data, insert .drz files generated in StarAlignment

Global Apply

Examine Processing Console data

Check High and Low Rejection Images for abnormalities with STF

For undersampled data, open Drizzle Integration tool

Insert .drz files generated in ImageIntegration

Output scale of 2 is recommended and you would need a massive amount of data to justify going higher (image size will be square of scale, i.e., 4x size for an output scale of 2)

Drop Shrink factor of .9 is fine (range is 0.7 to 1.0, smaller is sharper but with insufficient data may create artifacts such a “dry” pixels)

[*Note: I cannot say enough about the benefits of using drizzle if you take undersampled images. Undersampling is basically a measure of how good your resolution is on an arcsecond per pixel (arcsec/p) basis, where the higher the number, the lower the resolution and the more undersampled the image is. As an example, one of the two setups I use for image capture is an SBIG STF8300 with various Canon lenses, which result in images that have a range of 32 arcsec/p for my 35mm lens, down to 3.75 arcsec/p for my 300mm lens. Those compare to a resolution of 0.72 arcsec/p for my F16M/CDK12.5 combination. Obviously, the lens resolutions result in varying degrees of undersampling that drizzle does an amazing job of improving. The amount of improvement from small, blocky stars to well-rounded orbs is almost magical. The key is to take lots of subframes, the more ttter. I always take at least 30 subframes per channel so that I can be assured a set of at least 25 per channel to work with. Try it, you won’t be disappointed.*]

[*Note 2: I have started using Drizzle for narrowband images, even when not undersampled. I find that using drizzle really helps define the stars which typically suffer from the use of narrowband filters.*]

Do this separately for each channel set

For old Data sets calibrated outside PixInsight, use “Input Hints” as appropriate

(See ImageIntegration Documentation for nomenclature)

In ImageIntegration, as for image weighting, the choice of a reference image is theoretically irrelevant under ideal conditions. However, under less-than-ideal conditions, these points can help you to select an optimal reference image for integration:

- If there are varying gradients in the data set, select the image with the least/weakest gradients. Gradients complicate calculation of fundamental statistical properties such as scale and location.

- Try to select the best image in terms of SNR. In general, this corresponds to the image with the least noise estimate.

- Avoid selecting a reference image with strong artifacts, such as plane and satellite trails, etc.

The SubframeSelector and the Blink tool are your best friends in all image analysis and selection tasks

#### **ImageIntegration Usage Hints (by Juan Conejero)**

* Use the [SubframeSelector](http://pixinsight.com/doc/scripts/SubframeSelector/SubframeSelector.html) script and the [Blink](http://pixinsight.com/doc/tools/Blink/Blink.html) tool to analyze your data and grade your images, both quantitatively and qualitatively by visual inspection.
* Don't use the [BatchPreprocessing](http://pixinsight.com/doc/scripts/BatchPreprocessing/BatchPreprocessing.html) script to integrate your light frames. In most cases, BatchPreprocessing does a fine job for generation of master calibration frames, image calibration and registration. However, integration of light frames is a critical process requiring manual intervention to fine tune pixel rejection and image combination parameters. The integrated output of BatchPreprocessing can be used as a *quick preview* of the image that can be achieved, but it is not the optimal image by any means, and many times you're quite likely to get a grossly wrong result (e.g., invalid rejection of plane and satellite trails, etc.).
* Refine your pixel rejection parameters to achieve the highest possible effective noise reduction with appropriate rejection of spurious data (plane and satellite trails, cosmic ray impacts, CCD defective pixels, etc.). It is strongly recommend you read an excellent presentation by Jordi Gallego, where he describes this task with detailed practical examples and real-world tests. Although this presentation describes an old version of the ImageIntegration tool, the fundamental concepts exposed remain equally valid. Find it here: <http://www.astrosurf.com/jordigallego/articles.html>
* Experiment with different [scale estimators](http://pixinsight.com/doc/tools/ImageIntegration/ImageIntegration.html#scale_estimators) to discover which ones provide the best results for your data. We suggest you compare the results achieved with the IKSS, MAD and average absolute deviation estimators as a starting point.
* Unless you have a strong reason to do otherwise, use the noise evaluation [weighting method](http://pixinsight.com/doc/tools/ImageIntegration/ImageIntegration.html#image_weighting). In all of PI’s tests this method consistently leads to the highest SNR improvement in the integrated images.
* Never use [median combination](http://pixinsight.com/doc/tools/ImageIntegration/ImageIntegration.html#median_combination) for production work. Median combination will lead to a 20% loss of signal with respect to average combination (or more for small image sets). Always use [average combination](http://pixinsight.com/doc/tools/ImageIntegration/ImageIntegration.html#average_combination) and the appropriate [pixel rejection algorithm](http://pixinsight.com/doc/tools/ImageIntegration/ImageIntegration.html#pixel_rejection). Use median combination exclusively as a counter-test to evaluate rejection performance.
* Never use [min/max rejection](http://pixinsight.com/doc/tools/ImageIntegration/ImageIntegration.html#minmax_clipping) for production work. The min/max method rejects a fixed number of samples from each pixel stack without any statistical basis. It will lead to a constant loss of signal proportional to the square root of the number of clipped pixels. While the importance of this loss is inversely proportional to the number of integrated images, better results can *always* be achieved with more sophisticated rejection algorithms. Use min/max exclusively as a counter-test to evaluate the performance of other algorithms.
* For integration of master bias and master dark frames, you may want to disable the [Evaluate noise](http://pixinsight.com/doc/tools/ImageIntegration/ImageIntegration.html#evaluate_noise) option to accelerate the process, since a quality assessment is normally not necessary in these cases.
* Use [regions of interest](http://pixinsight.com/doc/tools/ImageIntegration/ImageIntegration.html#region_of_interest) (ROI) to accelerate repeated tests for the same data set.
* If you have to integrate images generated by other applications, use [input hints](http://pixinsight.com/doc/tools/ImageIntegration/ImageIntegration.html#format_hints) to adapt the alien data to the PixInsight platform. In particular, you probably will have to use the "upper-range 65535" input hint for the FITS format. A much better solution is: stop using those applications and calibrate and register your images with PixInsight.

**Fixing FITS Incompatibilities**

If you experience the types of incompatibilities I mentioned in the introduction, try the following until you hit on a solution:

*First*

* Open the Format Explorer (access under “View > Explorer Windows”)
* Double click on “FITS”
* Make sure the top two boxes are checked
	+ “Write scaling keywords . . .”
	+ “Signed Integer images . . .”

*Second*

* Where possible, particularly in the Image Calibration tool, add the following to the “input hints” line under “Format Hints”:
	+ lower-range 0 upper-range 65535

*Third*

* If the above does not work, insert the following in the “input hints” line:
	+ Signed-is-physical

Stop using input hints *AFTER* you have successfully saved an image after applying a PI tool; i.e, only use input hints for Light and Flat subframes in ImageCalibration and only use input hints for Dark and Bias subframes in ImageIntegration (after first PI application for a given file type, PI saves images in the proper PI format (0,1) and no longer need input hints)

For an excellent explanation of what is happening to cause this problem, check out the following threads from Juan Conejero:

<http://pixinsight.com/forum/index.php?topic=3118.msg21348>

<http://pixinsight.com/forum/index.php?topic=3474,msg23973>

<http://pixinsight.com/forum/index.php?topic=3800.msg26086>

Another solution that seems to work if you are suffering these types of issues is to switch from capturing your images in FITs format in the first place. For MaximDL users (and possibly others, though I only know MaximDL first hand) is that you can switch your capture format to unsigned TIFF format. Then, when you export those images to PI, they should work just fine and PI will then convert them to a PI friendly format (either FITs or .xisf, at your option) during the first pre-processing step (typically ImageCalibration). One note of caution, however. Saving your captures in the TIFF format will mean you lose the FITs header information. This is likely an issue if you use the Batch Pre-processing (BPP) routine since it relies in part on FITs header information. If this information is important to you, but you are still having the issues I describe above, you may want to capture using FITs, then do a batch convert to TIFF as an option.

*Black centers of stars*

Another issue some of us have seen using FITs files captured in third party software is a situation where, in PI, the centers of stars end up black and read zero counts in PI. A handy work-around that has worked for me and others is to do a batch conversion of those files into 32 bit floating point files. Note you may also need to use one or more of the solutions above, if the conversion alone does not fix all of your compatibility issues, but between these sets of solutions, you should be able to overcome any incompatibilities. Note, the TIFF fix mentioned above also seems to fix this problem as well so that is another option.

If, despite these ideas, you are still having issues, please contact me and post the problem on the PI Forum to see if we can find a solution.

***Linear Processes***

*Create Synthetic Lum:*

In general, the unweighted average of individual RGB components does not lead to an optimal result in signal-to-noise ratio terms. An optimal synthetic luminance must assign different weights to each color component, based on scaled noise estimates.

This process can be done very easily with the ImageIntegration tool. Open ImageIntegration and select the RGB files. Use one of the RGB channels as the reference for integration.

Then leave all tool parameters by default (you can click the Reset button to make sure) and click the Apply Global button. The relevant parameters are as follows:

- Combination = Average
- Normalization = additive with scaling
- Weights = Noise evaluation
- Scale estimator = iterative k-sigma
- Generate integrated image = enabled
- Evaluate noise = enabled
- Pixel rejection = No rejection
- Clip low range = disabled
- Clip high range = disabled

You can make several tests with different scale estimators and select the one that yields the highest noise reduction. The integration result is the optimal luminance image that you can treat in the usual way (deconvolve if appropriate, stretch to match the implicit RGB luminance, combine with RGB using LRGBCombination, etc).

