MaskGen
Journaling Tool

15 August 2018

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	
	11/3/16	Updated information regarding Create JPEG/TIFF group operation. Formatting	
	11/29/16	Updated QA and added Custom Plugins section to reflect new features	
	12/01/16	GraphViz visualize	
	12/30/16	Skip Processing	
	02/08/17	Installation and Execution Changes. Search for resources and plugins	
	03/08/17	Flushed out the definition of operation, made some critical edits and defined how to plugin image loaders and transformation rules	
	08/20/17	Batch Process. Remove composites.	

Table of Contents

List of Figures

1) OVERVIEW

The journaling tool can be used to keep track of media 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*.

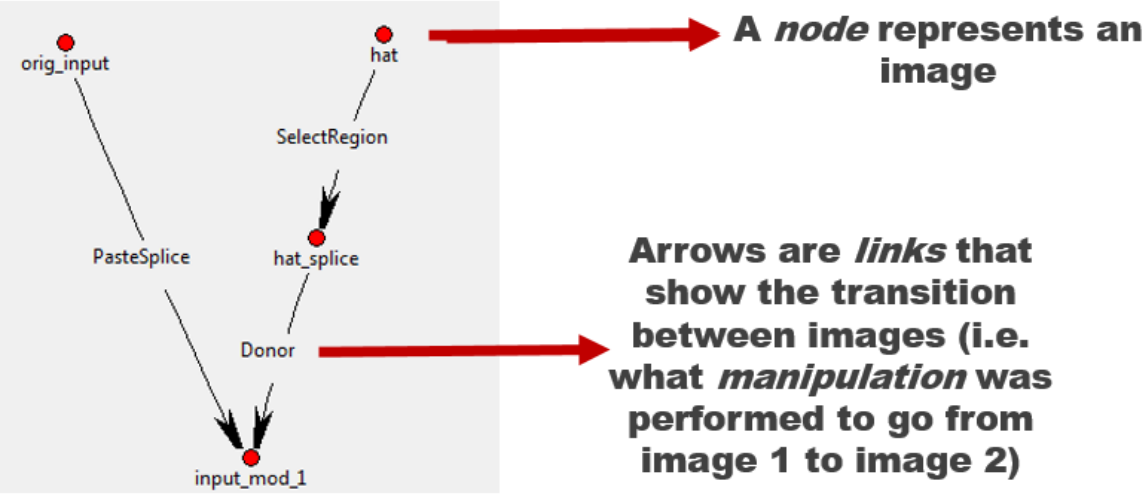


Figure 1: Example of a graph produced by the journaling tool. Specifically, a region of one image (hat) was spliced into another image (orig_input) to create a new image (input_mod_1).

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. One can define a group operation that documents co-operations, provided only one the grouped operations requires a change mask.

Please follow the quality assurance guidelines outlined in this document (13) QA Process).

2) 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.

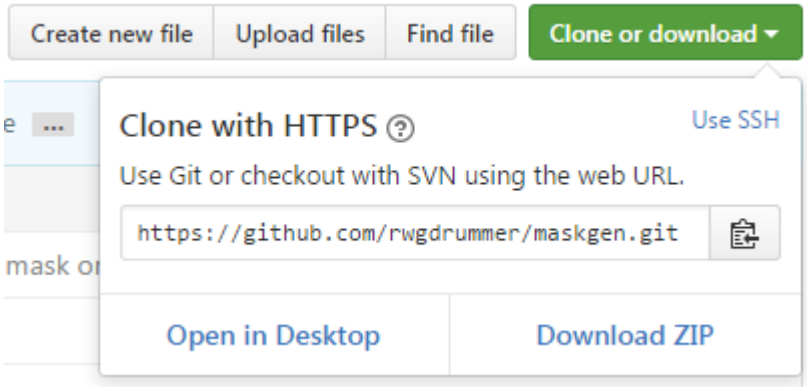


Figure 2: GitHub website download dialog

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.

2.1 Dependencies

There are several Python libraries required for operation listed below. Most of those dependencies are installed AUTOMATICALLY during the installation process. Read the steps carefully.

- Tkinter
- PIL/pillow
- numpy
- matplotlib
- opencv (cv2)
- networkx
- moviepy
- scikit-image
- tkintertable
- bitstring
- boto3 (for optional use of S3)
- hdf5
- rawpy
- graphviz
- pydot
- cachetools
- awscli

On windows, install graphviz and opencv before completing these steps. See 2.1.1. Most of these are installed automatically with the following procedure.

```
pip install setuptools
cd setuptools-version
python setup.py install
cd ..
python setup.py sdist
pip install -e .
```

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
conda install Pillow
```

The tool depends on the installation of 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 requires the installation of TCL and TK. Most Mac OS have TCL installed. The tool is tested with TCL 8.5.

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.

2.1.1 Installation scripts

Installation scripts are available the 'scripts' subdirectory.

Windows installer can be constructed following the guidelines in the windows readme installer text file. Installer construction requires a download of all the dependent programs (exiftool, ffmpeg, opencv, graphviz) prior to using the NSH installer builder. The result of the build process is an executable installer containing all dependencies.

2.1.2 Logging and Mac's OS

The default python on Mac contains an old version of logging and an old version of pip. The indication of a problem is an error message on starting the JT, indicating a unknown symbol 'utc'. The first step is to try to uninstall logging. This may leave an older version of pip in an unstable state, indicated by an error message trying to resolve a handler List. Here are the steps used to clean this problem.

```
pip uninstall logging
brew upgrade python
rm –r /usr/local/lib/python2.7/site-packages/logging
rm –r /usr/local/lib/python2.7/site-packages/logging-0.4.9.6.dist-info/
sudo easy_install pip
```

2.1.3 Graphviz

Mac:

```
brew install graphviz
pip install pygraphviz
```

See http://www.graphviz.org/Download_macos.php for other options.

Windows:

1. Download the current stable release from http://www.graphviz.org/Download_windows.php. Get the .msi, not the .zip.
2. Run the graphviz msi installer, and walk through the steps to install.
3. Add the graphviz "bin" directory to PATH variable. Most likely will be C:\Program Files (x86)\Graphviz2.38\bin
4. Restart computer to complete the install
5. Pull down the correct wheel fro http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygraphviz and perform:

```
pip install pygraphviz-1.3.1-cp27-none-win_amd64.whl
```

2.1.4 Rawpy installation issues

(1) pip install rawpy (may still give the same error)

(2) Pick and install the appropriate WHL from https://pypi.python.org/pypi/rawpy

(3) Install git and rerun the setup.

2.1.5 Anaconda, FFMPEG and OpenCV[1] on Windows

Some Python distributions come with some or all of these pre-installed, such as Anaconda. OpenCV is rarely included in such distributions and has a slightly more involved installation. Helpful instructions can be found in the OpenCV documentation

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 3.2.2.
- 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__*

2.1.6 PDF

To load PDF files:

```
pip install PyPDF2
```

2.1.7 HDF5

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

For Mac OS:

[1] http://opencv.org/

```
brew install homebrew/science/hdf5
```

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

2.1.8 FFMPEG 3.2.2 for Python 2.7 (not Anaconda)

Video processing requires ffmpeg 3.2.x to be installed and accessible via the PATH environment variable.

For Mac:

```
brew install ffmpeg --with-fdk-aac --with-sdl2 --with-freetype --with-libass --
with-libvorbis --with-libvpx --with-opus --with-x265 --with-xvid --with-openh264
```

2.2 Resources

The tool uses three key resource files:

- software.csv lists the permitted software and versions to select. This enables consistent naming.
- operations.json provides the description of all journaled operations and require parameters, along with defining validation rules, analysis requirements, and parameters.
- project_properties.json defines all final media node and project properties. Final media node properties are summarizations of activities that contributed to a final media node of a project.

If the user names should be constrained to a set list, then add a resources file called ManipulatorCodeNames.txt to the resource directory. The file contains newline separated list of usernames.

Resource files are stored in one of the following locations, searched in the order given:

- Directory as indicated by the MASKGEN_RESOURCES environment variable
- current working directory
- *resources* subdirectory
- The resource installation as determined by Python's sys path.

These files may downloaded from an S3 bucket, the files are placed the MASKGEN_RESOURCES directory, if set, or the current working directory.

Using S3 assumes the client is configured using *aws configure* (after installing awscli).

The files can be pulled from S3 within tool, or with the command line option –s3:

```
jtui --s3 bucket/path
```

3) GENERAL OPERATION

3.1 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:

```
jtui
```

Window users may consider using jtuiw instead, which does not require a console and redirects error messages to a log file in %APPDATA%.

```
jtuiw
```

Both commands can open a journal directory by supplying the optional project parameter directing the tool to open a journal maintained within the provided directory.

```
jtui --project folder
```

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

