Koala Documentation

Release 0.1alpha1

Chintalagiri Shashank

January 28, 2015

Part I

User Documentation

CHAPTER

ONE

INSTALLATION

For the moment, koala is not designed to be installed to the system (/usr, /opt, etc). Such installation scripts may be developed for later versions.

Setting up Koala is essentially a matter of cloning/checking out the approriate git/svn repository and providing the software with a suitable environment which contains all the necessary dependencies. The instructions listed here are for Ubuntu, and should work as-is on most Debian based Linux distributions.

The use of `pyenv` and `virtualenv` for this is recommended but not necessary. The use of these tools should create a reasonable stable environment until any cross-version kinks are found and worked out.

Compatibility of the scripts has not been tested with any python version other than the `Python 2.7.6` which comes with `Ubuntu 14.04.1 LTS Trusty Tahr`. They will almost certainly not work with `Python 3.x` in their current form and may or may not work with older point realeases of `Python 2.7`. While they may work, it's probably a good idea to avoid using Python versions < 2.7 with these scripts unit a full set of unit tests can be prepared and run.

1.1 Setting up pyenv

`pyenv` is needed to easily set up multiple python versions on your computer.

See http://davebehnke.com/python-pyenv-ubuntu.html for a more detailed explanation.

1. Install Git:

   ```
   sudo apt-get install git-core curl
   ```

3. Setup proxy, if any:

   ```
   export http_proxy=http://user:pass@192.168.1.254:3128
   export https_proxy=http://user:pass@192.168.1.254:3128
   export ftp_proxy=http://user:pass@192.168.1.254:3128
   ```

4. Tell `git` to use `https://` instead of `git://` to get around proxy issues:

   ```
   git config --global url."https://".insteadOf git://
   ```

5. Run the installer:

   ```
   curl -L \
   https://raw.githubusercontent.com/yyuu/pyenv-installer/master/bin/pyenv-installer \
   | bash
   ```

6. Insert the following at the end of `~/.bashrc`:

```
export PYENV_ROOT="${HOME}/.pyenv"
if [ -d "${PYENV_ROOT}" ]; then
    export PATH="${PYENV_ROOT}/bin:${PATH}"
    eval "$(pyenv init -)"
fi
```

7. **Install Build Dependencies for Python 2.7:**

```
sudo apt-get build-dep python2.7
sudo apt-get install build-essential wget \
    libreadline-dev libncurses5-dev libssl1.0.0 tk8.5-dev \
    zlib1g-dev liblzma-dev
```

8. Install Python 2.7.6:

   Python 2.7.x, where x>=6, should be fine. x<6 is untested. New features were intruduced in 2.7.5, 2.7.6 that may be necessary for the scripts to run. If system python is 2.7.6 or better, `pyenv` isn't strictly necessary. However, to standardize the environment in the absense of cross-version testing and intelligent installation scripts, the use of a version-specified python version (as opposed to `system`) is recommended.

   ```
   pyenv install 2.7.6
   ```

1.2 Getting the Code

The code can be obtained from the version control system. For users, the specific instance of `koala` applicable to the organization should be checked out from the locally controlled repository. This repository should be essentially `read-only` with a specific set of people administering the installation. Until the details can be worked out, use the following checkouts:

- Users:

  ```
  svn co svn://svnserver.qznet/koala
  ```

  Access control isn't set up. Please don't commit to this unless absolutely necessary, and even then run it by an administrator first.

- Administrators:

  ```
  svn co svn://svnserver.qznet/koala
  ```

  Access control isn't set up. Commits should generally be limited to instance-specific areas.

- Developers:

  ```
  git clone (something)
  ```

1.3 Setting up virtualenv

See http://simononsoftware.com/virtualenv-tutorial-part-2/ for a more detailed explanation.

1. Install `virtualenv` from the standard repository.

   ```
   sudo aptitude install python-virtualenv virtualenvwrapper
   ```

2. Create a directory for the virtual environments.

```
mkdir ~/.virtualenvs
```

3. Tell virtualenvwrapper where the folder you just created is. Put it into the bashrc so that you don't have to do it every time you restart.

   ```
   echo "export WORKON_HOME=$HOME/.virtualenvs" >> ~/.bashrc
   ```

   Start up a fresh shell.

4. Create a new `virtualenv` with the correct interpreter version. Don't use system packages.

   If `pyenv` is controlling the python version,

   ```
   cd /path/to/koala/checkout/trunk/
   mkvirtualenv -p `pyenv which python` --no-site-packages koala
   ```

   If you're just using `system` python,

   ```
   mkvirtualenv --no-site-packages koala
   ```

5. `mkvirtualenv` leaves you with the new virtualenv active. To deactivate,

   ```
   deactivate
   ```

   To reactivate the virtualenv, which you should do when running the scripts in a new terminal:

   ```
   workon koala
   ```

1.4 Installing the Dependencies

1. Install required python libraries (virtualenv should be active):

   ```
   cd /path/to/koala/checkout/trunk/
   pip install -r requirements.txt
   ```

2. Install `sofficehelpers`:

   `sofficehelpers` is a collection of scripts to deal with `libreoffice` documents. As of this version, this only includes a small script (ssconvertor) to convert a spreadsheet into csv files. The libreoffice python interface (`uno`) requires the use of the python bundled into libreoffice, and therefore is kept separate from the rest of koala. There are plenty of other (and simpler) ways to achieve the same effect, inculding a number of uno-based scripts to do this. The custom script is retained for the moment to maintain a functional base upon which additional functionality can be added on as needed. If another solution is to be used instead, appropriate changes should be made to `utils.libreoffice.XLFile._make_csv_files()` and `utils.libreoffice.XLFile._parse_sscout()`.

   (a) Install dependencies:

   ```
   sudo apt-get install python-uno
   ```

   (b) Determine `libreoffice` `python` version:

   TBD. Until then, refer to https://code.google.com/p/slidespeech/wiki/FindingLibreOfficePython

   (c) Checkout the `sofficehelpers` code:

   ```
   svn co svn://svnserver.qznet/scripts/libreoffice
   ```

(d) Navigate to the trunk folder of the obtained repository in a terminal.

(e) Install to system using:

```
sudo python setup.py install
```

3. To be able to generate the documentation, also install the following:

   TBD

CHAPTER

TWO

CONVENTIONS

2.1 gEDA Symbol Conventions

gEDA Symbols contain the primary data used by the scripts. As such, these symbols need to contain the necessary attributes in the correct format.

The conventions listed here are in addition to the standard guidelines and conventions listed on the gEDA wiki at gEDA/gaf Symbol Creation. In the few instances where the conventions listed here conflict with the standard guidelines, the conventions listed here should take precedence.

Some attributes listed here are not standard gEDA/gschem attributes, and as such will cause errors with `gsymcheck`. This is, unfortunately, unavoidable without introducing overly unwieldy annotation schemes.

2.1.1 Symbol Attributes

refdes As per standard gEDA usage.

device This is used as a device class field, with a restricted set of allowed values. See Device Classes.

value The content of this field is Device Class dependent. See Values for details.

footprint As per standard gEDA usage. This should ideally be a standard footprint name, though at present the scripts don't care.

description `(Optional)` Description string, usually the title of the datasheet.

symversion As per standard gEDA usage.

status `(Optional)` Defines the status of the symbol. The absence of the attribute entirely is interpreted as `Active`. See Symbol Status for more information.

fillstatus `(Optional)` Defines if a component is soldered or not. The symbol should not contain this attribute. The attribute should be added as needed in the schematic. `DNP` means Do Not Populate. Absence of the attribute altogether indicate normal usage.

group `(Optional)` Defines the group of components in the schematic in which the component is included. The symbol should not contain this attribute. The attribute should be added as needed in the schematic. The absence of the attribute entirely is interpreted as the component being in the `default` group. See Component Groups for an introduction to component groups.

package `(Optional)` Defines the package of the component. This should be the JEDEC package nomenclature for the component. This is intended to be used for 3D model generation.

2.1.2 Device Classes

The following are the allowed values for the Device Attributes. More should be added as needed.

RES SMD : SMD Resistors.

RES THRU : THRU Resistors.

RES POWER : Off-PCB power resistors for direct mounting onto heatsinks.

RES ARRAY THRU : THRU Resistor Arrays.

RES ARRAY SMD : SMD Resistor Arrays.

POT TRIM : Trimpots.

VARISTOR : Varistors and MOVs.

CAP CER SMD : SMD Ceramic Capacitors.

CAP TANT SMD : SMD Tantalum Capacitors.

CAP CER THRU : THRU Ceramic Capacitors.

CAP ELEC THRU : THRU Electrolytic Capacitors.

CAP POLY THRU : THRU Poly Capacitors.

CAP PAPER THRU : THRU Paper Capacitors.

INDUCTOR SMD : SMD Inductors.

INDUCTOR THRU : THRU Inductors.

FERRITE BEAD SMD : SMD Ferrite Beads.

TRANSFORMER HEAVY : Transformers.

TRANSFORMER SMD : SMD Transformers.

DIODE SMD : SMD Diodes.

DIODE THRU : THRU Diodes.

ZENER SMD : SMD Zener Diodes.

ZENER THRU : THRU Zener Diodes.

TRIAC : Triacs.

LED SMD : SMD LEDs.

LED THRU : THRU LEDs.

LED MODULE : LED Modules.

BRIDGE RECTIFIER : Bridge Rectifiers.

CRYSTAL AT : AT cut Crystals.

CRYSTAL TF : Tuning Fork Crystals.

CRYSTAL OSC : Integrated Crystal Oscillators.

TRANSISTOR THRU : THRU Transistors including MOSFETs.

TRANSISTOR SMD : SMD Transistors including MOSFETs.

IC THRU : THRU Hole ICs.

IC SMD : SMD ICs.

IC PLCC : PLCC ICs, separated because of their need for a socket.

IC POWER : Off-PCB power ICs for direct mounting onto heatsinks.

SOCKET STRIP : SIP sockets.

SOCKET DIP : IC sockets and bases.

RELAY : Relays.

MODULE : Modules.

PCB : Printed Circuit Board.

BUZZER : Buzzers.

CONN BANANA : Banana Connectors.

CONN BERG STRIP : Berg Strips.

CONN TERMINAL BLOCK : Terminal Blocks. Usually single-part.

CONN TERMINAL : Terminal Connectors. Usually two-part.

CONN DTYPE : DTYPE Connectors.

CONN INTERBOARD : Stackthrough Headers.

CONN FRC : FRC Connectors.

CONN MINIDIN : MiniDIN Connectors.

CONN MOLEX : Molex Connector.

CONN MOLEX MINIFIT : Molex Minifit Male (PCB Mount) connectors.

CONN BARREL : DC Power Jacks and similar barrel connectors.

CONN SIP : SIP connectors Male (PCB Mount).

CONN STEREO : Stereo Connectors.

CONN DF13 : Hirose DF13 Connectors.

CONN MODULAR : Modular Connectors.

SWITCH TACT : Tactile Switches.

SWITCH PUSHBUTTON : Pushbutton Switches.

TESTPOINT : Testpoints.

SOLDER DOT : Solder Dots.

2.1.3 Values

General

For the general case, value should include the manufacturer part number. The part number should be the minimal string necessary to uniquely locate components with all paramenters of interest, including Grade, Package, etc.

Unless otherwise specified, canonical representation for each class is constructed as `DEVICE VALUE FOOTPRINT`.

Resistors

- Applies to `RES SMD`, `RES THRU`, `RES POWER`, `RES ARRAY THRU`, `RES ARRAY SMD`, `POT TRIM`.
- The value contains the actual resistance value in a standard form.
- Order specifiers to be used are m, E, K, M, G. The `Ohm` symbol is excluded.
- The numerical part of the value should be greater than 1 (820E instead of 0.82K)
- For special cases, the full manufacturer part number can be used in place of the reistance value.
- Wattage can optionally (preferably) be specified within value, separated from the resistance value with a `/`.
- Tolerance, Temperature Coefficient, etc. can also be added similarly to Wattage if needed. If so, the conventions should be amended to reflect the correct order as well as code modifications to any relevent Script Dependencies.
- No spaces should be used.

Examples for Resistor Values :

- 10m/1W
- 10E/0.25W
- 10K/1W
- 10M/0.125W
- 10G/0.25W
- 8K2
- 8.2K (prefered)
- PTF561K0000BZEB

Capacitors

- Applies to `CAP CER SMD`, `CAP TANT SMD`, `CAP CER THRU`, `CAP ELEC THRU`, `CAP POLY THRU`, `CAP PAPER THRU`.
- The value contains the actual capacitance value in a standard form.
- Order specifiers to be used are p, n, u. The `F` symbol is included. (`pF, nF, uF`)
- The numerical part of the value should be greater than 1 (100nF instead of 0.1uF)
- For special cases, the full manufacturer part number can be used in place of the capacitance value.
- Voltage can optionally (preferably) be specified within value, separated from the capacitance value with a `/`. This voltage is interpreted as the minimum voltage necessary.
- If the `Voltage` is not specified, the voltage is assumed to be the `stdvoltage` parameter in the generator file, if any.
- For now, the `Voltage` should be specified to what is to be purchased (and not the minimum required).
- Tolerance, Temperature, etc. can also be added similarly to Voltage if needed. If so, the conventions should be amended to reflect the correct order as well as code modifications to any relevent Script Dependencies.
- No spaces should be used.

Examples for Capacitor Values :

- 100nF/50V

- 10uF/25V
- 2.2uF/10V
- 100nF
- 4700uF/63V

Standard Voltages :

CAP CER SMD 0805 : 100V

CAP TANT SMD TANT B : 25V

CAP TANT SMD TANT D : 25V

Diodes

- Applies to `DIODE THRU`, `DIODE SMD`, `ZENER THRU`, `ZENER SMD`, `LED THRU`, `LED MODULE`, `BRIDGE RECTIFIER`.
- The value contains the standard part number as far as possible.
- For LEDs, the value contains the Color. The size is determined by the footprint.
- LED Modules and other special LEDs have the necessary details in the value.
- Diodes not derived from standard part numbers should be manually handled in transform and map files.

Examples for Diode Idents :

- DIODE THRU 1N4007 ALF400-120
- DIODE THRU 1N5402 ALF600-200
- LED THRU RED LED3
- DIODE SMD LL4148 1206P
- BRIDGE RECTIFIER MB6S TO269AA
- ZENER SMD AZ23C3V6-7-F SOT23
- DIODE SMD PGB102ST23 SOT23

Inductors

- Applies to `INDUCTOR SMD`, `INDUCTOR THRU`.
- Given the complexity of Inductor specifications and sourcing, Inductor values should be full manufacturer part numbers.
- For low-end inductors locally obtained, the value attribute can contain the inductance value.
- Order specifiers to be used are n, u, m, with the *H* symbol included (`nH, uH, mH`)
- Additional specifications can be added by using */*. Spaces should be avoided.
- Further guidelines should be developed if inductors are used often.

Crystals

- Applies to `CRYSTAL AT`, `CRYSTAL TF`, `CRYSTAL OSC`.
- `VALUE` should contain the frequency of the crystal along with units. No spaces.
- For special cases, `VALUE` can be the full manufacturer part number.

Examples for Crystal Values:

- 11.0592MHz
- 16MHz
- 32.768KHz

Connectors

- `DEVICE` contains the connector family name as listed previously.
- `VALUE` contains the number of contacts, gender, direction (ST/RA), and any other parameters that may exist.
- `VALUE` can include spaces. However, every symbol for connectors of the same family should have a consistant structure.
- For highly specialized connectors, the `VALUE` attribute contains the manufacturer part number.
- `FOOTPRINT` almost always duplicates the information present in `DEVICE` and `VALUE`, and is therefore excluded from the ident string.

Constructors for Connector Idents:

- CONN INTERBOARD; ESQ-104-12-G-D
- CONN BERG STRIP; `2x05PIN 2R [ST/RA] [L?]`
- CONN BERG STRIP; `10PIN 1R [ST/RA] [L?]`
- CONN FRC; `10PIN [PM/CM] [ST/RA] [NL/WL]`
- CONN SIP; `10PIN [PM/CM] [ST/RA]`
- CONN DTYPE; `DB25 [PM/CM/WM] [ST/RA] [M/F]`
- CONN MOLEX MINIFIT; `10PIN [1R/2R] [M/F] [ST/RA]`
- CONN MOLEX; `04PIN PM RA`
- CONN TERMINAL; `02PIN [PM/CM] [ST/RA]`
- CONN TERMINAL BLOCK; `02PIN [ST/RA/ANG]`
- CONN MINIDIN; `04PIN PM [ST/RA]`
- CONN MODULAR; SS-60000-009
- CONN DF13; DF13A-5P-1.25H
- CONN BARREL; 2.1MM PM RA
- CONN STEREO; 6.3MM PM RA
- CONN THC; PCC-SMP-K-R
- CONN USB; B RA PM THRU
- CONN USB; mB RA PM SMD

2.1.4 Component Groups

HM

2.1.5 Symbol Status

Symbol status determines how the symbol is handled by the scripts. The `STATUS` attribute, if any, should be within the symbol and not added to the schematic. Within the schematic, the `STATUS` attribute should be visible or should be removed, depending on what the status is. (Details Follow). `STATUS` is, in some sense, an outer-loop version of gEDA's `symversion` attribute.

Allowed Status values:

Active : If the `STATUS` attribute is `Active` or does not exist, then the scripts treat the symbol as `Active`. This means the component is acceptable for normal use, and someone in the Company knows the details of procurement and usage of the component.

Experimental If the `STATUS` is `Experimental`, this means that the component is being considered for use. However, care should be taken because the symbol and footprint are likely untested, the component's sourcing details may not be finalized, so on.

Deprecated If the `STATUS` is `Deprecated`, this means a decision has been made to completely phase out use of this component. During redesign of any production PCB, the use of any `Deprecated` components should be looked at and removed if possible. Converting a `Deprecated` component back into `Active` use should involve a specific discussion of the relative merits. Under no circumstances should an Obsolete component be `Active`.

Generator The `STATUS` of `Generator` is a special case, indicating that the component represented by the symbol is not necessarily a real component. If such a symbol has a `VALUE` attribute, then the `VALUE` is the default value for the component and should be valid. The `VALUE` attribute of the symbol should be promoted to the schematic and set appropriately (or created if it does not exist in the symbol). Once the `VALUE` attribute is set, the `STATUS=Generator` attribute should be removed from the component in the schematic. The `VALUE` attribute of any symbol whose `STATUS` is not `Generator` should never be promoted / edited in the schematic under any circumstances.

2.1.6 Attribute Promotion

HM

2.1.7 Script Dependencies

At present, the scripts only depend on a subset of the full allowed range of attribute strings. For the sake of consistency, quality control, and painless additions of features to the scripts, the strings should follow the guidelines listed in this document. The actual requirements are listed here for information and to assist in a gradual migration plan.

`conventions.electronics`

Most of the strings listed here are defined in this module, along with string dependent functions.

`conventions.electronics.eln_ident_transform()` :

If the device string starts with any of the following, it's ident constructor leaves out the footprint.

- `CONN`

- MODULE
- CRYSTAL 4PIN

sourcing.electronics

IC If the device string begins with IC, the value is assumed to be a reasonably complete Manufacturer Part Number.

sourcing.digikey

Description

sourcing.digikey._search_preprocess():

Description

sourcing.digikey._get_device_catstrings():

Description

sourcing.digikey._tf_resistance_to_canonical():

Description

2.2 gEDA Project Folder Structure

The scripts assume a very specific folder structure for gEDA projects, including a set of minimally required files. It is strongly recommended that the project folders be laid out exactly as defined here to ensure comaptibilty with present and future functionality.

2.2.1 gEDA Tools

The minimal set of tools which the user should expect to interact with are :

- gschem, operating on schematics.
- refdes_renum, operating on schematics.
- gsch2pcb, operating on schematics to create / update the pcb.
- pcb, operating on pcb.

The gedaif package of koala contains scripts that use these and other gEDA tools to perform various functions, and assume a specific file and folder layout. Any deviation from the specified layout is likely to result in anything from runtime errors to silent omissions from the resultant data.

Some of the files documented in this section are standard gEDA files, while others are koala specific files.

2.2.2 A gEDA Project

A gEDA Project consists of exactly one PCB designed using gEDA tools. Note that this is different from a product, which can contain one or more projects (of the gEDA variety or other). The folder structure listed here is for the standard Quazar SVN gEDA project.

Nomenclature

gEDA projects use three levels of naming:

projname Project Name is the base name of the PCB, not including hardware revision and configuration information.

pcbname PCB Name is the actual name of the PCB for external fabrication, and therefore includes revision information but not configuration information.

confname Conf Name or Card Name includes full hardware revision as well as configuration information.

The rationale for separation of PCBs by project, pcb, and configuration is inherently ambiguous, and the developer / maintainer of a project and / or product line must use his / her discretion to make sure that the basis used is reasonable. Some suggested guidelines are listed here for reference :

- The `Project Name` can, in principle, exist for the entire duration of relevance of the Project itself. However, for the sake of simplicity, it is probably easier to change the project name at points where there is a significant change from the previous versions. Minimally, the `Project Name` should be sufficient to fully define the location of the project in the svn tree, which it does by specifying the folder name.
- The `PCB Name` must change, without exception, at every change requiring update of `gerber` data, however minor the change may be. The `PCB Name` must be suffcient information to fully define the data set to be sent to a bare PCB fabricator as well as handle inventory checks, etc. This should be the name printed on the PCB Silk as well as used in Purchase Orders, Indents, etc.
- The `Conf Name` must change along with `PCB Name`. Besides this, `conf names` are expected to change asynchronously with general PCB development in the form of configurations added or removed. The `Conf Name` should be suffcient to fully define the assembled PCB when read with information contained in the `configs.yaml` file and other standard gEDA files.

General Structure of Names:

```
[PROJECT-NAME-WITH-EMBEDDED-VARIABLES]-[HARDWARE-REVISION]-[SPECIAL-CONFIGURATION-FLAGS]
```

where:

- **`PROJECT-NAME-WITH-EMBEDDED-VARIABLES`** is a name including lower-case letters as placeholders for variable elements within the name. The variables may not be explicitly marked out, in which case the highest allowed value is the suggested representation.

  In the "Conf Name", the variables should all be completely resolved to the correct values.

  The `PROJECT-NAME` itself could generally be of the form:

  ```
  [PRODUCT LINE]-[PRIMARY PCB FUNCTION]-[ADDITIONAL INFORMATION]
  ```

  Examples:

  - `X-TCON-L1` for Xplore, Temperature Controller, Linear Design 1
  - `QASC-iSTRAIN` for QDA, Active Signal Conditioner, Interactive Strain
  - `QDAL41xB` for QDA, Low-end Data Acquisition Unit, 10^4 * 1Hz, x Channels, Type B
- **`HARDWARE-REVISION`** is a representation of the specific Hardware Revision of the PCB, generally represented as `Rn`, where `n` is a number. Further information in the hardware revision is probably not a good idea.
- **`SPECIAL-CONFIGURATION-FLAGS`** is a list of configuration flags to specify parameters that aren't represented in the embedded variables (ex. `-GR`). In the case of core circuit elements that are usually populated except in rare cases, the configuration flag can be a negation (ex. `-NOHV`).

Project Folder Structure

```
[projname]
`-- hardware
    |-- branches
    |-- tags
    |   |-- hR1
    |   |   `-- [full copy of trunk at a specific revision]
    |   ..
    `-- trunk
        |-- ChangeLog
        |-- gerber                          (all-generated-pcb-manual)
        |   |-- [projname].[layer].gbr or cnc
        |   ..
        |-- gerber.zip
        |
        |-- pcb
        |   |-- [projname].cmd              (generated-gsch2pcb)
        |   |-- [projname].dxf              (generated-koala)
        |   |-- [projname].net              (generated-gsch2pcb)
        |   |-- [projname].pcb
        |   `-- sourcing.yaml               (generated-koala-manual)
        |
        |-- schematic
        |   |-- [schname-1].sch
        |   ..
        |   |-- [schname-n].sch
        |   |-- attribs                     (project-template)
        |   |-- [projname].proj             (project-template-manual)
        |   |-- readme.txt                  (project-template-manual)
        |   `-- configs.yaml                (project-template-manual)
        |
        |-- electrical
        |   |-- [elschname-1].sch
        |   ..
        |   |-- [elschname-2].sch
        |   |-- attribs                     (project-template)
        |   |-- [projname]-electrical.proj  (project-template-manual)
        |   |-- readme.txt                  (project-template-manual)
        |
        `-- doc                             (all-generated-koala)
            |-- [projname]-masterdoc.pdf
            |-- [projname]-configs.pdf
            |-- [projname]-schematic.pdf
            |-- [projname]-pcb.pdf
            `-- confboms
                |-- [confname-1]-bom.pdf
                ..
                |-- [confname-m]-bom.pdf
                `-- conf-boms.csv
```

Part II

Developer Documentation

CHAPTER

THREE

GEDA INTERFACE MODULE DOCUMENTATION

3.1 gEDA ConfigsFile module documentation (gedaif.conffile)

class gedaif.conffile.**ConfigsFile**(*projectfolder*)
Bases: object

get_configs_file()

projectfolder

3.2 gEDA Project File module documentation (gedaif.projfile)

class gedaif.projfile.**GedaProjectFile**(*projectfolder*)
Bases: object

static strip_line(*line*)

schpaths

3.3 gEDA BOM Parser module documentation (gedaif.bomparser)

class gedaif.bomparser.**BomLine**(*line*, *columns*)
Bases: object

class gedaif.bomparser.**GedaBomParser**(*projectfolder*, *backend*)
Bases: object

generate_temp_bom(*backend*)

prep_temp_bom()

delete_temp_bom()

get_lines()

3.4 gEDA gschem module documentation (gedaif.gschem)

gedaif.gschem.**conv_gsch2pdf**(*schpath*)

3.5 gEDA gsymlib Module documentation (gedaif.gsymlib)

class gedaif.gsymlib.**GedaSymbol**(*fpath*)
Bases: object

_acq_sym(*fpath*)

ident

sym_ok

is_generator

is_virtual

class gedaif.gsymlib.**GSymGeneratorFile**(*sympath*)
Bases: object

_get_data()

values

gedaif.gsymlib.**gen_symlib**()

gedaif.gsymlib.**_jinja_init**()

gedaif.gsymlib.**seed_generators**()

gedaif.gsymlib.**export_symlib**()

CHAPTER

FOUR

ENTITY HUB MODULES DOCUMENTATION

4.1 EntityHub Transforms Module documentation (`entityhub.transforms`)

class `entityhub.transforms.`**`TransformFile`**(*tfpath*)
Bases: `object`

`get_canonical_repr`(*contextual*)

`get_contexual_repr`(*canonical*)

4.2 EntityHub Maps Module documentation (`entityhub.maps`)

class `entityhub.maps.`**`MapFile`**(*mappath*)
Bases: `object`

`get_idents`()

`get_upartnos`(*canonical*)

`get_apartnos`(*canonical*)

`get_all_partnos`(*canonical*)

`get_partnos`(*canonical*)

`get_strategy`(*canonical*)

`get_canonical`(*partno*)

`get_user_map`()

CHAPTER

FIVE

BOMS MODULE DOCUMENTATION

5.1 Electronic Boms Module documentation (boms.electronics)

This module contains various classes for handling structured data present in elctronics BOMs. The module is built for use with gEDA with specific additions and/or constraints to the standard gEDA project structure. Use with other EDA tools should be possible by replacing `gedaif` with an EDA tool specific package.

5.1.1 Module Summary:

EntityBase()	Placeholder class for potentially track-able objects.
EntityElnComp([item])	Object containing a single electronic component.
EntityGroup	
import_pcb(cardfolder)	Import PCB and return a populated EntityBom

5.1.2 Module Members:

class boms.electronics.**EntityElnComp**(*item=None*)

Bases: boms.entitybase.EntityBase

Object containing a single electronic component.

Accept a gedaif.bomparser.BomLine generated through gnetlist's bom backend. The attribs file should atleast contain:

- device
- value
- footprint
- fillstatus

Parameters **item** (*gedaif.bomparser.BomLine*) – BomLine containing details of component to be created

define(*refdes*, *device*, *value*, *footprint=''*, *fillstatus=''*)

Define the component.

Can be used directly for special cases when there is no *gedaif.bomparser.BomLine* to pass to the class *__init__* function.

Parameters

- **refdes** – Refdes string.
- **device** – Device string.
- **value** – Value string.
- **footprint** – Footprint string. Optional.
- **fillstatus** – Fillstatus string. Optional.

fillstatus
Fillstatus string. When `fillstatus` is `DNP`, the component is not included in BOMs irrespective of other configuration states.

ident

device
Component device string.

value
Component value string.

footprint
Component footprint string. `MY-` at the beginning of the footprint string is stripped away automatically.

class boms.electronics.**EntityElnGroup**(*groupname*, *contextname*)
Bases: boms.entitybase.EntityGroupBase

Container for a group of EntityElnComp objects.

Variables

- **groupname** – Name of the group
- **eln_comp_list** – List of EntityElnComp objects

insert(*item*)
Insert an electronic component into the EntityGroup.

Accept a BomLine and generate an EntityElnComp to represent the component if it's `fillstatus` is not `DNP`. Insert the created EntityElnComp object into the group.

Parameters **item** (*gedaif.bomparser.BomLine*) – `BomLine` representing the item to insert.

insert_eln_comp(*comp*)
Insert a manually created component into the EntityGroup.

This should be used for components not originating directly from gEDA's gnetlist output.

Parameters **comp** (*EntityElnComp*) – Existing component to insert

class boms.electronics.**EntityElnBomConf**(*configdata*)
Bases: object

get_configurations()

get_configsections()

get_sec_groups(*sectionname*, *config*)

get_configuration(*configname*)

class boms.electronics.**EntityElnBom**(*configfile*)
Bases: boms.entitybase.EntityBomBase

create_groups()

find_tgroup(*item*)
:rtype : EntityElnGroup

find_group(*groupname*)
:rtype : EntityElnGroup

populate_bom()

create_output_bom(*configname*)

boms.electronics.**import_pcb**(*cardfolder*)
Import PCB and return a populated EntityBom

Accept cardfolder as an argument and return a populated EntityBOM. The cardfolder should be the path to a PCB folder, containing the file structure described in somewhere.

See also:

- gedaif.projfile.GedaProjectFile
- gEDA Project Folder Structure

Parameters **cardfolder** (*str*) – PCB folder (containing schematic, pcb, gerber)

Returns Populated EntityBom

Return type EntityBom

Example

```
>>> import boms.electronics
>>> bom = boms.electronics.import_pcb('path/to/cardfolder')
```

CHAPTER

SIX

INVENTORY MODULE DOCUMENTATION

6.1 Electronics Inventory module documentation (inventory.electronics)

6.2 Inventory Acquire Module documentation (inventory.acquire)

class inventory.acquire.**StockXlsReader**(*xlf*, *sname*, *location*, *tfpath*)
Bases: object

_skip_to_header()

_get_cols(*row*)

static _is_balance(*item*)

_row_gen()

_tf_row_gen()

inventory.acquire.**get_stockxlsreader**(*xlpath*, *sname*, *location*, *tfpath*)

inventory.acquire.**get_reader**(*elec_inven_data_idx*)

inventory.acquire.**gen_canonical_transform**(*elec_inven_data_idx*)

inventory.acquire.**helper_transform**(*ident*)

CHAPTER

SEVEN

SOURCING MODULE DOCUMENTATION

7.1 Electronics Sourcing module documentation (sourcing.electronics)

sourcing.electronics.**gen_vendor_mapfile**(*vendor_obj*)

sourcing.electronics.**init_vendors**()

sourcing.electronics.**export_vendor_map_audit**(*vendor_obj*)

7.2 Vendors module documentation (sourcing.vendors)

class sourcing.vendors.**VendorBase**(*name*, *dname*, *pclass*, *mappath=None*, *currency_code='INR'*, *currency_symbol='INR '*)

Bases: object

name

pclass

mappath

map

currency

search_vpnos(*ident*)

get_vpart(*vpartno*, *ident=None*)

class sourcing.vendors.**VendorPrice**(*moq*, *price*, *currency_def*)

Bases: object

moq

native_value

class sourcing.vendors.**VendorPartBase**(*ident*, *vendor*)

Bases: object

add_price(*price*)

vpno

vqtyavail

manufacturer

mpartno

vpartdesc

ident

abs_moq

class sourcing.vendors.**VendorElnPartBase**(*ident*, *vendor*)

Bases: sourcing.vendors.VendorPartBase

package

datasheet

7.3 Digi-Key Sourcing Module documentation (sourcing.digikey)

class sourcing.digikey.**VendorDigiKey**(*name*, *dname*, *pclass*, *mappath=None*, *currency_code=None*, *currency_symbol=None*)

Bases: sourcing.vendors.VendorBase

get_vpart(*vpartno*, *ident=None*)

search_vpnos(*ident*)

static _search_preprocess(*device*, *value*, *footprint*)

static _process_product_page(*soup*)

static _get_device_catstrings(*device*)

_process_index_page(*soup*, *device*)

static _get_resultpage_row_pno(*row*)

static _get_resultpage_row_unitp(*row*)

static _get_resultpage_row_package(*row*)

static _get_resultpage_row_minqty(*row*)

static _get_resultpage_row_mfgpno(*row*)

_process_resultpage_row(*row*)

_get_resultpage_parts(*soup*)

static _find_exact_match_package(*parts*, *value*)

static _find_consensus_package(*parts*)

static _filter_results_unfiltered(*parts*)

static _filter_results_byfootprint(*parts*, *footprint*)

static _filter_results_bycpackage(*parts*, *cpackage*, *strategy*)

_filter_results(*parts*, *value*, *footprint*)

_process_results_page(*soup*, *value*, *footprint*)

_get_search_vpnos(*device*, *value*, *footprint*)

static _tf_resistance_to_canonical(*rstr*)

static _tf_capacitance_to_canonical(*cstr*)

static _tf_package_tant_smd(*footprint*)

_get_searchpage_filters(*soup*)

static _get_default_urlparams()

static _get_searchurl_res_smd()

static _get_searchurl_cap_cer_smd()

static _get_searchurl_cap_tant_smd()

_get_searchurl_filters(*searchurl*)

_get_pas_vpnos(*device*, *value*, *footprint*)

class sourcing.digikey.**DigiKeyElnPart**(*dkpartno*, *ident=None*, *vendor=None*)
Bases: sourcing.vendors.VendorElnPartBase

_get_data()

_get_prices(*soup*)

static _get_mpartno(*soup*)

static _get_manufacturer(*soup*)

static _get_package(*soup*)

static _get_datasheet_link(*soup*)

static _get_avail_qty(*soup*)

static _get_description(*soup*)

get_price(*qty*)

7.4 CSIL Sourcing Module documentation (sourcing.csil)

sourcing.csil.**get_csil_prices**(*params={'layers': 2, 'finish': 'Au', 'pcbname': 'Test', 'dY': '46.7', 'qty': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99], 'dX': '98.2', 'time': 15}*)

class sourcing.csil.**VendorCSIL**(*name*, *dname*, *pclass*)
Bases: sourcing.vendors.VendorBase

search_vpnos(*ident*)

get_vpart(*vpartno*, *ident=None*)

CHAPTER

EIGHT

DOX MODULE DOCUMENTATION

8.1 Production Dox module documentation (dox.production)

dox.production.**gen_pcb_am**(*projfolder*, *configname*, *outpath=None*, *sno=None*)

8.2 gEDA Project Dox module documentation (dox.gedaproject)

dox.gedaproject.**gen_confbom**(*projfolder*, *configname*)

dox.gedaproject.**gen_configdoc**(*projfolder*)

dox.gedaproject.**gen_schpdf**(*projfolder*)

dox.gedaproject.**gen_masterdoc**(*projfolder*)

dox.gedaproject.**gen_confpdf**(*projfolder*, *configname*)

8.3 Dox Render module documentation (dox.render)

dox.render.**jinja2_pdfinit**()

dox.render.**render_pdf**(*stage*, *template*, *outpath*)

CHAPTER

NINE

UTILS MODULE DOCUMENTATION

9.1 Config Module Documentation (utils.config)

9.2 Currency module documentation (utils.currency)

class utils.currency.**CurrencyDefinition**(*code*, *symbol=None*, *exchval=None*)
Bases: object

symbol

exchval

static _get_exchval(*code*)

class utils.currency.**CurrencyValue**(*val*, *currency_def*)
Bases: object

native_value

native_string

source_value

source_string

source_currency

9.3 Filesystem Utils Module Documentation (utils.fsutils)

class utils.fs.**VersionedOutputFile**(*pathname*, *numsavedversions=3*)
Create a new output file.

'pathname' is the name of the file to [over]write. 'numSavedVersions' tells how many of the most recent versions of 'pathname' to save.

close()

as_file()
Return self's shadowed file object, since marshal is pretty insistent on working w. pure file objects.

_replace_current_file()
Replace the current contents of self's named file.

_backup_current_file()
Save a numbered backup of self's named file.

_versioned_name(*revision*)
Get self's pathname with a revision number appended.

_current_revision()
Get the revision number of self's largest existing backup.

_revisions()
Get the revision numbers of all of self's backups.

_delete_old_revisions()
Delete old versions of self's file, so that at most self._numSavedVersions versions are retained.

utils.fs.**fsutils_cleanup**()

9.4 LibreOffice Utils Module Documentation (utils.ooutils)

class utils.libreoffice.**XLFile**(*filepath*)
Bases: object

get_csv_path(*sheetname*)

_make_copy()

_make_csv_files()

_parse_sscout(*string*)

close()

_clean()

utils.libreoffice.**get_xlf**(*fpath*)

utils.libreoffice.**ooutils_cleanup**()

9.5 PDF Utils Module documentation (utils.pdfutils)

This module contains small helper functions to handle PDF files. These functions are independent of the rest of the software and can be reused wherever needed.

utils.pdf.**merge_pdf**(*pdflist*, *outfilepath*)
Merges PDF files. Sources are PDF files whose paths are listed in pdflist and the combined PDF is written out to outfilepath.

Parameters

- **pdflist** – List of paths to PDF files to be merged.
- **outfilepath** – Path where output PDF file is to be written.

Returns outfilepath

utils.pdf.**conv_ps2pdf**(*pspath*, *pdfpath*)
Runs *ps2pdf* to convert a PS file specified in pspath to a PDF file writen out to pdfpath.

Parameters

- **pspath** – Path to PS file to be converted.
- **pdfpath** – Path of PDF file to be generated.

9.6 WWW Utils Module Documentation (utils.wwwutils)

utils.www.**strencode**(*string*)

utils.www.**dump_redirect_cache**()

class utils.www.**CachingRedirectHandler**

Bases: urllib2.HTTPRedirectHandler

http_error_301(*req*, *fp*, *code*, *msg*, *headers*)

http_error_302(*req*, *fp*, *code*, *msg*, *headers*)

utils.www.**_test_opener**(*openr*)

utils.www.**_create_opener**()

utils.www.**urlopen**(*url*)

utils.www.**get_soup**(*url*)

b

d

e

g

i

s

u

Symbols

A

B

C

D

E

F

G

H

I

J

M

N

O

P