*Narrow Band Imaging*

*Hubble Palette*

*This is a very challenging area and so I have included a number of suggested ways forward gleaned from a number of excellent suggestions on the PI Forum. This is very much an artistic effort. The Hubble palette and derivatives from it are very much a matter of taste since the color combinations are “artificial”; though many would argue that this applies to astrophotography as a whole and not just to narrowband images.*

*In any event, this is an area that may try your patience, but getting it right is hugely rewarding and therefore worth the effort; just like the rest of this great hobby/passion. And please drop me a line about your successes so I can share them with the group. We are all in this together.*

When applying the Hubble Palette, I used to use the following in ChannelCombination but this will leave you with lots of color tweaking to do since Ha as Green dominates:

* OIII = Blue Channel
* Ha = Green Channel
* SII\*0.8+Ha\*0.2 = Red Channel

*Note I have seen others who use different mixes, such as mixing 50/50 SII and Ha to create the Red Channel, 85/15 OIII and Ha for Green (yes I have that right, OIII is the major component), and 100% OIII for Blue. Still others go 80/20 SII/Ha for Red, 80/20 OIII/Ha for Green and 100 OIII for Blue (Juan Conejero calls this a more ‘natural’ representation; note he also says that if saturation is not an issue, then instead consider SII + 0.8\*Ha for Red, 0.2\*Ha + OIII for Green and pure OIII for Blue). Clearly this is an area to play with and I will be doing lots of experimenting.*

Note that you can simply open the SII, OIII, and Ha images and plug the formulas above into PixelMath in order to achieve the relevant Channel color mix. Note you likely will need to play with the individual colors using one of the methods described below to get everything looking the way you want.

*Option I (CurvesTransform and SCNR tools*)

For a great discussion of what is possible, take a look at the part of this post in the PI Forum that is the tutorial by Niall Saunders: <http://pixinsight.com/forum/index.php?topic=2346.0;topicseen>.

The following sets out the highlights of that tutorial, which I have used successfully myself so I can attest to the quality of the results if you take the time to adjust the learnings to your particular data set:

* Typically, because of how strong the signal is in Ha as compared to OIII or SII, an NB image will come out mostly green when first combined, so you need to start bringing those greens down to yellows. To do that, start by opening CurvesTransformation and select to hue adjustment (the box with the “H”). Note that this tool shows the current hue along the horizontal axis and the adjusted hue along the vertical axis. The goal here is to move the curve near the lower left so that the green in the horizontal is moved down to the brown/yellow range on the vertical. But you will want to also put a second point higher up and drag that back up so you create a curve in the form of an “S”. You want that curve to be reasonably smooth so all colors only get shifted gently. You can even add more points on the upper half of the curve to bring that part back to the original diagonal line, meaning that for those upper hues, no changes are made at this point. This takes practice but is worth the effort of getting the feel. And feel free to do the changes in small steps rather than one big adjustment. That prevents “blowouts” from occurring. Finally, use the preview window to see real time results of your adjustments before you apply them to the main image.
* Likely the image will still have too much of a green cast, particularly in the brighter sections. Fix this by opening the SCNR tool, select Green as the “Color to remove” and set the “protection Method” to “Maximum neutral”. Start with an amount of 0.50, but feel free to play with the setting.
* You may now have an excess of Red so, using the SCNR tool again, this time set the color to remove to Red and use Minimum neutral as the protection method. Start around 0.10, but play with this setting as well.
* If you need to boost saturation and brightness at this point, open ChannelExtraction, check the “L” box only, set it to CIE L\*a\*b\* and apply to the image. This will extract a lum image that you can now apply as a mask. This will allow you to boost saturation and brightness of the bright parts of your image without disturbing the dimmer regions. Once you have applied the mask, open CurvesTransformation and first, check the “S” box, for saturation and pull the curve to the upper left, without running either endpoint into the top or side (you want to always keep your curves smooth and all points fully on the graph). Once this is drawn, then click the “L” box (you can do more than one curve in the same graph) and do a similar curve to the Saturation curve. Again, use the real time preview to see how this looks before applying to the main image.
* At this point it is merely a matter of making whatever other adjustments may be necessary using the same techniques above to get the image where you want it. Trial and error is the key, but I think you will really come to enjoy this element of the workflow since it is so fulfilling to get it right. And, if you haven’t tried narrowband imaging, I hope this inspires you to give it a go.

*Option II (Using SHO-AIP script)*

*This is another powerful color mixer that deserves your attention. I am only beginning to play with it and will update this section as I gain more experience with the tool (if you read French, which I, much to my chagrin, do not, it may help since there are hints available in French in this post: http://pixinsight.fr/mixshonrvb.html).*

*Note, per a PI Forum discussion thread, apparently you are to use this tool in the non-linear phase of your processing.*

*Option III (L channel extraction)*

*Here is another suggested way forward, presented by Jose, listed as jmtanous in the PI Forum. As always, if you find this helpful, drop him a line to thank him:*

* *Manually mix the stretched NB channels (using ChannelCombination or PixelMath);*
* *Ignore the colors for the moment, which will likely be way off;*
* *Stretch the image a bit, but not too much;*
* *Extract the L channel;*
* *Aggressively tweak the color image using HistogramTransform, CurvesTransformation, or otherwise (such as SHO-AIP script). It doesn’t matter if image looks over processed at this stage;*
* *Recombine with the pristine L image using LRGBCombination, checking the color noise reduction option in the tool. This should produce a much cleaner image;*
* *Tweak with CurvesTransform to get contrast and saturation right.*

*Option IV: (Alejandro Tombolini’s master technique)*

*Finally, checkout this link for how one of the PixInsight masters works his magic:*

*http://pixinsight.com/forum/index.php?topic=5148.0*

[Note: For any of these options, make sure you check out the ColorMask script, described under Masks]

*Removing Magenta Stars in Hubble Palette*

This is a new routine suggested by troypiggo in the PI Forum and is worth looking at (I have simply copied his post in its entirety):

*Not sure if this has been done before, but I had this idea recently on how to get rid of the magenta stars in SHO narrowband images due to the red SII and blue OIII channels needing to be stretched so much to balance strong G Ha.  A common Photoshop method I've seen is to do a selection based on colour (magenta), and desaturate it.

It occurred to me that with PixelMath we may be able to achieve something like this.  We could detect if a pixel is magenta by checking if it's R and B channels were of similar value (within some acceptable range), and if they both were significantly brighter than the G channel.

With this in mind, I came up with the following formulae.  I found some stars needed a luminance rule in there too.

R: $T[0]
G: iif((CIEL($T)>MIN\_BRIGHTNESS)||((min($T[0],$T[2])/(max($T[0],$T[2])>MAGENTA\_DEFN))&&(mean($T[0],$T[2])>$T[1])),mean($T[0],$T[2]),$T[1])
B: $T[2]
Symbols: MAGENTA\_DEFN = 0.9, MIN\_BRIGHTNESS = 0.9

I'm just hoping it is a quick and simple solution to what seems to be a common issue with narrowband images.  Just drag and drop the attached process icon on your final narrowband image with magenta stars and they're gone.*

You can find the process icon he mentions in the last paragraph with his post in the PI Forum here (be sure to drop it into your Process Container for easy access): <http://pixinsight.com/forum/index.php?topic=7377.0>

For those of you, like me at first, who are not sure how to access the process, here are the steps:

* Save the process zip file and then unzip to it to a location you will remember (such a creating a subfolder to hold PI scripts)
* Go into PixInsight and right click anywhere in the empty workspace
* Choose "load process icon" and navigate to the file
* A process icon called " magenta\_ star\_reduction " should appear
* Just drag n drop that on your image
* Double click on the icon to see the PixelMath window with formulae etc.
* To put the process in your Process Container, open both the PixelMath window as above and your Process Container. Then simply drag the triangle at the bottom left of the PixelMath box into the Process Container. Once you have it there you can name it by clicking on the bottom right button re change description. Note this is how you would load any PixelMath formula, including the ones I have sprinkled throughout the cribsheet, into the Process Container so you have them all handy

*R and Ha Combine:*

*Option I (applying Ha highlights to RGB images – e.g., Ha regions in galaxies):*

Step 1:

Process RGB Image through Step 13 (ColorCalibration)

Open RGB and Ha images

In PixelMath RGB/K line enter the following (removes Ha background):

(([Ha]\*75)-([RGB]\*3))/(75-3)

Where:

[Ha] is the Ha image name

[RGB] is the RGB image name

75 is the approximate bandwidth of the Red filter (feel free to experiment)

3 is the bandwidth of the Ha filter (feel free to experiment)

Make sure “Rescale” is NOT checked

Click “Create New Image” button

New Image will be used as the Ha image in Step 2.

Step 2:

Open RGB and new Ha image from Step 1

Blends new Ha and Red channel into new HaR channel

Split the RGB into its component color channels

Open PixelMath and enter the following formula in the RGB/K line:

$T+([Ha]-Med([Ha]))\*5

Where:

[Ha] is the new Ha from Step 1

$T is the Red Channel

5 is the amount by which the Ha will be intensified (experiment recognizing that the higher the number, the greater the added noise)

Apply to the Red Channel and save result as new HaR image channel (replaces R in RGB).

Recombine HaRGB using ColorCombination Tool

*Option II (using new NBRGB Script; Note, my preferred method):*

Create RGB image through step 14 (ColorCalibration)

Perform Align RGB, Ha [and OIII, if using] images

Open RGB, Ha [and OIII] Images

Open NBRGB Script

Set appropriate bandwidths (F16M filter settings: ~75 for RGBs, 3 for NBs)

Set Ha [and OIII] multipliers, previewing with “Show NBRGB”

When satisfied, press OK

*Option III (general enhancements to Red channel):*

This is very easy to do with PixelMath in PixInsight. For example, to add a 25% of Ha to the red channel:

R: 0.75\*$T + 0.25\*Ha
G: $T
B: $T

with separate RGB expressions, applied to the RGB image.

*Dynamic Background Extraction:*

Stretch image using STF “autostretch”

Execute a DynamicCrop before running DBE to make sure image edges are clean

Open DynamicBackgroundExtraction.
Click on the image to bring up crosshairs.
Center crosshairs on object of interest.

Increase Tolerance to ~2

Generate matrix (settings around 8/20/0.25 seems to work well)

Alternatively, place matrix manually, making sure to hit the corners well

Delete any square over bright stars and areas of nebulosity.

