MaskGen

**Journaling Tool**

**20 July 2016**

Prepared for:

**Submitted by:**

|  |  |  |  |
| --- | --- | --- | --- |
| **REVISION RECORD** | | | |
| **REVISION** | **DATE** | **DESCRIPTION & PAGE AFFECTED** | **AUTHOR** |
|  | 7/20/16 | Initial Revision |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[1. Overview 1](#_Toc457993679)

[2. Installation 1](#_Toc457993680)

[2.1 Dependencies 2](#_Toc457993681)

[3. General Operation 2](#_Toc457993682)

[3.1 Starting the Tool 2](#_Toc457993683)

[3.2 Projects 4](#_Toc457993684)

[3.3 Processing 5](#_Toc457993685)

[3.4 Workflow 6](#_Toc457993686)

[3.4.1 JPEG Workflow 6](#_Toc457993687)

[4. Graph Operations 7](#_Toc457993688)

[4.1 Linking Nodes 7](#_Toc457993689)

[4.2 Interface Demonstration 8](#_Toc457993690)

[5. Link Descriptions 9](#_Toc457993691)

[5.1 Applying Filters (Plugins) 11](#_Toc457993692)

[5.2 The Paste/Splice Operation 11](#_Toc457993693)

[5.3 Link Inspection 12](#_Toc457993694)

[6. Other Metadata 12](#_Toc457993695)

[7. Mask Generation 12](#_Toc457993696)

[8. Analytics 13](#_Toc457993697)

[9. Plugins 13](#_Toc457993698)

[9.1 Arguments 14](#_Toc457993699)

[10. Group Manager 14](#_Toc457993700)

[11. Batch operation 14](#_Toc457993701)

[12. JSON 15](#_Toc457993702)

**List of Figures**

[Figure 1: Example of a graph produced by the journaling tool. Specifically, a region of one image (hat) was spliced into another image (orig\_input) to create a new image (input\_mod\_1). 1](#_Toc457993703)

[Figure 2: GitHub website download dialog 1](#_Toc457993704)

[Figure 3: Basic UI. The image nodes open here are from the example project based in the “Images” directory specified in the command line. 3](#_Toc457993705)

[Figure 4: File menu. 4](#_Toc457993706)

[Figure 5: Process menu. 5](#_Toc457993707)

[Figure 6: Description of user interface. 8](#_Toc457993708)

[Figure 7: Link description window. 9](#_Toc457993709)

[Figure 8: Example of an input mask one might use during seam carving to ensure preservation of the woman’s hat. 10](#_Toc457993710)

[Figure 9: Window for applying plugins to image nodes. 11](#_Toc457993711)

[Figure 10: EXIF Comparison Table 12](#_Toc457993712)

# Overview

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



Figure : 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).

# 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 : GitHub website download dialog

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

## Dependencies

There are several Python libraries required for operation:

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

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

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

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

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

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

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

# General Operation

## Starting the UI

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

python src/python/MaskGenUI.py --imagedir *images*

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

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

python src/python/MaskGenUI.py --imagedir *images* --base *images/baseimage.jpg*

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

python src/python/MaskGenUI.py --imagedir *images --*s3 *bucket/path*

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

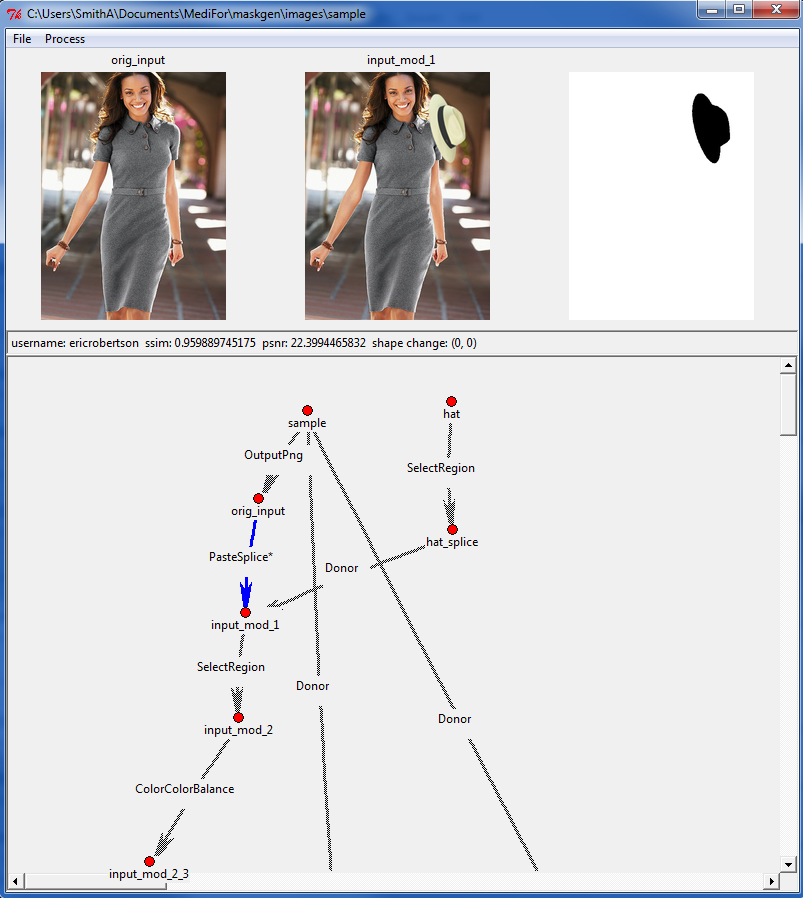


Figure : Basic UI. The image nodes open here are from the example project based in the “Images” directory specified in the command line.

