MaskGen

**Journaling Tool**

**3 November 2016**

Prepared for:

**Submitted by:**

|  |  |  |  |
| --- | --- | --- | --- |
| **REVISION RECORD** | | | |
| **REVISION** | **DATE** | **DESCRIPTION & PAGE AFFECTED** | **AUTHOR** |
|  | 7/20/16 | Initial Revision |  |
|  | 8/25/16 | Batch and Video Enhancements |  |
|  | 9/8/16 | New Video/Image install and execution procedures. |  |
|  | 10/11/16 | New installation procedures. Consolidation of Clone operations. Removal of Paste Duplicate |  |
|  | 11/02/16 | New installation procedures. Consolidation of Clone operations. Removal of Paste Duplicate |  |
|  | 11/3/16 | Updated information regarding Create JPEG/TIFF group operation. Formatting. |  |
|  |  |  |  |

**Table of Contents**

[1) Overview 1](#_Toc467166607)

[2) Installation 1](#_Toc467166608)

[2.1 Dependencies 2](#_Toc467166609)

[2.1.1 Anaconda, FFMPEG and OpenCV on Windows 3](#_Toc467166610)

[2.2 HDF5 4](#_Toc467166611)

[2.3 FFMPEG for Python 2.7 (not Anaconda) 4](#_Toc467166612)

[3) General Operation 4](#_Toc467166613)

[3.1 Starting the UI 4](#_Toc467166614)

[3.2 Projects 5](#_Toc467166615)

[3.3 Processing 7](#_Toc467166616)

[3.4 Workflow 8](#_Toc467166617)

[3.4.1 JPEG Workflow 9](#_Toc467166618)

[4) Graph Operations 10](#_Toc467166619)

[4.1 Linking Nodes 10](#_Toc467166620)

[4.1.1 Video Displays 11](#_Toc467166621)

[4.2 Interface Demonstration 12](#_Toc467166622)

[12](#_Toc467166623)

[5) Link Descriptions 13](#_Toc467166624)

[5.1 Input Masks 14](#_Toc467166625)

[5.2 Paste Sample 15](#_Toc467166626)

[5.3 The PasteSplice Operation 16](#_Toc467166627)

[17](#_Toc467166628)

[5.4 Shadow from Donor Image 17](#_Toc467166629)

[5.5 Output PNG and Image Rotations 18](#_Toc467166630)

[5.6 Applying Filters (Plugins) 20](#_Toc467166631)

[5.7 Link Inspection 21](#_Toc467166632)

[5.8 Video Link Descriptions 22](#_Toc467166633)

[5.9 Other Metadata 24](#_Toc467166634)

[6) Mask Generation 24](#_Toc467166635)

[6.1 Video Masks 25](#_Toc467166636)

[6.2 Composite Mask 25](#_Toc467166637)

[7) Analytics 26](#_Toc467166638)

[8) Plugins 27](#_Toc467166639)

[8.1 Arguments 28](#_Toc467166640)

[8.2 Plugin Builder 28](#_Toc467166641)

[9) EXPORT 29](#_Toc467166642)

[10) Group Manager 30](#_Toc467166643)

[11) Validation 30](#_Toc467166644)

[12) QA Process 30](#_Toc467166645)

[13) Batch operation 31](#_Toc467166646)

[Different arguments will trigger different functionality. 32](#_Toc467166647)

[13.1 Creating a Project 32](#_Toc467166648)

[13.2 Extending a Project 32](#_Toc467166649)

[13.3 Capping a Project: Antiforensics 33](#_Toc467166650)

[13.4 Image File Names 33](#_Toc467166651)

[14) JSON 33](#_Toc467166652)

[15) Project properties 38](#_Toc467166653)

[15.1 Example Edge with Specific Operation 38](#_Toc467166654)

[15.2 Example Edge with Specific Operation and Argument Value 38](#_Toc467166655)

[15.3 Example with a Rule 38](#_Toc467166656)

[15.4 Example Project Property without a Rule 39](#_Toc467166657)

**List of Figures**

[Figure 1: Example of a graph produced by the journaling tool. Specifically, a region of one image (hat) was spliced into another image (orig\_input) to create a new image (input\_mod\_1). 1](#_Toc467166586)

[Figure 2: GitHub website download dialog 2](#_Toc467166587)

[Figure 3: Basic UI. The image nodes open here are from the example project based in the “Images” directory specified in the command line. 5](#_Toc467166588)

[Figure 4: File menu. 6](#_Toc467166589)

[Figure 5: Process menu. 7](#_Toc467166590)

[Figure 6: Description of user interface. 12](#_Toc467166591)

[Figure 7: Link description window. 13](#_Toc467166592)

[Figure 8: Parameter Edit 14](#_Toc467166593)

[Figure 9: Example of an input mask one might use during seam carving to ensure preservation of the woman’s hat. 15](#_Toc467166594)

[Figure 10: Paste Splice for Copy/Paste 16](#_Toc467166595)

[Figure 11: Rules or Paste Splice 17](#_Toc467166596)

[Figure 12: Shadow Paste 18](#_Toc467166597)

[Figure 13: Image Orientation Prompt 19](#_Toc467166598)

[Figure 14: Summary of Orientation Workflow 20](#_Toc467166599)

[Figure 15: Window for applying plugins to image nodes. 21](#_Toc467166600)

[Figure 16: Link Inspection 22](#_Toc467166601)

[Figure 17: EXIF Comparison Table 22](#_Toc467166602)

[Figure 18: Video Link Inspection 23](#_Toc467166603)

[Figure 19: Video Mask Table Menu 24](#_Toc467166604)

[Figure 20: Composite Mask 26](#_Toc467166605)

[Figure 21: Plugin Builder Interface, filled out with an “AutoGammaCorrection” plugin 29](#_Toc467166606)

# Overview

The journaling tool can be used to keep track of image manipulations and software according to NIST guidelines. The tool has the primary function of generating mask images that highlight changes made for every individual manipulation. These masks will be used later to evaluate accuracy of manipulation detection. The journaling tool will also create a graph of manipulations performed, an example of which is shown here, introducing *nodes* and *links*.



Figure 1: Example of a graph produced by the journaling tool. Specifically, a region of one image (hat) was spliced into another image (orig\_input) to create a new image (input\_mod\_1).

Quality and thorough journaling is critical. More information captured during journaling provides insurance that future post processing activities have the required information.

Do not combine manipulations together. For example, do not convert to JPG and perform a crop at the same time.

Please follow the quality assurance guidelines outlined in this document (11 QA Process).

# Installation

The most direct way to obtain the most recent version of the tool is to download it from GitHub. Navigate to <https://github.com/rwgdrummer/maskgen> and click the green “Clone or Download” button. The user can either download everything in a ZIP, or clone using GitHub’s desktop application. Experienced users may choose to clone via the *git* command line tool.



Figure 2: GitHub website download dialog

If the user has opted to download the ZIP file, simply extract to the desired location. The tool is run within a Python 2.7 interpreter.

## Dependencies

There are several Python libraries required for operation:

* Tkinter
* PIL/pillow
* numpy
* matplotlib
* [opencv](http://opencv.org/) (cv2)
* networkx
* moviepy
* scikit-image
* tkintertable
* bitstring
* boto3 (for optional use of S3)
* hdf5
* rawpy

Most of these can be installed easily are installed with the following procedure.

python setup.py develop

For use with using plugins that write TIFF files, install tifffile. For Mac users, XCode needs to be installed. For Windows users, Microsoft Visual C++ 9.x for python needs to be installed. An alternative is to find a prebuilt libtiff library for Mac or Windows. Download at your own risk.

pip install tifffile

Anaconda users may need to need install a different Pillow library. The instructions are as follows.

conda remove PIL

conda remove pillow

pip install Image

The tool depends on the installation of [exiftool](http://www.sno.phy.queensu.ca/~phil/exiftool/). An alternate tool can be provided, setting the MASKGEN\_EXIFTOOL environment variable prior to running the tool. Additionally, for use of S3, the user must also install awscli (*pip install awscli*) and configure using *aws configure*.

[TKinter](http://www.tkdocs.com/tutorial/install.html) requires the installation of TCL and TK. Most Mac OS have TCL installed. The tool is tested with [TCL 8.5](https://www.tcl.tk/software/tcltk/8.5.html).

If there is a cv2 import error when using the tool, try setting the PYTHONPATH to the location of the python site packages. For example:

export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python2.7/site-packages

A common issue is using more than on python version on a single machine. In this case, ‘pip’, the python installer may reference the incorrect python. Try using ‘python –m pip’ instead of ‘pip’ if import errors occur and ‘pip’ confirms the existence of the module.