Add squares as necessary to ensure coverage in the corners

[*Tool tip: after highlighting a square, hitting “Delete” continues down column*]
In the Target Image Correction section, change “Correction” to “Subtraction”
Use “Background Model” to examine pattern.
Execute

[*Note, you may want to run a second instance of DBE after you do your RGB combine and color calibration routines. Sometimes, despite background neutralization and color calibration, you may still have large background hue differences (best seen if you use the accelerated STF button) that these processes can’t fix. If you have that situation, I have had success with the following:*

* *First, using a luminance mask to protect the background, enhance the saturation of the target and stars using CurvesTransFormation;*
* *Next, invert the luminance mask so that you are only working on the background;*
* *Run DBE and use the sample generation routine to cover as much of the background as you can (for my images I use settings of 20/30/0.25). You can place the matrix manually, but in this instance the generation routine works well. Make sure that none of the matrix covers the target, including faint nebula, or any medium to bright stars;*
* *Run the routine using subtraction, checking both the resulting image and the background model.*]

One area that may prove difficult for DBE is with images having lots of nebulosity since it is hard to find places with true background. Note that, in the case of narrowband images, you really only need a few points; 5 or 6 should do you. You might also try using Automatic Background Extraction or ABE, with the default settings. Others have had success with this technique.

*Linear Fit*

Open integrated R,G, & B images

Open LinearFit tool

Make best looking image (one showing greatest detail) the reference image

Apply to each of the other two images (drag “apply” triangle to images)

[*Note: Linear Fit should balance the histograms for the three color images. If using Linear Fit properly balances the colors for the resulting RGB combined image, it may not be necessary to apply either the Background Neutralization or ColorCalibration tools to the RGB. They may still be necessary if Linear Fit does not solve a particularly difficult color balance issue, however*.]

[*Note 2: There have been questions raised about whether using LinearFit is appropriate since it might harm color balance. As such, I have been taking the time to run some tests by using LinearFit in one case and the Background Neutralization and ColorCalibration tools in the other case, to see the differences, if any, that result. So far, I have found that LinearFit actually gives me a better result, but that may be because my Astrodon filters are very close to 1:1:1 combined color weights. I encourage you to do testing with your own data.*]

[*Note 3: LinearFit has an important role, mentioned later, in making sure you have matching fluxes between your Lum and RGB masters before you do an LRGB Combination. See Section on the LRGB Combination tool for details.*]

*Color Combine:*

Open aligned R, G, & B (or Ha, OIII, and SII) images

Open ColorCombine

Allocate stacks to appropriate color line

Apply

DynamicCrop RGB image

*Background Neutralization (use with RGB images)*

Open BackgroundNeutralization

Create a Preview of background sky

Select Preview as working image in Background Neutralization

Apply

*Color Calibration (use with RGB images):*

Open ColorCalibration
Create a preview of the object of interest
In the ColorCalibration dialog, select this preview as the White Reference
Create a preview of a dark part of the image
In the ColorCalibration dialog, select this preview as the Background Reference
Turn on "Output white reference mask"
Turn off "Structure detection"
Adjust Lower Limit starting at 0.001, if necessary
Apply

*Deconvolution (Based on Mike Wiles’ Video Tutorial and the discussion in the following tutorial:* [*http://www.pixinsight.com/examples/M81M82/index.html*](http://www.pixinsight.com/examples/M81M82/index.html)*):*

Apply to Linear Images only!

Use on Lum or Lum equivalent (SynLum or HA in NB images)

Create Masks:

Local Support Deringing Mask:

As a first step try deconvolution without local deranging support. For many images, global deranging alone may be sufficient. If you need local deranging support, then build the deranging mask as follows:

Mask bright stars, but not just “super bright”

Look for strong red on bright stars, no protection on target

Pre-process image before applying StarMask

Duplicate image

Use MLT to remove 1st and residual wavelet layers in a 5 layer decomposition

Use StarMask on resulting image per below

Use Star Mask tool

Starting settings:

Noise threshold: 0.01

Scale: 3

Structure growth: 1-4-2

Smoothness: 16

Check “aggregate” and “binarize”

Do not check “invert”

If any of the target area remains (such as bright portions of galaxies), use CloneStamp tool to remove while leaving highlighted stars untouched

Smooth mask by applying a Gaussian Convolution

Start at settings of StdDev 5, Shape 2, and Aspect Ratio 1

Once satisfied, rename “Deringing mask”

Minimize for later use

It is VERY important to build a suitable local support for deringing: Try using a StarMask that covers well the brightest stars with 10-15 pixel of smoothness and truncation at about 0.75. Then fine trim deringing with a very small (usually less than 0.02) amount of global deringing.

Luminance Mask:

Should only exclude background – covers target and stars

Provides smooth transition b/t high and low SNR zones

Make clone of image

Do an auto STF stretch on clone

Copy STF settings to HT tool (pull STF bottom left triangle to bottom bar of HT)

Apply HT to clone to make permanent stretch

Move Shadows just to left of first light

Rename “luminance mask”

Minimize for later use

Create Composite PSF:

Open DynamicPSF tool

Select ~60-70 stars of medium brightness (unsaturated)

Moffat only

Type “Control – a” to select all

Sort list by increasing order of MAD (median absolute deviation)

Highlight best ~50 stars

Hit the little camera button on bottom of tool to create a composite PSF image

Minimize for later use

Deconvolution Process Steps:

Open Deconvolution Tool

Insert PSF image using “External PSF” tab

Select Dark, Light and Stars Previews to test

Protect original image using Luminance Mask (background Red)

Activate “Local Deringing” and use Star Mask image

Start with Global Dark Deringing at 0.02

[*A setting of 0.02 may not do much and if not, try lowering. I have gone as low as 0.002 and stepped up from there. Again, this is one tool that rewards lots of experiment with the Global Dark Deringing setting*]

Apply Regularized Lucy-Richardson

Start with 25 interations

Open Previews of one or more target areas and of background

Run iterations to test for sharpening and ringing

* Up iterations to improve sharpening but try not to go beyond 50
* If ringing in stars, increase Global Dark Deringing (in ~0.02 increments)
* If no improvement consider an order of magnitude reduction in the Global Dark Deringing setting to the 0.002 and above range

*Alternate Deconvolution to Correct Elongated Stars:*

Bring up Deconvolution tool
Make a small preview
In Gaussian PSF, StdDev at 1.7, Shape 1.55

Adjust aspect ratio and rotation to approximate star elongation
Set iterations to 10
Turn on Deringing
Adjusting Deringing Global dark anywhere from 0.001 to 0.2
Apply and evaluate

*ScreenTransferFunction*

STF is a great tool for examining images in the linear stage, but sometimes it gets a bit “wonky”, at least on my computer, especially after I have done a lot of linear processing of an image. What I experience is the autoSTF function “blowing out” my images, making them hard to analyze, even when I use the normal setting. Fixing this problem by trying to adjust the sliders on the tool doesn’t work, at least for me, since I always end up with an image that is too dark or blown-out again. If you experience this problem, a simple fix is to open the STF tool, hit the little wrench icon on the lower left side and simply adjust the numbers to get the result you want.

***Masks***

*Star Mask (using HDRMT enhanced Image for Mask base image):*

Prep Image for StarMask purposes as follows:

Compress dynamic range aggressively using HDRMT to flatten image

Apply noise reduction to help isolate stars

Open StarMask tool
Set Mode to “Star Mask”
Set Noise Threshold to 0.25
Set Scale to 8. Test with lower and higher settings but this is where to start

In Growth Structure set to 2:1:2

Set Smoothness to 16, leave all boxes unchecked

Set midtones to 0.4

Use Convolution tool to smooth star mask
Optional: Use erosion and dilation filters to shrink or inflate mask structures

Optional: Use histogram to clip shadows if necessary

Execute, evaluate created mask, adjust as necessary
When satisfied, apply mask to image
Do NOT close star mask image as it is still being used. Minimize and set aside
Invert mask if required “Invert Mask” tool bar button

Red areas are protected
Hide mask by clicking “Show Mask” tool bar button

[Note: if you are having trouble generating good star masks, especially for larger stars (scale >7) consider the suggestion by MortenBalling on the PI Forum. He says he has had excellent results by cloning your image, resample it to 50% scale, make the star mask, then resample the mask at 200% scale.

Let me know how this works for you.]

*Brightness Mask:*

Option 1:

Create a clone

Invert clone

Adjust using HistogramTransformation

Blur using Convolution, if necessary

When satisfied, apply mask by dragging side left tab of mask to side bar of image

Do NOT close Mask image as it is still being used. Minimize and put to the side

Invert mask if required “Invert Mask” tool bar button

Red areas are protected

Hide Mask by clicking “Show Mask” tool bar button

Option 2:

Use Range Selection Tool (click “real time preview” to test)

Slowly increase lower limit to protect only brightest areas

Click Apply to set Mask

Blur using ATWT settings from RGB Noise Reduction, below

Do NOT close Mask image as it is still being used. Minimize and put to the side

Option 3:

Use HistogramTransformation, Curves Transformation and the Convolution tools (see Mask discussion below under “Noise Reduction and Sharpening”)

*Mask Combine:*

Open Pixel Math

In text box type: “range\_mask+star\_mask”

Uncheck “rescale”

Check “Create New Image”

Apply

Right Click to Set Image Identifier to “combined\_mask”

*Creating Target-Only Mask (for subject, minus stars):*

Pixel Math method:

Create a luminance mask, stretching if necessary and clipping the blacks a bit

Create a nice star mask, usually taking the target image and running an aggressive HDRMT on it first

Option 1 (preferred): Apply this PixelMath expression to the luminance mask: iif(star\_mask>0.2,min($T,0.8-star\_mask),$T)

Option 2 (quick and dirty): Use this PixelMath expression: [LumMask]-[StarMask]

(note; replace bracketed names with actual mask names)

Blur the resulting mask a bit by removing layers 1..4 with ATWT

Softening/blurring a Mask:

For all of the above masks, in the event you want to soften them a bit so they do not have such hard edges, consider using the Convolution tool. Here you can apply various levels of convolution to achieve various degrees of blurring/softening.

*Creating a Nebula only mask*

[*Note: This is another one I picked up in the PI forum that may be worth a try for building a nebula only mask using the following procedure*.]

* Create a mask using RangeSelection which selects the nebula and stars
* Create a star mask
* Remove stars from range\_mask using the PixelMath expression, range\_mask - star\_mask. At this point the nebula will be bright, noise in the background will be grey and the stars will be black.
* Apply a contrast curve using CurvesTransformation to darken the background and brighten the nebula.
* Blur the image by deselecting the detail in all wavelet layers except the residual layer with ATrousWavelets & apply to the image
* Repeat steps 4 and 5 until there is a smooth mask where the nebula portion is bright and the background is very dark.
* Subtract the stars out of the range mask again using PixelMath: range\_mask - star\_mask

This creates a mask that selects the nebula alone. By iteratively applying contrast curves and smoothing, you can effectively darken the background of the mask (removing the noise along the way) while still preserving the transitions between the nebula and the background.