## Projects

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

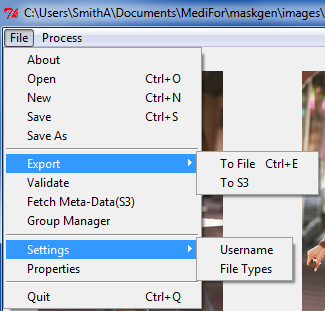


Figure : File menu.

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

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

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

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

**Export – To File**: will create a compressed archive file of the project, including all images and masks.

**Export – 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 ‘/’.

**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.csv and software.csv from an S3 bucket. The user is prompted for the bucket/folderpath, separated by '/'.

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

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

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

## Processing

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

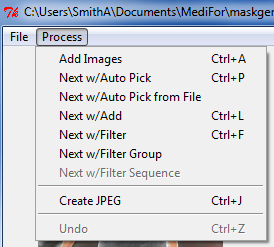


Figure : Process menu.

Several options are available:

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

## Workflow

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

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

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

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

The journaling tool has the ability to automatically select the next modified image in the directory (see section 3.3: Processing). To use this feature, image manipulation steps should be saved out in numerical order (e.g. image\_mod\_1.PNG, image\_mod\_2.PNG, etc).

### JPEG Workflow

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

1. Import the original JPEG image into the tool as the base image.
2. Export the JPEG image as PNG (lossless) and import the PNG image as the first step. This can be done automatically using the SaveAsPNG plugin (skip step 3).
3. Create a link with the operation ‘OutputPNG’ between the two nodes.
4. Perform manipulations as described above.
5. Use the Journaling Tool’s ‘Create JPEG’ 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.

# Graph Operations

## Linking Nodes

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

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

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

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

## Interface Demonstration

Mask Represented By Link

## 

Link Statistics

Image at Start of Selected Link

Currently Selected Link

Final Result

Starting Node

(Shown on Left)

Ending Node

(Shown in Middle)

Base Image

Image at End of Link

Figure : Description of user interface.

# Link Descriptions

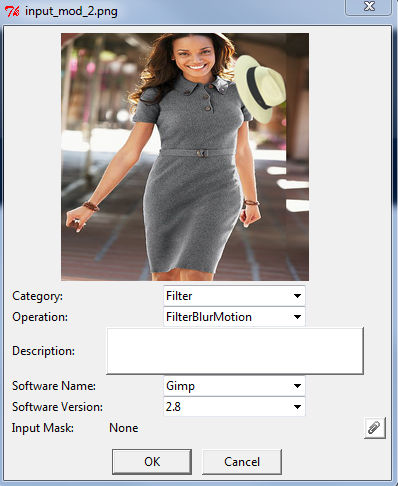
Link descriptions include a category of operations, an operation name, a free-text description (optional), and software with version that performed the manipulation. The category and operation are either derived from the operations.csv file provided at the start of the tool or the plugins. Plugin-based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

Figure : Link description window.

## Input Masks

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



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

## Paste/Clone vs. Paste/Duplicate

Two similar operations are the Paste/Clone and Paste/Duplicate. The key difference between them is that Paste/Duplicate is a direct copy that requires an input mask, while Paste/Clone does not.

Paste/Clone 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. A Paste/Duplicate operation would involve directly copying pixels from one area and transferring them to another area in the same image, using a selection tool. This operation should include an input mask highlighting the area used to create the copy.

## The Paste/Splice 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.

## 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 : Window for applying plugins to image nodes.

## Link Inspection

Regardless if the link is read-only (for links created by plugins) or created through one of the image connection operations, a link may be selected and inspected. The inspection includes all arguments used by the plugin, if applicable, and EXIF comparison analysis, as shown in the figure below.



Figure : EXIF Comparison Table

# Other Metadata

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

# Mask Generation

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

The mask generation algorithm gives special treatment to manipulations that alter the image size. The algorithm first finds most common pixels in the smaller image to match the larger. This is useful in cropping OR framing. Interpolation applied when expanding an image may distort many pixels, causing a full change mask. Smaller manipulated images are produced when cropping or seam cutting is applied. Seam cutting is typically done by finding an optimal cut. Seams can be cut both vertical or horizontally. Seam cutting may be considered as operations of region removal, sliding over the remaining pixels, and cropping. It is expected that each cut is a separate operation. For the tool to recognize seams cuts, two things must be present: an image size change, either vertically or horizontally, and a mismatched region. Do not confuse seam cutting with a splice and crop--two separate manipulations. Although the procedure to create splice involves a cut, paste, move and crop (of the remaining space), the entire effect is detected by a single non-linear line through an image.

When performing manipulations, it is important to consider what is detectable in a modified image. A crop may not detectable, depending on the compression configuration, since the initial image is absent in the analysis. A move manipulation, in itself, resembles an insert. It is acceptable to group manipulations so long as their final result can be represented as one of the accepted singular operations configured with the tool. A pure crop does not produce a mask with identified changes. Thus, it is important to the manipulation operation to understand the operation.

## Composite Mask

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

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

# 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:

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

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

# Plugins

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

1. 'operation()' that returns a list of five items:
2. 'operation name'
3. 'operation category'
4. 'description'
5. 'python package'
6. 'package version'
7. 'transform(im, source, target, \*\*kwargs)' consumes a PIL Image the location of the result file and a set of arguments. The function returns True if the EXIF metadata should be copied from the source to target.
8. ‘arguments()’ returns a list of tuples or None. Each tuple contains an argument name and a default value.

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

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

The tool creates a copy of the source image in a new file. The path (i.e. location) of the new file is provided in the second argument (imgfilename). 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. If the transform returns True, then the tool copies the EXIF from the source image to the new image file. This is often required since PIL(Pillow) Images do not retain all the EXIF data, with the exception of working with TIFF.

## Arguments

There are two special arguments: 'donor' and 'inputmaskpathname'.

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

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

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

# Group Manager

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

# Validation

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

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

# QA Process

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

* 1. Open the journaling tool to the project directory
  2. Run File -> Validate to automatically search for any obvious errors
  3. Make sure the start and end images are the same format (jpg, tiff, etc.)
  4. If JPEG was the base node, the graph should end with two links: AntiForensicExifQuantizationTable and AntiForensicCopyExif. This can be done easily with Process -> Create JPEG.
  5. Right click and select “View Composite” on the final node. Make sure all relevant local changes are represented (particularly copy/paste-type operations)
  6. Make sure all images in the project directory correspond to a link (mask images, input masks), or a node. Extra images from errors should be deleted.
  7. Send the project back to the original manipulator with any feedback.

# Batch operation

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

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

python src/python/batch\_process.py <args>

Mandatory arguments:

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

One (and only one) of these three arguments must be present (see below for an explanation):

--sourceDir <DIR>: directory of images

--plugin <pluginName>: plugin to perform

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

These arguments may be used only if using sourceDir:

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

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

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

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

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

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

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

Optional arguments:

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

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

Different arguments will trigger different functionality:

* + 1. Using both --sourceDir and --endDir will create new projects, using the images in sourceDir as base, and link them with the specified operation. This will also create the project directories and JSON files if necessary.

python src/python/batch\_process.py --projects <DIR> --sourceDir <DIR> --endDir <DIR> --op ColorColorBalance --softwareName GIMP --softwareVersion 2.8

* + 1. Using --sourceDir without --endDir will add the images in the source directory to the current project and link them to the most recent node with the specified operation.

python src/python/batch\_process.py --projects <DIR> --sourceDir <DIR> --op ColorColorBalance --softwareName GIMP --softwareVersion 2.8

* + 1. Use --plugin to specify a plugin to perform on the most recent image node.

python src/python/batch\_process.py --projects <DIR> --plugin ColorEqHist

* + 1. Using --jpg will perform antiforensic jpeg export and exif copy on existing projects.

python src/python/batch\_process.py --projects <DIR> --jpg

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

For example:

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

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

# JSON

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

{

"directed": true,

"graph": {

"igversion": "0.1",

"idcount": 21,

"name": "sample",

"typespref": []

},

"nodes": [],

“links”: [],

"multigraph": false

}

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

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

{

"username": "ericrobertson",

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

"psnr": 29.379891227475664,

"description": "\n",

“shape change": "(0,0)",

"source": 16,

"editable": "yes",

"ssim": 0.9918523088886719,

"softwareName": "OpenCV",

"inputmaskownership": "no",

"softwareVersion": "2.4.13",

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

"inputmaskname": "",

"target": 0,

"op": "FilterBlurMotion",

"argments": {“x”:1},

“exifdiff": {…}

}

Link properties include:

* ***source*** - identifies the source node in the order it appears in the node list
* ***target*** - identifies the target node in the order it appears in the node list
* ***maskname*** - the assigned mask file. It is usually composed with the source and target image node names.
* ***description*** - a user provided description of the operation performed on the source node to create the target. For plugin operations, the description is provided by the plugin.
* ***editable*** - ‘yes’ if the link was not generated by a plugin or internal tool operation
* ***username*** - the name of the user that created the link
* ***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
* ***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]

***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.*