Card Assembler

Release 1.4.0

Martin Brajer

Dec 13, 2020

CONTENTS:

CHAPTER

ONE

CARD ASSEMBLER OVERVIEW

Card Assembler is a free and open-source plug-in for Gimp. Assembles an image according to an XML file blueprint. Meant for board game card creation.

Source code can be found on Github.

1.1 Features

- Gimp tools: single-color layer, textbox, import layers from a given XCF file etc.
- Load card layout information from an XML file blueprint.
- Command chaining, making templates possible.
- Export colors used in blueprint to Gimp as a palette.

1.2 Installation

Copy all PY files from `src/` into `card-assembler/` in your Gimp plug-in folder: `*user*/AppData/Roaming/GIMP/*version*/plug-ins/`. Then **restart Gimp**.

1.3 How to use this plug in

There is a new item in the menu bar - *Card Assembler* containing:

1. *Card Assembler*: Create board-game cards.
   - *Data Folder*: Data files location. Mainly the following xml file.
   - *XML file*: Name of the blueprint (XML) you want to use.
   - *CardIDs*: Path to the starting node within the blueprint.
     – Omit the root node. Steps are separated by spaces.
     – Write each CardID entry on a separate line.
     – Add “keepCmdOpen” as one of the IDs to keep Gimp’s cmd open.
   - *Save*: Save the image into a data folder subfolder as `*image name*.xcf`.
2. *Palette creator*: Export colors used in a blueprint to Gimp palette.

- *Data Folder*, *XML file*: Same as above.
- *PaletteID*: Path to the colors starting node (assuming there is a special color subtree).
- *Name*: The new palette's name.

1.4 Compliance

Versioning follows Semantic Versioning 2.0.0.
Following PEP8 Style Guide coding conventions.
Testing with `unittest` and pycodestyle.

1.5 License

Card Assembler is licensed under the MIT license.

CHAPTER

TWO

CREATE A BLUEPRINT

Card Assembler works with an XML tree structure. When calling the plug-in in Gimp, you specify which node to start at. Its children are then alphabetically called to formulate respective layer builders.

Warning: Each layer must include `layer_type` tag! Accepted values are listed in section *Layer types*.

Note: Term *"layer"* is used a bit vaguely here. Among real layers (e.g. background, imported image) are also other commands: selection, load a file etc.

2.1 Parsing

If a parameter `type` is not `str`, add attribute `parse` to the parameter's tag with value `int`, `float` or `tuple` as needed. For example:

```
<position parse="tuple">100, 125</position>
```

2.2 Newline

To get multiline text, use `\n` or input `text` tag multiple times.

```
<text>First line\nSecond line</text>
<text>Third line</text>
```

2.3 Nesting

Any node can contain any number of `next` tags. Intentional usage is inserting card specific tags first and then let `next` point to some more general version (i.e. template).

Properties of `next` tag:

- points to another node in the data tree whose children are appended to the initial node
- can only add new tags, not rewrite the ones already in place
- is executed at the end of a layer definition regardless o its position

- relative order to other `next` tags is retained in the execution

It's recommended to place the tag at the beginning of a layer definition for easier readability.

It is useful to create separate color dedicated subtree. It can be later exported into Gimp as a palette.

```
<card>
    <command02_background>
        <next>color black</next>
        <layer_type>monochrome</layer_type>
        <size parse="tuple">800, 500</size>
    </command02_background>
</card>
...
<color>
    <black>#ffffff</black>
</color>
```

2.4 Examples

There are two example blueprints in examples folder. The first one provides a minimal blueprint to start with. The second one shows intended use.

- `Minimal blueprint.xml`

  To assemble the this card, use "example" as **CardIDs**. There are three commands in the file:

  1. Blank canvas.
  2. White background.
  3. Text "Example 1" in the middle.

- `Blueprint using a template.xml` + `Data image.xcf`

  To assemble the this card, use "unique spell soothingWinds" as **CardIDs**.

  The first tag (under the mentioned path) refers to the template for all spell cards, where universal details (such as title position) are specified. The main part continues by adding card specific details to the template-defined layers. Try to follow all `next` tags to discover its full structure.

  Take notice of the last part tagged `color`. Using this definition, colors can be exported by filling "color" into **PaletteID**.

CHAPTER

THREE

LAYER TYPES

All layer nodes can use `next` tag, see *Creating blueprint > Nesting*. In the following list of layer types parameters are formatted as follows:

- **name** (`type`, `default if optional`) Description

Note: Setting `add_to_position = -1` adds the layer to a recently defined group.