```
jtui --project images --base images/baseimage.jpg
```

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

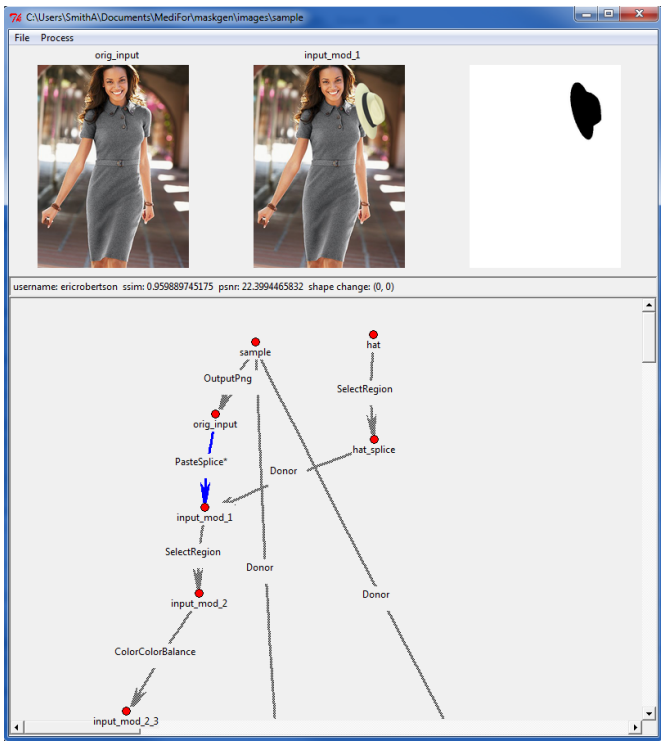


Figure 3: Basic UI. The media nodes open here are from the example project based in the project directory specified in the command line.

3.2 Projects

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

Projects are given a type based on the initial media node. The type is image, audio or video.

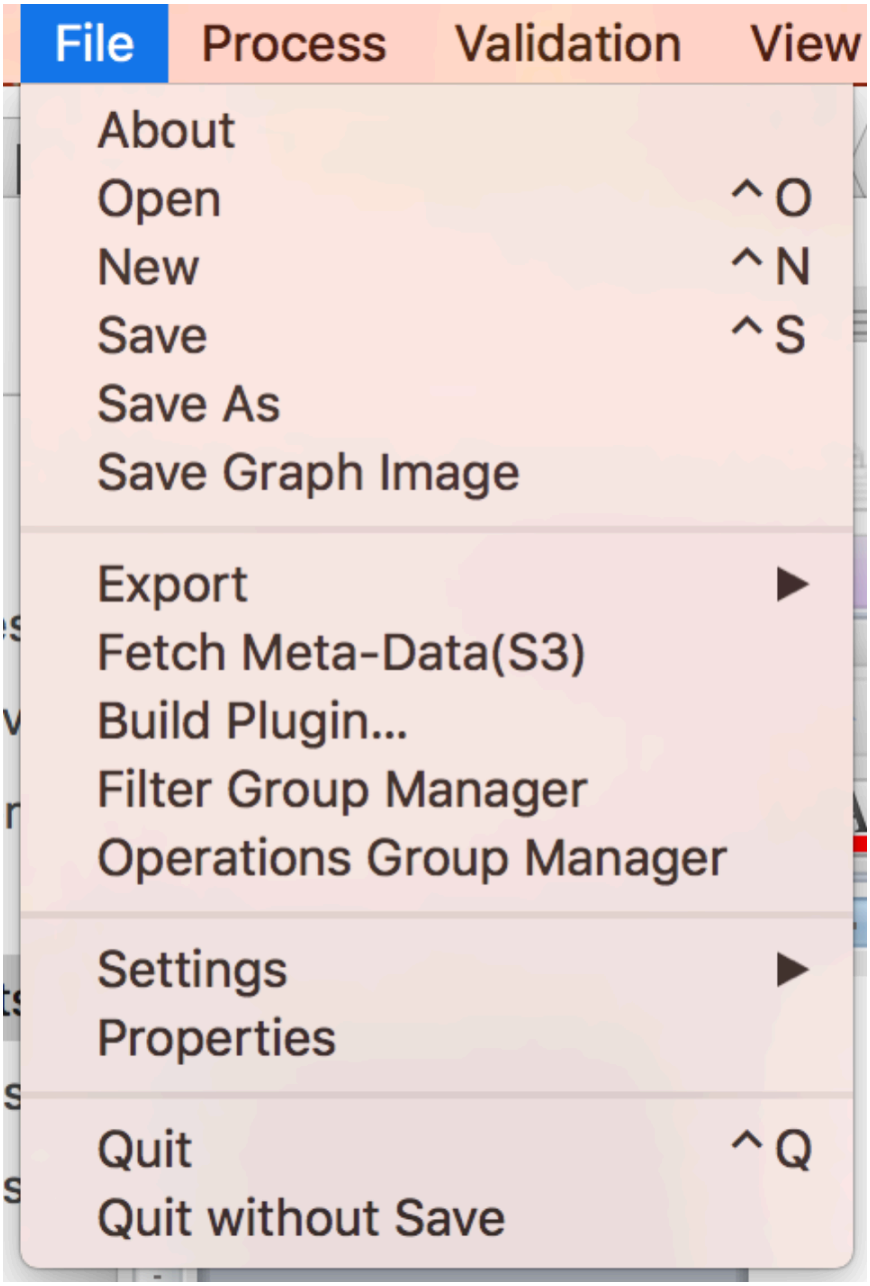


Figure 4: File menu.

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 media file. The directory containing that media file becomes the project directory. The name of the project is based on the name of the media file, sans the file type suffix. All media 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.

Save Graph Image: Save a graph depiction of the project with thumbnails in a PNG file (using graphviz/dot)

Export – To File: will create a compressed archive file of the project, including all media and masks. Export also validates the project, running the graph rules, builds composite masks and calculates summary information for final media nodes as described by rules in project_properties.json (those properties with 'node' : true). Export creates a 'dot' graphic called _overview_.png in the archive.

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 media nodes as described by rules in *project_properties.json* (those properties with 'node' : true). Export creates a 'dot' graphic called _overview_.png in the archive.

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 '/'. The files are pulled down to the current working directory or the directory as indicated in the environment variable MASKGEN_RESOURCES, if set.

Filter Group Manager: Opens a separate dialog to manage groups of plugin filters. Filters can be run in a sequence, where the output of one feeds another, or as a spread, sharing the same input node.

Operations Group Manager: Opens a separate dialog to manage groups of operations. Grouping operations treats the group as one atomic operation (on one link). This is only possible if the individual operations do not require their own change mask.

Rename to Base: Rename a project to match one of the base media.

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 media.

Settings - AutoSave: Set the number of seconds to automatically save the project.

Settings – Skip Link Compare: Allows the user skip link comparison (mask generation) until validation or export time. Link comparison may be time consuming for video journals. See batch options (**Error! Reference source not found. Error! Reference source not found.**) to perform link comparisons on projects in batch.

Settings –Trello API Key: Notfications of journal export may be sent directly to a Trello card. The trello list is the user name. The card is the project name.

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.

3.3 Processing

Media can be added to the current project via the 'Process' menu, on the toolbar, as depicted in Figure 5.

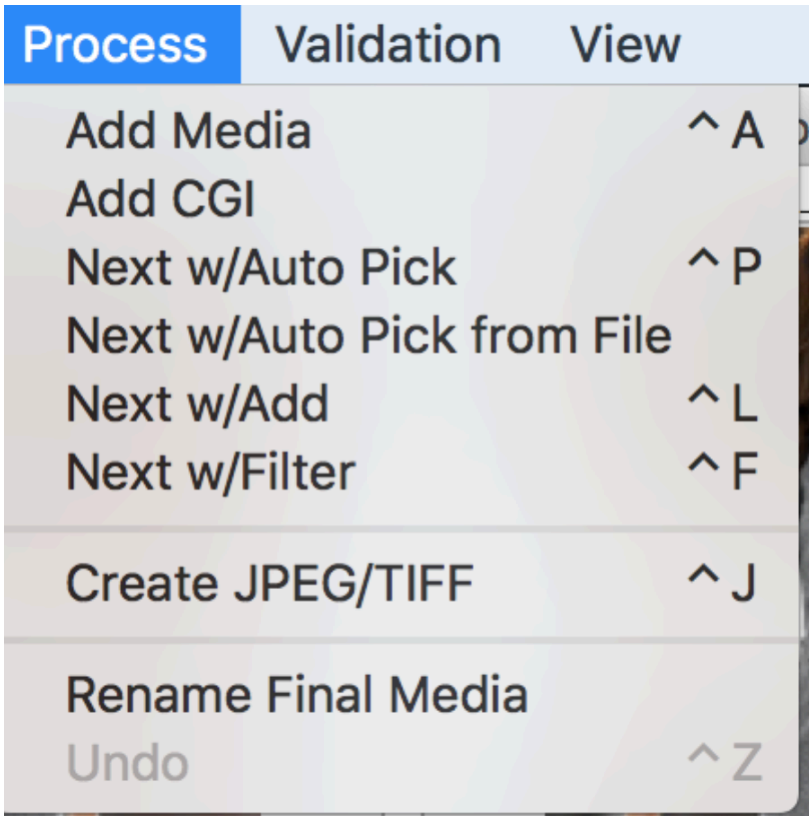


Figure 5: Process menu.

Several options are available:

- **Add Media**: Add a media file selected via file browser to project. The media can be linked to other media nodes within the graph.

- **Add CGI** Similar to add media, adds media to the tool. The media is marked a computer generated (e.g. model, GAN, etc.).

- **Next w/Auto Pick**: Automatically picks an media node without neighbors. The chosen media node is the next node in found in lexicographic order. Preference is given to those media 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 media as confirmation. A link is formed to the current media to the next selected media node.

- **Next w/Auto Pick from File**: Finds a modified version of current media file from the project directory. The modified version of the file contains the same name as the initial media 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 media as confirmation. A link is formed from the current media to the next selected media 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 media node. Unlike the other two 'next' functions, the set of operation is limited to those available from the tool's plugins. Furthermore, the media shown in the dialog window is the current selected media to which the selected modification is applied.

- **Next w/Filter Group**: Runs a group of plugin transforms against the selected media, creating an media node for each transform and a link to the new media 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 media. Each transform results in a new media node. The result from one transform is the input into the next transform. Links are formed between each media 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 media in JPEG format using the same quantization tables as the base media. It will also copy the EXIF data from the base media. For TIFFs, this will just save the media as TIFF and copy as much metadata as possible. This option should only be used at the end of processing.

- **Rename File Media**: Rename final media to their MD5 name, retaining the suffix.

- **Undo**: Removes the last operation performed. The tool does not support undo of an undo.

3.3.1 Difficulty Loading Media

Some raw image formats, TIFF files and PDF files can pose a problem for cv2, tifffile, Pillow, rawpy and PyPDF2, thus exhausting the packages loaded into the tool. The tool's primary image file is PNG. Upon difficulty loading an image, remove the image node, create a copy of the image in PNG format with the file name ending with '_proxy.png', select the image node right-button menu, select 'Proxy', select the proxy PNG file. The PNG will serve as a visual proxy.

For example, suppose the desired file is an odd TIFF format with the name 'troubleimage.tif'. Create a PNG version of that image called 'troubleimage_proxy.png' within the project directory prior to adding the TIFF image file.

3.3.2 Image Loading Plugins

An alternative to loading raw images into the tool is to provide a custom plugin. These plugins are not incorporated into the tool be default to manage the dependencies. A plugin is a python function that accepts a string file name and returns a tuple:

- *numpy* ndarray of the image (height, width, channel).
- The string mode according the PIL Image specifications *(e.g. RGB)*

The function is registered with the tool by registering any entry point called maskgen_image the setup.py. The definition is composed of a list of equations of the form:

File type suffix = package.module:function

```
entry_points=
       {'maskgen_image': [
            'cr2 = rawphoto_wrapper.opener:openRawFile'
        ]
       },
```

The package layout is as follows:

```
topfolder/
    setup.py
    __init__.py
    packagename/
      modulename.py
```

Upon installing the package, a JT can discover the function through the entry point.

3.4 View

The manipulation graph may grow fairly complex. The graph view permits LASSO of several nodes to move nodes in bulk, in the effort to format the graph for readability. Another option is

to use the 'Reformat' graph option. The reformat attempts to come up with a graph layout that is easy read, using Graphviz's[2] directed-graph *dot* layout.

3.5 Workflow

There are two primary ways to use the journaling tool: The first is to directly export manipulation steps into separate media 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** for media and raw or lossly compressed format for other media. An example workflow for this method would be:

Open Image -> Export as PNG -> Crop -> 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 media 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 -> Save as PNG -> Add image node to graph -> Crop -> Save -> Add Image node to graph-> Adjust Contrast -> Save -> Add image node to graph -> Clone -> Save -> Add image node to graph…

CAUTION: Make sure the tool works on the saved result. If the export is different in format from the working copy, the journal will be incorrect. For example, exporting to a lossless format while working with a lossy format introduces inconsistencies in the media.

The journaling tool has the ability to automatically select the next modified media in the directory (see section 3.3: Processing). To use this feature, media manipulation steps should be saved out in numerical order (e.g. media_mod_01.PNG, media_mod_02.PNG, etc).

3.5.1 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.

[2] http://www.graphviz.org/

3.5.1.1 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 (or TIFF). 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.

4) 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 media node. A base media represents an un-manipulated starting media. The base media may either be the basis for a final media or a donor to the final media. A donor media provides some part of the media to a final media.

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

4.1 Linking Nodes

Links record a single action (manipulation) taken on one media to produce another. A media node has only one primary input link. Some links allow donor links. These links are supporting the operation link. They are represented graphically as links for clarity.

A media node can contribute to multiple different manipulation paths resulting in many different media. Therefore, a media node may have several output links. A helpful graphical description of the UI is shown on the following page.

Media nodes may be selected, changing media display. The media associated with selected media node is shown in the left most media box. The right two boxes are left blank. Media nodes can be removed, and all input and output links to that node are removed as well. Media can be

connected to another media node. When 'connect to' is selected, the cursor changes to a cross. Select another media node that is either a media node to which to connect.

Media nodes may be exported. Exporting a media 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 media name (replacing '.' with '_').

Links may also be selected, changing the media 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.

4.1.1 Video Displays

When working with video nodes, media 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 media display is a button. Click on the media opens the media in the default system viewer. In the case of videos, the default system viewer is a movie player.

4.2 Interface Demonstration

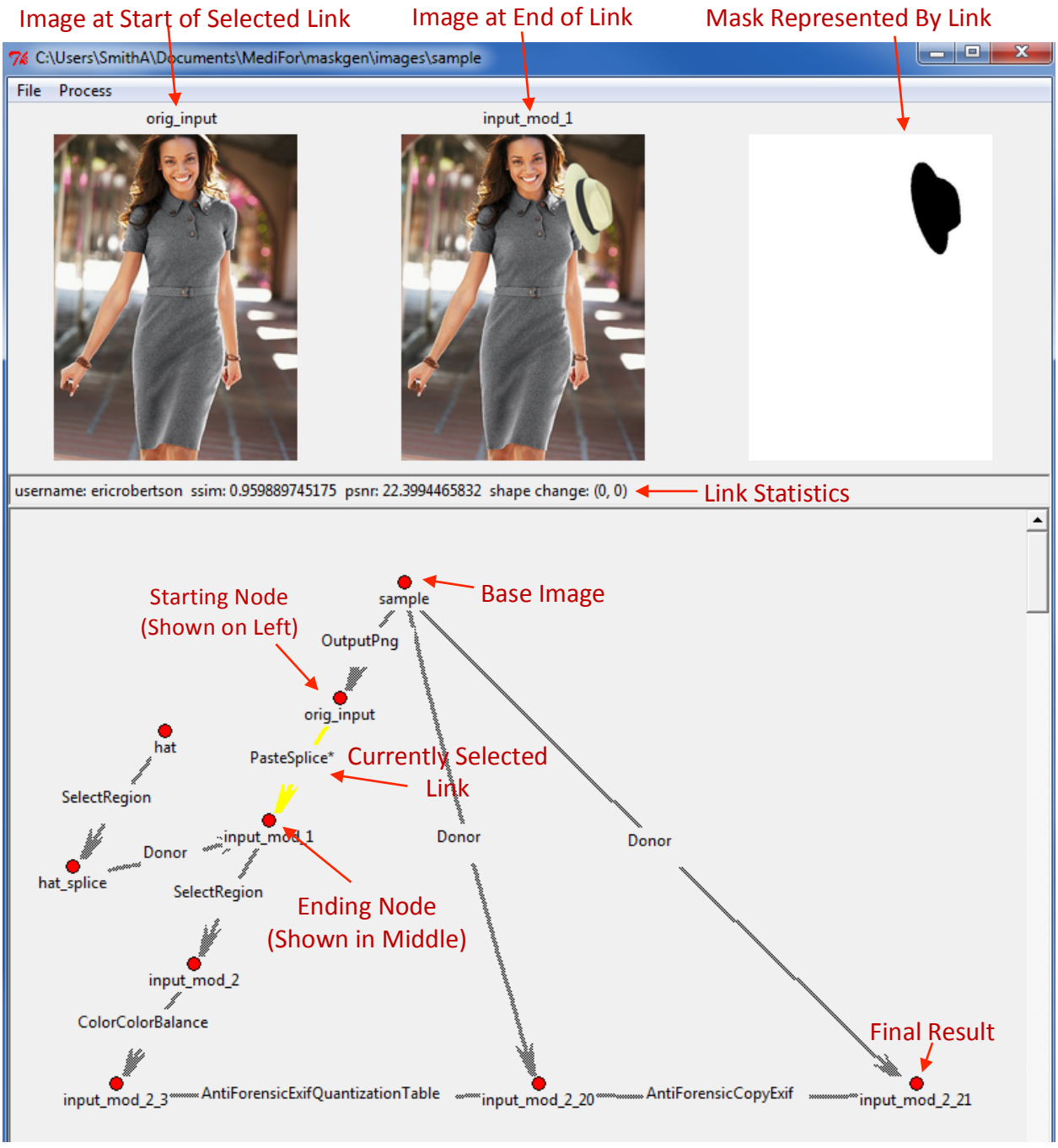


Figure 6: Description of user interface.

5) 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 media, the user can override the version set, as the versions associated with each software may be incomplete. It is important to maintain the software.csv to with the appropriate versions and work with centralize management of software.csv for consistency. Invalid versions and software fail the project validation.

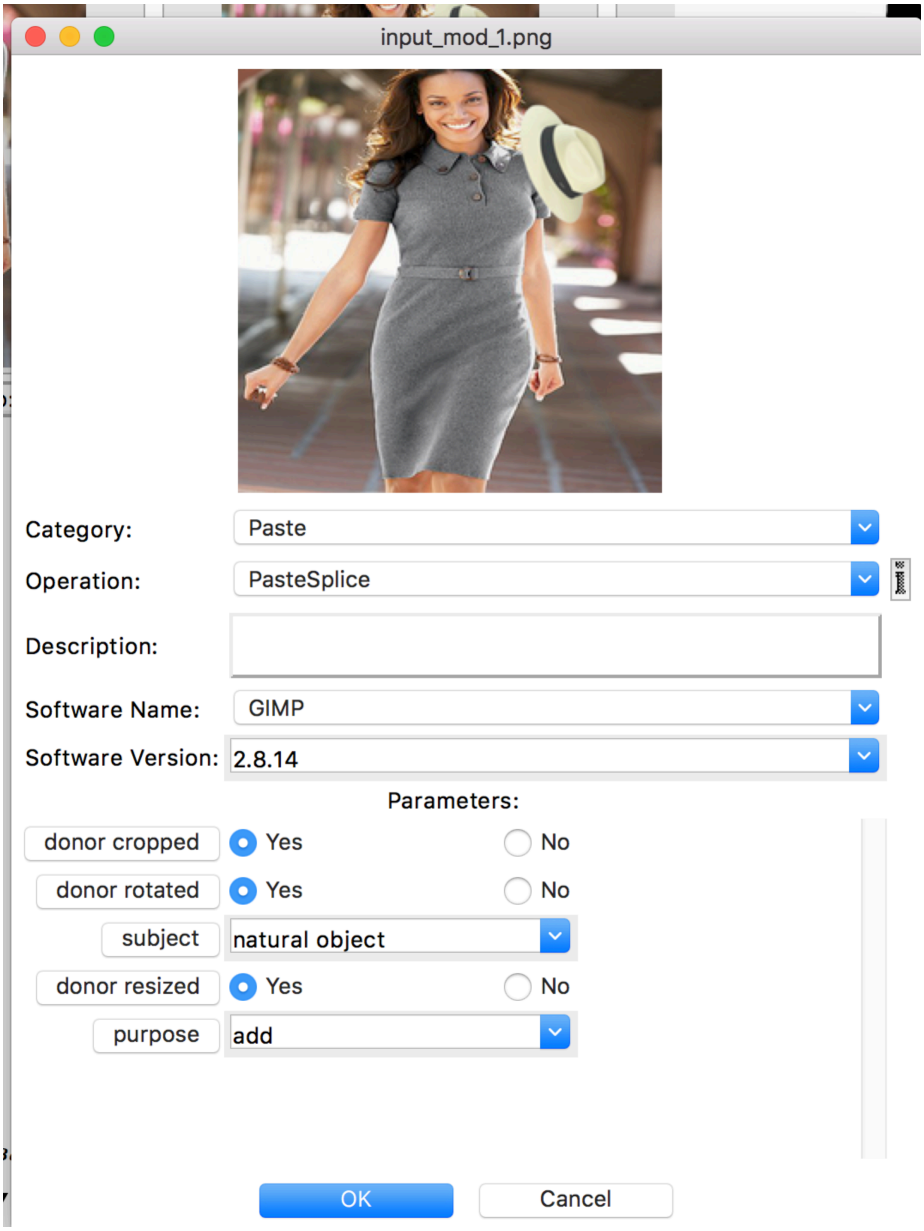


Figure 7: Link description window.

Each parameter in the parameters list is editable. The description for the parameter is shown in a dialog by clicking on the parameter name (button). 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 description.

Some parameters are mandatory. The link edit window cannot be closed without providing valid values for each mandatory parameter.

5.1 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 media. 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 media. When first attached to the description, the mask is not shown in the description dialog. On subsequent edits, the media is both shown and able to be replaced with a new attachment.

Input masks are inverted from changes masks--high intensity (255) indicated selection. Low intensity (0) indicates no selection.

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

Figure 8: Example of an input mask one might use during seam carving to ensure preservation of the woman's hat.

5.2 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 journaled 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.

5.3 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.

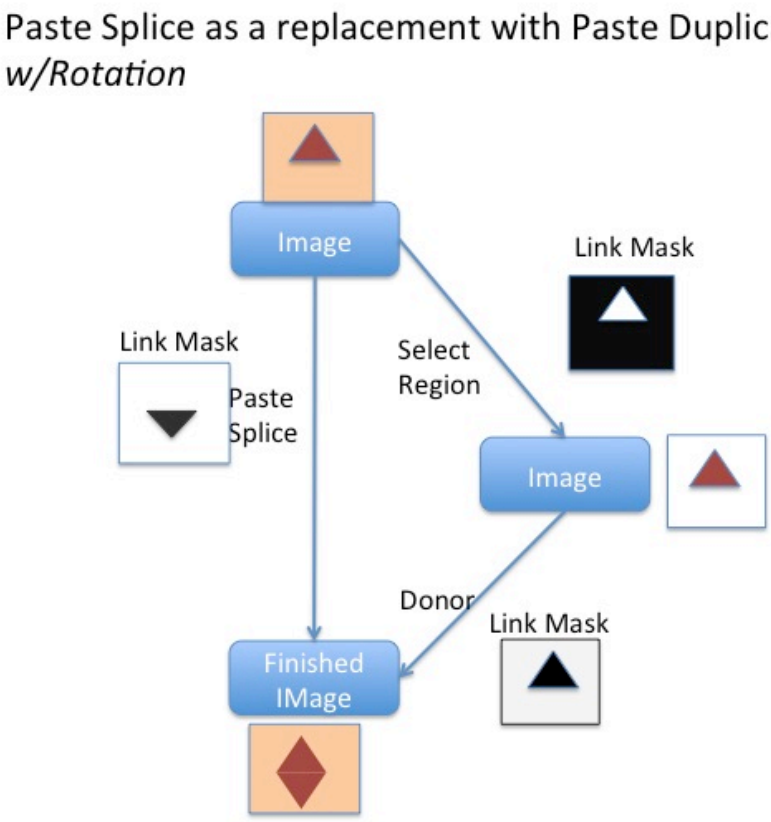


Figure 9: Paste Splice for Copy/Paste

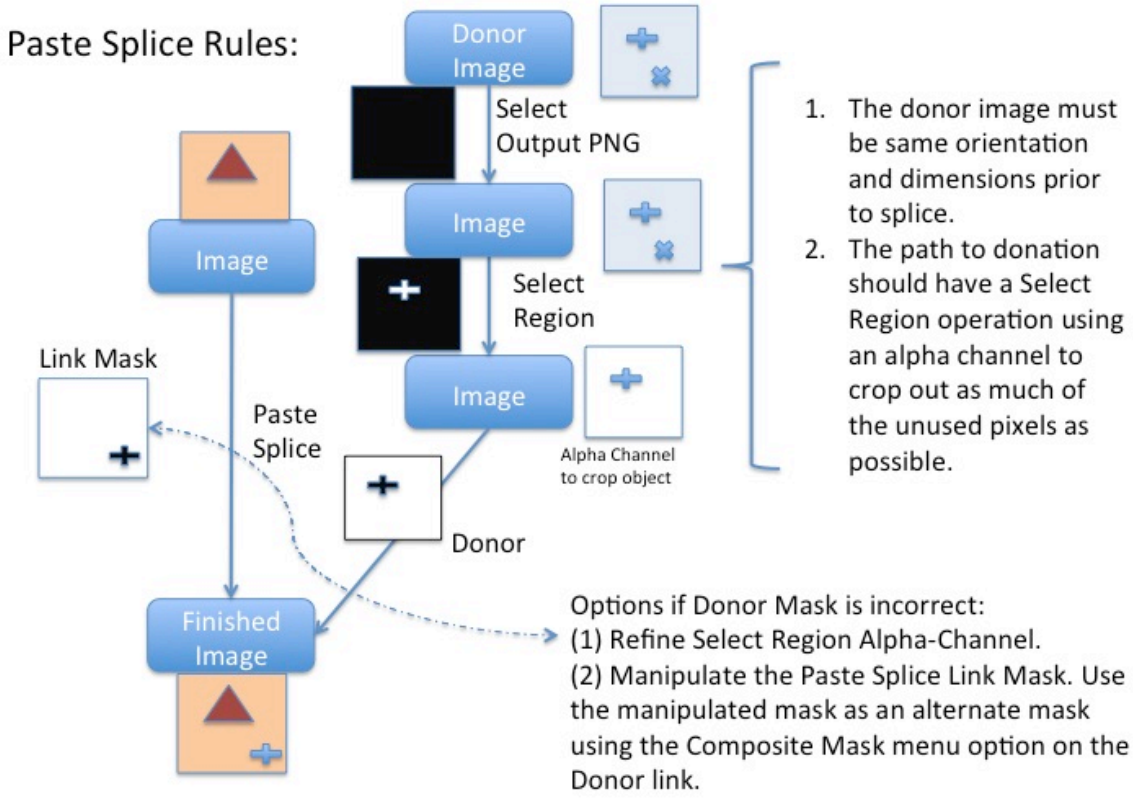


Figure 10: Rules or Paste Splice

5.4 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.

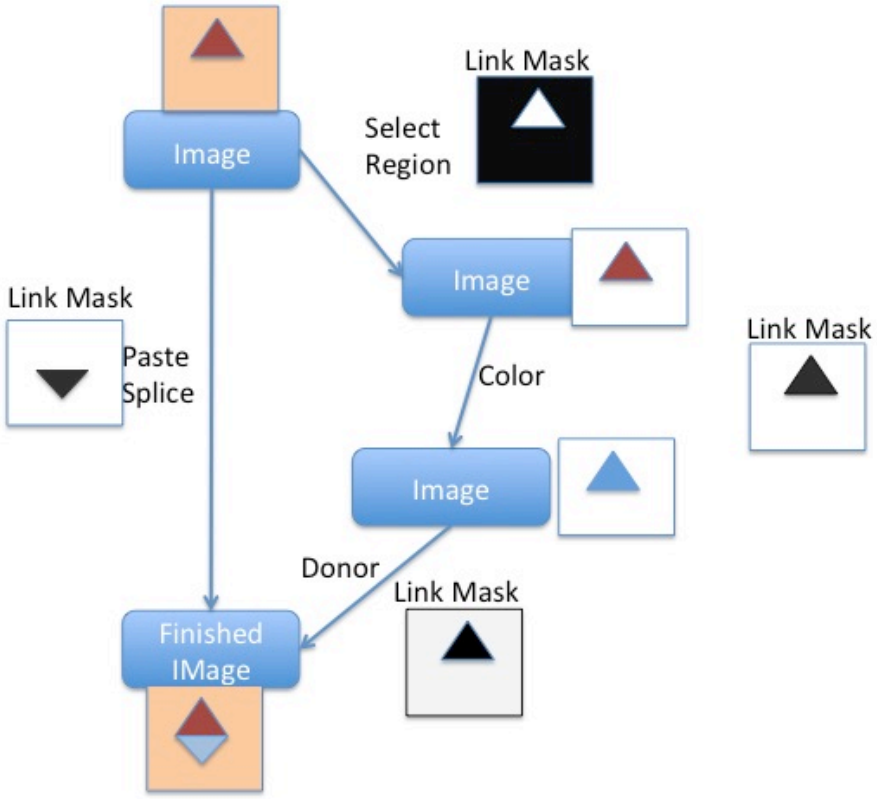


Figure 11: Shadow Paste

5.5 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.

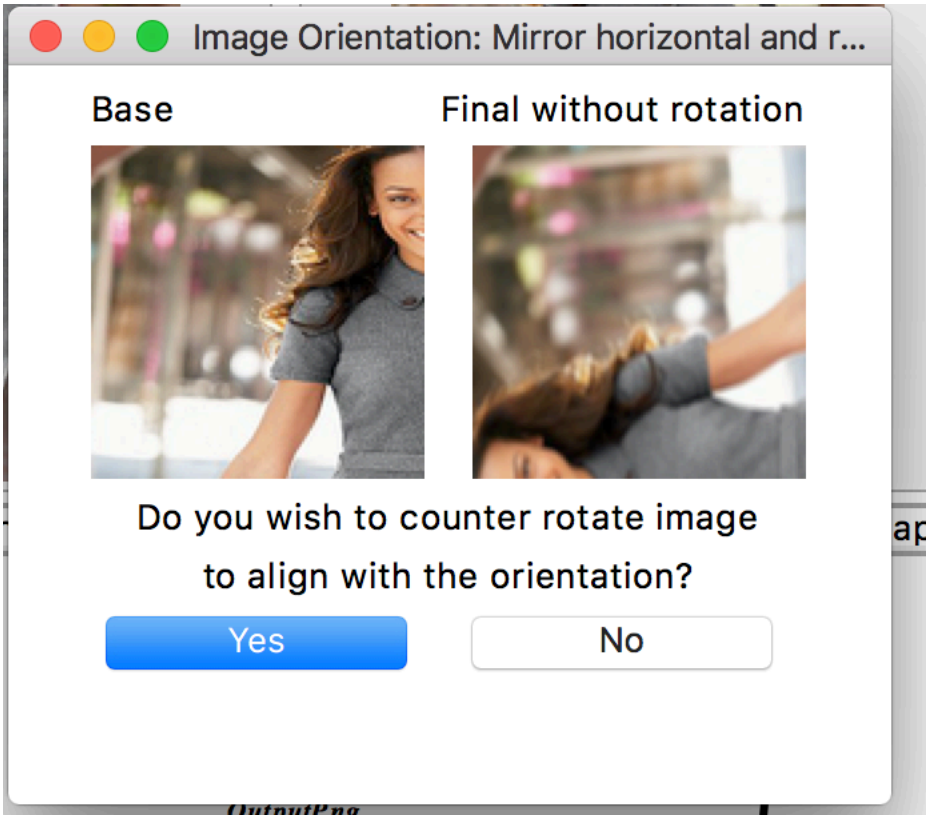


Figure 12: Image Orientation Prompt

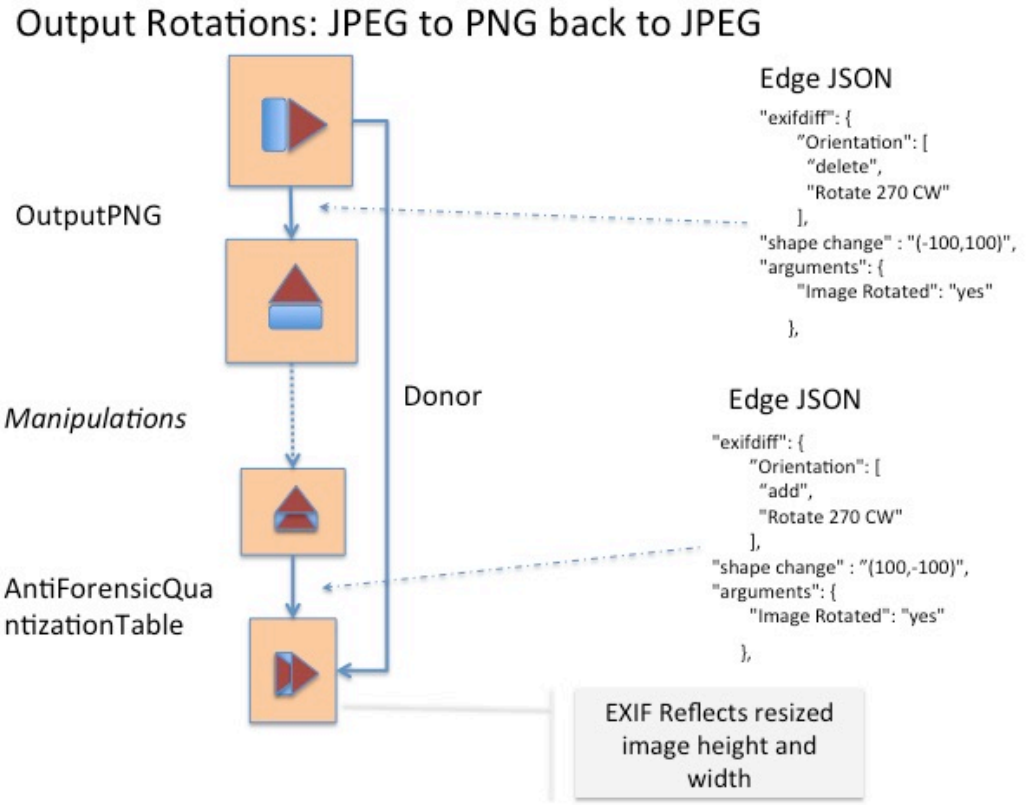


Figure 13: Summary of Orientation Workflow

5.6 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.

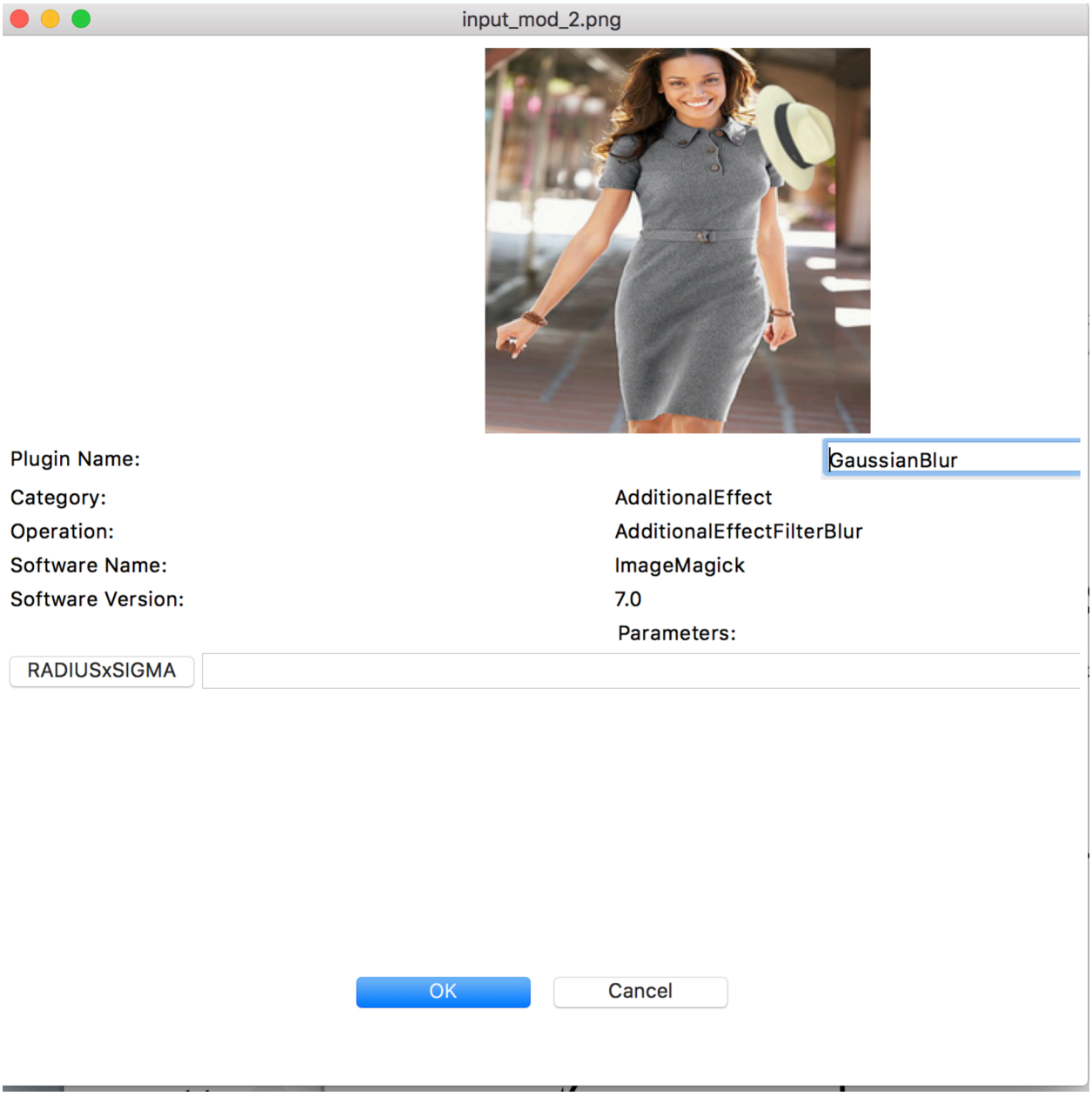


Figure 14: Window for applying plugins to media nodes.

5.7 Link Inspection

Regardless if the link is read-only (for links created by plugins) or created through one of the media 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.

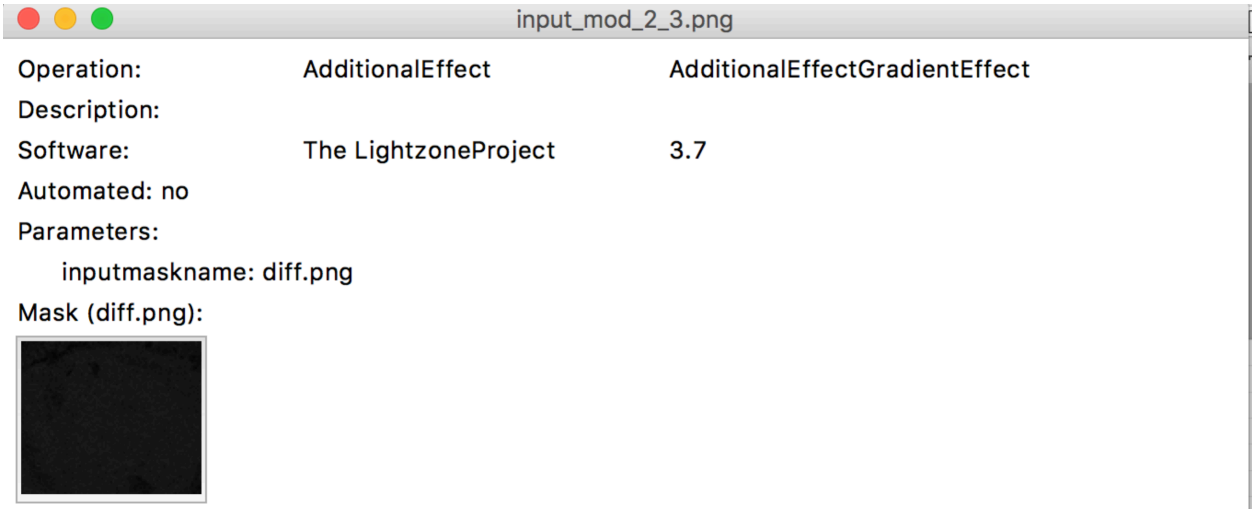


Figure 15: Link Inspection

EXIF Changes:

	Operation	Old	New
Modify Date	add		2016:07:14 13:19:14
File Inode Change Date/Time	change	2016:07:21 08:44:38-04:00	2016:07:21 10:22:30-04:00
File Name	change	input_mod_2_3_1_2.png	tmpjq4ix8.png
File Size	change	250 kB	251 kB
File Modification Date/Time	change	2016:07:18 11:19:39-04:00	2016:07:21 10:21:47-04:00
Pixels Per Unit Y	add		2835
Pixels Per Unit X	add		2835
XMP Toolkit	add		Image::ExifTool 10.19
Pixel Units	add		meters

Figure 16: EXIF Comparison Table

5.8 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.

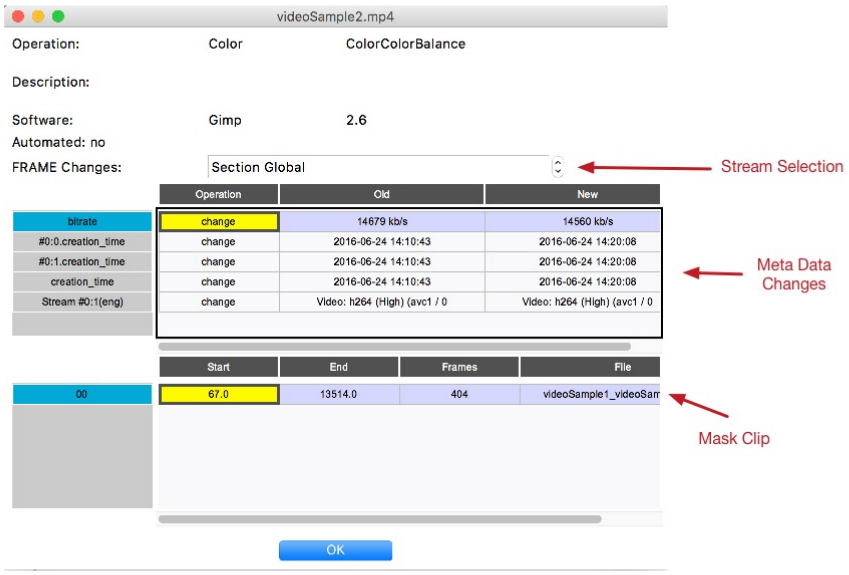


Figure 17: Video Link Inspection

Masks are stored in HDF5[3] 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.

[3] https://support.hdfgroup.org/HDF5/

```
  "vid_suffix":"mv4",
  "vid_codec":"XVID"
```

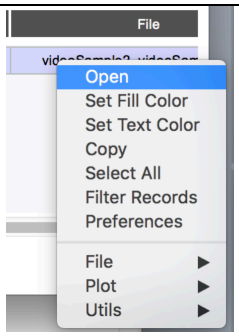


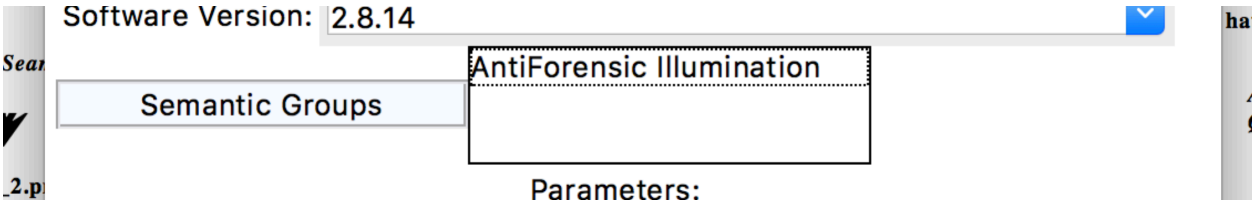
Figure 18: Video Mask Table Menu

5.9 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 media. See the dedicated JSON section of this document.

6) SEMANTIC GROUPS

Semantic groups are groups of operations that are used to achieve a specific semantic goal. Operations can be added or removed from a Semantic Group within an expandable section of the link edit window.



Groups of edges can be assigned to a semantic group by selecting (lasso or shift-click) the source and target nodes for all the links the participate in the semantic operation, followed triggering the right click menu within the graph canvas and selecting the pop-up menu option ‘Semantic Groups’.

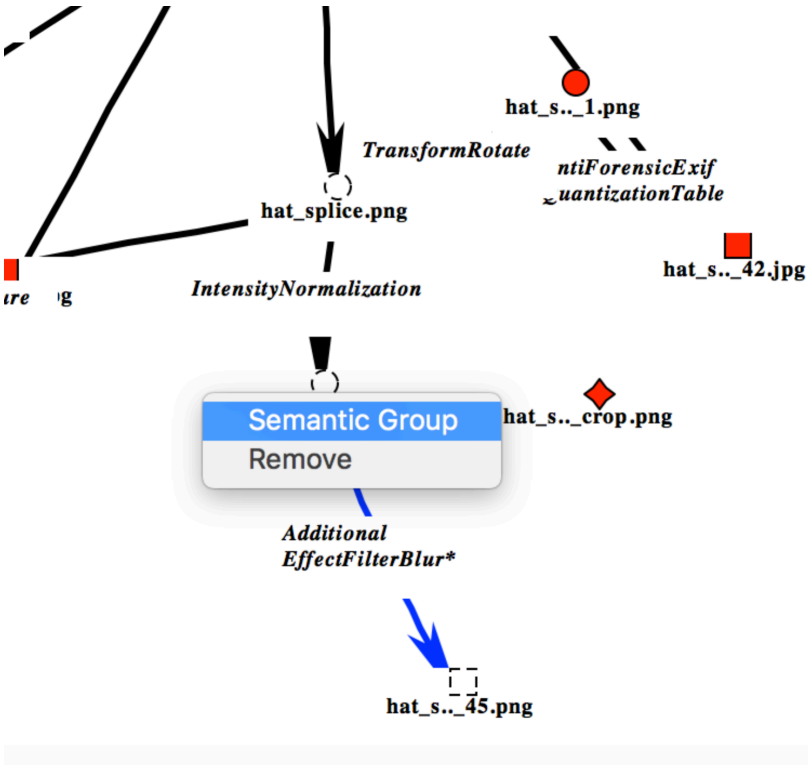


Figure 19 Semantic Groups

Semantic group participation is also verified for 'blue' links in the QA review window. The semantic groups appear between brackets next two the operation name.

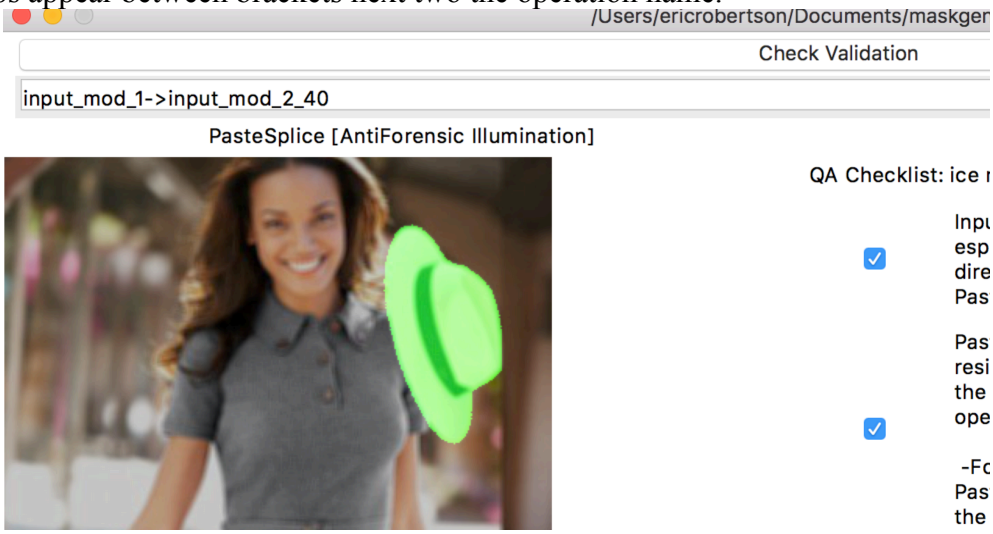


Figure 20 Semantic Group Review

7) MASK GENERATION

It most cases, mask generation is a comparison between before and after manipulations of an media. Full media 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.

Many operations and the donor link use SIFT[4] and RANSAC[5] to generate a transformation matrix used for probe (composite) and donor mask generation. These masks are transformed masks aligned to their final or base image, respectively. For details on how to use these parameters, visit section 13) QA Process.

7.1 Video Mask

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. Each set of clips is stored in HDF5 files[6]. Clips are viewable by conversion to a video format. The tool will do this automatically upon opening a clip, provided OpenCV is installed properly

7.2 Probe/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.

[4] https://en.wikipedia.org/wiki/Scale-invariant_feature_transform
[5] https://en.wikipedia.org/wiki/Random_sample_consensus
[6] https://support.hdfgroup.org/HDF5/

The media manipulator should not include all masks in the project, otherwise the composite would likely be completely black, representing a complete change in pixels. The media 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.

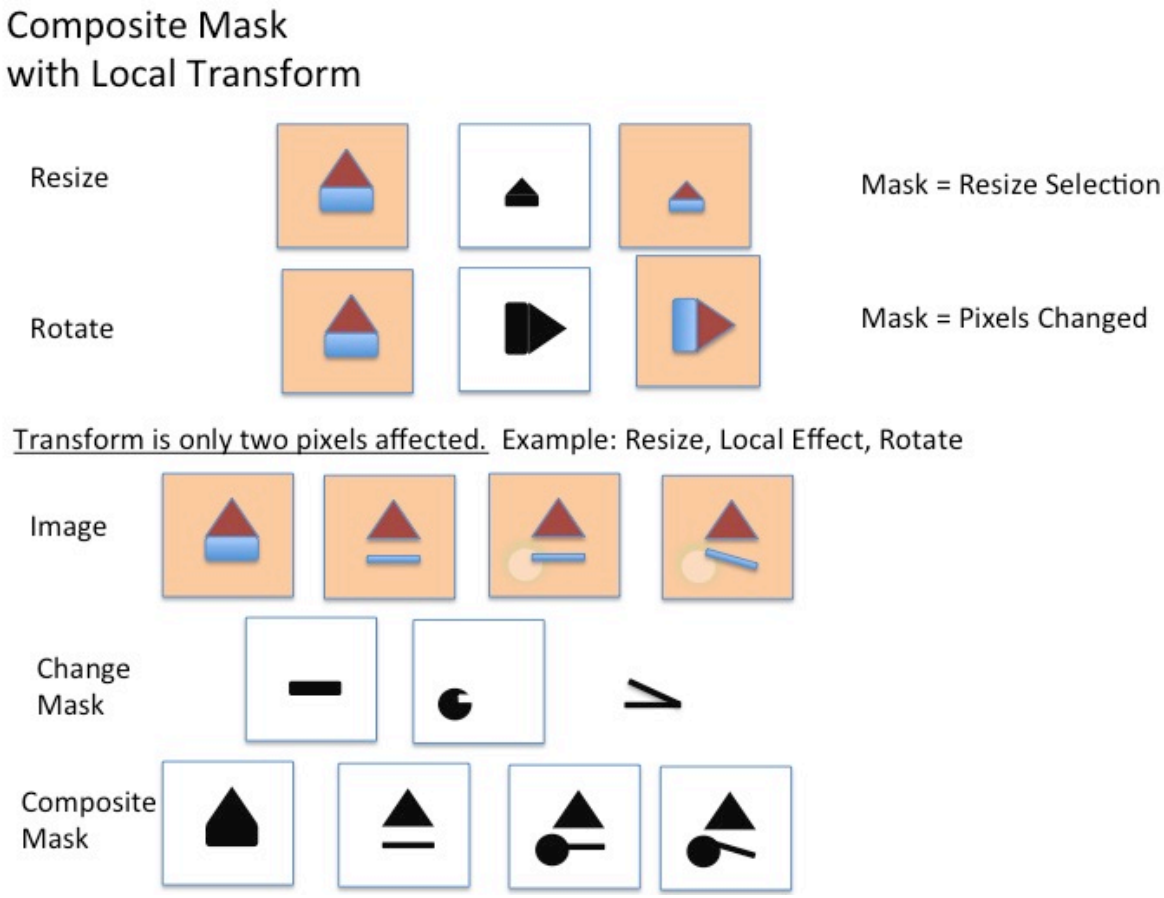


Figure 21: Composite Mask

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.

An example set of special mask generation treatment is presented in the table below. Composite mask generation involves modifying a mask according to each subsequent operational step along the path to a final media node. Donor mask generation involves modifying a donor mask according to each prior operational step along the path the donor media. Thus donor mask generation involves inverse transformations.

Operation	**Mask Description**	**Composite/Donor Description**

PasteSplice/Donor	The Donor mask for a PasteSplice attempts to use SIFT/RANSAC to find to identify the portion of the donor image cropped, rotate and scaled prior to Paste. When this fails, as I the case of fairly uniform texture images or small pasted images, the user must provided the selected donor from the image, using an alpha transparency to mask out unselected pixels.	The donor mask is the base mask for donor mask calculation. The PasteSplice mask the the base mask image for the composite mask. An optional paste mask can be used to identify the pasted region of the image when blends occur with the PasteSplice.
TransformCrop	The cropped region is identified by searching the original image for the cropped region	Composite generation applies the cropped region to the composite mask.
SelectRemove	Remove a portion of the image, the masks reflects the area removed	SelectRemove is considered a type of cut. When generating the composite mask, the pixels removed are also removed from the composite mask. For Donor mask computation, the SelectRemove is an affine transform that aligns donor mask pixels to the donor image.
TransformSeamCarving	Seam carving is	As with SelectRemove, the

	another form of cut. The composite mask reflects the pixels removed.	pixels identified in the change mask are removed from the composite mask. For Donor mask computation, the TransformSeamCarving is an affine transform that aligns donor mask pixels to the donor image.
TransformRotate	Rotation is applied to the image prior to mask generation. A perfect 90 rotation does not require interpolation. In these cases, the mask should reflect NOT change to pixels.	The composite mask is rotated accordingly.
TransformFlip	As with rotate, the image is flipped prior to mask generation. Thus, the mask should reflect no change to the image.	The composite mask is rotated accordingly.
TransformResize	Resize involves interpolating or filtering pixels. The entire image is assumed to change. Resize can be applied to a local area. In this case, the mask is a localized.	When global, the resize is applied to the mask. Otherwise, the resize does not have an effect on the composite mask generation.
Other Image Transforms	All other transforms are treated as affine transforms. The mask, itself is	The transformation matrix is applied to composite mask generation.

	just a measure of change.	

8) 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
2. 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.

9) PLUGINS

Plugin filters are python scripts. They are located under a plugins directory. Plugins from plugin directory be loaded from in one of the following locations in the given order:

- ./plugins
- environment variable MASKGEN_PLUGINS
- the python system installation directory

A plugin is not loaded twice, choosing the first one found in the provided search order.

Each plugin is a directory with a file **__init__**.py. The **init** module must provide three functions:

(1) 'operation()' returns a dictionary describing the plugin operation. The description includes operation name, category, description, software package, software version, arguments and valid transitions. The operation name and category must match operation name and category in the master operations JSON file. A transition is composed of the type of source of file and type of destination file, in terms of video and image. Valid transitions are 'image.video','video.image', 'image.image' and 'video.video'.

Arguments follow the same structure as the master operations JSON file. There is one additional name/value pair in the arguments: defaultValue. Arguments listed in

the operation definition are in addition the master operation arguments. If the argument is redefined, only the defaultValue is used.