### Anaconda, FFMPEG and OpenCV on Windows

Some Python distributions come with some or all of these pre-installed, such as [Anaconda](https://www.continuum.io/downloads). OpenCV is rarely included in such distributions and has a slightly more involved installation. Helpful instructions can be found in the [OpenCV documentation](http://docs.opencv.org/3.1.0/d5/de5/tutorial_py_setup_in_windows.html#gsc.tab=0)

*DO NOT install openv from menpo or ffmpeg using a conda install*. Follow these instructions. The menpo version of opencv is 2.4.11. JT requires opencv 2.4.13

Reproduced from: http://mathalope.co.uk/2015/05/07/opencv-python-how-to-install-opencv-python-package-to-anaconda-windows/

NOTE: The instructions below us *vc11*; *vc12* is also acceptable.

* Dowload and install FFMPEG 3.1.1.
* Download and install opencv 2.4.13 fro opencv.org using the appropriate architecture (x86 or x64). Make sure it is built with FFMPEG. This is confirmed by looking for opencv/build/x64/vc11/opencv\_ffmpeg2413\_64.dll. Follow build-from-source instructions for opencv to include FFMPEG.
* Copy opencv/build/python/2.7/x64/cv2.pyd (or x86/cv2.pyd for the x86 architecture) to Anaconda2/Lib/site-packages
* Set environment variables:
  + 32-bit: Set OPENCV\_DIR to the absolute path of opencv/x86/vc11
  + 64-bit: Set OPENCV\_DIR to the absolute path of opencv/x64/vc11
  + Add to PATH: %OPENCV\_DIR%\bin
  + Add to Path: FFMPEG’s bin directory
* Test :
  + run *python*
  + *import cv2*
  + *print cv2.\_\_version\_\_*

## HDF5

If the initial setup fails due to an hdf5 error, trying installing hdf5 separately.

For Mac OS:

brew install homebrew/science/hdf5

For Windows, use the latest instructions on https://www.hdfgroup.org/

## FFMPEG for Python 2.7 (not Anaconda)

Video processing requires ffmpeg to be installed and accessible via the PATH environment variable.

For Python 2.7:

pip install ffmpeg

# General Operation

## Starting the UI

Open an instance of command prompt (Windows)/terminal (Mac/Linux), and navigate (cd) to the root maskgen folder. This will likely be called just “maskgen,” or it might be called “maskgen-master” if downloaded via ZIP. Run here using:

python –m maskgen.MaskGenUI --imagedir images

* The *imagedir* argument is an initial project directory or project (JSON) file in the project directory. A project directory is simply the location where manipulated image steps are stored.

If the project JSON is not found and the imagedir contains is a set of images, then the images are sorted by time, oldest to newest. The first image file in the sorted list is used as the base image of the project and as a basis for the project name. All images in the imagedir are imported into the project. An alternative base image can be chosen using the --base command parameter:

python –m maskgen MaskGenUI --imagedir images --base images/baseimage.jpg

By default, the tool will assume three files: operations.json, project\_properties.json and software.csv are located in the same directory as the tool (or in the resources directory). If these files are to be downloaded from an S3 bucket, then use *aws configure* (after installing awscli). Then add the s3 argument:

python –m maskgen.MaskGenUI --imagedir images --s3 bucket/path

Running the tool specifying only the imagedirectory will open an interface that looks like that in Figure 3. The project shown is an example project that is included with the tool.

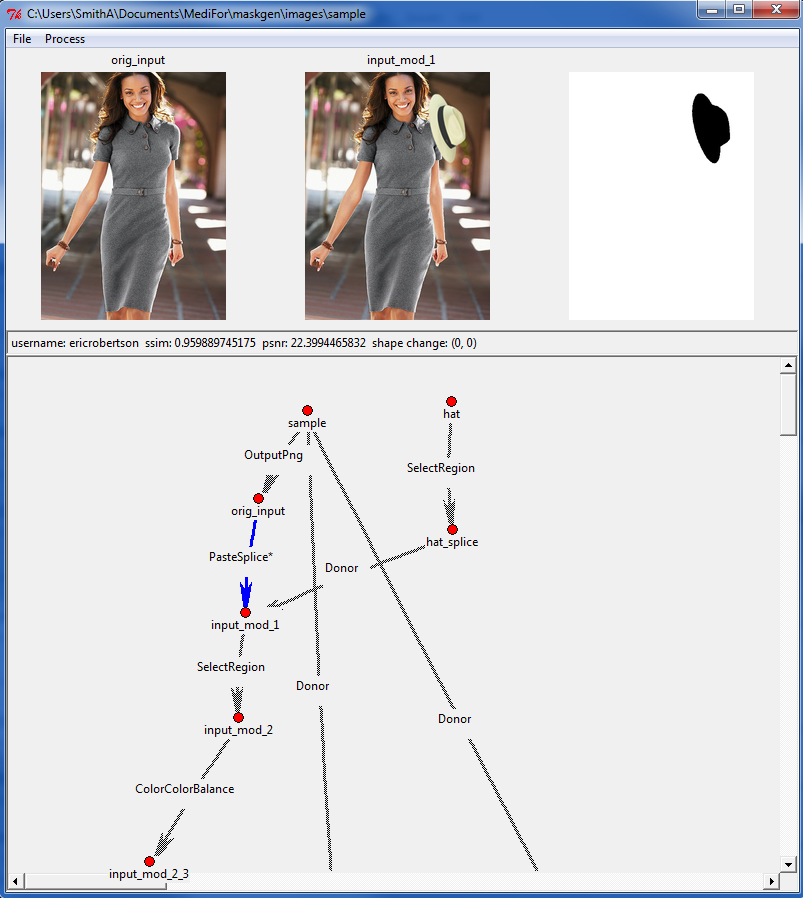


Figure 3: Basic UI. The image nodes open here are from the example project based in the “Images” directory specified in the command line.

## Projects

Projects are at the core of the journaling tool. The journaling tool represents a project simply as a directory that contains images (both original and manipulated), generated image masks, and a project file (.json). **A project directory can have only one project file**.

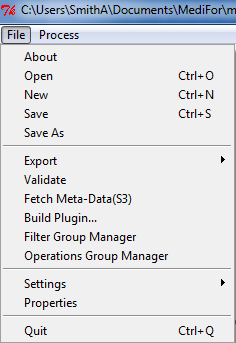


Figure 4: File menu.

A new project directory can be created or opened in the ‘File’ menu on the toolbar.

**New**: Creates a new project. Creating a new project will prompt the user for to select a base image file. The directory containing that image file becomes the project directory. The name of the project is based on the name of the image file, sans the file type suffix. All images in the directory are automatically imported into the project as nodes.

**Save**: Saves the current instance into the JSON data, and can be re-opened at a later time with **Open**.

**Save As**: Copies the contents of the current project directory into another directory, with the option of re-naming the project.

**Export – To File**: will create a compressed archive file of the project, including all images and masks. Export also validates the project, running the graph rules, builds composite masks and calculates summary information for final image nodes as described by rules in project\_properties.json (those properties with ‘node’ : true).

**Export – To S3**: creates a compressed archive file of the project and uploads to an S3 bucket and folder. The user is prompted for the bucket/folderpath, separated by ‘/’. Export also validates the project, running the graph rules, builds composite masks and calculates summary information for final image nodes as described by rules in project\_properties.json (those properties with ‘node’ : true).

**Validate**: Runs a set of validation rules on the project. Errors are displayed in a list box. Clicking on each error highlights the link or node in the graph, as if selected in the graph.

**Fetch Meta-Data (S3)**: Prompts the user for the bucket and path to pull down operations.json project\_properties.json and software.csv from an S3 bucket. The user is prompted for the bucket/folderpath, separated by '/'.

**Build Plugin…**: Opens a separate dialog to allow the user to easily construct a plugin from a command line image processing command.

**Group Manager**: Opens a separate dialog to manage groups of plugin filters.

**Settings - Username**: Allows the user to set the username attached to project links.

**Settings - Filetypes**: Allows the user to add additional file types to loading images or videos.

**Properties**: Allows user to give project a description and technical summary. This may detail the specific scenario assigned to the project, and/or any other relevant information.

## Processing

Images can be added to the current project via the ‘Process’ menu, on the toolbar, as depicted in Figure 5.

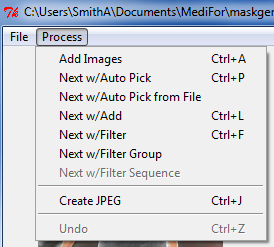


Figure 5: Process menu.