If you just want the background only, skip steps 2, 3 and 7. This will give a mask selecting the nebula and stars which can be inverted to select the background.

*Noise Reduction Masks*

See Mask discussion below under “Noise Reduction and Sharpening”

*ColorMask*

This is another script that I have just started testing and deserves your attention. It offers the ability to isolate various color channels for adjustment without affecting the image as a whole. It was developed my RickS on the PI forum and if you find it helpful please drop him a line to thank him for the effort. He provided the following advice with I asked him his technique on a particular image:

*“The main things I did with ColorMask were remove Magenta, make the yellows more orange and generally adjust the cyans and blues to taste:*

*• Magenta mask: Curves reduce R and B, boost G, slight desat*

*• Yellow mask: boost R*

*• Cyan/Blue masks: adjust a\*, b\* curves, lightness and saturation”*

This only scratches the surface of what can be accomplished with this tool. The key is to experiment. Select a color you are interested in enhancing or subduing and create the appropriate mask using the best color approximation (e.g., if you are looking to enhance your blues, work with the cyan and blue buttons). The other thing is to move in very small increments using the CurvesTransformation tool once you have applied the colormask to your image. And, depending on what you are trying to achieve, try using the “a” and “b” channels to make your adjustments as well. Give it a try, it’s very impressive and does amazing with NB images**.**

***Permanent Stretching***

[*Note: The MaskedStretch Tool creates an image that lacks contrast but does a substantially better job of producing natural, non-blown out, stars. HistogramTransform creates a nice, contrasty image but at the expense of stars being blown out. Harry, from Harry’s Astroshed, has a tutorial that shows how to combine the two to get the best of both worlds. I have the basics here but please go to Harry’s website at* [*http://www.harrysastroshed.com/*](http://www.harrysastroshed.com/) *and have a look around at this and his other excellent tutorials.*]

*MaskedStretch (base method):*

Apply ScreenTransferFunction to a linear image and leave the STF tool open.

 While holding down the control key, click on the “hazard” circle.

 Note the “Target Background” number from the pop-up window.

 Open the MaskedStretch tool. Insert new Target Background number from above

 Create a preview of a background area and use as “Background Reference” image

 Leave rest of settings at default values

 Apply to linear image.

 Clean up contrast using CurvesTransformation tool

*Histogram Stretch (Alternative 1):*

Open image, STF tool and HistogramTransform tool

Apply STF tool to linear image

Take triangle from bottom left of STF tool and drag to bottom middle of HT tool

Apply to image to create HT that mirrors STF tool parameters

[Note: make sure to cancel STF to get proper view of stretched image]

*(Alternative 2):*

Open HistogramTransformation
Select the appropriate image
Click Preview to open up a real time preview
Adjust the midtones triangle over until it's near the lefthand side of the histogram
Zoom in on the left hand side until the image curve can be seen properly
Move the dark end triangle over until it is just left of the main histogram curve

Ensure that dark pixel clipping is kept to a minimum
Bring midtones triangle to the left until the image has the proper brightness
When happy with the preview, Apply the tool to the image

[Be sure to reset the Preview (small circle at top left) for multiple tweaks]
Save new non-linear image

[*Note: Do not stretch background too much; use mask if necessary. Stretch Midtones 1st; then shadows*]

*Getting best of both worlds (Combining MS and HT stretch per Harry’s Astroshed):*

* Open a linear image and create a clone (simply click on the image name at in left side of image and drag to right to create a duplicate)
* Apply MaskedStretch as above to one image and HT Alternative 1 to the other
* Create StarMask of the HT image and apply to HT image
* Open PixelMath and in the expression editor, apply name of MaskedStretch image so that the RGB/K line only shows the MaskedStretch image name
* Apply PixelMath to HT image, replacing HT stars with MS image stars.
* Send a note to Harry, thanking him for his tutorials

*LRGB Combination*

I had not previously realized the importance of getting this right, but read the following from Juan Conejero:

*“[A]chieving a good adaptation between RGB and L is very important, and also the most difficult part. The "intrinsic" L in your RGB image will basically be replaced with the L image. You should stretch L first, then stretch RGB to achieve similar levels in its luminance (watch the L display channel, or select the RGB+L pixel readout mode and compare readouts, or extract L from the stretched RGB and compare statistics). The LRGBCombination tool also has a luminance midtones transfer function to fine tune the adaptation. The goal is to achieve an optimal adaptation between luminance and chrominance. Too much luminance means more chrominance noise to achieve the required color saturation. Too much chrominance means more noise in the luminance*.”

Luckily, Juan also explains that in PI, we have a nice way to match the fluxes between RGB and L: namely the LinearFit tool (which in this case will be used with non-linear images). This is the basic procedure:

- Apply the initial nonlinear histogram transformations to your RGB and L images. Adjust the L image first, to the desired brightness and contrast. Then try to match the overall illumination of L when you transform RGB. Do it roughly by eye using the CIE L\* display mode (Shift+Ctrl+L, Shift+Cmd+L on the Mac). Don't try to do a particularly accurate job here; it will get much better in the next steps.

- Extract the CIE L\* component of RGB with the ChannelExtraction tool (select the CIE L\*a\*b\* space, uncheck a\* and b\*, and apply to RGB).

- Open the LinearFit tool (ColorCalibration category) and select the L image as the reference image. Apply to the L\* component of RGB that you have extracted in the previous step.

- Reinsert the fitted L\* in the RGB image with the ChannelCombination tool.

Now your RGB and L images have been matched very accurately. Use the LRGBCombination tool with them. You shouldn't change the luminance transfer function, nor the channel weights, as LinearFit has already done the matching job much better than anything you could do manually.

*Color Saturation*

*Using LRGBCombination:*

In the section above, you were admonished not to adjust the luminance transfer function as a part of using the LRGBCombination tool. But if you take a look at that tool, you will notice that the transfer function also includes a Saturation slider. You can simply use that slider to enhance saturation while, at the same time, taking advantage of the fact that this tool has chrominance noise reduction built in (you just have to check the box). One thing you need to keep in mind with this adjustment, however, is that the slider works backwards to how you might think, namely the smaller the number, the bigger the saturation effect.

*Using CurvesTransformation:*

 Create Luminance Mask

* Use HT to darken the shadows
* Use ATWT to denoise mask by removing layers 1-3
* Apply Mask

Curves Transformation

* Select “Saturation” setting (“S” button at right)
* Apply Saturation 1
* Repeat with second Saturation 1, as necessary
* Form slight s-shaped contrast enhancing curve on all RGB

*Contrast Enhancement:*

There are several tools such as DarkStructureEnhancement and LHE that allow you to sharpen an image through contrast adjustments. But a simple way to make small changes is mentioned above, but deserves a bit more explanation. Using the CurvesTransformation tool, create an “S” curve by first placing a control point on the exact center of the vertical line. Then, when you grab and adjust the curve, it is pivoting around that point. The effect is that you are affecting contrast but not otherwise changing the image. Give it a try.

***Noise Reduction***

[Note: Masks are hugely important in noise reduction to ensure that while you clean up the noise, you do not damage image detail. In that regard, let me quote some words on Masks in PI from the master Jedi himself, Juan Conejero (and if you don’t know who he is you have not spent enough time on the PI Website and in the PI forum):

*Inverted lightness masks have been used routinely as noise reduction masks in PixInsight since the first version of PixInsight LE (2003). You can use a variety of tools to stretch and clip a duplicate of the image (or an extracted brightness or lightness component) as necessary, including HistogramTransformation, CurvesTransformation, PixelMath, RangeSelection, etc. StarMask is not required at all for a noise reduction mask.*

*A linear mask is much more efficient for noise reduction of linear images than a stretched nonlinear mask. By preserving linearity of the mask we ensure that noise reduction is strictly proportional to the SNR of the image. A linear mask can be generated very easily with PixelMath, but you can also use MultiscaleMedianTransform with the Linear Mask and Preview Mask options enabled.*

Take these words of wisdom to heart. I also followed this up with a question to Juan of how to “very easily” build a linear mask with PixelMath and here was his response:

*As implemented in the MLT and MMT tools, a linear mask is just a duplicate of the linear image multiplied by an amplification factor, so the corresponding PixelMath expression can be as simple as:*

*$T\*k*

*with k defined as a constant symbol such as k=100 for example. In the case of a color image one can apply this either to the duplicate RGB image (because in PixInsight you can use RGB masks with RGB images), or to single components such as H or I (extracted with ChannelExtraction working in the HSV and HSI color spaces, respectively). The linear mask should always be low-pass filtered to soften edges and make it more robust to local variations; this can be done easily with the Convolution tool applying a Gaussian filter.*

Juan, however, also points out that the PixelMath method is automated through the process of building a mask using the MLT or MMT preview mask function described above and this is probably the easier way to achieve the same result.]

*Noise Reduction*

*[Note, In my personal workflow, I find the MLT gives superb results in the linear phase and both the “Removing first wavelet layer” and the “General Noise Reduction” methods under MLT below have become my preferred noise reduction tools in the linear phase. To date I do not find that MLT noise reduction does anything in the non-linear phase, but that is likely due to the fact that I haven’t played with it enough. My preference therefore, at least to date, is to use MLT in the linear phase and TGVD in the non-linear phase. But, as always, only use this as a guide and explore all the tools to see what works best with your data.]*

*Multiscale Linear Transformation for Noise Reduction:*

*Removing First Wavelet Layer:*

In the linear phase, use MLT, as follows, to get rid of the first wavelet layer which most likely is little more than noise anyway (though an inverted luminance mask to protect the target is highly recommended):

Open image and MLT tool

Set MLT for 5 layers

Make sure that “Noise Reduction” is UNCHECKED for each layer

For all but the first layer, make sure that “Bias Level” is CHECKED

For layer 1 only, UNCHECK “Bias Level” (layer 1 will have a red “x”)

Apply to the image

*General Noise Reduction*

Used in Linear stage;

Settings:

* Click Dyadic
* Layers 5
* Initial Layer settings (set in “Noise Reduction”)
	+ Layer 1: S(5.000,1.00,1)
	+ Layer 2: S(3.500,0.90,1)
	+ Layer 3: S(2.500,0.80,1)
	+ Layer 4: S(1.500,0.80,1)
	+ Layer 5: S(0.500,0.70,1)
* Check Detail Layer: Bias 0.000 in each layer
* Check Linear Mask
	+ set both sliders to 200
	+ Check “invert”
* Rest unchecked\*
* Target: RGB/K Components

Make sure noise reduced image does not end up ‘plastic’ looking or with background blobs or similar artifacts. Do not try to get an overly smooth background. A little noise will make your images look more natural.