3.1 image

Create new image. Needed for any layer creation.

- **size** (`tuple`) Image dimensions in pixels note
- **name** (`str`, `Card Assembler Image`) Image name

3.2 monochrome

Single color filled layer.

- **size** (`tuple`) Layer dimensions in pixels
- **color** (`str`) Layer color in hex code
- **name** (`str`, `Monochrome`): Layer name
- **position** (`tuple`, `0, 0`)
- **add_to_position** (`int`, `0`) Position among layers (`-1` for *group*)

3.3 import_layer_load

Load new data image. The file has to be in the data folder. Filename is specified in the XML file. Name parameter is used in the XML file to reference the imported file. That's why it's not an optional parameter.

- **filename** (`str`) See description
- **name** (`str`) Name the data for future use by *import_layer*

3.4 import_layer

Copy layer from a data image.

- **target_file** (`str`) Use **name** filled in *import_layer_load*
- **target_layer** (`str`) Name of the layer to be imported in the target file
- **add_to_position** (`int`, `0`) Position among layers (`-1` for *group*)
- **name** (`str`, **target_layer**) Layer name
- **position** (`tuple`, `0, 0`)

3.5 group

Create new layer group. To fill next layers in, set theirs **add_to_position** parameter to `-1`.

- **add_to_position** (`int`, `0`) Position among layers (`-1` for *group*)
- **name** (`str`, `Group`) Group name

3.6 text

Text layer.

- **text** (`str`) Text
- **font** (`str`) Font name
- **font_size** (`int`) Font size
- **font_scale** (`float`, `1`) Multiply **font_size**
- **add_to_position** (`int`, `0`) Position among layers (`-1` for *group*)
- **name** (`str`, Gimp default = `Text Layer`) Layer name
- **color** (`str`, `#000000`) Text color in hex code
- **size** (`tuple`, autosize) Layer dimensions in pixels
- **line_spacing** (`float`, `0`) Line separation change
- **letter_spacing** (`float`, `0`) Letters separation change
- **justification** (`int`, `0`) Either left (`0`), right (`1`), center (`2`) or fill (`3`)
- **position** (`tuple`, `0, 0`)

3.7 select

New selection by percentage of image size.

- **mode** (`str`, `select`) Either `select`, `select_invert` or `deselect`
- **left** (`float`, `0`) Left edge position in percentage of the image size
- **right** (`float`, `100`) Right edge position in percentage of the image size
- **top** (`float`, `0`) Top edge position in percentage of the image size
- **botton** (`float`, `100`) Bottom edge position in percentage of the image size

3.8 mask

Mask layer. Create a mask for the given layer from the given selection.

- **target_layer** (`str`) Layer to be masked
- other parameters are passed to *select*

3.9 hide

Ignore command. Used for overrides, i.e. hiding a predefined (template) layer.

CHAPTER

FOUR

LIBRARY

4.1 cardassembler module

Main script which handles communication with Gimp.

All code using gimpfu is here. The rest can be found in `blueprint` module. Written for Gimp 2.10.18 which uses Python 2.7.18.

Probably: Gimp runs this script through `eval()` function inside its installation folder and direct import from different folder raises *access denied error*. See `_run_code()` function in `Gimp installation folder/lib/python2.7/runpy.py`.

`cardassembler.`**`card_creator`**(*data_folder*, *xml_file*, *card_IDs*, *save*)

Create board-game cards.

Registered function by `gimpfu.register()`. Main plugin functionality. Add "keepCmdOpen" among **cardIDs** to keep the cmd window open.

Parameters

- **`data_folder`** (*`str`*) – Blueprints (XML) and data images (XCF) folder
- **`xml_file`** (*`str`*) – Blueprint to be used (with extension)
- **`card_IDs`** (*`str`*) – Newline-separated paths to starting nodes.
- **`save`** (*`bool`*) – Save the images after generation

Raises `ValueError` – If cardIDs are empty.

`cardassembler.`**`palette_creator`**(*data_folder*, *xml_file*, *palette_ID*, *name*)

Create palette.

Registered function by `gimpfu.register()`. Supplemental plugin functionality.

Parameters

- **`data_folder`** (*`str`*) – Blueprints (XML) folder
- **`xml_file`** (*`str`*) – Blueprint to be used (with extension)
- **`palette_ID`** (*`str`*) – Path to the starting node.
- **`name`** (*`str`*) – Name of the created palette

Raises `ValueError` – If the paletteID is empty.