Several options are available:

* **Add Image (Video)**: Add an image (or Video) selected via file browser to project. The image can be linked to other images within the graph.
* **Next w/Auto Pick**: Automatically picks an image node without neighbors. The chosen image node is the next node in found in lexicographic order. Preference is given to those image nodes that share the same prefix as the currently selected node. A dialog appears for to capture the manipulation information including the type and additional description (optional). The dialog displays the next selected image as confirmation. A link is formed to the current image to the next selected image node.
* **Next w/Auto Pick from File**:Finds a modified version of current image file from the project directory. The modified version of the file contains the same name as the initial image file, minus the file type suffix, with some additional characters. If there is more than one modified version, they are processed in lexicographic order. A dialog appears for each modification, capturing the type of modification and additional description (optional). The dialog displays the next selected image as confirmation. A link is formed from the current image to the next selected image file.
* **Next w/Add**: See Next w/Auto Pick, except the journaling tool will instead prompt the user to add the next file in manipulation sequence, instead of automatically selecting it.
* **Next w/Filter**: Will add a new, original node created by a applying an operation to the currently selected image node. Unlike the other two 'next' functions, the set of operation is limited to those available from the tool's plugins. Furthermore, the image shown in the dialog window is the current selected image to which the selected modification is applied.
* **Next w/Filter Group**: Runs a group of plugin transforms against the selected image, creating an image node for each transform and a link to the new images from the selected range. A group must first be created via File -> Group Manager.
* **Next w/Filter Sequence**: Runs a group of plugin transforms in a sequence starting with the selected image. Each transform results in a new image node. The result from one transform is the input into the next transform. Links are formed between each image node, in the same sequence.
* **Create JPEG/TIFF**: Runs either the ‘CompressAs’ (JPEG) or ‘OutputTIFF’ (TIFF) plugins on all end nodes, using the base node as donor. For JPEGs, this will save the end node images in JPEG format using the same quantization tables as the base image. It will also copy the EXIF data from the base image. For TIFFs, this will just save the images as TIFF and copy as much metadata as possible. This option should only be used at the end of processing.
* **Undo**: Removes the last operation performed. The tool does not support undo of an undo.

## Workflow

There are two primary ways to use the journaling tool: The first is to directly export manipulation steps into separate images as the manipulations are made. Exporting every step will be very tedious, but also helpful if a mistake is made. **All intermediary steps should be exported as .PNG** or another lossless file type. An example workflow for this method would be:

**Open Image -> Crop -> Export as PNG -> Adjust Contrast -> Export as PNG -> Clone -> Export as PNG -> Run Journaling Tool -> Add image nodes to graph**

The second way is to “save-as-you-go,” that is, save over the same image each edit. The journaling tool makes copies of the imported images, and therefore it is possible to load multiple instances of the same file as different nodes. An example workflow for this method would be:

**Open Image -> Run Journaling Tool -> Add image node-> Crop -> Save -> Add Image node to graph-> Adjust Contrast -> Save -> Add image node to graph -> Clone -> Save -> Add image node to graph…**

The journaling tool has the ability to automatically select the next modified image in the directory (see section 3.3: Processing). To use this feature, image manipulation steps should be saved out in numerical order (e.g. image\_mod\_01.PNG, image\_mod\_02.PNG, etc).

### JPEG Workflow

To avoid artifacts from recompression, all edits to a JPEG image should be made on a lossless version of the image. Manipulators using the tool should:

1. Import the original JPEG image into the tool as the base image.
2. Export the JPEG image as PNG (lossless) and import the PNG image as the first step. This can be done automatically using the SaveAsPNG plugin (skip step 3).
3. Create a link with the operation ‘OutputPNG’ between the two nodes.
4. Perform manipulations as described above.
5. Use the Journaling Tool’s ‘Create JPEG/TIFF’ option to save the image back as a JPEG and copy the metadata. This will use the base JPEG image as the donor for compression information.

#### A note regarding the ‘Create JPEG/TIFF’ option

The JPEG functionality of the Create JPEG/TIFF feature is a fairly powerful processing option to save all end image nodes in JPEG format using the compression (quantization tables) and metadata of the project’s base image. It will also embed a thumbnail if the base image has one as well. If the user selects this feature, the journaling tool will automatically determine the base image and the relevant nodes to branch from, requiring no further input.

However, this feature is limited in the sense that the base image node must be a jpeg. Individual operations may be performed by selecting the end node and performing the CompressAs plugin (Process--Next w/Filter). This allows the user to select any node as a donor for quantization tables and metadata.

# Graph Operations

The graph viewer is an interactive display. The left mouse enables moving of a single node or a group of nodes (Lasso). When selecting a group of nodes, the selection box must include the nodes full text. The right mouse button (and double click) selects a node or edge, prompting the user with a menu. Node menu options include inspecting composite masks, removing nodes, exporting and all its dependencies to an archive (subset project). Edge menu options include edge inspection, edge editing, edge removal, and composite mask selection and inclusion.

All nodes are effectively the same. From a terminology standpoint, a node that does not have any in-bound links is a base image node. The base image represents an un-manipulated starting image. The base image may either be the basis for a final image(s) or a donor to the final image(s). A donor image provides some part of the image to a final image(s).

Final images do not have any outgoing edges. A project may produce more than one final image. It is best practice to have a single base image (not including donors) for all final images. Thus, each project represents all manipulated artifacts applied to single base image.

## Linking Nodes

Links record a single action (manipulation) taken on one image to produce another. An image node can only have one input link (with the exception of Paste/Splice, see below). An image node can contribute to multiple different manipulation paths resulting in many different images. Therefore, an image node may have several output links. A helpful graphical description of the UI is shown on the following page.

Image nodes may be selected, changing image display. The image associated with selected image node is shown in the left most image box. The right two boxes are left blank. Image nodes can be removed, and all input and output links to that node are removed as well. Images can be connected to another image node. When 'connect to' is selected, the cursor changes to a cross. Select another image node that is either an image node without any links (input or output) links OR an image node with one input link with operation PasteSplice (again, see Paste Splice below).

Image nodes may be exported. Exporting an image node results the creation of compressed archive file with the node and all edges and nodes leading up to the node. The name of the compressed file and the enclosed project is the node's image name (replacing '.' with '\_').

Links may also be selected, changing the image display to show the output node, input node and associated difference mask. Editing a link permits the user to change the operation and description. Links created using a plugin operation ([Process Next w/Filter [Ctrl-f]) cannot be edited. They may be inspected. Inspection opens a separate window with the description of the link.

### Video Displays

When working with video nodes, image displays contain a frame from the video. The tool selects the most diverse in color ranges over the first 25 frames of the video.

Each image display is a button. Click on the image opens the image in the default system viewer. In the case of videos, the default system viewer is a movie player.

## Interface Demonstration

## 

Image at Start of Selected Link

Image at End of Link

Mask Represented By Link

Link Statistics

Currently Selected Link

Final Result

Starting Node

(Shown on Left)

Ending Node

(Shown in Middle)

Base Image

Figure 6: Description of user interface.

# Link Descriptions

Link descriptions include a category of operations, an operation name, a free-text description (optional), and software with version that performed the manipulation. The category and operation are either derived from the operations.json file provided at the start of the tool or the plugins. Plugin-based manipulations prepopulate descriptions. The software information is saved, per user, in a local user file. This allows the user to select from software that they currently use. Adding a new software name or version results in extending the possible choices for that user. Since each user may use different versions of software to manipulate images, the user can override the version set, as the versions associated with each software may be incomplete. It is important to reach out the management team for the software.csv to add the appropriate version. Invalid versions and software fail the project validation.

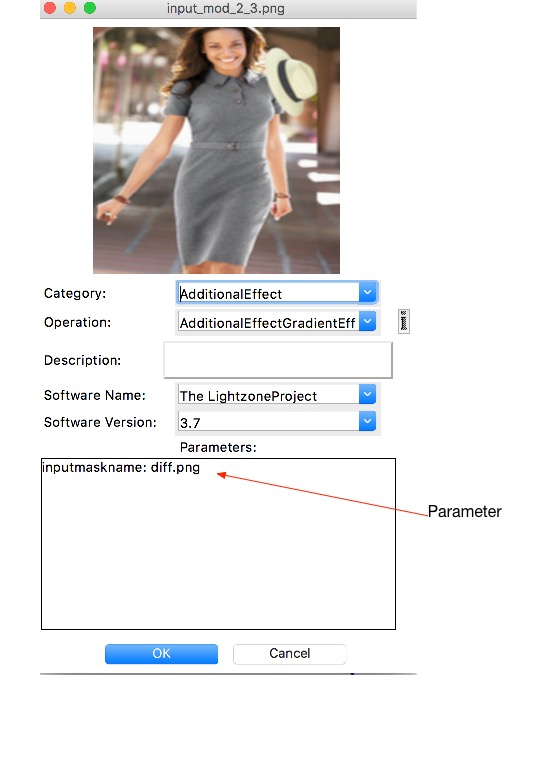


Figure 7: Link description window.