\* I tested this assumption by testing both the K-Sigma Noise Thresholding and Large-Scale Transfer Functions. The Notes suggest that K-Sigma should not be used, however, I found that applying a threshold of 3.0 and an Amount of 80%, with both Soft Thresholding and Multi-Resolution Support checked, gave me a noticeably cleaner image without loss of detail. I also tried various settings in the Large Scale Transfer Function but did not see any improvement, and some degradation at certain settings. Based on my experience, I encourage you to test these settings yourself to see if they improve your images as well.

*TGVD Initial Settings:*

Build and apply a permanently stretched clone/luminance mask

Measure Background Std Dev (using the Statistics tool) and use as the “Edge Protection” setting

Decrease value if edges lost, otherwise increase a bit

*For non-linear images:*

Start with default settings

Need 300-500 iterations

Use Local Support for non-linear images as follows:

Extract lightness (IMAGE > Extract > Lightness); select it as a Local Support

With HT, use midtones balance and shadows clipping to increase contrast

Consider removing first wavelet layer if image really noisy

Disable “Preview” mode

Set Noise reduction to 1

Use RGB/K for RGBs or for LRGB where noise is similar in both color and luminance

Use CIE Lab for Lums or for LRGB when color and luminance noise differ

* When using CIE Lab, perform separate runs in each mode if using different support images for each, otherwise can be done together
* Be much more aggressive in Color noise reduction

Continue to tweak settings until able smooth background WITHOUT introducing clumpiness to background. Key parameters to play with are the strength and edge protection settings. Trial and error is the key.

*Alternative Settings for Non-Linear Noise Reduction in TGVD:*

* Strength: 2.5 0
* Edge protection: 2 -3
* Smoothness: 2 0
* Local Support: NONE

*For linear images:*

* decrease strength by 2 orders of magnitude
* decrease smoothness by one order of magnitude

Need 300-500 iterations

When using TGVD in Linear mode, need to set local support as follows:

Open Image to be de-noised

Open STF Tool

Stretch support image to achieve good contrast and minimize noise

Click on “crescent wrench” icon

Pop-up corresponds to black-point, midtone, and highlight settings

Import these into the TGVD (note settings are NOT in the same order)

*Base Settings to try for TGVD in Linear Stage:*

* Strength: 9.99 at -2
* Edge Protection: 7.35 at -3
* Smoothness: 2 at -1
* Local Support: Midtones 0.05, Shadows 0.16, Highlights 0.0

*Convolution for Background Noise Reduction*

If done carefully, Convolution can be very effective as a noise reduction tool for the background sky. You need an excellent mask that protects everything but the background.

* Build and apply strong luminance mask
* Leave Aspect Ratio at 1.00
* Experiment with Shape at, above, and below 2 (I have had good success at 1)
* Experiment with Different StdDev levels)
* Avoid over doing where background looks artificial or plastic

*ACDNR (second round of Non-linear Noise Reduction):*

[This is now becoming an old technique. Try it out but it is dated compared to some of the newer tools such as MLT and TGVD]

Build and apply good Luminance Mask

Starting Lightness settings:

StdDev: 2.0

Amount: 1.0

Iterations: 4

Robustness: 3x3 Morphological Median

Starting Chrominance settings:

StdDev: 3.0

Amount: 1.0

Iterations 4

Robustness: 3x3 Unweighted Average

***Sharpening***

*[Note, be sure to use a quality mask for any sharpening activity and you will likely want the mask limited to the target only. Look at the Mask Section above for details on how to create a target only mask.]*

*AtrousWaveletTransform:*

Combine a Star Mask and Background Mask to protect all but target

Set to 4+ layers

Set starting Bias for layers 1 and 3 in the 0.05 range

Set starting Bias for layer 2 in the 0.1 to 0.3 range

*HDR Wavelets to Increase Detail:*

[Note: I have been playing with this tool recently and it can be very powerful. One of the things to keep in mind is to go into this tool with what looks like a “blown-out” image, meaning one that looks too bright, particularly the target, but that is not actually saturated. The challenge is to achieve that without making the background too bright. As such, HistogramTransformation may be the wrong tool. You may want to try using the CurvesTransform tool. To get an idea of what I am talking about take a look at the dynamic range compression discussion in the following tutorial: <http://www.pixinsight.com/examples/M81M82/index.html> Personally, I am not a great fan of this tool since I think it gives images an strange, almost hyper-detailed, look that to my eyes seems unnatural, but you absolutely should try it out]

To bring out details:
Bring up “HDRWaveletTransformation” tool.
Set layers to 3 to start.

Values of 5 and 6 are good for images with high dynamic range.
Number of iterations set to 1 (default)
Scaling function set to “3x3 Linear”
Turn on “To luminance”
Turn on Luminance mask
Turn on Deringing

Set Deringing Large-scale to 0.0
Set Deringing Large-scale to about 0.25 (experiment to find proper value)
Apply tool to image

To bring out finer details, lower number of layers to say 2
lower Large-scale details to about 0.15
Apply tool to image

Note: Keep star mask around for next step

*Morphological Transformation:*

Click "Show Mask" in tool bar
Invert Mask so mask is protecting the field (i.e. red field) instead of the stars
Hide mask by clicking "Hide Mask" in tool bar

Bring up Morphological Transformation tool
Operator should be "Erosion (Minimum)"
Put Amount at about 0.32 to 0.4 or to taste in "Morphological Filter" section
Pick a Size of "5 (25 elements)" in "Structuring Element" section
Click the "Circular Structure" widget in "Structuring Element" section
Put iterations at 2
Apply and evaluate

*(Note: I really like this tool so long as you do not overdo it. Make sure to use a mask and play with the “Amount” slider to get the effect right by reducing, but not over-reducing the stars that detract from the overall image)*

*DarkStructureEnhancement and LHE*

*(Note: I have not discussed either of these in the past though they have always been in my workflow. They are both powerful, and in the case of DSE, almost too powerful. Please give them a try, but as always, a little goes a long way).*

LHE is my “go to” final tool for sharpening an image and I rarely complete an image without at least a delicate application of LHE to bring out some faint details that otherwise would remain hidden. Spend a lot of time getting to know what this tool can do for you and you won’t be disappointed.

DSE, on the other hand, has to be used very very delicately or you will end up with an over contrasted mess, but for the right image with the right settings it will really bring out details by subtly enhancing contrast between the bright and dark portions of your target.

***Miscellaneous Image Processing techniques***

***Layers***

For those of you, like me, who have migrated to PixInsight from Photoshop, hopefully you have realized, as I have, that there is really no comparison between the two when it comes to the depth of control you can exert over your images. That said, one thing that I have seen people complain about is the lack of a Layers tool in PI, though I understand the developers are working on one.

But our friend, “oldwexi” (Gerald Wechselberger)has come to the rescue, by, along with Hartmut Bornemann, building a great javascript on one originally developed by Mike Reid that should give you everything you could want in this regard. To find it, click on the following address:

<http://www.werbeagentur.org/oldwexi/Daten/Blend_StarMask_english.js>

(He also has a German language version available at:

<http://www.werbeagentur.org/oldwexi/Daten/Blend.js>)

Gerald reports that it can also be used with a mask and that the latest version can also create a starmask from the actual Image. What a great tool!!

If you are unfamiliar with how to open javascripts in PI, simply save the process file to a location you will remember (such a creating a subfolder to hold PI scripts). Then, go to the scripts tab along the top and click.  Near the bottom, four options up, just below a dividing line, is an option to “execute script file”.  Click on that and then navigate to where you saved the script.  Just click on the javascript icon in that folder and the tool should open right up.

And, if you find it useful, be sure to drop by the PI Forum and give oldwexi the thanks he and the others mentioned above deserve (as I note later, he also taught me everything I know about PixelMath through his great tutorials on his website).

***Using Previews for Multiple Processing Steps***

Here is a nice feature that I will be using more of. One thing I always found frustrating is that I could only do one operation on a preview image at a time, so I couldn’t see what multiple steps would do without having to apply them to the main image. Juan gives the following work-around:

1. Apply a (previewable) process to a preview. Now we say that the preview has a volatile state. A volatile state can be undone/redone with the Preview > Undo/Redo command. The volatile state will be replaced if you apply a new process to the preview. This is the "normal" preview operation.

2. Select Preview > Store. Now the preview has a stored state.

3. Repeat steps 1-2 to accumulate stored preview states. When a preview has one or more stored states, you can use the History Explorer window to navigate throughout the list of states, in a similar way to a main view. You can select Preview > Reset to destroy all stored preview states (similar to Image > Undo All, but destroyed preview states cannot be recovered).

*CanonBandingReduction (should be named HorizontalBandingReduction)*

This is another very useful script I have just started using. The critical thing to realize is that is works with any images, no matter how generated. It is most definitely NOT restricted to images taken with a Canon camera. I use a high end Apogee Alta F16M which, unfortunately started out with horizontal banding issues and after getting that fixed, now produces irritating vertical bands. Before finding this script, I had to hand fix using the CloneStamp tool, which is both very slow and sloppy and not in keeping with the PI philosophy of staying true to the signal. I was really dismayed until I saw a post explaining that this would work for any image and have been thrilled with the results I am able to achieve. It’s saved the F16M as far as my imaging system is concerned.

Note, this script “only” works with horizontal banding, but that is easily remedied. If you are dealing with vertical banding, when you are ready for this step, simply rotate the image 90% using Rotation or FastRotation.

I have tried the script in the linear stage and have had good results, but seem to get the best outcome if I apply after stretching.

You can play with the settings, but try the default settings first. I have found that it does everything I need on the default settings.

*Star Blurring (always use with a mask):*

Use Convolution tool

Set Kernel to 5 to start

*Image Crop for Printing:*

Rotate as necessary using Fast Rotation (if increments of 900) or Rotation tool

Crop using DynamicCrop Tool, “paint” the image and for my F16M, best fit is long side set at 3894 pixels and short side at 3009 pixels. Main point is you need to find Width and Height settings that are multiples of whatever image size you are trying to create. Position as desired, then crop.

Open Resample tool

Select Image from View List

In Resolution Section, set:

Resolution units: Inches

 Enter: Horizontal = Vertical = 300

 Check: Force Resolution

In Dimensions Section, change width and height to 8.5 and 11 in the inches boxes

Select “Preserve Aspect Ratio” to allow resizing while preserving 8.5 x 11 size

Use Default settings unless issues arise

Apply Process to Image

***PixelMath***

*(I want to thank Gerald Wechselberger who has some excellent PixelMath tutorials and goes by oldwexi in the PI Forum; my understanding of the power of PixelMath is wholly attributable to him; several of the items below are from his tutorials or suggestions in the forum).*

*It is important to note that these examples only begin to scratch the surface of PixelMath. I encourage you to explore this powerful tool in much greater detail. You may want to start by looking at the PixelMath section I have at the end of the Cribsheet under Tool Details.*

*Clean Up Dark Pixels*

Use PixelMath expression:

Basic function: iif($T < 0.08, med($T), $T) [*stronger, more uniform*]

Advanced function: iif($T < 0.08, ((med($T)-$T/2)+$T, $T) [*subtler, less uniform*]

Examine image to set highlighted number based on where break point is between normal and overly dark background pixels

*PixelMath Calibration of Individual Images*

Use PixelMath expression:

Single image: ([light]-[dark]) / [flat]; or for just flat fix: [light] / [flat]

Several images: ($T – [dark]) / [flat]

*Note: [name] means substitute actual image name*

Another option for processing multiple images would be to place several images in the Image Container, then drag the Image Container Operator unto the bottom panel of the PixelMath Tool

*Channel Selection*

$T[0] = red channel

$T[1] = green channel

$T[2] = blue channel

**Tool Details**

*StarMask Process Details:*

The StarMask process (MaskGeneration category) is the best tool in PixInsight to build star masks of many types, including deringing supports. StarMask operates by extracting the luminance from the target image (or a duplicate of the image if it's in the grayscale color space) and applying a multiscale algorithm that detects and extracts all image structures within a given range of scales (read sizes). The algorithm is based on the à trous wavelet transform and on a proprietary multiscale morphological transform.

It is very important that you understand the different parameters available to control generation of masks with the StarMask tool. Refer to Figure 18 to identify them on the StarMask interface.

*Mode*— You can select one between four available operation modes:

■Star Mask, which will generate an actual star mask that you can use to process images. Of course, this mode is used to produce actual deringing supports for deconvolution.

■Structure Detection, which generates a special mask with all detected structures, also known as a structure map. A structure map is useful to know exactly which image structures are being detected. It can also be used for actual image processing purposes, especially as the starting point of other mask generation procedures.

■Star Mask Overlay. In this mode, StarMask generates an 8-bit RGB test image where the red channel contains the generated star mask superposed to the target image, while the green and blue channels have no mask contribution. The base image to build this overlayed image is the target image after applying the initial histogram transform (see the Shadows Clipping, Midtones Balance and Highlights Clipping parameters below).

■Structure Detection Overlay. This mode is essentially the same as Star Mask Overlay, but instead of the star mask, the structure map is overlayed on the target image.

*Shadows, Midtones, Highlights*— These parameters correspond to a histogram transform that is applied to the target image prior to structure detection and mask generation. In fact, this histogram transform is an important preparatory step in the StarMask algorithm. These parameters have default values of 0.0, 0.5 and 1.0, respectively, which define an identity transformation (no change). However, usually you'll need to apply lower values of the midtones balance parameter, especially working with linear images, mainly for two reasons:

■To improve overall structure detection. In linear images, the structure detection algorithm may need you to improve local contrast of small structures in order to separate them from the noise.

■To block structure detection over bright parts of the image, where you don't want the mask to include structures that are not stars, for example, but actually small-scale nebular features.

On the example shown on Figure 20, note how the mask built with a midtones balance value of 0.1 includes much more stars and, at the same time, prevents inclusion of nonstellar structures from the brightest parts of the main object. We'll see later why this is particularly important for deconvolution deringing supports.

Increasing the Shadows parameter may also help to improve detection slightly; however, if you set it to an excessive value, clipping will occur in the shadows, which will prevent inclusion of dim structures. Generally, the highlights parameter is left with its default 1.0 value.

*Scale*— This parameter is the number of (dyadic) wavelet layers used to extract image structures. The larger value of Scale, the bigger structures will be included in the generated mask. Always try to set this parameter to the lowest value capable of extracting all required image structures; the range between 4 and 6 wavelet layers (scales from 16 to 64 pixels) covers virtually all deep-sky images.

*Growth*— This is actually a category that comprises three StarMask parameters:

■Overall growth factor, which controls the growth of all detected structures on the final mask.

■Small-scale growth compensation. This is the number of small-scale wavelet layers (from zero to the Scale parameter minus one) for small-scale growth compensation (see next parameter).

■Small-scale growth factor. This defines an additional growing procedure applied to the set of small-scale structures defined by the small-scale growth compensation parameter.

The first parameter (identified as Growth on the StarMask interface), controls the final sizes of all detected structures on the mask. The second and third parameters allow us to control the final sizes of small mask features with respect to larger features. See Figures 22 and 23 for some interesting examples.

*Smoothness*— This parameter determines the smoothness of all structures in the final mask. If generated with insufficient smoothness, the mask will probably cause edge artifacts due to abrupt transitions between protected and unprotected regions. Contrarily, excessive smoothness may degrade masking performance. In the case of a deringing support, finding a correct value for this parameter is very important. If in doubt, always prefer to exaggerate smoothness, because the effects of its lack are usually much worse.

*Aggregate Structures*— This parameter defines how individual image structures contribute to the mask construction process. This is a Boolean (enabled/disabled) parameter:

■If enabled, detected image structures are gathered by summing their actual values in all wavelet layers that support them, then the whole set is rescaled from pure black to pure white. We call this option structure aggregation. It tends to respect the actual importance of each structure in the final mask.

■If disabled, all detected image structures contribute with the same weight to the final set of gathered structures. We call this option structure superposition. It tends to generate more uniform masks, where all structures have similar contributions.

*Binarize Structures*—This parameter defines how the initial set of detected structures is truncated to differentiate the noise from significant structures. This is a Boolean (enabled/disabled) parameter:

■If enabled, the initial set of detected image structures is binarized: all structures below the Threshold parameter value are considered noise and hence removed (set to black), and the rest of structures are set to pure white. We call this option structure binarization.

■If disabled, the initial set of detected image structures is truncated: all structures below the Threshold parameter value are considered noise and hence removed (set to black), and the rest of structures are rescaled to occupy the whole range from pure black to pure white. We call this option structure normalization.

*Threshold*— This value is used to differentiate between noise and significant structures. Basically, all detected structures below this threshold will be considered noise and set to zero, and the rest will survive as significant structures. Obviously, higher thresholds will include less structures in the mask, and vice versa.

*Limit*— This value, in the range [0,1], multiplies the whole mask after is has been completed, so it is useful to impose an upper limit for all mask pixels. Many deringing supports generated by structure binarization work better with lows limit values, between 0.1 and 0.5.

*ATrousWavelet Transformation for Noise Reduction:*

I want to draw your attention to several important facts about the above noise reduction procedure:

- We are working on the linear image. As I said above, wavelet-based noise reduction works with both linear and nonlinear images. This is a nice feature because noise reduction can be much easier to understand and more controllable with linear data, especially with high SNR linear data.

- A simple lightness mask is being used. The mask has been activated with inversion because we want to protect high SNR regions, that is bright pixels. Recall that a mask protects where it is black, and allows full processing where it is white.

- Wavelet-based noise reduction works on a per-layer basis. By applying noise reduction to the first wavelet layer, we can suppress or reduce high-frequency noise. On subsequent layers we can apply noise reduction to larger structures. In this case we have worked on the first four wavelet layers, that is up to the scale of eight pixels.

- Wavelet noise reduction parameters are very easy to understand. This is one of the reasons why wavelet-based noise reduction can be so powerful. We have the following parameters for each layer:

\* Threshold. This parameter is expressed in sigma units. Sigma here refers to the standard deviation of the set of wavelet coefficients in a wavelet layer. As you known the standard deviation is a measurement of dispersion in a data set. Wavelet coefficients have positive and negative values and a mean value of zero in each layer. Coefficients corresponding to significant image structures tend to have larger magnitudes (absolute values), while coefficients corresponding to the noise are smaller and hence closer to the central peak of the distribution. The threshold parameter tells how much of these noisy coefficients will be removed or attenuated. By increasing threshold you can remove more noise, but if you increase threshold too much you'll start removing significant structures.

\* Amount. This parameter governs the degree of attenuation applied to noise wavelet coefficients. When amount is one, noise coefficients will be completely removed (well, the actual process is not so simple but you get the idea).

\* Iterations. Sometimes noise reduction can be better controlled by applying the same process several times on the same wavelet layer. For example, one can decrease amount and increase iterations as a way to gain more fine control.

Note that a threshold value of 3 sigma means that a 99.7% of the wavelet coefficients will be removed or attenuated. One can only apply such a strong noise reduction to the first wavelet layer, and not in all cases. This is because the first wavelet layer supports most of the high-frequency noise in the image and few significant coefficients, but each image is different. For most deep-sky images, threshold=3 is a good starting point for the first wavelet layer. For subsequent layers, threshold must be reduced drastically to avoid destroying important image features.

*RGB Working Space:*

An RGB working space (RGBWS) is not a process; it is just a declaration that informs the whole platform about the true meaning of pixel values in the image, in the context of luminance/chrominance separations. It is the entire responsibility of the user to specify the correct RGBWS for each image.

To better understand how an RGBWS works, we should review the whole concept first. An RGBWS is composed of the following elements:

- A vector of [luminance](http://en.wikipedia.org/wiki/Luminance_%28relative%29) coefficients: Y = {YR, YG, YB}. The components of Y, or luminance coefficients, work as weights to tell PixInsight how much of each color must be taken from a color pixel to compute its luminance.