`class` `cardassembler.`**`Toolbox`**(*data_folder*, *xml_file*)

Blueprint-to-image manipulation tool.

This class offers means for creating common card components (i.e. text, icons). Then completes the image and optionally saves it. Probably you'll want to fine-tune the image manually.

Parameters

- **`data_folder`** (*`str`*) – Blueprints (XML) and data images (XCF) folder
- **`xml_file`** (*`str`*) – Blueprint to be used (with extension)

`create_image`(*card_ID*)

Blueprint to image.

Layout consists of commands which are called alphabetically by name.

Parameters **`card_ID`** (*`str`*) – Path to the starting node.

Raises

- **`RuntimeError`** – If there is no blueprint
- **`KeyError`** – If any of the layers has no type
- **`ValueError`** – If any of the layers has unknown type

`_layer_image`(*size*, *name='Card Assembler Image'*, ***kwargs*)

Create new image. Needed for layer creation.

Parameters

- **`size`** (*`tuple`*) – Image dimensions in pixels
- **`name`** (*`str`*) – Image name, defaults to "Card Assembler Image"

`_layer_monochrome`(*size*, *color*, *name='Monochrome'*, *position=(0, 0)*, *add_to_position=0*, ***kwargs*)

Single color filled layer.

Parameters

- **`size`** (*`tuple`*) – Layer dimensions in pixels
- **`color`** (*`str`*) – Layer color in hex code
- **`name`** (*`str, optional`*) – Layer name, defaults to "Monochrome"
- **`position`** (*`tuple, optional`*) – Defaults to (0, 0)
- **`add_to_position`** (*`int, optional`*) – Position among layers (-1 adds the layer to a recently defined group), defaults to 0

Raises **`RuntimeError`** – If there is no image

`_layer_import_layer_load`(*filename*, *name*, ***kwargs*)

Load new data image.

The file has to be in the data folder. Filename is specified in the XML file. Name parameter is used in the XML file to reference the imported file. That's why it's not an optional parameter.

Parameters

- **`filename`** (*`str`*) – See description
- **`name`** (*`str`*) – Name the data for future use by `import_layer`

Raises **`RuntimeError`** – If there is no image

`_layer_import_layer`(*target_file*, *target_layer*, *add_to_position=0*, *name=None*, *position=(0, 0)*, ***kwargs*)

Copy layer from a data image.

Parameters

- **`target_file`** (*`str`*) – Use **name** filled in `import_layer_load`
- **`target_layer`** (*`str`*) – Name of the layer to be imported in the target file
- **`add_to_position`** (*`int, optional`*) – Position among layers (-1 adds the layer to a recently defined group), defaults to 0
- **`name`** (*`str or None, optional`*) – Layer name, defaults to **target_layer**
- **`position`** (*`tuple, optional`*) – Defaults to (0, 0)

Raises **`RuntimeError`** – If there is no image

`_layer_group`(*add_to_position=0*, *name='Group'*, ***kwargs*)
Create new layer group.

To fill next layers in, set theirs **add_to_position** parameter to `-1`.

Parameters

- **`add_to_position`** (*`int, optional`*) – Position among layers (-1 adds the layer to a recently defined group), defaults to 0
- **`name`** (*`str, optional`*) – Group name, defaults to "Group"

Raises **`RuntimeError`** – If there is no image

`_layer_text`(*text*, *font*, *font_size*, *font_scale=1*, *add_to_position=0*, *name=None*, *color='#000000'*, *size=None*, *line_spacing=0*, *letter_spacing=0*, *justification=0*, *position=(0, 0)*, ***kwargs*)
Text layer.

Parameters

- **`text`** (*`str`*) – Text
- **`font`** (*`str`*) – Font name
- **`font_size`** (*`int`*) – Font size
- **`font_scale`** (*`float, optional`*) – Multiply **font_size**, defaults to 1
- **`add_to_position`** (*`int, optional`*) – Position among layers (-1 adds the layer to a recently defined group), defaults to 0
- **`name`** (*`str or None, optional`*) – Layer name, defaults to None (Gimp default)
- **`color`** (*`str, optional`*) – Text color in hex code, defaults to "#000000"
- **`size`** (*`tuple or None`*) – Layer dimensions in pixels, defaults to None (autosize)
- **`line_spacing`** (*`float, optional`*) – Line separation change, defaults to 0
- **`letter_spacing`** (*`float, optional`*) – Letters separation change, defaults to 0
- **`justification`** (*`int, optional`*) – Either left(0), right(1), center(2) or fill(3), defaults to 0
- **`position`** (*`tuple, optional`*) – Defaults to (0, 0)

Raises **`RuntimeError`** – If there is no image

`_layer_select`(*mode='select'*, *left=0*, *right=100*, *top=0*, *bottom=100*, ***kwargs*)
New selection by percentage of image size.