Each parameter in the parameters list is editable by clicking the parameter name. A dialog window appears prompting for the new parameter value. The tool prompts with a file selection window for parameters associated with file names. The tool validates parameter values collected from text entry type dialogs. Guidance on the semantics and syntax for each parameter is given within the parameter capture dialog.

Mandatory parameters are marked with an asterisk (‘\*’). The link edit window cannot be closed without providing valid values for each mandatory parameter.

NOTE: Occasionally, the property dialogs lose focus. Try pressing the escape key and selecting the parameter in the parameters list again.

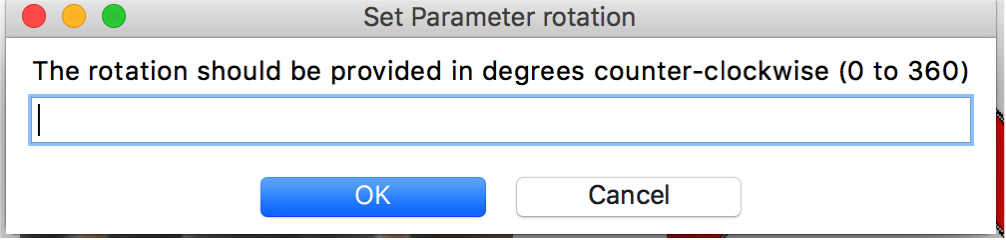


Figure 8: Parameter Edit

## Input Masks

Link descriptions can include an input mask. An input mask is a mask used by the software as a parameter or set of parameters to create the output image. For example, some seam carving tools request a mask describing areas targeted for removal and areas for retention. This mask may be used as the input mask for the journaling tool as well. The input mask is an optional attachment, but it **should be used** for any operations that operate on a specific region of the image. When first attached to the description, the mask is not shown in the description dialog. On subsequent edits, the image is both shown and able to be replaced with a new attachment.

Video input masks may be a single image applied to multiple frames (detected in the change mask), or a set of frames as a movie clip.



Figure 9: Example of an input mask one might use during seam carving to ensure preservation of the woman’s hat.

## Paste Sample

Paste Sample involves using a content aware tool to merge pixels from one area of an image to another area of the same image, using some sort of interpolation. It can be used semantically for healing, removing or cloning. The semantics are captured with a mandatory parameter ‘purpose’.

* Remove– sample parts of the the image serving background to effectively remove an object.
* Heal—Repair image after Paste Splice or other transformations that distort specific areas of an image (e.g. edge effects).
* Clone—Clone is a combination of add and remove. Do **not** use this operation as a replacement for Copy/Paste, where an object is copied to another part of an image. Copy/Paste are to be journalled with Paste Splice.

When possible, capture sample areas of the image as a separate exported image using the alpha transparency to mask out non-sampled areas. Supply this image file using the input mask parameter. Since this may not be possible in some circumstances (e.g. Heal), the parameter is optional.

Paste Sample should not be used for copy/paste of an object. The next section discusses the appropriate procedure for copy/paste.

## The PasteSplice Operation

Paste Splice is a special operation that expects a donor image. This is one of the few operations where an image node can have two incoming links. The first link is PasteSplice, recording the difference from the prior image (the mask is only the spliced in image). The second link is a donor image. The tool enforces that a second incoming link to a node is a donor link. The tool does not force donor links to exist. Instead, the tool reminds the user to form the link. There may be several steps to create a donor image from an initial image (e.g. blurring, cropping, etc.). Examples of this behavior can be found in many of the graph image examples throughout this document.

The Paste Splice operation often includes rotation, resize and cropping of the donor image or copied region. These operations require indication through ‘yes/no’ parameters if these actions took place. Furthermore, if the rotation amount degrees or the resize difference are known, these parameters should be provided.

Copy/Paste within the same image requires a PasteSplice. The image being altered to be both a donor and the recipient of the pasted pixels. The diagram below summarizes the basic approach.

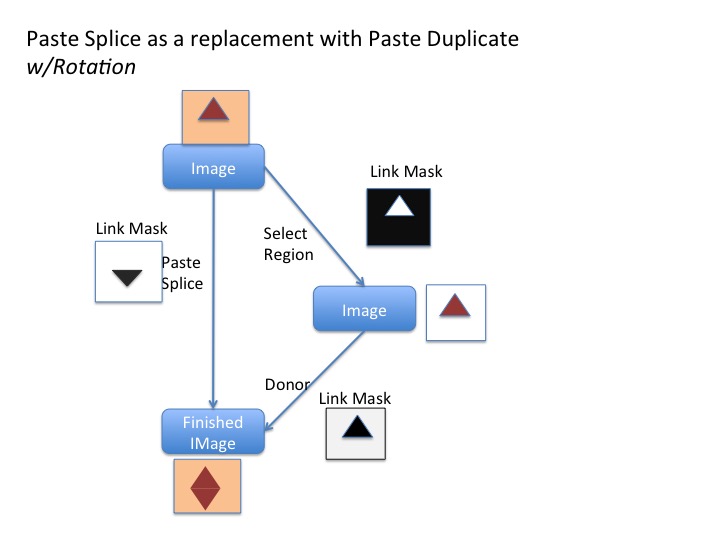


Figure 10: Paste Splice for Copy/Paste

## 

Figure 11: Rules or Paste Splice

## Shadow from Donor Image

It is important the journal captures where the shadow comes from copied from a donor image. The resulting mask created by a Paste operation (Paste Clone, Paste Splice) represents the newly placed shadow on the image in its final resting place, size, and orientation.  The journal should also reflect that the select region is adjusted (opacity).

Here is one approach:

The originating image node will have two links leaving it.

The first link is SelectRegion.  The link's destination image is the same dimension as the source image, with unselected region completely transparent.   The input mask for the operation is the selected region.   The link mask is the inverse (black on all unselected areas).  The image manipulator may alter the destination image further by applying Color.ColorOpacity, creating the appropriate link(s).

The second link from the originating image node represents a Paste Splice operation to a destination image node containing the shadowed object in its final resting place.  The link mask reflects the added shadow to the originating image.  The same destination node has an additional incoming ‘donor’ link from the last node on the path starting with the SelectRegion link.

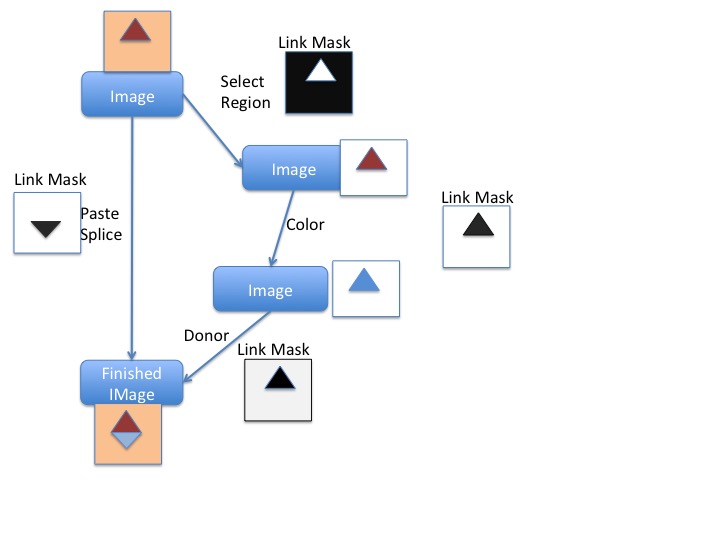


Figure 12: Shadow Paste

## Output PNG and Image Rotations

When exporting a JPEG image to PNG as the start of the manipulation process, the EXIF is stripped from the image. EXIF can have an Orientation attribute describing the orientation of the image. Some export software (e.g. Adobe Photoshop) automatically rotates the image in accordance to the Orientation during export, as the exported image needs to reflect the proper orientation--the guidance from EXIF is not longer present. Software may automatically rotate, prompt to rotate or not rotate on export. The manipulator must record whether the image was rotated during the OutputPNG operation when creating the OutputPNG link.

During a final ‘Create JPEG’, the operation adds the EXIF back into the final image, using the original JPEG as a Donor. Thus, the Orientation is re-applied to the image. The manipulator must decide if the image should be counter-rotated as part of this process.

In general, the manipulator should counter-rotate if the manipulation software rotated the software during the initial OutputPNG step or the manipulator rotated the image manually, journaled as a separate step with a TransformRotate.

If an Orientation (other than standard) exists in the EXIF, the JT will present the user with a dialog window showing the base JPEG image and the final image with the Orientation applied if counter-rotation is not performed.

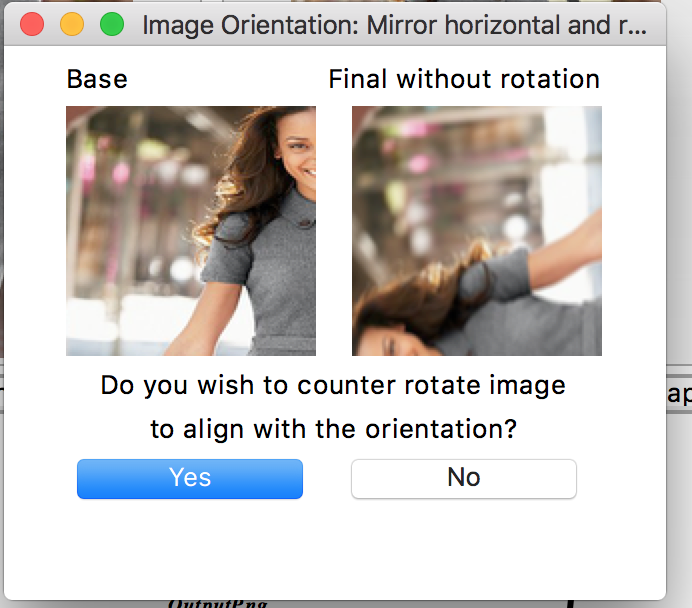


Figure 13: Image Orientation Prompt

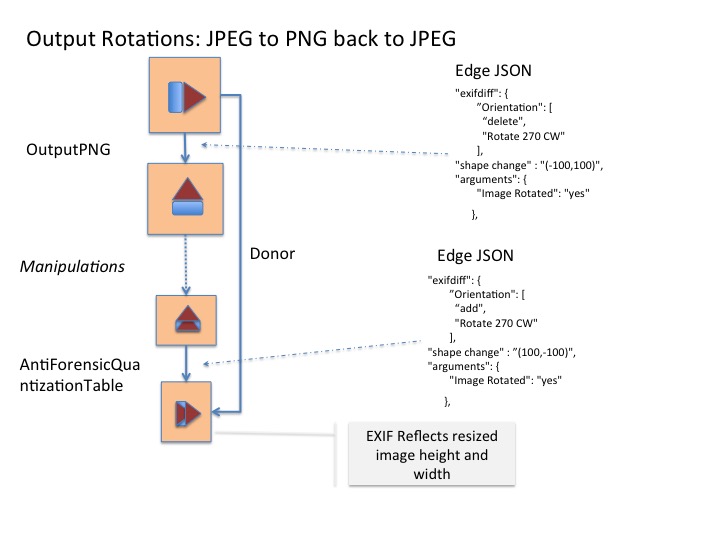


Figure 14: Summary of Orientation Workflow

## Applying Filters (Plugins)

A special kind of link description is available for plugins. When using a filter made available by a tool (Process -> Next w/Filter, etc), a dialog appears with different options. These filters are performed directly, and as such the new link and node are automatically generated. The user must supply the plugin name and parameters (if applicable). The Category, Operation, Software Name, and Software Version are all included in the plugin. For more information on writing plugins, see the corresponding section of this document.



Figure 15: Window for applying plugins to image nodes.

## Link Inspection

Regardless if the link is read-only (for links created by plugins) or created through one of the image connection operations, a link may be selected and inspected. The inspection includes all parameters, an input mask if provided, and an EXIF comparison analysis, as shown in the figures below.

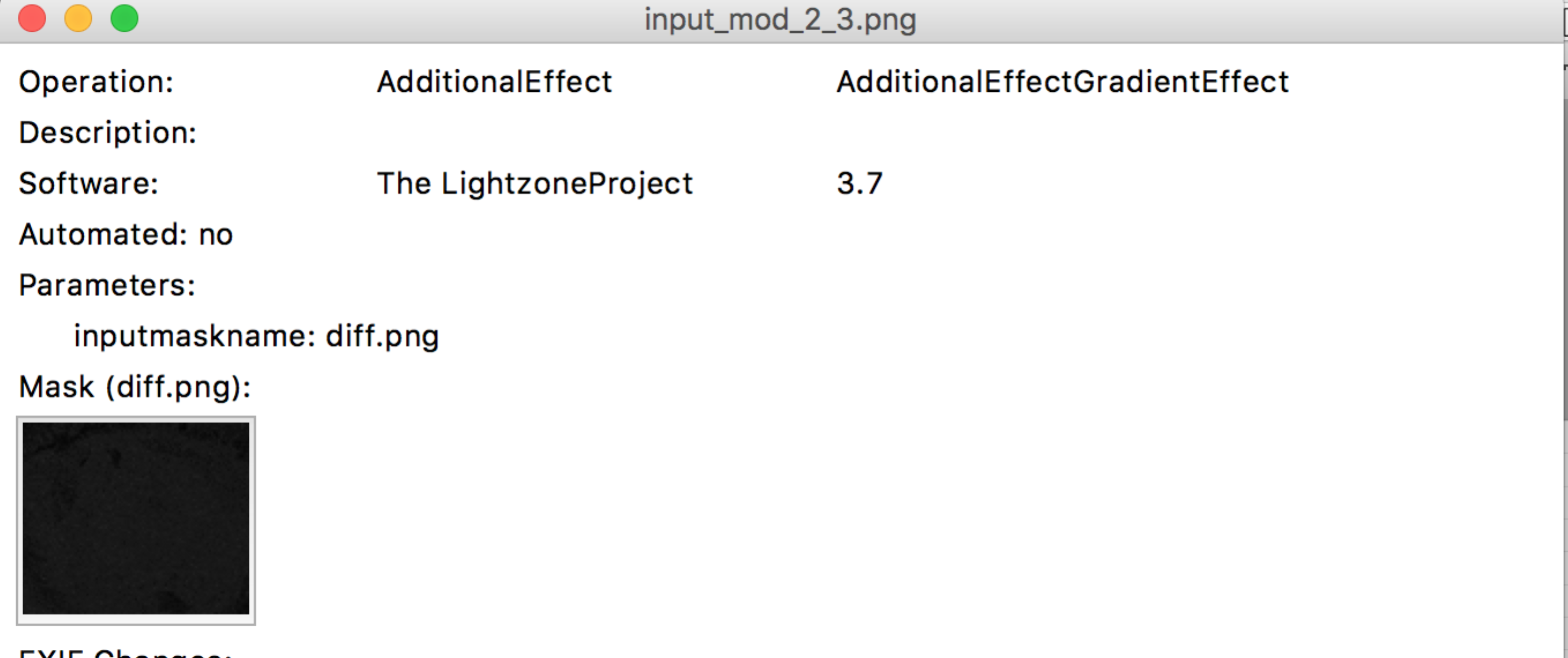


Figure 16: Link Inspection



Figure 17: EXIF Comparison Table

## Video Link Descriptions

Video Links contain changes across the entire video stream. Changes are organized by steam. A selection drop down controls the section of data viewed in the subsequent meta-data change table. Global meta-data changes represent changes for the entire video file.

Meta data changes include property value changes, addition of frames and removal of frames. Frames are sequence and compared by presentation time stamp.

The tool displays additional sequence of frames organized by the presentation time stamp. The ‘New’ value is N:=>T where N is the number of frames added and T is the presentation time stamp of the last added frame.

The tool displays removed sequence of frames organized by the presentation time stamp. The ‘Old’ value is N:=>T where N is the number of frames removed and T is the presentation time stamp of the last removed frame.

The tool displays presentation timestamps in fractional seconds (e.g. 13.480133).

Video masks are visible as a set of clips. The video masks are inverted: white indicates change. The mask name includes the presentation time stamp of the first frame the mask applies, shared between both streams. When adding or removing frames, the mask reflects those added and removed sequences.

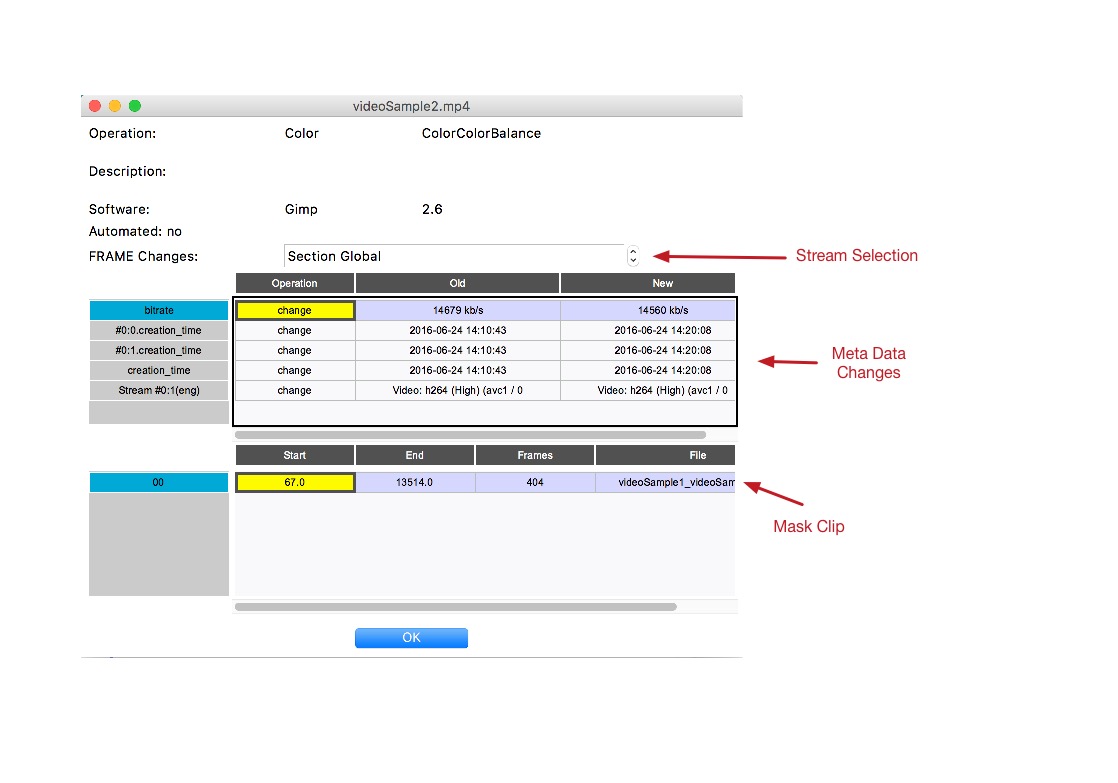


Figure 18: Video Link Inspection

Masks are stored in HDF5 format with the journal. Mask clips may be viewed with system default movie player by selecting ‘Open’ using the table row’s right mouse-button enabled menu. Masks are converted on demand to a playable format. The tool chooses m4v with the AVC1 codec. This default can be changed by editing .maskgen2 JSON preferences file in the user’s home directory as shown in the next example.

"vid\_suffix":"mv4",

"vid\_codec":"XVID"

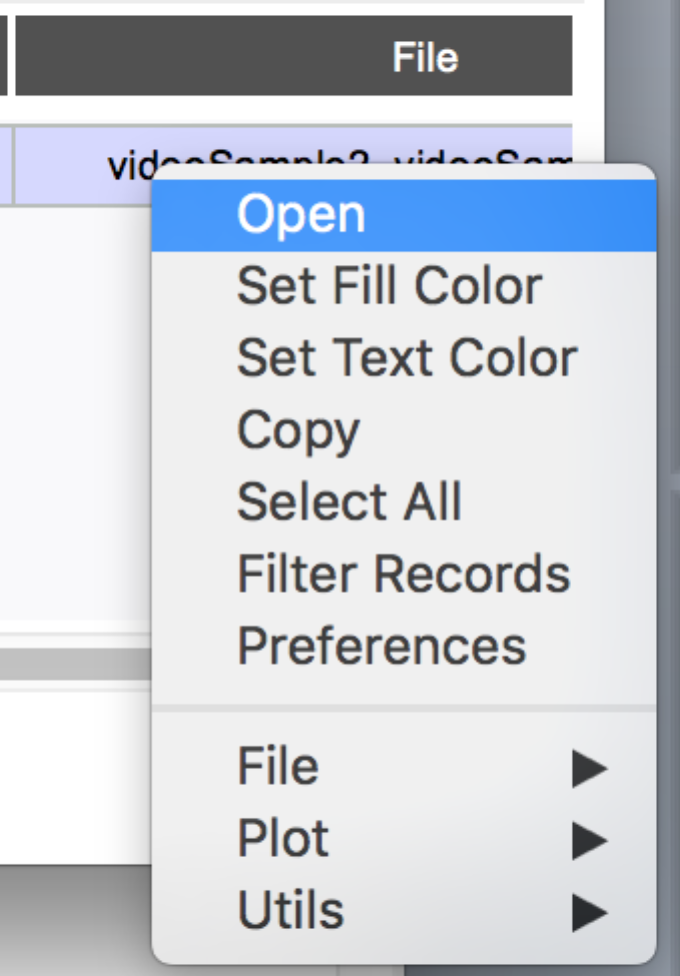


Figure 19: Video Mask Table Menu

## Other Metadata

Not shown in the UI, the project JSON file also contains the operating system used to run the manipulation and the upload time for each image. See the dedicated JSON section of this document.

# Mask Generation

It most cases, mask generation is a comparison between before and after manipulations of an image. Full image operations like equalization, blur, color enhance, and anti-aliasing often effect all pixels resulting in a full image mask. Since these operations may only effect some pixels (e.g. anti-aliasing), the mask does represent the scope of change.

The mask generation algorithm gives special treatment to manipulations that alter the image size. The mask is the same size as the source node artifact. The algorithm finds most common pixels in the smaller image to match the larger. This is useful in cropping OR framing. When cropping in image, the mask should include the frame around the cropped image. When expanding image, the mask represents the comparison of the most closely related area. If the expansion is due adding a frame, rather than some interpolation, then the mask will not reflect any change since the original pixels have not changed.

The mask generation algorithm also is sensitive to rotations, generating mask reflecting the image after rotation. A pure rotation with interpolation should have an empty change ask. Rotations are counter-clockwise in degrees. Consider a 90-degree rotation: interpolation is not needed and the mask is empty. When rotating 45 degrees, the size of the resulting image must increase to accommodate entire image. Since the mask size is the size of the initial image, the mask only indicates some distortion that occurred during the rotation. The mask is created by first reversing the rotation back to the original image orientation and size.

## Video Masks

Video masks are organized by video clips per section of the video affected by the change. The masks are labeled with the start time in the source video where the change is detected. There may be more than one video clip.

## Composite Mask

The tool support creation of an aggregate summary mask, composed of masks from a leaf manipulated node to a base node. By default, all masks that involve marking or pasting specific regions of the node are included in the composite mask. Those links are colored blue and the link operation name is appended with an asterisk. The status of the link can changed with the Composite Mask menu option. Furthermore, the mask used for the composite can override the link mask, as a substitute.

Obviously, the image manipulator should not include all masks in the project, otherwise the composite would likely be completely black, representing a complete change in pixels. The image manipulator should therefore ensure the composite mask accurately reflects ALL LOCALIZED changes, and only localized changes, to a manipulated node from the base node. For example, a global saturation adjustment should not be included in the mask, while a Paste Sampled should.

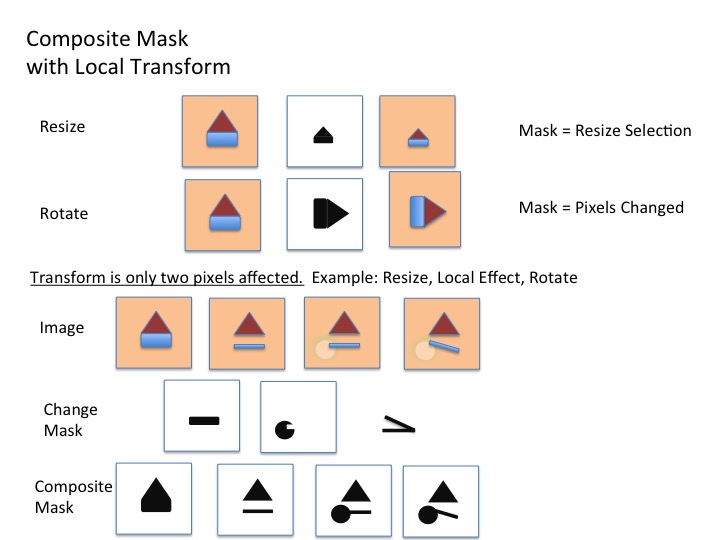


Figure 20: Composite Mask

Composite masks are composed of RGB colored pixels, each color is associated with a single link. When a pixel is modified by more than one operation, the color of the last operation is applied to that pixel.

# Analytics

During mask generation, analytics are processed on the images and shown just beneath the images when a link is highlighted. The purpose of these analytics is primarily to show how much a given manipulation changed the source image. These analytics include:

1. [Structural Similarity](https://en.wikipedia.org/wiki/Structural_similarity)
2. [Peak Signal to Noise Ratio](https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio)

NOTE: Structural Similarity produces a warning on the tool command line output that can be safely ignored. There is a bug within the scikit-image package where the compare\_ssim function calls a known deprecated function for multichannel images. Furthermore, the deprecation warning module reinstates the warning filter prior to issuing the warning, thus overriding warning suppression.

# Plugins

Plugin filters are python scripts. They are located under a plugins directory. Each plugin is a directory with a file \_\_**init\_\_**.py. The **init** module must provide three functions:

1. 'operation()' that returns a list of five items. The operation name and category must match operation name and category in the master operations JSON file.
2. 'operation name'
3. 'operation category'
4. 'description'
5. 'python package'
6. 'package version'
7. 'transform(im, source, target, \*\*kwargs)' consumes a maskgen.image\_wrap.ImageWrapper. The image may be converted to a numpy array (numpy.asarray()), or a PIL Image (ImageWrapper.toPIL()). The ImageWrapper is designed to support 16 bit multi-channel images, not supported by PIL.

The source file and target (result) file names are provided. The tool creates a temporary target file to be used by the plugin.

The plugin must augment the source image and save the result, overwriting the contents of the target file. When the plugin is complete, the tool moves the temporary file to a permanent location in the project.

A dictionary of arguments is provided to the transform method, including mandatory arguments captured by the tool. The function returns None or additional arguments it used (in a dictionary).

1. ‘arguments()’ returns a list of tuples or None. Each tuple contains an argument name a default value and a description. These arguments represent arguments that may be in addition to those normally collected by the specific operation, as defined in the operations JSON file.

The python package and package version are automatically added to the list of software used by the manipulator.

Plugins may provide a fourth function called 'suffix()'. The function returns the file suffix of the image file the plugin expects (e.g. .tiff, .jpg). The expectation is that the plugin overwrites the contents of the file with data corresponding the suffix.

The tool creates a copy of the source image in a new file. The path (i.e. location) of the new file is provided in the target argument. The transform changes the select contents of that image file. The image provided in the first argument the transform is a convenience, providing a copy of the image from the file. The image is disconnected from the file, residing in memory.

The transform can optionally return a dictionary of additional arguments to record in the journal.

Argument values are collected as parameters from the invoking tool.

## Arguments

There are two special argument names: 'donor' and 'inputmaskname'.

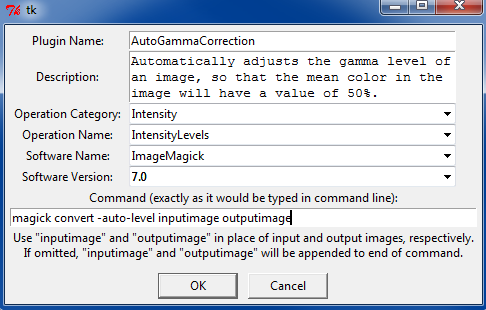
The system will prompt a user for an image node to fulfill the obligation of the donor. The transform function will be called with the user selected image (e.g. donor=image). Upon completion, separate Donor link is made between the donor image node and the image node created from the output of the transform operation.

The system prompts for an image file to fulfill the obligation of the inputmaskname. The path name is provided the transform function (e.g. inputmaskname='/somepath'). The tool does not load the image in this case. The image will be preserved within the project as the inputmask of the link, which references the image, upon completion of the operation.

All other parameters collected by the user will be provided as strings to the transform function.

## Plugin Builder

The Plugin Builder is an alternate way of creating simple plugins for the journaling tool. The image operation must be able to be run from the command line. To access the Builder from the journaling tool, go to **File -> Build Plugin…** and you will see a blank version of the interface below.



magick convert -auto-gamma {inputimage} {outputimage}

Figure 21: Plugin Builder Interface, filled out with an “AutoGammaCorrection” plugin

With this tool, the user can create a plugin from any command line tool. “{inputimage}” and “{outputimage}” are required keywords, as they will be swapped out when the plugin is run. This type of “custom plugin” is saved via JSON with the following format:

{

"name": "Gamma Correction",

"operation": {

"name": "IntensityNormalization",

"category": "Intensity",

"description": "Automatically adjusts the gamma level of an image, so that the mean color in the image will have a value of 50%.”,

"softwarename": "ImageMagick",

"softwareversion": "7.0.3"

},

"command": ["magick",

"convert",

"-auto-gamma",

"{inputimage}",

"{outputimage}"

]

}

With this template, new plugins can easily be written, even without the use of the plugin builder. Modifications and new arguments can also be added to the “command” list of the JSON file.

# EXPORT

Export performs the following steps:

* Validates the project. The user is prompted to continue if errors are found.
* Computes composite images for all final nodes.
* Performs final analysis for final image nodes using rules within the project\_properties.json. The results are placed in the ‘pathanalysis’ attribute of the final image nodes.
* Strips out unused images and videos.
* Saves the project.
* Creates a gzipped tar file of the used contents of the project along with the project JSON file.

# Group Manager

The Group Manager allows the user to create, remove and manage groups. Groups are sets of plugin image transforms. Only those transforms which do not require arguments are permitted within the group at this time.

# Validation

The journaling tool contains several built-in checks to help ensure a project is complete and without errors. This functionality can be accessed via File-Validate. Below is a short list of the items it will check for:

* Nodes without any links attached
* Operations which seem to change image size that should not.
* Operations which seem like they should change image size but do not.
* Seam carving operations which alter more than one dimension of an image
* Paste operations without an associated donor image

# QA Process

Once a manipulation has been journaled with the tool, it should be reviewed by a peer for completeness and accuracy. For simplicity, here is an example QA workflow to follow:

* 1. Open the journaling tool to the project directory
  2. Run File -> Validate to automatically search for any obvious errors
  3. Make sure the start and end images are the same format (jpg, tiff, etc.)
  4. Make sure all manipulations are isolated. The only manipulations that are ‘combined’ involve fitting a pasted image into a image.
  5. If JPEG was the base node, the graph should end with an additional link attached to each final node.: AntiForensicExifQuantizationTable. This can be done in one operation with Process -> Create JPEG/TIFF.
  6. Right click and select “View Composite” on the final node. Make sure all relevant local changes are represented (particularly copy/paste-type operations)
  7. Where possible, all input masks should be provided to operations that request the mask as a parameter.
  8. Make sure all images in the project directory correspond to a link (mask images, input masks), or a node. Extra images from errors should be deleted.
  9. Send the project back to the original manipulator with any feedback.

# Batch operation

The journaling tool currently supports a rudimentary batch processing feature. This is designed to operate on large quantities of images with the same type of simple manipulation. For example, 100 images are manipulated to have varying levels of saturation. These images can be specified with the tool’s batch feature, and will automatically generate the project, including the mask image and graph.

The typical flow for batch operations includes:

1. Project creation. The tool is executed with a source directory of un-manipulated images, an empty project directory, and either a plugin operation or a target directory containing the first set of manipulated images (*image names aligned to source images*).
2. Project Extension. The tool is executed with a source directory of a set of manipulated images, all manipulated by the same operation applied to the images last added to the projects contained in a project folder. The tool may also apply an operation to the set of images last added to the project through a plugin. Project Extension may occur over many iterations.
3. Project Capping. Adding a final set of Antiforensics operations to the images last added to the projects.

At its core, the batch tool requires only 1 directory, a directory of project directories. It can be run with the following:

python –m maskgen.batch.batch\_process <args>

Mandatory arguments:

--projects <DIR>: directory of project directories.

At least one of these three arguments must be present (see below for an explanation):

--sourceDir <DIR>: directory of images

--plugin <pluginName>: plugin to perform

--jpg: Copies quantization tables and exif data from base image to save new jpeg image.

These arguments may be used only if using sourceDir:

--op <operation>: operation performed (required)

--softwareName <name>: manipulation software used (required)

--softwareVersion <version>: manipulation software version (required)

--endDir <DIR>: directory of manipulated images (optional)

--inputMaskPath <DIR>: directory containing input masks (optional)

--description <"descr">: description of manipulation performed (optional)

--additional <name1 value1 name2 value2...>: additional operation arguments, such as rotation angle (optional)

These arguments are required when first creating a project:

--projectDescription <quoted description>

--technicalSummary <quoted technical summary with key words from assignment>

--username <name> Only required if the operating system user name does not match your assignment ID

Optional arguments:

--continueWithWarning: use this tag to ignore warnings that check for valid operations, software, etc.

--s3 <bucket/path>: if included, will automatically upload projects to specified S3 bucket after performing operation

## Different arguments will trigger different functionality.

## Creating a Project

Using both --sourceDir and --endDir will create new projects, using the images in sourceDir as base, and link them with the specified operation. This will also create the project directories and JSON files if necessary.

When creating a project, a technical summary is required for the project. The user name is only required if the operating system user ID is not the identifier to be associated with the project. This case may occur when using a shared machine.

python –m maskgen.batch.batch\_process --projects <DIR> --sourceDir <DIR> --endDir <DIR> --op ColorColorBalance --softwareName GIMP --softwareVersion 2.8

Using --sourceDir without --endDir will add the images in the source directory to the current project and link them to the most recent node with the specified operation.

Projects may also be created when first applying a plugin to a set of image nodes. Use --plugin to specify a plugin to perform on the most recent image node.

python –m maskgen.batch.batch\_process --projects <DIR> --plugin ColorEqHist

## Extending a Project

Projects are extended by providing a source directory containing manipulated images and an operation applied to those images.

In absence of a source directory, a plugin can be applied to all leaf node images in all projects within the projects directory.

## Capping a Project: Antiforensics

Antiforensic techniques are often applied as a last step to project creation. The tool provides a single two-step JPEG creation process. Using the --jpg option, the tool performs antiforensic jpeg export

and exif copy on existing projects. This option can also be added to any other operation.

python –m maskgen.batch.batch\_process --projects <DIR> --jpg

## Image File Names

All images that are to be placed in the same project should have the same basename. Manipulated images should be appended with an underscore followed by some text and a number (i.e. image.jpg, image\_01.jpg).

For example:

|  |  |
| --- | --- |
| sourceDir | endDir |
| imageA.jpg | imageA\_01.png |
| imageB.jpg | imageB\_01.png |
| imageC.jpg | imageC\_01.png |

It is recommended the user view generated graphs by opening the projects in MaskGenUI once the processing is complete to verify.

# JSON

The JSON is made up of two key parts: Nodes and Links. The structure of the JSON document is as follows. The name of the project is the graph name. The graph structure with the JSON document contains other project meta-data include file type preferences, software version, and id counter used to generate unique file names.

{

"directed": true,

"graph": {

"igversion": "0.1",

"idcount": 21,

"name": "sample",

"typespref": []

},

"nodes": [],

“links”: [],

"multigraph": false

}

The graph mapping includes project properties including categorical information as described in project\_properties.json, for those properties where ‘node’ is false.

Each node with the nodes list is a structured describing an image node within the project.

{

"xpos": 619,

“file": "cropTest\_1.png",

"ypos": 33,

"seriesname": "cropTest\_1",

"ownership": "yes",

"id": "cropTest\_1",

"ctime": "2016-07-13 17:05:50"

}

Node properties include:

* ***xpos and ypos*** - describe the location of the node in the tool graph viewer.
* ***file*** - the name of the image file within the project directory
* ***seriesname*** - describes a path from base image to one or more manipulated images
* ***id*** - an image name minus the file type suffix
* ***ctime*** - the creation time of the image file within the project
* ***ownership*** - “yes” if the image file was created by a tool operation or copied into the project from another location
* *donormaskname*- The file name of the last computed donor mask corrected for transformations from the unaltered donor image. The donor mask is attached the node modified by a Paste Splice.
* *compositemaskname* – The file name of the last computed composite mask attached to nodes of final images.
* "*composite change size category":* "small", "medium" or "large" depending on the number of pixels represented in the mask.
* *pathanalysis* – a structure containing values associated with property rules. The property rules are defined in project\_properties.json, labeled with ‘node’:true. These properties summarize all the activities leading up to the final image node. An example image structure is shown below:

"pathanalysis": {

"healinglocal": "no",

"natural": "no",

"people": "no",

"color": "no",

"otherenhancements": "no",

"contrastenhancement": "no",

"manmade": "no",

"blurlocal": "no",

"remove": "no",

"face": "no",

"othersubjects": "no",

"compositepixelsize": "small",

"largemanmade": "no",

"histogramnormalizationglobal": "no",

"clone": "no",

"landscape": "no",

"imagecompression": "no"

}

A link is a connection between source and target nodes. The nodes are referenced by a number in accordance to the order of nodes list from 1 to N (N being the total number of nodes).

{

"username": "ericrobertson",

"maskname": "cropTest\_cropTest\_1\_mask.png",

"psnr": 29.379891227475664,

"description": "\n",

“shape change": "(0,0)",

"source": 16,

"editable": "yes",

"ssim": 0.9918523088886719,

"softwareName": "OpenCV",

"inputmaskownership": "no",

"softwareVersion": "2.4.13",

"opsys": "Darwin 15.5.0 Darwin Kernel Version 15.5.0: Tue Apr 19 18:36:36 PDT 2016; root:xnu-3248.50.21~8/RELEASE\_X86\_64",

"inputmaskname": "",

"target": 0,

"op": "FilterBlurMotion",

"argments": {“x”:1},

“exifdiff": {…}

}

Link properties include:

* ***source*** - identifies the source node in the order it appears in the node list
* ***target*** - identifies the target node in the order it appears in the node list
* ***maskname*** - the assigned mask file. It is usually composed with the source and target image node names.
* ***description*** - a user provided description of the operation performed on the source node to create the target. For plugin operations, the description is provided by the plugin.
* ***editable*** - ‘yes’ if the link was not generated by a plugin or internal tool operation
* ***username*** - the name of the user that created the link
* ***automated –***‘yes’ if the link is created through a batch or automated process
* ***opsys*** - the operating system used to run the operation that generated the target image from the source image
* ***op*** - the standard operation name describing the operation used to generate the target image from the source image
* ***softwareName*** - the software that performed the operation to create the target image from the source image. The plugin provides a describing the software library used
* ***softwareVersion*** - the version of software that performed the operation to create the target image from the source image
* ***arguments*** – the set of argument captured and used by a plugin
* ***inputmaskname*** - an optional parameter containing the name of an input image file used by the software as a parameter to the operation to create the target image from the source image. For example, a seam carving algorithm may use an input file masking regions to keep and discard from the source image.
* ***Inputmaskownership*** - ‘yes’ if the tool copied the inputmaskname into the project folder
* ***selectmaskname*** - an optional parameter containing a image used in the composite creation, overriding the change mask.
* ***selectmaskownership*** - ‘yes’ if the tool copied the selectmaskname into the project folder
* ***recordCompositeInMask*** –‘yes’ if the mask for this link should be included in the composite mask for a successor node.
* ***masks count*** – the number of video masks
* ***videomasks*** – structures describing the masks for each video.
* ***metadatadiff*** – video meta data comparisons. The structure is an array. The first element is the global metadata. All other elements are structures labeled with the stream identifier (e.g. 0,1,2 etc.)
* ***psnr*** - a measure to signal to noise ratio
* ***shape change***- a measure in both x and y dimensions the change in shape from the source image.
* ***ssim*** - the structure similarity between source and target images. The range is -1 to 1. -1 indicates opposite similarity, 1 indicates exactly the same and 0 indicates completely dissimilar.
* ***exifdiff –*** a structure that defines changes to EXIF tags. Each key is the tag name. Each value is a list of one of the following
* [‘change’, old value, new value]
* [‘add’, new value]
* [‘delete’, old value]
* ***compositecolor*** – The assigned RGB color of the link when assigned to affected pixels in the composite mask.
* ***transform matrix*** – A description of a 3x3 transformation matrix used to realign images to their original after a transformation, applied masks during construction of the composite mask.

***A Note about Donors:*** *Donor images provide data to be placed in a source image. Currently there is only one operation that expects a Donor image: Paste/Splice.*

***A Note about Image Names:*** *The manipulator is responsible for using image names as they coincide with image databases. The tool, when copying an image into project, does not change the image base name. It may add a postfix to the name into ensure uniqueness in the project.*

# Project properties

Project Properties is a JSON file that describes properties captured by the user at the project level. The JSON file also describes properties assigned to each final image done during export.

A project property is defined for a final image node if the ‘node’ property attribute is true. Properties may automatically be determined based on rules. The rules are setup in three ways:

1. Existence of an edge with a specified operation
2. Existence of an edge with a specified operation and argument with a specific valued.
3. A general rule (python function)

Project level properties inspect all edges. Final image node properties inspect those edges from the final node to the base node, ignoring paths of edges starting with a donor.

## Example Edge with Specific Operation

{  
 "description": "Image Compression",  
 "name": "imagecompression",  
 "type": "yesno",  
 "operation" : "AntiForensicExifQuantizationTable",  
 "node" : true  
},

## Example Edge with Specific Operation and Argument Value

{  
 "description": "Other Subjects",  
 "name": "othersubjects",  
 "type":"yesno",  
 "operation" :"PasteSplice",  
 "parameter":"subject",  
 "value":"other",  
 "node" : true  
},

## Example with a Rule

{  
 "description": "ColorEnhancement",  
 "name": "color",  
 "type": "yesno",  
 "\_comment": " any color category operation",  
 "rule": "colorGlobalRule",  
 "node" : true  
},

## Example Project Property without a Rule

{  
 "description": "Manipulation Category",  
 "name": "manipulationcategory",  
 "type": "list",  
 "values": [  
 "Provenance",  
 "2-Unit",  
 "4-Unit",  
 "6-Unit"  
 ]  
},