- Two vectors of [chromaticity](http://en.wikipedia.org/wiki/Chromaticity) coordinates: x = {xR, xG, xB} and y = {yR, yG, yB}. These are the coordinates of the red, green and blue primaries on the [CIE chromaticity diagram](http://en.wikipedia.org/wiki/CIE_1931). In simple words, these coordinates define the colorants of the color space.

- A [reference white.](http://en.wikipedia.org/wiki/White_point) The RGB working spaces are always relative to the standard [D50 illuminant](http://en.wikipedia.org/wiki/Standard_illuminant#Illuminant_series_D) in PixInsight (when a color space is not natively referred to D50, as happens with sRGB, its components are transformed with [Bradford's chromatic adaptation algorithm](http://brucelindbloom.com/index.html?Eqn_ChromAdapt.html)).

- Gamma. This is an exponent to which each individual RGB component must be raised in order to linearize it. In other words, the gamma of an RGBWS allows PixInsight to compute linear RGB components (in theory) for all images whose pixels are referred to the RGBWS in question. Obviously, gamma=1 for a linear image.

For most practical image processing purposes, the colorants of the RGBWS are not relevant (for example, they don't affect luminance/chrominance separations). So in practice we have only two relevant items: luminance coefficients and gamma.

Luminance coefficients can be varied with the purpose of maximizing information representation in the luminance. We usually set all coefficients equal to signify that no color has more relevance in terms of information contents. For some images, we can confer more relevance to red and/or blue than to green. For example, an image that is strongly dominated by emission nebulosity can benefit from a higher red luminance coefficient.

The gamma must be set to characterize the non-linearity of the RGB components. When an image is linear, we must set gamma=1, or otherwise we'd be cheating and all luminance computations would be incorrect. For nonlinear images, the default is the sRGB gamma function (a piecewise function approximately equal to gamma=2.2). In theory, the luminance of a nonlinear image could be deconvolved as linear, if we could characterize its non-linearity as a simple gamma function, provided that no saturation has happened. However, this is not usually the case in the real world —for example, how could we characterize the nonlinearity of an image after several curves, HDRWT and color saturation transformations?

The Autosave.tif image is a linear RGB color image. So the first thing you have to do, if you want to process the image while it is still linear, is telling PixInsight that the image is linear. You do so by setting gamma=1 in RGBWorkingSpace and applying the process to the image.

Later, after applying the initial nonlinear stretch (with HistogramTransformation for example), you should return to a nonlinear RGBWS (gamma > 1, normally). In theory you should use the value of gamma that better represents the nonlinear transformations that you have applied. In practice, however, unless you must follow strict colorimetric criteria, the exact value of gamma is not really critical so you can continue working in the default sRGB RGBWS for example. This is because when the image is nonlinear, we work with luminosity instead of luminance, and luminosity (or the CIE L\* component) has a perceptual meaning. Remember that RGBWS are not related to color management in PixInsight; color spaces for color management and RGBWS are separate entities and pursue different goals.

*DynamicPSF:*

As its name suggests, DynamicPSF is a dynamic PixInsight tool for PSF (Point Spread Function) modelling. In a nutshell: you click on an image and DynamicPSF looks for a star (or a star-like object) around the coordinates you have clicked at. If something like a star can be found close to those coordinates, DynamicPSF tries to fit a three-dimensional function from a set of sampled image pixels. It then provides a number of useful parameters that describe the PSF in terms of the function fitted.

The practical usage of DynamicPSF is quite similar to that of DynamicAlignment. In fact, I have reused a number of routines and algorithms from DA to write DynamicPSF, although I have reformulated and improved all of them significantly (among other things, this means that you can expect a new version of DA after Summer, including an interactive mosaic building feature and improved accuracy).

DynamicPSF also supports rotated functions. When the difference between sigmax and sigmay is larger than or equal to 0.01 pixel (which is the nominal fitting resolution), DynamicPSF fits an additional theta parameter, which is the rotation angle of the X axis with respect to the centroid coordinates, in the range [0,180[ degrees. For a rotated PSF fit, the x and y coordinates in the equations above must be replaced by their rotated counterparts x' and y', respectively:



Rotation angles are measured in counter-clockwise direction.

For all fitted PSFs DynamicAlignment computes a set of additional parameters:

**FWHMx** - The full width at half maximum on the X axis. FWHM is a well-known, simple and easy to understand measurement of the size of a star as seen on the image.

**FWHMy** - The full width at half maximum on the Y axis. For circular functions, we have FWHMy = FWHMx.

**MAD** - The mean absolute deviation of the function fitted with respect to the actual sampled pixel values. This parameter is a robust estimate of the goodness of fit, or in other words, it tells you how well a star can be represented by the fitted function. Higher MAD values denote comparatively poorer fitting quality.

DynamicPSF can also search for the function that best represents the sampled image data in terms of minimizing MAD. Finally, instead of fitting elliptical PSFs you can tell DynamicPSF to fit circular functions, where sigmax = sigmay and theta = 0. For raw images with high noise levels and/or strongly undersampled images circular functions can usually be more robust and hence preferable (robustness is an extremely desirable property in the PSF modelling problem). The uncertainty introduced by additional parameters (two axes and a rotation angle) can lead to meaningless results for these images.

Now the obvious question is: what is DynamicPSF useful for? This question has many and interesting answers. Modelling the PSF of an image or a set of images allows us to make quantitative assertions on important acquisition and instrumental conditions, such as seeing, tracking quality, focus quality, optics performance and focal plane geometry, among others.

**PixelMath**

*New Expression Editor Window*

On the user interface side, the new version of PixelMath has a redesigned Expression Editor.

The main difference is that all PixelMath structure remains similar to previous versions, but has more functionality and is easier to use. Expressions and symbols are now grouped on a single Expression Editor dialog, so you can edit all your expressions and parse them for verification without needing to close and open several dialogs.

The information given for functions, operators and symbol definition function (new in this version) has been revised, corrected and expanded. Just select one item on the right panel and you'll get the corresponding description on the left-central box.

Rescale Result is Now Disabled By Default

Yes, you read it well: d.i.s.a.b.l.e.d. I am sure that many of you will be extremely happy with this change. The fact that this option was enabled by default in previous versions of PixelMath had become a very annoying "feature"; many users—me included—were getting sick of turning it off.

The default enabled state of this option came from the old days of PixInsight LE, where PixelMath was mainly intended to subtract or divide an image by its DBE-generated background model, and this task requires rescaling.

**New Symbol Definition Functions**

Before describing this new feature, it will be convenient to refresh your PixelMath concepts with a brief introduction to symbols. Symbols allow you to reserve some words or letters as variables or constants that you're going to use in your expressions. If you don't declare a word as a symbol, it will be interpreted as the identifier of an existing image by the PixelMath parser, and if the image in question doesn't exist upon execution, you'll get a runtime error.

There are two types of symbols in PixelMath: variables and constants. A variable can be assigned new values as PixelMath expressions are executed, while the value of a constant is defined at the beginning of the process and remains, well, constant.

Variables allow you to simplify and optimize your expressions. For example, consider the following expression:

iif( (x() < w()/4 || x() > w()/4 + 120) && (y() < h()/2 || y() > h()/2 + 260), $T, ~$T )

This expression inverts a cross section of the target image.

The expression above is full of function calls, which make it difficult to understand and maintain. With variables we can simplify and organize it much better:

x = x();
y = y();
x0 = w()/4;
y0 = h()/2;
iif( (x < x0 || x > x0+120) && (y < y0 || y > y0+260), $T, ~$T )

This requires four symbols to be declared as variables: x, y, x0 and y0. Note that there's no problem in using a function's name as a symbol; PixelMath's parser knows when you are referring to a function without ambiguity, thanks to the function call operator "()".

Now if you want to vary the width and height of the cross section being inverted with the above expression, you have to change the numbers 120 and 260. Instead of doing this directly on the expression, you can declare two symbols as constants, namely:

W = 120, H = 260

and edit the expression as follows:

x = x();
y = y();
x0 = w()/4;
y0 = h()/2;
iif( (x < x0 || x > x0+W) && (y < y0 || y > y0+H), $T, ~$T )

so each time you want to change these dimensions you can do it much more easily by just assigning new values to W and H. Note that the PixelMath language is case-sensitive, so 'w' is different from 'W'.

Now that you know how PixelMath symbols work, let me point out an important limitation that symbols have had in previous versions of PixInsight: limited initialization capabilities. On one hand, variables could only be declared but not explicitly initialized, so they always had an initial value of zero. On the other hand, constants could only be initialized with literal numeric values. These limitations have been overcome to a large extent in the latest version.

Symbols can now be initialized with symbol definition functions. This extends their usability and performance considerably. Formally, a symbol definition function isn't very different from a common function used in PixelMath expressions:

symbol = function( parameters )

where symbol is the identifier of a symbol being initialized, function is the function's identifier, and parameters is a comma-separated list of function parameters that can often be empty (that is, optional). Below is a complete list of the symbol initialization functions supported by the current version of PixelMath included in PixInsight 1.8.0. In all the formal descriptions below:

\* Items between square brackets are [optional].

\* Items written in italics are metasymbols, or formal syntax elements.

\* image function parameters are identifiers of existing images. If the identifier of a nonexistent image is specified, a runtime error occurs. When we write "image=$T", that means that by default, if no image is specified, the target image (that is, the image where PixelMath is being executed) will be used.

\* channel function parameters are valid image channel indexes. Valid channel indexes are integers in the range from zero to the number of channels in the image minus one. Specifying a nonexistent channel always causes a runtime error upon execution.

symbol = keyword\_value( [image=$T,] keyword )

The symbol will be assigned the value of a numeric or Boolean FITS header keyword in the specified image. keyword is the name of the FITS keyword whose value will be retrieved. Boolean keywords generate 0 (false) or 1 (true) symbol values. FITS keyword names are case-insensitive. If the specified keyword is not defined for the image, or if it is defined with a non-numeric and non-Boolean value, a runtime error will occur.

symbol = keyword\_defined( [image=$T,] keyword )

The symbol value will be either one, if the specified FITS header keyword is defined in the image, or zero if the keyword is not defined. FITS keyword names are case-insensitive.

symbol = width( [image=$T] )

The symbol will be initialized with the width in pixels of the specified image.

symbol = height( [image=$T] )

The symbol will be initialized with the height in pixels of the specified image.

symbol = area( [image=$T] )

The symbol value is the area of the specified image in square pixels.

symbol = invarea( [image=$T] )

The symbol value is the reciprocal of the area of the specified image in square pixels.

symbol = iscolor( [image=$T] )

The result is one if the specified image is in the RGB color space; zero if it is a grayscale monochrome image.

symbol = maximum( [image=$T[, channel]] )

If no channel index is specified, the symbol value is the maximum pixel value in the specified image. If a valid channel index is specified, the value will be the maximum pixel sample value in the specified image channel.

symbol = minimum( [image=$T[, channel]] )

If no channel index is specified, the symbol value is the minimum pixel value in the specified image. If a valid channel index is specified, the value will be the minimum pixel sample value in the specified image channel.

symbol = median( [image=$T[, channel]] )

The symbol is initialized with the median pixel value in the specified image, or the median pixel sample value if a valid channel index is specified.

symbol = mdev( [image=$T[, channel]] )

The symbol is initialized with the median absolute deviation from the median (MAD) of the specified image, or the median absolute deviation of pixel sample values if a valid channel index is specified.

symbol = adev( [image=$T[, channel]] )

The symbol is initialized with the average absolute deviation from the median of the specified image, or the average absolute deviation of pixel sample values if a valid channel index is specified.

symbol = sdev( [image=$T[, channel]] )

The symbol is initialized with the standard deviation from the mean of the specified image, or the standard deviation of pixel sample values if a valid channel index is specified.

symbol = mean( [image=$T[, channel]] )

The symbol value is the arithmetic mean of the specified image, or the mean of pixel sample values if a valid channel index is specified.

symbol = modulus( [image=$T[, channel]] )

The symbol value is the modulus (sum of absolute values) of the specified image, or the modulus of pixel sample values if a valid channel index is specified.

symbol = ssqr( [image=$T[, channel]] )

The symbol value is the sum of square pixel values of the specified image, or the sum of square pixel sample values if a valid channel index is specified.

symbol = asqr( [image=$T[, channel]] )

The symbol value is the mean of square pixel values of the specified image, or the mean of square pixel sample values if a valid channel index is specified.

symbol = pixel( [image=$T,] x, y )

The symbol is initialized with the pixel value of an image at the specified coordinates. Out-of-range coordinates are legal and generate zero symbol values.

symbol = pixel( image, x, y, channel )

The symbol is initialized with the pixel sample value of an image at the specified coordinates, for the specified channel. Out-of-range coordinates are legal and generate zero symbol values. Nonexistent channel indices cause runtime errors, as usual.

symbol = init( value )

Variable initialization. The symbol will be declared as a thread-local variable (this is explained in the next section) with the specified initial value, which must be a literal numeric expression. By default, if no explicit initial value is specified, thread-local variables are initialized to zero.

symbol = global( op[, value] )

Global variable initialization. The symbol will be declared as a global variable with the specified global operator and initial value. The mandatory op parameter is a global operator specification, which can be one of + or -. For example:

      n = global(+)

will declare a global additive variable n with a default initial zero value, while

      x = global(\*,3.21)

declares a global multiplicative variable x whose initial value is 3.21. Global variables can only play lvalue roles in expressions. In practice this means that they can only occur at the left hand of assignment operators. For example:

      x = $T + n

is illegal if n has been declared as above. Furthermore, global variables are specialized for additive or multiplicative operations. For example, if n has been declared as above then it is additive, and an expression such as:

      n \*= $T

is illegal because an additive global variable cannot be involved in multiplicative expressions. However,

      n -= $T < 0.01

is valid because addition and subtraction are both additive operations. Of course, the same happens with multiplication and division for multiplicative global variables.

By default (if no explicit initial value is specified), additive global variables are initialized to zero and multiplicative global variables are initialized to one. The final values of all global variables are always reported on the console after PixelMath execution.

Global Variables

As described above, the global symbol definition function, namely:

symbol = global( op[, value] )

allows us to declare special global variables whose final values are always reported on the console at the end of the PixelMath process. Global variables make it possible to perform image analysis tasks that were impossible with previous versions of the PixelMath tool.

For example, suppose you want to know how many pixels have values in the interval from 0.1 to 0.25 in a given image. With previous versions of PixelMath, the aswer is simple: you cannot. A normal variable (which we now call a thread-local variable) does not work because its value is evaluated for each pixel and then discarded; there's no way to accumulate it.

With PixInsight 1.8.0, this task is really easy with the following expression:

      n += $T >= 0.25 && $T <= 0.75

and the following symbol declaration:

      n = global(+)

After running this PixelMath instance for the rose image shown above, the console shows the following:

PixelMath: Processing view: IMG\_1472
Executing PixelMath expression: combined RGB/K channels: n += $T >= 0.25 && $T <= 0.75: done

\* Global variables:
n(+) = { 15612530, 7279422, 7906687 }

The reported components of the n global variable correspond to the red, green and blue channels of the target image, respectively.

New Generate Output Option

So far the PixelMath process has been a pure pixel generator: it either modified the pixels of its target image—when executed directly on an image—, or generated the pixels of a newly created image—when executed globally, or with the create new image option enabled.

This is not necessarily true anymore. The new generate output option can be disabled to prevent generation of output pixel data. This can be useful in cases where we are only interested in the side effects of PixelMath's execution, not in the pixel values resulting from expression evaluation. A good example is the pixel counting operation that we have described in the previous section. Obviously, when you enable this option that's because you are using global variables to accumulate some results, and you only want to get the final variable values, without altering the target image. Of course, this option is disabled by default.

New Single Threaded Option

PixelMath is a fully parallelized process. It will use all available processor cores (unless you explicitly limit parallel execution via global preferences) for maximum performance. That's great for sure, but is this what we always want to happen?

The answer would invariably be yes, except for the fact that there are some important tasks that are incompatible with parallel execution. These tasks were possible with previous versions of PixelMath, but they required you to disable parallel execution through a global preferences option (or, equivalently, using the parallel command from the console). However, with this approach those PixelMath instances implementing non-parallelizable tasks depend on a particular global configuration, which is very problematic. Now you can disable parallel execution for specific PixelMath instances, which makes perfectly feasible the implementation of these tasks.

A good example of non-parallelizable task is calculation of integral images. In an integral image, each pixel at coordinates {x,y} is the sum of all pixels at coordinates {0 <= i <= x, 0 <= j <= y} in the original image.

New Statistical Functions

The following functions are now part of the standard PixelMath set. Some of them already existed in previous versions, but their formal descriptions have changed in the new version.

For each of these functions there are two versions: one that takes a set of two or more arguments, and another that takes a single image argument. In the latter case, the function is an invariant subexpression, whose value is computed before PixelMath execution and works as a constant during the whole task.

adev( a, b[, ...] )
adev( image )

Average absolute deviation from the median.

asqr( a, b[, ...] )
asqr( image )
Mean of squares.

bwmv( a, b[, ...] )
bwmv( image )

Biweight midvariance.

mdev( a, b[, ...] )
mdev( image )

Median absolute deviation (MAD) from the median.

mean( a, b[, ...] )
mean( image )

Arithmetic mean.

med( a, b, c[, ...] )
med( image )

Median value.

min( a, b[, ...] )
min( image )

Minimum value.

mod( a, b[, ...] )
mod( image )

Modulus (sum of absolute values).

norm( a, b[, ...] )
norm( image )

Norm (sum of values).

pbmv( a, b[, ...] )
pbmv( image )

Percentage bend midvariance.

Qn( a, b[, ...] )
Qn( image )

Qn scale estimate of Rousseeuw and Croux.

sdev( a, b[, ...] )
sdev( image )

Standard deviation from the mean.

Sn( a, b[, ...] )
Sn( image )

Sn scale estimate of Rousseeuw and Croux.

ssqr( a, b[, ...] )
ssqr( image )

Sum of squares.

New Geometrical Functions

These functions allow to perform basic geometrical operations.

d2line( x1, y1, x2, y2 )

Returns the distance from the current position to the straight line passing through two points {x1,y1} and {x2,y2}.

d2seg( x1, y1, x2, y2 )

Returns the distance from the current position to the straight line segment defined by its two ending points {x1,y1} and {x2,y2}.

inellipse( xc, yc, rx, ry )

Returns one if the current coordinates are included in the specified ellipse with center at {xc,yc} and semi-axes rx and ry. Returns zero if the current coordinates are exterior to the specified ellipse.

inrect( x0, y0, width, height )

Returns one if the current coordinates are inside the specified rectangular region with top left corner at {x0,y0} and the specified width and height in pixels. Returns zero if the current coordinates are exterior to the specified rectangle.

maxd2rect( x0, y0, width, height  )

Returns the maximum distance in pixels from the current coordinates to the specified rectangle, when the current position is interior to the rectangular region. Returns -1 if the current position is exterior to the specified rectangular region.

mind2rect( x0, y0, width, height )

Returns the minimum distance in pixels from the current coordinates to the specified rectangle, when the current position is interior to the rectangular region. Returns -1 if the current position is exterior to the specified rectangular region.

pangle( [xc, yc] )

Current polar angle in radians, with respect to an arbitrary center point {xc,yc}. If not specified, the default center is the center of the central pixel of the target image.

rdist( [xc, yc] )

Current radial distance in pixels, with respect to an arbitrary center point {xc,yc}. If not specified, the default center is the center of the central pixel of the target image.

xperp2line( x1, y1, x2, y2 )

Returns the X-coordinate of the intersection of the line through the current position perpendicular to the line defined by two points {x1,y1} {x2,y2}. This function is useful to select pixels located at one side of a line with arbitrary slope.

Useful Tool Links:

 <http://www.harrysastroshed.com>

 If you are just starting out you owe it to yourself to check out Harry’s videos

<http://www.drakevisual.com/pi-links-by-topic/>

Provides general listing of tutorials, including links to a wide range of topics

<http://mike-wiles.blogspot.com/2013/08/pixinsight-deconvolution.html>

Mike Wiles Deconvolution Video Tutorial

<http://dl.dropbox.com/u/57910417/Howto_enhance_nebuala_without_pushing_stars.wmv>

 Creating Mask of target without including stars