```
def operation():
    return {
            'name':'AntiForensicCopyExif',
            'category':'AntiForensicExif',
            'description':'Copy Image metadata from donor',
            'software':'exiftool',
            'version':'10.23',
            'arguments':{
                'donor':{
                    'type':'donor',
                    'defaultValue': None,
                    'description': 'Image/video with donor metadata.'
                    }
                },
            'transitions':[
                'image.image',
                'video.video'
                ]
            }
```

(2) '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.

(3) 'suffix()' determines the suffix type of the target file. The function may return None if the suffix of the source file is used to create the target file. The function may return a specific suffix, such as '.jpg'. The function may also return the name of parameter of type 'donor', indicating the donor's image file suffix should be used for the target file.

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

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 parameters to record in the journal.

Argument values are collected as parameters from the invoking tool.

9.1 Arguments

There are three special argument keys: 'donor' , 'semanticGroups' and 'imagefile'.

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 userselected 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 imagefile. The path name is provided the transform function (e.g. inputmaskname='/somepath'). The tool does not load the image in this case.

In either case, donor and imagefile arguments are provided to the plugin as path names to the image file.

Semantic groups are added to the resulting link given the 'semanticGroups' key.

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

9.2 Return Parameters

The transform may return additional parameters for recordation in the journal. There are some actionable parameters:

- rename_target – the JT chooses the target name. The plugin can request the target file be renamed upon completion of the operation by providing the base path name.
- override_target – the plugin may choose to save results to a different file name, providing the target file full path name with this parameter key, ignoring the JT provided target file name.
- output_files – plugins may produce additional output files for retention in the journal. The JT would normally interpret the file names as strings, not including the actual file in the journal. This key associates a list of full path names to files that should be included in the journal.

9.3 Custom Plugins

The Journaling Tool also comes with a built-in Plugin Builder, capable of creating plugins from command line tools. To access this feature, go to File → Plugin Builder…

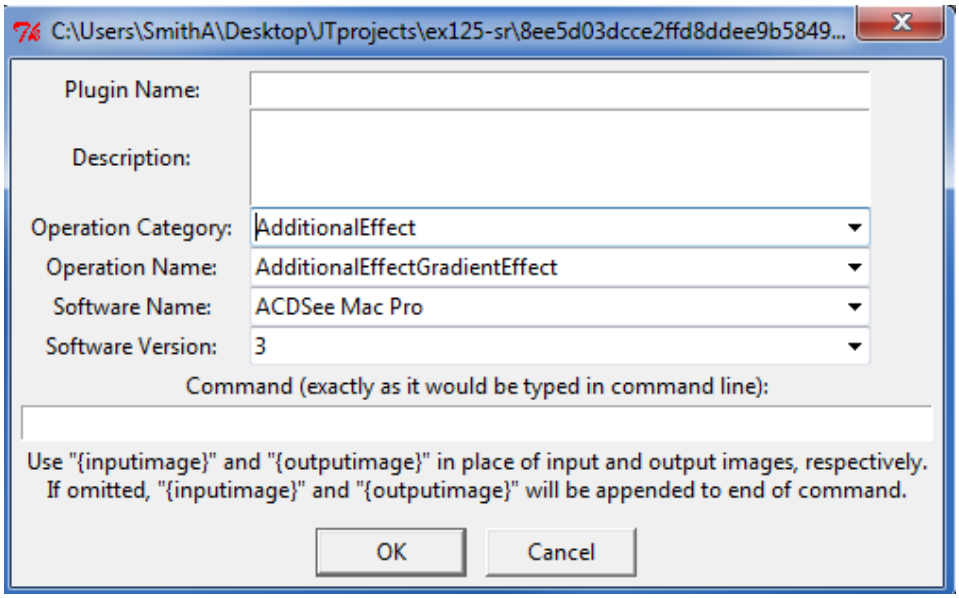


Figure 22: Custom Plugin Creation Window

Here, the user should enter the basic plugin information: Name, Description, Operation, and the Software used. Additionally, the command, exactly as it would be typed into the command line, should be entered. For example, if the user desired to create a plugin that used ImageMagick to apply an auto-gamma correction to an image, they would enter:

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

After hitting okay, the plugin data is stored in JSON format in the maskgen/plugins/Custom folder.

The JSON have three keys: name, operation and command.

The operation is defined in terms of the master operations JSON, defining the operation name, category, software, arguments and transitions. A transition is composed of the type of source of file and type of destination file, in terms of video and image. Valid transitions are 'image.video','video.image', 'image.image' and 'video.video'.

The command is composed of list of command line parameters in order of appearance for the command line. Arguments can be substituted into parameters by placing the argument name in between curly braces (e.g. {outputimage}). If {inputimage} and {outputimage} are special arguments representing the source and target file names.

```
{
    "name": "Gamma Correction",
    "operation": {
```

```
      "name": "IntensityNormalization",
      "category": "Intensity",
      "description": "Apply gamma adjustment so the mean image color will have a
value of 50% (https://www.imagemagick.org/script/command-line-options.php#auto-
gamma)",
      "softwarename": "ImageMagick",
      "softwareversion": "7.0",
      "transitions": ["image.image"]
    },
    "command": ["magick",
                "convert",
                "-auto-gamma",
                "{inputimage}",
                "{outputimage}"
                ]
}
```

More technical customizations can be made to the plugin by editing the JSON directly. One key in particular "arguments" can be added to specify arguments in the command sequence, as shown here:

```
 {
    "name": "GaussianBlur",
    "operation": {
      "name": "AdditionalEffectFilterBlur",
      "category": "AdditionalEffect",
      "description": "Automatically apply a motion blur with the given radius and
sigma value. (https://www.imagemagick.org/script/command-line-options.php#blur)",
      "software": "ImageMagick",
      "version": "7.0",
      "arguments":{
        "RADIUSxSIGMA":{
          "type": "string",
          "defaultvalue": null,
          "description": "Blur specifications, must be entered with an x in
between (e.g. 25x2)."
        }
      },
     "transitions": ["image.image"]
    },
    "command": ["magick",
                "convert",
                "-blur",
                "{RADIUSxSIGMA}",
                "{inputimage}",
                "{outputimage}"
                ]
}
```

arguments should be specified with a type (consisten with operation definitions), description and defaultvalue: The name of the argument, it's default value (use **null** for no default), and a description of the argument. The argument should also be added to the **command** parameters as a substitution indicated by surrounding the argument name with curly braces {}.

Once the plugin is saved, the user will be able to select it as if it were any other plugin. The plugin will call the command line operation, and automatically replace the input image, output image, and additional arguments with the required information.

10) EXPORT

Export performs the following steps:

- Validates the project. The user is prompted to continue if errors are found.
- 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 an output PNG file.
- Creates a gzipped tar file of the used contents of the project along with the project JSON file.

11) GROUP MANAGER

The Group Manager allows the user to create, remove and manage groups. There are two types of groups:

(1) Groups are sets of plugin image transforms. Only those transforms that do not require arguments are permitted within the group at this time.

(2) Groups are sets of operations that are collectively used to manipulate an image or video. The group represents a new operation listed under the 'Groups' category. When used, the group operation name is placed in the link 'op' description. The journal JSON file, under the 'graph' key, contains 'groups' key that lists the operations used as part of the the group operation, thus retaining the definition of that group at the time it was used in the specific journal.

```
{
   "graph": {
    "groups": {
      "myop": ["AdditionalEffectFilterBlur","AdditionalEffectFilterSharpening"]
     }
    }
}
```

12) 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

13) QA PROCESS

Once a manipulation has been journaled with the tool, it must be reviewed by a peer for completeness and accuracy. For simplicity, a QA ("quality assurance") feature has been built into the tool. To access it, click "QA…" in the Validation toolbar menu. Journals will not be accepted if they have not been reviewed.

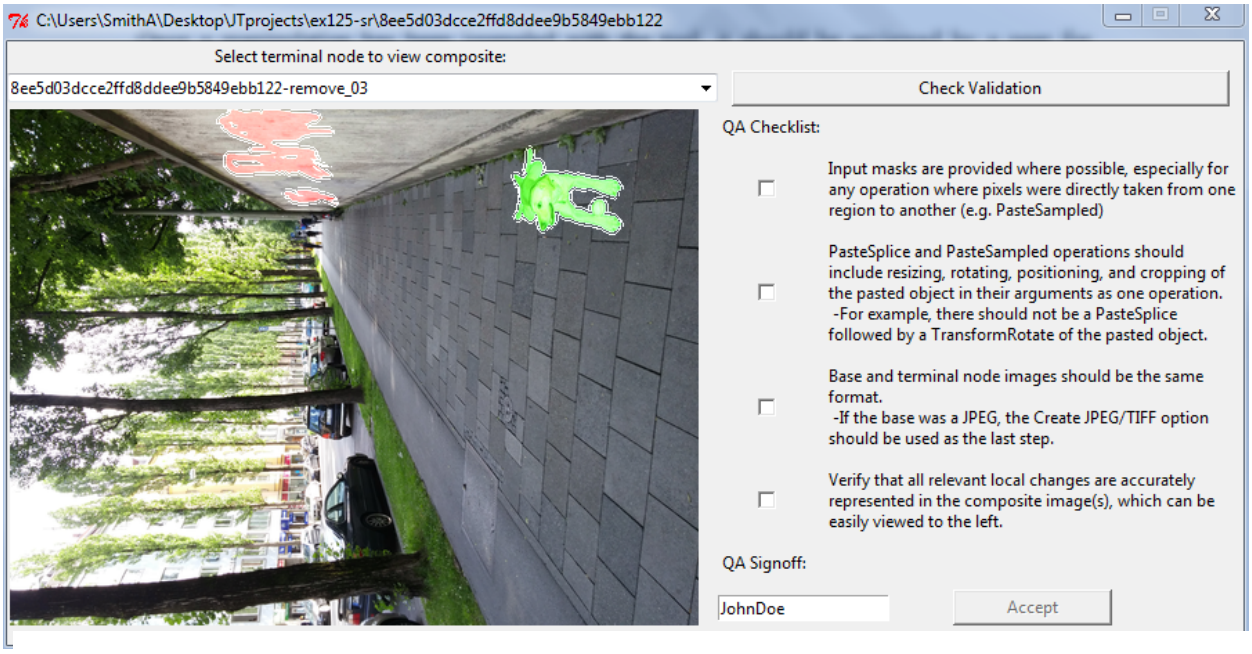


Figure 23: QA dialog window

After opening the QA dialog, it is recommended to first click "Check Validation". This has the same functionality as clicking the Validate menu item. The reviewer should first check every validation error in the list that appears. If there are unexplainable errors, the journal should not be accepted.

Review items consist of donor masks and link masks, for those links marked for inclusion in the composite (colored blue), transformed to overlay the base image, for donor masks, and final images, for link masks. The overlay must be precisely represented. Imprecise overlays represent in a missing or erroneously defined operation along the path to the base of final image, respectively.

> NOTE: SIFT is used to construct a transform matrix use to transforms masks to overlay properly on base or final images. Sometimes SIFT (or RANSAC) is not able to find sufficient features to construct the transform matrix. Many of the transform operations have the option of skipping SIFT or configuring RANSAC's sensitivity. If transforms

cause issues, go back to those problematic links, change the RANSAC parameter and then recompute the mask for that link (using the link's *Recompute Mask* sub-menu option). When applying a recomputed on a donor link, the user is prompted for the RANSAC parameter since the donor link is not a formal operation, thus not having operation arguments. When using RANSAC of None for a Donor link, the alpha channel in the donor image is used to determine the mask.

The reviewer must go through and check each review item associated with each donor and link mask overlay. There are four review items:

1. Input masks are provided where possible, especially for any operation where pixels were directly taken from one region to another (e.g. PasteSampled).
2. PasteSplice and PasteSampled operations should include resizing, rotating, positioning, and cropping of the pasted object in their arguments as one operation.
    - For example, there should not be a PasteSplice followed by a TransformRotate of the pasted object.
3. Base and terminal node images should be the same format.
    - If the base was a JPEG, the Create JPEG/TIFF option should be used as the last step.
4. Verify that all relevant local changes are accurately represented in the composite image(s), which can be easily viewed on the left side of the window.

Once all review item boxes have been checked, the journal may be accepted by entering your manipulator code name (will be auto-filled based on what name you've stored) and clicking the "Accept" button.

If the reviewer accepts the journal:

1. Click File->Export->To S3, and enter the proper Bucket/Path as usual.
2. When the Project Properties dialog appears, make sure that the User Name field has the original manipulator's user name, that "Validation" is checked "Yes", and "Validated By" and "Validation Date" fields have been filled*.

If the reviewer rejects the journal:

1. Do not export the journal.
2. Contact the manipulator to give feedback. A mechanism for this may be implemented in the future - for now, an email is recommended.
3. Once the manipulator has made the changes you suggest, re-evaluate. Do not accept the journal unless it passes all criteria.

*The QA validation will be deleted if the project is saved by a user with a different username than the reviewer. These fields are not filled, it is most likely because the reviewer's username was changed recently. Simply re-open the QA dialog, check the boxes, and accept the journal with the new username.

14) BATCH OPERATION

The journaling tool supports a batch processing including creation, extension and export. See the Batch Documentation for details.

15) OPERATION JSON

The operations defined in the operation.json file is the core of the JT behavior. The file is composed of operations.

```
{
      "category": "PostProcessing",
      "name": "Recapture",
      "mandatoryparameters": {
        "Output Device": {
          "type": "list",
          "values": [
            "Gloss Paper",
            "Flat Paper",
            "LCD",
            "LED",
            "CRT",
            "Screen Shot",
            "Other Device"
          ],
          "description": "Type of image display"
        },
         "Resolution": {
          "type": "text",
          "description": "Resolution of output device e.g. 300 DPI of print or PPI
of screen. https://www.noteloop.com/kit/display/pixel-density/"
        },
         "Capture Distance": {
          "type": "int[0:100000]",
          "description": "Distance in centimeters from camera to print"
        },
        "Magnification": {
          "type": "float[0:100000]",
          "description": "Magnification = (hi/ho) = -(di/do), where hi = image
height, ho = object height, and di and do = image and object distance"
        }
      },
      "analysisOperations": [
        "maskgen.tool_set.siftAnalysis"
      ],
      "rules": [
      ],
      "optionalparameters": {
        "Capture Camera ID": {
```

```
          "type": "text",
          "description": "ID of camera used to capture image if not screen shot"
        },
        "Capture Printer ID": {
          "type": "text",
          "description": "ID of printer used to compose the image"
        }
```

15.1 Rules

Operation rules are functions that accept three parameters to identify the project graph and the link to be validated.

- masken.software_loader.Operation instance
- maskgen.image_graph.ImageGraph
- start node id
- end node id

A rule function returns None if the link is valid, otherwise the function returns a tuple:

- Severity as defined in maskgen.validation.core
- Error String

`Example:` (Severity.WARNING, **'Starting node appears to be compressed'**)

15.2 Compare Parameters

Compare components configure the construction of the change masks and associated meta-data.

```
"compareparameters" : {
  "function":"maskgen.tool_set.resizeCompare"
  "video_function":"maskgen.video_tools.resizeCompare
  "convert_function":"maskgen.tool_set.extractAlpha
},
```

The *function* key determines the function used to create a mask for images. The *video_function* key determines the function used to create a mask for video and audio. The *convert_function* key determines the function used to adjust image to a 16 bit gray scale prior to comparison. It currently only works for images

All other key-value pairs in the parameters list is provided to the indicated function as key-value arguments.

The signature for an image function is:

```
def function_name(img1, img2, arguments=dict()):
  “” ”
     Return mask and initial analysis results such as a location, rotation amount,
etc.
    :return mask is the same of the source image (img1)
    @type img1: maskgen.image_wrap.ImageWrapper
    @type img2: maskgen.image_wrap.ImageWrapper
    @rtype:(numpy.ndarray,dict)
   “” ”
```

The signature for an convert function is:

```
def function_name(img1, img2):
  “” ”
     Return mask and initial analysis results such as a location, rotation amount,
etc.
    :return mask is the same of the source image (img1)
    @type img1: maskgen.image_wrap.ImageWrapper
    @type img2: maskgen.image_wrap.ImageWrapper
    @rtype:numpy.ndarray
   “” ”
```

The signature for a video function is:

```
def function_name (fileOne, fileTwo, name_prefix, time_manager, arguments=None,analysis={}
);
“””
    Function fills analysis dictionary.
    Function returns a tuple:
     1. list of masks.  Each mask is a dictionary.
     2. list of errors
    :param source: source file name
    :param target: target file name
    @type fileOne: str
    @type fileTwo: str
    @type name_prefix:
    @type time_manager: maskgen.tool_set.VidTimeManager,
    @type arguments: dict
    @type analysis: (list of dict, list of str)
“””
```

Each mask dictionary has the following keys:

- startframe = frame number from 1
- endframe = the last manipulated frame
- framecount = endframe – startframe + 1
- starttime = start time starting at 0
- endtime = time of the last manipulated frame
- type = ‘audio’ or ‘video’
- rate = frames per second. Each frame consumes 1000.0/rate milliseconds
- videosegment = filename of the hdf5 file containing the spatial masks

15.3 Analysis Operations

```
optionalSiftAnalysis(analysis, img1, img2, mask=None, linktype=None,
arguments=dict(),directory='.')
```

Each analysis operation is function, defined by the *package.module.function name*. The function is defined with the following parameters:

- Analysis -> a dictionary to put the results

- Img1 -> maskgen.image_wrap.ImageWrapper for the source image
- Img2 -> maskgen.image_wrap.ImageWrapper for the destination image (result of the operation)
- Mask -> maskgen.image_wrap.ImageWrapper containing the mask generated when comparing the images
- linkType -> one of 'video.video', 'image.video','video.image','image.image','audio.video','video.audio'.
- arguments -> a dictionary of arguments provided to the operation (collected from the link parameters).
- Directory -> the project directory

15.4 Probes and Mask Transformation Function

Mask Transformation is used to align manipulation's spatial and temporal masks to probe media. Probe media is a final node. Each probe is the manipulation aligned to the final media to which it contributed to. Any manipulation can alter the affects of prior manipulation. These alterations must be applied to the mask to support realignment. Realignment includes frame and time references as frames move around in a video, pixel adjustments as prior manipulated pixels are further move or adjusted. Some manipulations may remove prior manipulations such as TransformCrop, SelectRemove, Paste operations (pasting over prior changes), cut frames, overlaying frames, etc.

Each manipulation operation is associated with mask transformation functions that apply their action to masks from prior manipulations. These may be as simple as resizing an image or apply a homography. Some operations can be quite complex, such as Recapture which may resize, rotate and crop to fit the probe mask into target dimensions.

Composite masks are probes for the same final node collapse into one JPEG2000 or Color Image. Often, the term composite mask and probe mask are used interchangeably.

Donors are components of a probe. Each probe may have an associated donor describing the contributing donor media. The donor must be transformed in the reverse direction to align with the base donor media. Each transformation function supports both directions: affectively applying the transformation in the composite (final) node direction and inversely to the donor.

A probe (maskgen.mask_rules.Probe) is defined as:

edgeId = the tuple (source,target) graph link identifier
targetBaseNodeId = the based node id
finalNodeId = the final media node id
composites = a dictionary describing the location of this image in a one or more composite images
donorBaseNodeId = the donor base media node id
donorVideoSegments = the list video and audio donor segments
targetMaskImage = the ImageWrapper target mask image (aligned to target final node)

donorMaskImage= the ImageWrapper donor mask image (aligned to donor)
targetMaskFileName = the file name of the target mask image (aligned to target final node)
donorMaskFileName = the file name of the donor mask image (aligned to donor)
targetVideoSegments = the list video and audio final node segments
targetChangeSizeInPixels = (0,0)
level = 0

A segment is defined as:

rate = frames per second
starttime = in milliseconds; display time of the first frame manipulated
startframe = integer number; the first frame manipulated
endtime = in milliseconds; display time of the last frame manipulated
endframe = integer number; the last frame manipulated
frames = integer total frames changed in segment (endframe-startframe + 1)
filename = string

A probe for an image is the image mask.
A probe for an audio component is set segments.
A probe for a video component is a set of segments with a video mask file (hdf5)

Mask transformation functions are defined for a specific media type:

```
"maskTransformFunction": {
  "video": "maskgen.mask_rules.select_cut_frames",
  "video_preprocess": "maskgen.mask_rules.select_cut_frames_preprocess",
  "audio_preprocess": "maskgen.mask_rules.select_cut_frames_preprocess",
  "audio": "maskgen.mask_rules.delete_audio"
},
```

In some cases, the probe masks (not donors) generated by the mask generation code needs to be preprocessed. The reason is because the masks are in terms of the source dimensions, frame rates and durations. The preprocessor applies the affect to move the mask into the target space.

The preprocessor is specific to media type. The pre-processors are functions associated by the media type followed by '_preprocess'. In the example, SelectCutFrames operation requires a preprocessor to mask the frames as the prior frame and the following frame of the cut. The reason is that the probes for cut frames cannot references frames that no longer exist in the final video.

During probe and donor image construction, the transformation function takes the current mask and transforms it according to the parameters provided to the associated link operation. For

example, a TransformResize link resizes the composite or donor mask according the resize specification. Donor mask construction is the inverse of composite construction. Thus, if the image size changed from (500,500) to (450,450), then the donor transformation would resize a (450,450) mask into a (500,500) mask.

```
def resize_transform(build_state):
```

Each function must return a CompositeImage object.

Each transform function is defined by the *package.module.function name*. The function is defined to consume a build_state of type *maskgen.mask_rules.BuildState* which contains the following parameters:

- edge -> a dictionary containing all the edge arguments and analysis results
- edgeMask -> the numpy ndarray of the edge mask
- compositeMask -> CompositeImage for video or numpy array for image. The contained mask values are set by level starting with 1. Alterations should be careful to preserve the numbering. One strategy is to process each pixel at a time, convert the level to 255 and then back again upon completion.
- donorMask -> CompositeImage for video or numpy array for image. The mask values are 0 and 255. The donorMask attribute is not None, then perform the transformation in reverse using the embeddded mask.
- source -> source node id for edge
- target -> target node id for edge
- directory -> location of journal
- pred_edges -> list of edge ids for predecessor edges to the source node.
- graph -> the ImageGraph of the journal

```
CompositeImage = namedtuple('CompositeImage', ['source', 'target', 'media_type',
'videomasks'])
```

Videomasks is a list of dictionaries. Each dictionary has the following keys:

- startframe = frame number from 1
- endframe = the last manipulated frame
- framecount = endframe – startframe + 1
- starttime = display time starting at 0 in milliseconds of the first manipulated frame
- endtime = display time of the last in milliseconds manipulated frame
- type = ‘audio’ or ‘video’
- rate = frames per second. Each frame consumes 1000.0/rate milliseconds

15.5 Transitions

An operation is established for a limited set of transitions, called link types, between different types of media. Allowed values are: of 'video.video', 'zip.video', 'zip.image', 'image.video',' video.image', 'image.image', 'audio.video', and 'video.audio'.

Zip are treated similar to videos with an audio stream.

15.6 Parameters

There are two sets of parameters, mandatory and optional. The parameters are mappings of parameter name to a structure defining the parameter.

A parameter is defined with:

- type:
  - float[<start>:<end>] – example float[0.0:1.0]
  - int[<start>:<end>] – example int[1:100]
  - text
  - yesno
  - list
  - time
  - frame _or_time -> picks time if audio, frame if video depending on the source file type
  - string
  - coordinates
  - listfromfile:<filename> -- a text file containing a list of possible entries
  - file:image,
  - file:<suffix> -- example file:png or file:xmp
  - folder:<location> -- example folder:plugins/QTTables
- values: the list of values in the case of list type
- description: a string description

A coordinate type parameter has the value format (n,m) where n and m are numbers.

15.7Other Components

- includeInMask -> a dictionary indicating if a difference mask and information is include in probe by default (blue link). The key 'default' covers all media type. Example:
  { 'default': false, 'video':true}
- generateMask -> one of 'all', ',meta', 'audio', 'frames'. Perhaps the most confusing parameter, frames is used to capture meta–data on frames, excluding spatial masks. 'all' and 'audio' include spatial masks.

15.8 Adding new rule functions

Rule functions are used in both operations and project properties. Add new rule functions follows a similar approach to loading new image loading plugins. The Setuptools entry point 'maskgen_rules' permits the discovery of new rules installed outside the maskgen core module.

```
entry_points=
       {'maskgen_rule': [
            'ruleid = apackage.amodule:aRuleFunction'
        ]
       },
```

The package layout is as follows:

```
topfolder/
    setup.py
    __init__.py
    apackage /
      amodule.py
```

16) PROJECT 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. This includes semantic Groups:

```
graph:[
    "validationtime": "19:19:03",
    "projectdescription": "DeepFake Face Swap",
    "creator": "dupre",
    "validatedby": "",
    "jt_upgrades": [
```

```
      "0.5.0515.afee2e2e08",
      "0.5.0515.49f20fd1fa"
    ],
    "creator_tool": "jtui",
    "autopastecloneinputmask": "no",
    "idcount": 8,
    "provenance": "no",
    "manipulationcategory": "",
    "semanticgroups": [
      "Face Manipulations"
    ],
    "skipped_edges": [],
    "username": "dupre",
    "igversion": "0.5.0515.49f20fd1fa",
    "validationdate": "08/14/2018",
    "edgeFilePaths": {
      "maskname": "",
      "videomasks.videosegment": "",
      "arguments.XMP File Name": "",
      "arguments.model image": "",
      "arguments.convolutionkernel": "",
      "selectmasks.mask": "",
      "arguments.pastemask": "",
      "arguments.qtfile": "",
      "inputmaskname": "inputmaskownership",
      "arguments.PNG File Name": ""
    },
    "nodeFilePaths": {
      "KML File": "",
      "proxyfile": "",
      "compositemaskname": "",
      "donors.*": ""
    },
    "qadata": {},
    "groups": {},
    "updatetime": "2018-08-14 21:19:02",
    "projecttype": "video",
    "name": "29522a7c74eb85e4aee67ad7dd86a235",
    "typespref": [
      [
        "jpeg files",
        "*.jpg"
      ],
      [
        "png files",
        "*.png"
      ],
      [
        "tiff files",
        "*.tiff"
      ],
      [
        "*.*",
```

```
        "*.*"
      ],
      [
        "*.mpg",
        "*.mpg"
      ]
    ],
    "modifier_tools": [
      "jtui",
      "jtprocess"
    ],
    "organization": "Wonderland",
    "validation": "no",
    "technicalsummary": "DeepFake Face Swap.  Fake App 2.0."
  },
```


The graph properties include:

- ***modifier_tools*** – tools used to modify journal
- ***organization*** – organization that is responsible for creating journal
- ***validation*** – is journal validated
- ***technicalSummary*** – technical description of journal's intent
- ***nodeFilePaths*** – attributes of a node that reference a file to be included in the journal
- ***edgeFilePaths*** – attributes of an edge that reference a file to be included in the journal
- ***creator_tool*** – tool that created journal
- ***username*** – user who created journal
- ***validatedby*** – person who validated the journal
- ***validationdate*** – when the journal was validated. Example format: "06/08/2018"
- ***validationtime*** – when the journal was validated. Example format: "21:19:03"
- ***projectdescription*** – general description of journal
- ***qacomment*** – comment by validator
- ***qadata*** – detailed recorded per edge on what has been validated
- ***updatetime*** – Example format: "2018-08-14 21:19:02"
- ***semanticgroups*** – list of all semantic groups used in the journal (as a summary)
- ***manipulationcategory –*** indication of complexity in journal by depth (1-Unit,2-Unit, etc.)

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
- ***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).


```
{
   "source": 16,
   "target": 0,
   "username": "ericrobertson",
```

```
   "maskname": "cropTest_cropTest_1_mask.png",
   "description": "\n",
   "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": "",
     "op": "FilterBlurMotion",
   "argments":  {“x”:1},
   "semanticGroups": [
        "Date Burn-in"
      ],
   "change size category": "large",
      "psnr": 9.785041050027957,
   "change size ratio": 0.8456920469992365,
      "shape change": "(0, 0)",
    "exifdiff":  {…},
    "recordMaskInComposite": "no",
    "linkcolor": "85 85 85",
    "metadatadiff": {…},
    "errors": [],
    "location": "(90, 350)",
    "tool": "jtui",
    "videomasks": [],
    "automated": "no",
    "experiment_id": "no"
}
```


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. For videos, the mask is a snapshot of the spatial mask.
- ***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 composite mask, aligned with the final image node.

- ***recordCompositeInMask*** –'yes' if the mask for this link should be included as a probe 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.)

- ***global*** – the global/local indicator is part of an auto change detection analysis to deteremine if an operation affected a localized set of pixels or a diffuse set of pixels across the entire image. Validation rules notify journal reviewers if there is an inconsistency between the *global* and *recordCompositeInMask* properties. The *global* property does not exist for every operation. Rather, it exists for operations that produce global or local results.

- ***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]

- ***linkcolor*** – 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.

- ***experiment_id*** – Filled by the Auto Batch JT function to label some experiment identifier (optional).

- ***tool*** – the tool used to make the node including jtproject, jtprocess (extension) and jtui.

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.*

16.1.1Meta Data Diff

Meta Data Diff is a list with two entries: (1) Global changes (2) Frame Changes.
Global changes are represented as a dictionary where each key is the stream number ':' the attribute:

```
"0:coded_picture_number": [
            "change",
            "4",
            "3"
          ],
```

Since stream's can change position, the JT forces the video to be in position 0. Audio tracks are assumed to align.

The frame changes is a dictionary keyed by stream id (0,1,etc.) associated with a list of changes recorded on frames:

```
"0": [
            "change",
            [
              [
                "change",
                3,
                1,
                0.016683,
                {
                  "coded_picture_number": [
                    "change",
                    "4",
                    "3"
                  ]
                }
              ],
              [
                "change",
                6,
                4,
                0.066733,
                {
                  "pict_type": [
                    "change",
                    "B",
                    "P"
                  ]
                }
              ],
…
     ]
]
```

17) 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.

Since operations may be used for different types of media (image and video), operation based rules can be restricted to a media type given the ‘nodetype’ key with values of ‘image’ or ‘video’.

Multiple operations can be represented by a rule by replace the ‘operation’ key with an ‘operations’ key. The value for the operations key is a list of operation names.

17.1 Example Edge with Specific Operation for Media Type Video

```
{
  "description": "Post Process Crop Frames",
  "name": "postprocesscropframes",
  "node" : true,
  "operation": "TransformCrop",
  "nodetype": "video",
  "type": "text",
  "information": "Post Process Crop Frames"
}
```

17.2 Example Edge with Specific list of Operations for Media Type Image

```
{
  "description": "Image Compression",
  "name": "imagecompression",
  "type": "yesno",
  "operations" :
["AntiForensicExifQuantizationTable",["AntiForensicJPEGCompression"],
  "nodetype" : "image",
  "node" : true
},
```

17.3 Example Edge with Specific Operation and Argument Value

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

17.4 Example with a Rule

```
{
```

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

17.5 Example Project Property without a Rule

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

18) API

18.1 Open and Interrogate a Journal

```
from maskgen.scenario_model import ImageProjectModel,Description

#project can be the path to the JSON file, the directory of the journal
# or the tgz (tarred zip file) of the journal
scModel = maskgen.scenario_model.ImageProjectModel(project)

# obtain list of final node IDs
scModel. finalNodes()
# find base node for given node id
scModel.getBaseNode(node)
# get the name of the project
scModel.getName()
# get the list of based node ids (typical only one)
scModel.baseNodes()
# get successor node ids to node
scModel.getGraph().successors(nodeid)
# get predecessor node ids to node
scModel.getGraph().predecessor(nodeid)
# returns the Description object describing the link
description  = scModel. getModificationForEdge(start,end)

#obtain all descriptions for all links
scModel.getDescriptions()
```

Description is:

- operationName = None

- additionalInfo = description
- inputMaskName = filename of inputmask
- maskSet = list of mask items
- recordMaskInComposite = Record the link in the probe. Uses 'no' and 'yes' to mirror JSON
- arguments = dictionary arguments used by the operation
- software = software name
- automated = 'no'
- errors = list of mask generation error text
- generateMask = as set in the operation
- username = user's name as recorded by the tool
- ctime = time
- start = start node id
- end = end node id
- semanticGroups = list of group names

Video Mask Set consists of a list of dictionaries, each describing a single segment. See Compare Parameters.

18.2 Create Probes

Create CSV file for all the probes and target probe files.

```
from maskgen.services.probes import archive_probes
archive_probes( 'project_directory/project_json_file.json' , directory='.',
archive=True, reproduceMask= True)
```

Generate with API:

```
from maskgen.scenario_model import ImageProjectModel
from maskgen.mask_rules import Probe

scModel = maskgen.scenario_model.ImageProjectModel(project)
probes = scModel.getProbeSet(compositeBuilders=[EmptyCompositeBuilder])
```