Parameters

- **mode** (*str*) – Either “select”, “select_invert” or “deselect”, defaults to “select”
- **left** (*float, optional*) – Left edge position in percentage of the image size, defaults to 0
- **right** (*float, optional*) – Right edge position in percentage of the image size, defaults to 100
- **top** (*float, optional*) – Top edge position in percentage of the image size, defaults to 0
- **botton** – Bottom edge position in percentage of the image size, defaults to 100

Raises

- **RuntimeError** – If there is no image
- **ArithmeticError** – If width is not positive
- **ArithmeticError** – If height is not positive
- **ValueError** – If mode is unknown

_layer_mask(*target_layer*, ***kwargs*)

Mask layer.

Create a mask for the given layer from the given selection.

Parameters

- **target_layer** (*str*) – Layer to be masked
- **kwargs** (*various, optional*) – Additional named arguments are passed to `select`

_layer_hide(***kwargs*)

Ignore command.

Used for overrides, i.e. hiding a predefined (template) layer.

save_image()

Save the image.

Filemane: **image.name**.xcf into folder `saveDirectory` (subfolder of `dataFolder`).

create_palette(*palette_ID*, *name*)

Blueprint to palette.

Colors are sorted by their branch hight and then alphabetically.

Parameters

- **palette_ID** (*str*) – Path to the starting node.
- **name** (*str*) – Created palette name

4.2 blueprint module

Supplemental script which handles data manipulation.

Read an XML file and produce a layout list, which is then used by the main script *cardassembler*.

Note: Run this script directly to run a `unittest`.

class blueprint.**Blueprint**(*file_path*)

Blueprint information handling class.

Can read XML file, produce layout list and palette.

Parameters file_path (*str or None*) – Folder containing XML blueprint

SPECIAL_TAGS = ['next', 'text']

_load(*file_path*)

Load XML file blueprint into a dictionary tree.

Parameters file_path (*str*) – Path to the XML file to load

Returns Tree structure of cards data

Return type dict

_ElementTree_to_dict(*parent*)

Translation from `xml.etree.ElementTree` to `dict` tree from the given node down.

Tags in *SPECIAL_TAGS* are stored in a `list`.

Parameters parent (`ElementTree.Element`) – A node of ElementTree

Returns Dictionary representation of the given tree

Return type dict

_parse(*text, target_type*)

ElementTree.element.text to various python types.

Input parsed as tuple can have any length. Its elements will be parsed as `int`

Parameters

- **text** (*str*) – Text to be parsed
- **target_type** (*str*) – Either "int", "float" or "tuple"

Raises ValueError – If target type is not known

Returns Parsed value

Return type int or float or tuple

generate_layout(*start_by*)

Generate card layout given starting position.

Starting position children are sorted alphabetically (name them accordingly).

Parameters start_by (*str*) – Space separated path through data tree leading to the starting node

Returns Layout of the chosen card

Return type dict

`_step_in`(*layout*, *this_step*)

Browse data guided by the `next` tag.

Do not overwrite (first in stays). Further `next` tags are served successively in the order according to the XML file. If there are multiple `text` tags, join them by `\n`

Parameters

- **`layout`** (*`dict`*) – Input layout
- **`this_step`** (*`str`*) – Where does this step leads

Returns Filled layout

Return type dict

`_goto`(*next_steps*)

Find target dict tree node and return its sub tree.

Analogous to successive application of `dict.get()`.

Parameters **`next_steps`** (*`str`*) – Space separated key sequence.

Raises **`KeyError`** – If one of the given keys doesn't exist.

Returns Sub-tree of the `self.data` dict tree.

Return type dict

`generate_palette`(*start_by*)

Make palette out of colors used by cards.

To be used in another mini plug-in to import palette into Gimp.

Parameters **`start_by`** (*`str`*) – Path through the data tree (space separated)

Returns Pairs of name and color

Return type list

`_harvest_leaves`(*color_tree*)

Find the path to the leaves of the given tree, whose tag is `color`.

Kinda inverse to *`_goto()`*.

Parameters **`color_tree`** (*`dict`*) – Part of the data (dict tree) to look for colors in

Returns List colors as `tuple` of space delimited path and color code

Return type list

CHAPTER FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX