Koala Documentation

Release 0.1alpha1

Chintalagiri Shashank

August 04, 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.

motif `(Optional)` Identifier for the motif the component is part of (**'Motif'_**).

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 Motifs

Attribute Syntax Structure : `[MOTIF_CLASS].[REFDES]-[MOTIF_ELEMENT]`

Examples : `DLPF1.1:R1`, `DLPF1.1:R2`, `DLPF1.1:C1`, `DLPF1.1:C2`, `DLPF1.1:C3`

2.1.6 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.

2.1.7 Attribute Promotion

HM

2.1.8 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-koala)
        |   |-- [projname].[layer].gbr or cnc
        |   ..
        |-- [projname]-gerber.zip           (generated-koala)
        |
        |-- 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)
        |
        `-- 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

Koala

CHAPTER

THREE

KOALA PACKAGE

3.1 Subpackages

3.1.1 koala.boms package

Submodules

koala.boms.electronics module

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.

`EntityElnComp`([item])	Object containing a single electronic component.
`EntityGroup`	
`import_pcb`(cardfolder)	Import PCB and return a populated EntityBom

Module Summary:

Module Members:

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

Bases: *`koala.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 *koala.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 `koala.boms.electronics.`**`EntityElnGroup`**(*groupname*, *contextname*)
Bases: *`koala.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 `koala.boms.electronics.`**`EntityElnBomConf`**(*configdata*)
Bases: `object`

`get_configurations`()

`get_configsections`()

`get_sec_groups`(*sectionname*, *config*)

`get_configuration`(*configname*)

`get_configuration_motifs`(*configname*)

`get_configuration_gens`(*configname*)

`get_configuration_sjs`(*configname*)

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

create_groups()

find_tgroup(*item*)
:rtype : EntityElnGroup

find_group(*groupname*)
:rtype : EntityElnGroup

populate_bom()

get_motif_by_refdes(*refdes*)

create_output_bom(*configname*)

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

koala.boms.entitybase module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class koala.boms.entitybase.**EntityBase**
Bases: `object`

Placeholder class for potentially track-able objects.

Depending on the implementation used, this class should inherit from an external class built for this purpose instead of from `object`.

define(**args*, ***kwargs*)

defined
State of the component. The component should be used only when it is fully defined.

This is a read-only property.

ident

refdes
Refdes string.

class koala.boms.entitybase.**EntityGroupBase**(*groupname*, *contextname=''*)
Bases: *koala.boms.entitybase.EntityBase*

insert(*item*)

ident

define(*contextname*, *groupname*)

class koala.boms.entitybase.**EntityBomBase**
Bases: *koala.boms.entitybase.EntityBase*

ident

create_output_bom(**args*, ***kwargs*)

define(**args*, ***kwargs*)

koala.boms.outputbase module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class koala.boms.outputbase.**OutputElnBomDescriptor**(*pcbname*, *cardfolder*, *configname*, *configurations*, *multiplier=1*)
Bases: object

class koala.boms.outputbase.**OutputBomLine**(*comp*, *parent*)
Bases: object

add(*comp*)

quantity

class koala.boms.outputbase.**OutputBom**(*descriptor*)
Bases: object

sort_by_ident()

find_by_ident(*ident*)

insert_component(*item*)

multiply(*factor*, *composite=False*)

class koala.boms.outputbase.**CompositeOutputBomLine**(*line*, *colcount*)
Bases: object

add(*line*, *column*)

quantity

subset_qty(*idxs*)

merge_line(*cline*)

class koala.boms.outputbase.**CompositeOutputBom**(*bom_list*)
Bases: object

get_subset_idxs(*confignames*)

insert_bom(*bom*, *i*)

insert_line(*line*, *i*)

find_by_ident(*ident*)

sort_by_ident()

dump(*stream*)

collapse_wires()

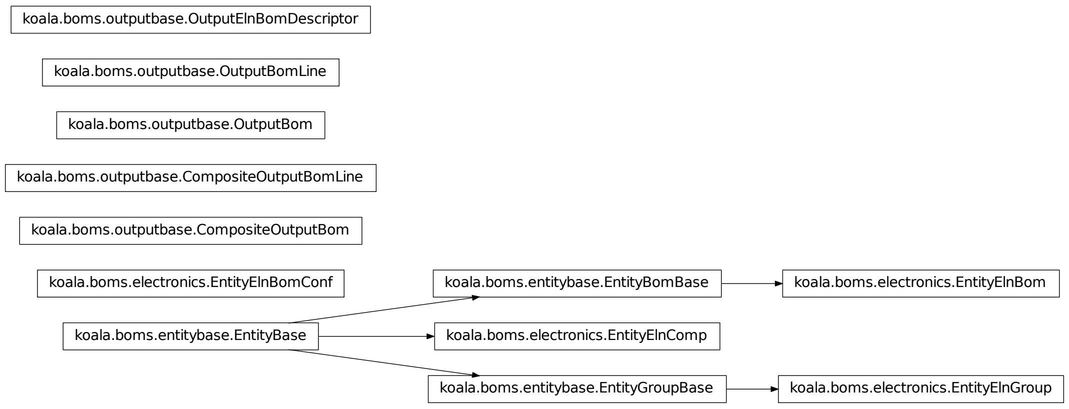
koala.boms.products module

This file is part of koala See the COPYING, README, and INSTALL files for more information

Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

Inheritance Diagram


koala.boms.outputbase.OutputElnBomDescriptor
koala.boms.outputbase.OutputBomLine
koala.boms.outputbase.OutputBom
koala.boms.outputbase.CompositeOutputBomLine
koala.boms.outputbase.CompositeOutputBom
koala.boms.electronics.EntityElnBomConf
koala.boms.entitybase.EntityBomBase
koala.boms.electronics.EntityElnBom
koala.boms.entitybase.EntityBase
koala.boms.electronics.EntityElnComp
koala.boms.entitybase.EntityGroupBase
koala.boms.electronics.EntityElnGroup

3.1.2 koala.conventions package

Subpackages

koala.conventions.motifs package

Submodules

koala.conventions.motifs.DLPF1 module This file is part of koala See the COPYING, README, and INSTALL files for more information

This file needs to be refactored quite a bit

class `koala.conventions.motifs.DLPF1.`**`MotifDLPF1`**(*identifier*)

Bases: *`koala.conventions.motifs.motifbase.MotifBase`*

`configure`(*configdict*)

`_set_biases`()

`Fcm`

`Fdiff`

`R1`

`R2`

`C1`

`C2`

`C3`

`validate`()

`get_configdict_stub`()

koala.conventions.motifs.ING_AD8421 module This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.conventions.motifs.ING_AD8421.`**`MotifING_AD8421`**(*identifier*)

Bases: *`koala.conventions.motifs.ingbase.MotifInampGainBase`*

`res_to_gain`(*res*)

`gain_to_res`(*gain*)

koala.conventions.motifs.LPF1 module This file is part of koala See the COPYING, README, and INSTALL files for more information

This file needs to be refactored quite a bit

class `koala.conventions.motifs.LPF1.`**`MotifLPF1`**(*identifier*)

Bases: *`koala.conventions.motifs.motifbase.MotifBase`*

`configure`(*configdict*)

`Fc`

`R1`

`C1`

`validate`()

`get_configdict_stub`()

koala.conventions.motifs.LREGS1 module This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.conventions.motifs.LREGS1.`**`MotifLREGS1`**`(`*identifier*`)`
Bases: *`koala.conventions.motifs.motifbase.MotifBase`*

`get_configdict_stub``()`

`configure``(`*configdict*`)`

`_autoset_r2``()`

`validate``()`

`Vout`

`R1`

`R2`

`Vref`

`Imin`

`Il`

`listing`

koala.conventions.motifs.ingbase module This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.conventions.motifs.ingbase.`**`MotifInampGainBase`**`(`*identifier*`)`
Bases: *`koala.conventions.motifs.motifbase.MotifBase`*

`validate``()`

`get_configdict_stub``()`

`configure``(`*configdict*`)`

`res_to_gain``(`*res*`)`

`gain_to_res``(`*gain*`)`

`R1`

`gain`

koala.conventions.motifs.motifbase module This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.conventions.motifs.motifbase.`**`MotifBase`**`(`*identifier*`)`
Bases: `object`

`columns` **= [’refdes’, ‘device’, ‘value’, ‘footprint’, ‘fillstatus’, ‘group’, ‘package’, ‘status’]**

`refdes`

`_line_generator``()`

`get_configdict_stub``()`

`configure``(`*configdict*`)`

`get_line_gen``()`

`get_elem_by_idx``(`*idx*`)`

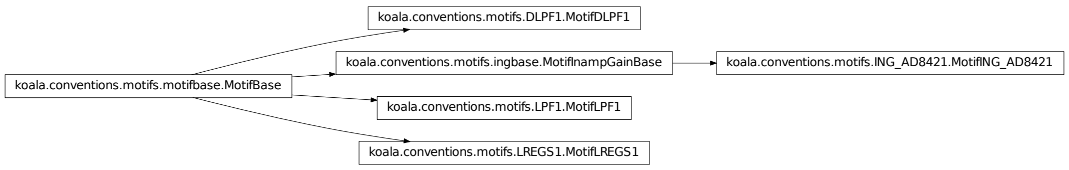
add_element(*bomline*)

validate()

listing

Module contents This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.conventions.motifs.**create_motif_object**(*motifst*)


koala.conventions.motifs.motifbase.MotifBase
koala.conventions.motifs.DLPF1.MotifDLPF1
koala.conventions.motifs.ingbase.MotifInampGainBase
koala.conventions.motifs.ING_AD8421.MotifING_AD8421
koala.conventions.motifs.LPF1.MotifLPF1
koala.conventions.motifs.LREGS1.MotifLREGS1

Inheritance Diagram

Submodules

koala.conventions.electronics module

Electronics Conventions Module documentation (`conventions.electronics`)

koala.conventions.electronics.**fpismodlen**(*device*)

koala.conventions.electronics.**fpiswire**(*device*)

koala.conventions.electronics.**fpiswire_ident**(*ident*)

koala.conventions.electronics.**ident_transform**(*device*, *value*, *footprint*, *tf=None*)

Auto-generated ident string from the component `device`, `value` and `footprint` attributes.

If the device string starts with any of the strings in `nofp_strings`, the `footprint` is excluded from the `ident` string.

When applied to gEDA generated data (BOMs, gsymlib), the return value is the `Canonical Representation` for the component. The burden of ensuring uniqueness and consistency is on the gEDA schematic files and authors thereof. See conventions.rst listed in `somewhere` for relevant guidelines.

For all other forms of data, the specific modules must provide a transform csv file to ensure correct mapping into the canonical form. For the format of this file, see `somewhere`.

koala.conventions.electronics.**parse_ident**(*ident*)

koala.conventions.electronics.**construct_resistor**(*resistance*, *wattage=None*)

koala.conventions.electronics.**construct_capacitor**(*capacitance*, *voltage*)

koala.conventions.electronics.**construct_inductor**(*inductance*)

koala.conventions.electronics.**construct_crystal**(*frequency*) `koala.conventions.electronics.`**`parse_resistor`**(*value*)

`koala.conventions.electronics.`**`parse_capacitor`**(*value*)

`koala.conventions.electronics.`**`parse_inductor`**(*value*)

`koala.conventions.electronics.`**`parse_crystal`**(*value*)

`koala.conventions.electronics.`**`normalize_resistance`**(*res*)

`koala.conventions.electronics.`**`check_for_std_val`**(*ident*)

Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

Inheritance Diagram

3.1.3 koala.dox package

Submodules

koala.dox.customs module

This file is part of koala See the COPYING, README, and INSTALL files for more information

`koala.dox.customs.`**`gen_declaration`**(*invoice*, *target_folder*, *copyt*, *serialno*)

`koala.dox.customs.`**`gen_valuation`**(*invoice*, *target_folder*, *serialno*)

`koala.dox.customs.`**`gen_rsp_declaration`**(*invoice*, *target_folder*, *serialno*)

`koala.dox.customs.`**`gen_authorization`**(*invoice*, *target_folder*, *serialno*)

`koala.dox.customs.`**`gen_tech_writeup`**(*invoice*, *target_folder*, *serialno*)

`koala.dox.customs.`**`gen_submitdocs`**(*invoice*, *target_folder*, *serialno*)

`koala.dox.customs.`**`gen_verification_sections`**(*invoice*, *target_folder*, *serialno*)

`koala.dox.customs.`**`gen_verification_checklist`**(*invoice*, *target_folder*, *serialno*)

`koala.dox.customs.`**`gen_verificationdocs`**(*invoice*, *target_folder*, *serialno*)

`koala.dox.customs.`**`generate_docs`**(*invoice*, *target_folder=None*, *serialno=None*, *register=False*, *efield=None*)

koala.dox.docstore module

This file is part of koala See the COPYING, README, and INSTALL files for more information

`koala.dox.docstore.`**`insert_document`**(*sno*, *docpath*, *series*)

koala.dox.gedaproject module

gEDA Project Dox module documentation (`dox.gedaproject`)

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

`koala.dox.gedaproject.`**`gen_configdoc`**(*projfolder*, *namebase*)

`koala.dox.gedaproject.`**`gen_schpdf`**(*projfolder*, *namebase*)

`koala.dox.gedaproject.`**`gen_masterdoc`**(*projfolder*, *namebase*)

`koala.dox.gedaproject.`**`gen_confpdf`**(*projfolder*, *configname*, *namebase*)

`koala.dox.gedaproject.`**`gen_cobom_csv`**(*projfolder*, *namebase*)

`koala.dox.gedaproject.`**`gen_pcb_pdf`**(*projfolder*)

`koala.dox.gedaproject.`**`gen_pcb_gbr`**(*projfolder*)

`koala.dox.gedaproject.`**`gen_pcb_dxf`**(*projfolder*)

`koala.dox.gedaproject.`**`gen_pcbpricing`**(*projfolder*, *namebase*)

`koala.dox.gedaproject.`**`generate_docs`**(*projfolder*)

koala.dox.indent module

Production Dox module documentation (`dox.production`)

`koala.dox.indent.`**`gen_stock_idt_from_cobom`**(*outfolder*, *sno*, *title*, *carddict*, *cobom*)

koala.dox.invoice module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.dox.labelmaker module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.dox.labelmaker.`**`LabelBase`**(*code*, *ident*, *sno*, *branding=None*, *logo=None*, *include_qr=True*, *include_logo=True*)

Bases: `object`

`templatefile` = None

`code`

`sno`

`ident`

`branding`

`include_logo`

`include_qr`

`logo`

`qrcode`

`_gen_qrcode`(*wfpath=None*)

class `koala.dox.labelmaker.`**`LabelCW1`**(*code*, *ident*, *sno*, *branding=None*, *logo=None*, *include_qr=True*, *include_logo=True*)
Bases: *`koala.dox.labelmaker.LabelBase`*

`templatefile` **= 'labels/CW1_template.tex'**

`lpp` **= 88**

class `koala.dox.labelmaker.`**`LabelP1`**(*code*, *ident*, *sno*, *branding=None*, *logo=None*, *include_qr=True*, *include_logo=True*)
Bases: *`koala.dox.labelmaker.LabelBase`*

`templatefile` **= 'labels/CW1_template.tex'**

`lpp` **= 88**

class `koala.dox.labelmaker.`**`LabelP2`**(*code*, *ident*, *sno*, *branding=None*, *logo=None*, *include_qr=True*, *include_logo=True*)
Bases: *`koala.dox.labelmaker.LabelBase`*

`templatefile` **= 'labels/P2_template.tex'**

`lpp` **= 120**

class `koala.dox.labelmaker.`**`LabelD1`**(*code*, *ident*, *sno*, *branding=None*, *logo=None*, *include_qr=True*, *include_logo=True*)
Bases: *`koala.dox.labelmaker.LabelBase`*

`templatefile` **= 'labels/CW1_template.tex'**

`lpp` **= 88**

class `koala.dox.labelmaker.`**`LabelIDT`**(*code*, *ident*, *sno*, ***kwargs*)
Bases: *`koala.dox.labelmaker.LabelBase`*

`templatefile` **= 'labels/IDT_template.tex'**

`lpp` **= 88**

`qty`

`ident`

`koala.dox.labelmaker.`**`get_labelbase`**(*code*)

class `koala.dox.labelmaker.`**`LabelSheet`**(*base*, *code*)
Bases: `object`

`code`

`base`

`labels`

`add_label`(*label*)

`nl`

`generate_pdf`(*targetfolder*, *force=True*)

`clear_sno_label`(*sno*)

class `koala.dox.labelmaker.`**`LabelMaker`**
Bases: `object`

`add_label`(*code*, *ident*, *sno*, ***kwargs*)

`_get_sheet`(*code*)

_sheetdict

generate_pdfs(*targetfolder*, *force=False*)

nl

_clear_sno_label(*sno*)

koala.dox.labelmaker.**get_manager**()

koala.dox.labelmaker.**dump_manager**()

koala.dox.production module

Production Dox module documentation (dox.production)

koala.dox.production.**gen_pcb_am**(*projfolder*, *configname*, *outfolder*, *sno=None*, *productionorderno=None*, *indentsno=None*, *register=False*)

koala.dox.production.**gen_production_order**(*outfolder*, *prod_sno*, *sourcedata*, *snos*, *sourcing_orders=None*, *root_orders=None*)

koala.dox.purchaseorder module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.dox.purchaseorder.**render_po**(*stage*, *templateid*, *outpath*)

koala.dox.render module

Dox Render module documentation (dox.render)

koala.dox.render.**format_currency**(*value*)

koala.dox.render.**escape_latex**(*string*)

koala.dox.render.**jinja2_pdfinit**()

koala.dox.render.**render_pdf**(*stage*, *template*, *outpath*, *remove_sources=True*, ***kwargs*)

koala.dox.render.**render_lineplot**(*outf*, *plotdata*, *title*, *note*)

koala.dox.testing module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.dox.wallet module

This file is part of koala See the COPYING, README, and INSTALL files for more information

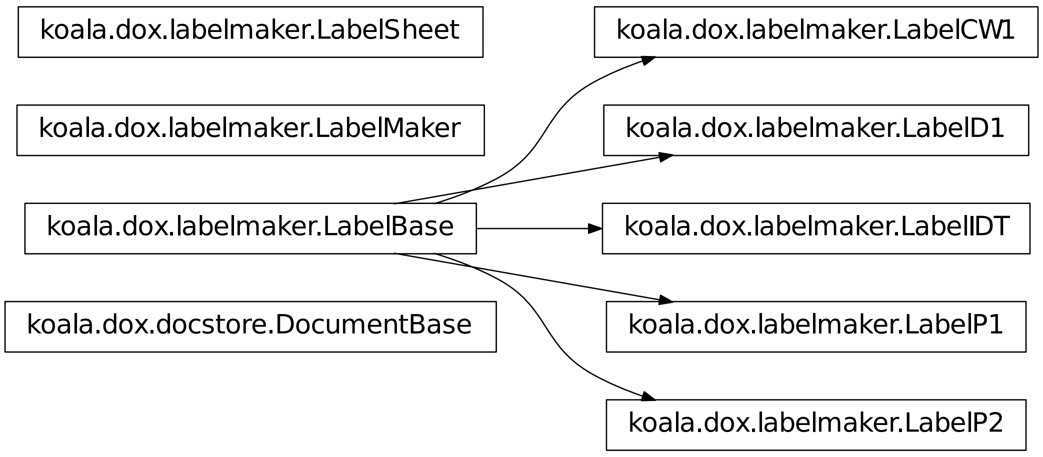
koala.dox.wallet.**get_document_path**(*key*)

koala.dox.wallet.**is_in_wallet**(*fpath*)

Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

Inheritance Diagram


koala.dox.labelmaker.LabelSheet
koala.dox.labelmaker.LabelMaker
koala.dox.labelmaker.LabelBase
koala.dox.docstore.DocumentBase
koala.dox.labelmaker.LabelCW1
koala.dox.labelmaker.LabelD1
koala.dox.labelmaker.LabelIDT
koala.dox.labelmaker.LabelP1
koala.dox.labelmaker.LabelP2

3.1.4 koala.entityhub package

Submodules

koala.entityhub.guidelines module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.entityhub.guidelines.`**`QtyGuidelineTableRow`**(*ide*, *(oqty_min*, *oqty_multiple*, *baseline_qty*, *excess_min_pc*, *excess_min_qty*, *excess_max_qty)*)

Bases: `object`

`id`

`oqty_min`

`oqty_multiple`

`baseline_qty`

`excess_min_pc`

`excess_min_qty`

`excess_max_qty`

class `koala.entityhub.guidelines.`**`QtyGuidelines`**(*guidelinefile*)

Bases: `object`

`_load_guidelines`()

`get_guideline_table`()

static `_get_full_guideline`(*gldict*)

get_compliant_qty(*ident*, *qty*, *handle_excess=True*, *except_on_overrun=True*, *handle_baseline=False*)

koala.entityhub.maps module

EntityHub Maps Module documentation (entityhub.maps)

class koala.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()

koala.entityhub.products module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.entityhub.projects module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.entityhub.projects.**is_project_folder**(*folder*)

koala.entityhub.projects.**get_project_doc_folder**(*projectfolder*)

koala.entityhub.projects.**parse_total_costing**(*filename*)

koala.entityhub.projects.**get_card_indicative_cost**(*cardname*)

koala.entityhub.projects.**get_projects**(*basefolder=None*)

koala.entityhub.serialnos module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.entityhub.serialnos.**get_series**(*sno*)

koala.entityhub.transforms module

EntityHub Transforms Module documentation (entityhub.transforms)

class koala.entityhub.transforms.**TransformFile**(*tfpath*)

Bases: object

get_canonical_repr(*contextual*)

get_ideal_repr(*contextual*)

get_status(*contextual*)

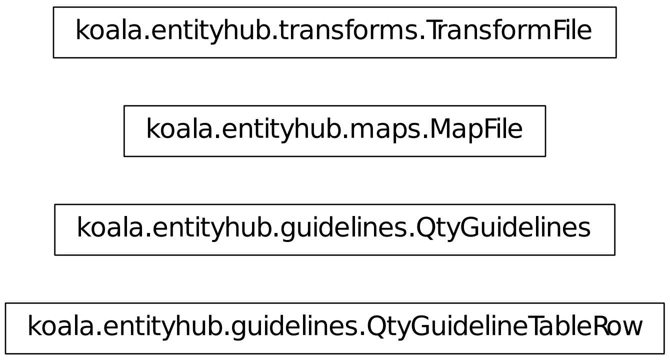
get_contextual_repr(*canonical*)

has_contextual_repr(*contextual*)

Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

Inheritance Diagram


koala.entityhub.transforms.TransformFile
koala.entityhub.maps.MapFile
koala.entityhub.guidelines.QtyGuidelines
koala.entityhub.guidelines.QtyGuidelineTableRow

3.1.5 koala.frontend package

Subpackages

koala.frontend.blueprints package

Subpackages

koala.frontend.blueprints.conventions package

Submodules

koala.frontend.blueprints.conventions.views module This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.frontend.blueprints.conventions.views.**get_iec60063_params**()

koala.frontend.blueprints.conventions.views.**get_iec60063_contextpart**(*stype*, *series*, *start=None*, *end=None*)

koala.frontend.blueprints.conventions.views.**main**(**args*, ***kwargs*)

Module contents This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.frontend.blueprints.entityhub package

Submodules

koala.frontend.blueprints.entityhub.views module This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.frontend.blueprints.entityhub.views.**cards**(**args*, ***kwargs*)

koala.frontend.blueprints.entityhub.views.**pcbs**(**args*, ***kwargs*)

koala.frontend.blueprints.entityhub.views.**projects**(**args*, ***kwargs*)

koala.frontend.blueprints.entityhub.views.**main**(**args*, ***kwargs*)

Module contents This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.frontend.blueprints.gsymlib package

Submodules

koala.frontend.blueprints.gsymlib.views module This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.frontend.blueprints.gsymlib.views.**is_geda_folder**(*path*)

class koala.frontend.blueprints.gsymlib.views.**Subfolder**(*name*, *path*)

Bases: tuple

_asdict()

Return a new OrderedDict which maps field names to their values

_fields = ('name', 'path')

classmethod _make(*iterable*, *new=<built-in method __new__ of type object at 0x90aa40>*, *len=<built-in function len>*)

Make a new Subfolder object from a sequence or iterable

_replace(*_self*, ***kwds*)

Return a new Subfolder object replacing specified fields with new values

name

Alias for field number 0

path

Alias for field number 1

`koala.frontend.blueprints.gsymlib.views.`**`get_geda_browser_context`**`(`*path*`)`

`koala.frontend.blueprints.gsymlib.views.`**`get_geda_symbol_context`**`(`*ident*`)`

`koala.frontend.blueprints.gsymlib.views.`**`get_geda_generator_context`**`(`*gen*`)`

`koala.frontend.blueprints.gsymlib.views.`**`main`**`(`**args*, ***kwargs*`)`

Module contents This file is part of koala See the COPYING, README, and INSTALL files for more information

Submodules

koala.frontend.blueprints.doc module This file is part of koala See the COPYING, README, and INSTALL files for more information

`koala.frontend.blueprints.doc.`**`root`**`(`**args*, ***kwargs*`)`

`koala.frontend.blueprints.doc.`**`static_proxy`**`(`**args*, ***kwargs*`)`

Module contents This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.frontend.pages package

Submodules

koala.frontend.pages.views module

`koala.frontend.pages.views.`**`home_page`**`()`

Module contents

koala.frontend.startup package

Submodules

koala.frontend.startup.assets module This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.frontend.startup.init_app module

`koala.frontend.startup.init_app.`**`init_app`**`(`*app*, *db*`)`
Initialize Flask applicaton

`koala.frontend.startup.init_app.`**`init_error_logger_with_email_handler`**`(`*app*`)`
Initialize a logger to send emails on error-level messages. Unhandled exceptions will now send an email message to app.config.ADMINS.

koala.frontend.startup.reset_db module

`koala.frontend.startup.reset_db.`**`reset_db`**`(`*app*, *db*`)`
Delete all tables; Create all tables; Populate roles and users.

`koala.frontend.startup.reset_db.`**`add_user`**`(`*app*, *db*, *username*, *full_name*, *email*, *password*`)`
Create UserAuth and User records.

koala.frontend.startup.settings module

Module contents

koala.frontend.users package

Submodules

koala.frontend.users.forms module

class `koala.frontend.users.forms.`**`MyRegisterForm`**(*formdata=<class flask_wtf.form._Auto>, obj=None, prefix='', csrf_context=None, secret_key=None, csrf_enabled=None, *args, **kwargs*)

Bases: `flask_user.forms.RegisterForm`

`full_name` = **<UnboundField(StringField, ('Full name',), {'validators': [<wtforms.validators.DataRequired object at 0x**

`_unbound_fields` = **None**

`_wtforms_meta` = **None**

class `koala.frontend.users.forms.`**`UserProfileForm`**(*formdata=<class flask_wtf.form._Auto>, obj=None, prefix='', csrf_context=None, secret_key=None, csrf_enabled=None, *args, **kwargs*)

Bases: `flask_wtf.form.Form`

`full_name` = **<UnboundField(StringField, ('Full name',), {'validators': [<wtforms.validators.DataRequired object at 0x**

`submit` = **<UnboundField(SubmitField, ('Save',), {})>**

`_unbound_fields` = **None**

`_wtforms_meta` = **None**

koala.frontend.users.models module

class `koala.frontend.users.models.`**`User`**(***kwargs*)

Bases: `flask_sqlalchemy.Model`, `flask_user.UserMixin`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

`id`

`email`

`confirmed_at`

`active`

`full_name`

`user_auth`

`roles`

`_sa_class_manager` = **<ClassManager of <class 'koala.frontend.users.models.User'> at 2ae429281990>**

class `koala.frontend.users.models.`**`UserAuth`**`(`***kwargs*`)`

Bases: `flask_sqlalchemy.Model`, `flask_user.UserMixin`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

`id`

`user_id`

`username`

`password`

`reset_password_token`

`active`

`user`

`_sa_class_manager` = **<ClassManager of <class 'koala.frontend.users.models.UserAuth'> at 2ae429281d08>**

class `koala.frontend.users.models.`**`Role`**`(`***kwargs*`)`

Bases: `flask_sqlalchemy.Model`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

`id`

`name`

`description`

`_sa_class_manager` = **<ClassManager of <class 'koala.frontend.users.models.Role'> at 2ae4295c2050>**

class `koala.frontend.users.models.`**`UserRoles`**`(`***kwargs*`)`

Bases: `flask_sqlalchemy.Model`

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

`id`

`user_id`

`role_id`

`_sa_class_manager` = **<ClassManager of <class 'koala.frontend.users.models.UserRoles'> at 2ae4295c22a0>**

koala.frontend.users.views module

`koala.frontend.users.views.`**`user_profile_page`**`(`**args*, ***kwargs*`)`

Module contents

Submodules

koala.frontend.app module

This file is part of koala See the COPYING, README, and INSTALL files for more information

Module contents

3.1.6 koala.gedaif package

Submodules

koala.gedaif.bomparser module

gEDA BOM Parser module documentation (`gedaif.bomparser`)

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

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

    `generate_temp_bom`(*backend*)

    `prep_temp_bom`()

    `delete_temp_bom`()

    `get_lines`()

class `koala.gedaif.bomparser.`**`MotifAwareBomParser`**(*projectfolder*, *backend*)
    Bases: *`koala.gedaif.bomparser.GedaBomParser`*

    `get_motif`(*motifst*)

    `get_lines`()

    `get_motifs`()

koala.gedaif.conffile module

gEDA ConfigsFile module documentation (`gedaif.conffile`)

exception `koala.gedaif.conffile.`**`NoGedaProjectException`**
    Bases: `exceptions.Exception`

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

    `get_configs_file`()

    `doc_folder`

    `indicative_pricing_folder`

    `projectfolder`

    `description`(*configname=None*)

    `configurations`

status

pcbdescriptors

koala.gedaif.gschem module

gEDA gschem module documentation (gedaif.gschem)

class koala.gedaif.gschem.**GschPoint** (*parent*, *x*, *y*)
Bases: *koala.utils.types.cartesian.CartesianPoint*

unit = 'mil'

parent

class koala.gedaif.gschem.**GschLine** (*parent*, *p1*, *p2*)
Bases: *koala.utils.types.cartesian.CartesianLineSegment*

parent

class koala.gedaif.gschem.**GschElementBase** (*parent*, *lines*, ***kwargs*)
Bases: object

add_element (*element*)

_get_multiline (*lines*)

parent

active_point

active_points

passive_line

passive_lines

write_out (*f*)

class koala.gedaif.gschem.**GschElementComponent** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

class koala.gedaif.gschem.**GschElementNet** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

class koala.gedaif.gschem.**GschElementBus** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

class koala.gedaif.gschem.**GschElementPin** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

class koala.gedaif.gschem.**GschElementLine** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

class koala.gedaif.gschem.**GschElementBox** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

class koala.gedaif.gschem.**GschElementCircle** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

class koala.gedaif.gschem.**GschElementArc** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

class koala.gedaif.gschem.**GschElementText** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

_get_multiline (*lines*)

class koala.gedaif.gschem.**GschElementPicture** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

_get_multiline (*lines*)

class koala.gedaif.gschem.**GschElementPath** (*parent=None*, *lines=None*, ***kwargs*)
Bases: *koala.gedaif.gschem.GschElementBase*

_get_multiline (*lines*)

class koala.gedaif.gschem.**GschFile** (*fpath*)
Bases: object

add_element (*element*)

_get_version (*lines*)

static _get_next_element (*parent*, *lines*)

_load_file ()

write_out (*f*)

koala.gedaif.gschem.**conv_gsch2pdf** (*schpath*, *docfolder*)

koala.gedaif.gschem.**conv_gsch2png** (*schpath*, *outfolder*)

koala.gedaif.gsymlib module

gEDA gsymlib Module documentation (gedaif.gsymlib)
class koala.gedaif.gsymlib.**GedaSymbol** (*fpath*)
Bases: object

_acq_sym (*fpath*)

_img_repr ()

img_repr_fname

ident

sym_ok

is_generator

is_deprecated

is_experimental

is_virtual

is_wire

is_modlen

datasheet_url

genident

genpath

generator

idents

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

`type`

`igenerators`

`ivalues`

`iunits`

`_get_data`()

`values`

`koala.gedaif.gsymlib.`**`get_folder_symbols`**(*path*, *template=None*, *resolve_generators=True*, *include_generators=False*)

`koala.gedaif.gsymlib.`**`gen_symlib`**(*path='/home/chintal/gEDA2/symbols'*, *recursive=True*, *resolve_generators=True*, *include_generators=False*)

`koala.gedaif.gsymlib.`**`_jinja_init`**()

`koala.gedaif.gsymlib.`**`get_generator`**(*gen*)

`koala.gedaif.gsymlib.`**`is_recognized`**(*ident*)

`koala.gedaif.gsymlib.`**`get_symbol`**(*ident*, *case_insensitive=False*, *get_all=False*)

`koala.gedaif.gsymlib.`**`get_symbol_folder`**(*ident*, *case_insensitive=False*)

`koala.gedaif.gsymlib.`**`find_capacitor`**(*capacitance*, *footprint*, *device='CAP CER SMD'*, *voltage=None*)

`koala.gedaif.gsymlib.`**`find_resistor`**(*resistance*, *footprint*, *device='RES SMD'*, *wattage=None*)

`koala.gedaif.gsymlib.`**`export_gsymlib_audit`**()

koala.gedaif.pcb module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.gedaif.pcb.`**`PCBPoint`**(*parent*, *x*, *y*)
Bases: *`koala.utils.types.cartesian.CartesianPoint`*

`unit` **= 'mm'**

`parent`

class `koala.gedaif.pcb.`**`PCBLine`**(*parent*, *p1*, *p2*)
Bases: *`koala.utils.types.cartesian.CartesianLineSegment`*

`parent`

class `koala.gedaif.pcb.`**`PCBElementBase`**(*tokens*)
Bases: `object`

`parent`

class `koala.gedaif.pcb.`**`PCBLayerLine`**(*tokens*)
Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBLayerArc`**(*tokens*)
Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBLayerText`**(*tokens*)
Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBLayerPolygonVertex`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBLayerPolygonHole`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBLayerPolygon`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBLayer`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBAttribute`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBSymbolLine`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBSymbol`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBElementLine`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBElementArc`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBPin`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBPad`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBElement`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBConnect`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBNet`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBNetList`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBVia`**(*tokens*)
    Bases: *`koala.gedaif.pcb.PCBElementBase`*

class `koala.gedaif.pcb.`**`PCBGrid`**(*step*, *offset_x*, *offset_y*, *visible*)
    Bases: `object`

class `koala.gedaif.pcb.`**`PCBDRC`**(*bloat*, *shrink*, *drill*, *line*, *silk*, *ring*)
    Bases: `object`

class `koala.gedaif.pcb.`**`PCBStyles`**(*string*)
    Bases: `list`

class `koala.gedaif.pcb.`**`PCBFile`**(*tokens=None*)
    Bases: `object`

    `_load_from_pptokens`(*tokens*)

class `koala.gedaif.pcb.`**`StringFlags`**(*s*)
    Bases: `object`

From : PCB file parser written by Lilith Byrant 2014 http://pastebin.com/2TqbDfKf GPLv2

`L` **= Suppress:(“(”)**

`R` **= Suppress:(”)”)**

`PartStart` **= Re:(‘[^,\\(\\)]*’)**

`CSV` **= Re:(‘[^,\\(\\)]*’) [, Re:(‘[^,\\(\\)]*’)]...**

`Part` **= Group:({Re:(‘[^,\\(\\)]*’) Group:([{{Suppress:(“(”) Re:(‘[^,\\(\\)]*’) [, Re:(‘[^,\\(\\)]*’)]...} Suppress:(”)”)}])})**

`Parts` **= Group:({Re:(‘[^,\\(\\)]*’) Group:([{{Suppress:(“(”) Re:(‘[^,\\(\\)]*’) [, Re:(‘[^,\\(\\)]*’)]...} Suppress:(”)”)}])}) [, G**

`add`(*s*)

`remove`(*s*)

`add_thermal`(*n*, *ttype*)

`remove_thermal`(*n*)

`remove_all_thermal`()

`koala.gedaif.pcb.`**`length`**(*s*, *loc*, *toks*)

`koala.gedaif.pcb.`**`introspect`**(*s*, *loc*, *toks*)

`koala.gedaif.pcb.`**`build_bnf`**(*pp=<module ‘pyparsing’ from ‘/home/chintal/code/virtualenvs/koala/lib/python2.7/site-packages/pyparsing.pyc’>*)

From : PCB file parser written by Lilith Byrant 2014 http://pastebin.com/2TqbDfKf GPLv2

`koala.gedaif.pcb.`**`conv_pcb2pdf`**(*pcbpath*, *docfolder*, *projname*)

`koala.gedaif.pcb.`**`conv_pcb2gbr`**(*pcbpath*)

`koala.gedaif.pcb.`**`conv_pcb2dxf`**(*pcbpath*, *pcbname*)

koala.gedaif.projfile module

gEDA Project File module documentation (`gedaif.projfile`)

class `koala.gedaif.projfile.`**`GedaProjectFile`**(*projectfolder*)

Bases: `object`

static **`strip_line`**(*line*)

`schpaths`

koala.gedaif.validate module

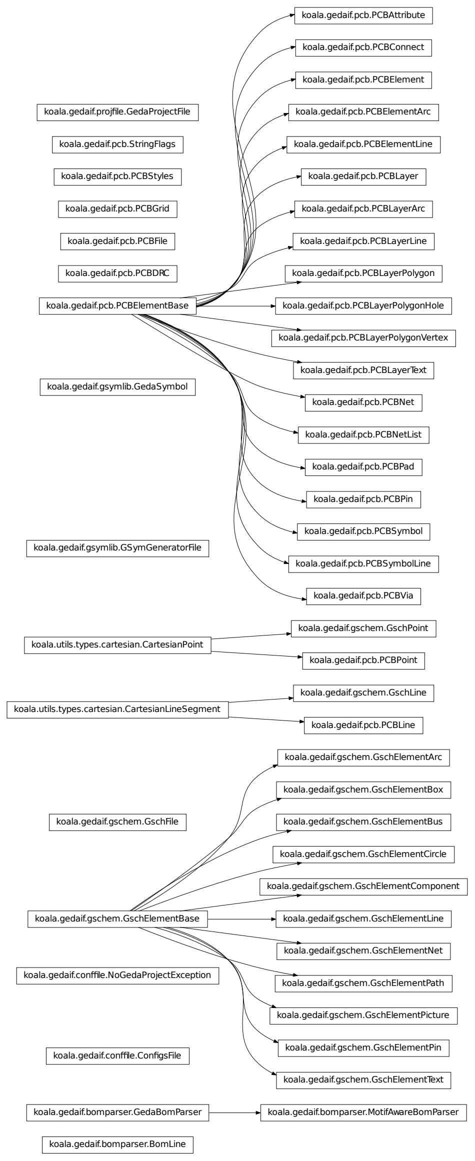
This file is part of koala See the COPYING, README, and INSTALL files for more information

`koala.gedaif.validate.`**`check`**(*projectfolder*)

Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

Inheritance Diagram


koala.gedaif.pcb.PCBAttribute
koala.gedaif.pcb.PCBConnect
koala.gedaif.pcb.PCBElement
koala.gedaif.projfile.GedaProjectFile
koala.gedaif.pcb.PCBElementArc
koala.gedaif.pcb.StringFlags
koala.gedaif.pcb.PCBElementLine
koala.gedaif.pcb.PCBStyles
koala.gedaif.pcb.PCBLayer
koala.gedaif.pcb.PCBGrid
koala.gedaif.pcb.PCBLayerArc
koala.gedaif.pcb.PCBFile
koala.gedaif.pcb.PCBLayerLine
koala.gedaif.pcb.PCBDRC
koala.gedaif.pcb.PCBLayerPolygon
koala.gedaif.pcb.PCBElementBase
koala.gedaif.pcb.PCBLayerPolygonHole
koala.gedaif.pcb.PCBLayerPolygonVertex
koala.gedaif.pcb.PCBLayerText
koala.gedaif.gsymlib.GedaSymbol
koala.gedaif.pcb.PCBNet
koala.gedaif.pcb.PCBNetList
koala.gedaif.pcb.PCBPad
koala.gedaif.pcb.PCBPin
koala.gedaif.pcb.PCBSymbol
koala.gedaif.gsymlib.GSymGeneratorFile
koala.gedaif.pcb.PCBSymbolLine
koala.gedaif.pcb.PCBVia
koala.gedaif.gschem.GschPoint
koala.utils.types.cartesian.CartesianPoint
koala.gedaif.pcb.PCBPoint
koala.gedaif.gschem.GschLine
koala.utils.types.cartesian.CartesianLineSegment
koala.gedaif.pcb.PCBLine
koala.gedaif.gschem.GschElementArc
koala.gedaif.gschem.GschElementBox
koala.gedaif.gschem.GschFile
koala.gedaif.gschem.GschElementBus
koala.gedaif.gschem.GschElementCircle
koala.gedaif.gschem.GschElementComponent
koala.gedaif.gschem.GschElementBase
koala.gedaif.gschem.GschElementLine
koala.gedaif.gschem.GschElementNet
koala.gedaif.conffile.NoGedaProjectException
koala.gedaif.gschem.GschElementPath
koala.gedaif.gschem.GschElementPicture
koala.gedaif.conffile.ConfigsFile
koala.gedaif.gschem.GschElementPin
koala.gedaif.gschem.GschElementText
koala.gedaif.bomparser.GedaBomParser
koala.gedaif.bomparser.MotifAwareBomParser
koala.gedaif.bomparser.BomLine

3.1.7 koala.inventory package

Subpackages

koala.inventory.db package

Submodules

koala.inventory.db.controller module This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.inventory.db.controller.**get_inventorylocationcode**(*name*, *create=False*)

koala.inventory.db.controller.**populate_inventorylocationcodes**()

koala.inventory.db.model module This file is part of koala See the COPYING, README, and INSTALL files for more information

class koala.inventory.db.model.**InventoryLocationCode**(***kwargs*)

Bases: *koala.utils.db.BaseMixin*, sqlalchemy.ext.declarative.api.Base

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

name

_sa_class_manager = <ClassManager of <class 'koala.inventory.db.model.InventoryLocationCode'> at 2ae42a7e5868

id

Module contents This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.inventory.guidelines package

Module contents This file is part of koala See the COPYING, README, and INSTALL files for more information

Submodules

koala.inventory.acquire module

Inventory Acquire Module documentation (inventory.acquire)

class koala.inventory.acquire.**InventoryReaderBase**(*location*, *tfpath*)

Bases: object

class koala.inventory.acquire.**InventoryDBReader**(*location*, *tfpath*)

Bases: *koala.inventory.acquire.InventoryReaderBase*

class koala.inventory.acquire.**StockXlsReader**(*xlf*, *sname*, *location*, *tfpath*)

Bases: *koala.inventory.acquire.InventoryReaderBase*

_skip_to_header()

_get_cols(*row*)

static _is_balance(*item*)

_row_gen()

_tf_row_gen()

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

koala.inventory.acquire.**get_reader**(*elec_inven_data_idx*) `koala.inventory.acquire.`**`gen_canonical_transform`**(*elec_inven_data_idx*, *regen=True*)

koala.inventory.electronics module

Electronics Inventory module documentation (`inventory.electronics`)

class `koala.inventory.electronics.`**`InventoryLine`**(*ident*, *qty=None*, *parent=None*)

Bases: `object`

`ident`

`avail_qty`

`reserved_qty`

`reserve_qty`(*value*, *earmark*)

`earmarks`

`_reservation_gen`()

`get_reservation_gen`()

class `koala.inventory.electronics.`**`InventoryLocation`**(*name*, *dname*, *reader*)

Bases: `object`

`_get_code`()

`name`

`_load_from_reader`()

`get_ident_qty`(*ident*)

`get_reserve_qty`(*ident*)

`reserve_ident_qty`(*ident*, *qty*, *earmark*)

`earmarks`

`_reservation_gen`()

`get_reservation_gen`()

`koala.inventory.electronics.`**`init_inventory_locations`**()

`koala.inventory.electronics.`**`get_total_availability`**(*ident*)

`koala.inventory.electronics.`**`get_total_reservations`**(*ident*)

`koala.inventory.electronics.`**`reserve_items`**(*ident*, *qty*, *earmark*, *die_if_not=True*)

`koala.inventory.electronics.`**`export_reservations`**(*folderpath*)

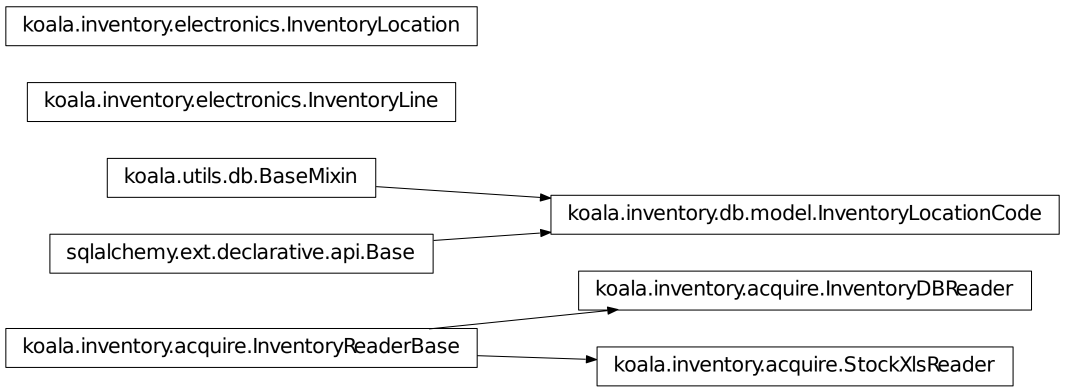
koala.inventory.modules module

This file is part of koala See the COPYING, README, and INSTALL files for more information

Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

Inheritance Diagram


koala.inventory.electronics.InventoryLocation
koala.inventory.electronics.InventoryLine
koala.utils.db.BaseMixin
koala.inventory.db.model.InventoryLocationCode
sqlalchemy.ext.declarative.api.Base
koala.inventory.acquire.InventoryDBReader
koala.inventory.acquire.InventoryReaderBase
koala.inventory.acquire.StockXlsReader

3.1.8 koala.scripts package

Submodules

koala.scripts.gencostsummary module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.scripts.gendox module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.scripts.genpcbpricing module

This file is part of koala See the COPYING, README, and INSTALL files for more information

`koala.scripts.genpcbpricing.`**`generate`**`()`

`koala.scripts.genpcbpricing.`**`flushpcbpricing`**`()`

koala.scripts.genvmapaudits module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.scripts.genvmaps module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.scripts.order module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.scripts.production module

This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.scripts.validate module

This file is part of koala See the COPYING, README, and INSTALL files for more information

Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

3.1.9 koala.sourcing package

Submodules

koala.sourcing.csil module

CSIL Sourcing Module documentation (`sourcing.csil`)

`koala.sourcing.csil.`**`get_credentials`**`()`

`koala.sourcing.csil.`**`get_csil_prices`**`(`*params={'layers': 2, 'finish': 'Au', 'pcbname': 'QASC-', 'dY': '151', '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], 'dX': '109', 'time': 10}, rval=None*`)`

class `koala.sourcing.csil.`**`VendorCSIL`**`(`*name, dname, pclass, mappath=None, currency_code=None, currency_symbol=None, username=None, password=None*`)`

Bases: *`koala.sourcing.vendors.VendorBase`*

`search_vpnos``(`*ident*`)`

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

`get_optimal_pricing``(`*ident, rqty*`)`

`_generate_purchase_order``(`*path*`)`

class `koala.sourcing.csil.`**`CSILPart`**`(`*vpartno, ident, vendor*`)`

Bases: *`koala.sourcing.vendors.VendorPartBase`*

`_load_descriptors``()`

`_load_prices``()`

`descriptors`

`get_price``(`*qty*`)`

koala.sourcing.csil.**flush_pcb_pricing**(*projfolder*)

koala.sourcing.csil.**generate_pcb_pricing**(*projfolder*, *noregen=True*, *forceregen=False*)

koala.sourcing.customs module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class koala.sourcing.customs.**CustomsSection**(*code*, *hsdict*)
Bases: object

_load_duties(*dutiesdict*)

code

name

desc

idents

folders

class koala.sourcing.customs.**CustomsClassifier**
Bases: object

_load_sections()

hs_from_ident(*ident*)

class koala.sourcing.customs.**CustomsInvoice**(*vendor*, *inv_yaml*, *working_folder=None*)
Bases: *koala.sourcing.vendors.VendorInvoice*

given_data

_process_other_costs()

insurance

includes_freight

landing

cif

added_insurance

added_handling

source_folder

working_folder

source_files

idxs

_acquire_lines()

hssections

getsection_lines(*hssection*)

getsection_idxs(*hssection*)

getsection_qty(*hssection*)

getsection_assessabletotal(*hssection*)

unclassified

assessabletotal

bcd

cvd

cec

cshec

cvdec

cvdshec

acvd

dutypayable

effectiverate_cif

effectiverate_fob

class koala.sourcing.customs.**DutyComponent**(*title*, *rate*, *notification*, *value*)

Bases: object

class koala.sourcing.customs.**CustomsInvoiceLine**(*invoice*, *ident*, *vpno*, *unitp*, *qty*, *idx=None*, *desc=None*)

Bases: *koala.sourcing.vendors.VendorInvoiceLine*

dutypayable

idx

hs_section

bcd

cvd

cec

cshec

cvdec

cvdshec

acvd

invoice_fraction

freight

insurance

cifprice

handling

assessableprice

print_duties()

koala.sourcing.digikey module

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

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

Bases: *`koala.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_tolerance_to_canonical`(*tstr*)

static `_tf_capacitance_to_canonical`(*cstr*)

static `_tf_package_tant_smd`(*footprint*)

static `_tf_package_crystal_at`(*footprint*)

`_get_searchpage_filters`(*soup*)

static `_get_default_urlparams`()

static `_get_searchurl_res_smd`()

static `_get_searchurl_res_thru`()

static `_get_searchurl_cap_cer_smd`()

static `_get_searchurl_cap_tant_smd`()

static **_get_searchurl_crystal**()

_get_searchurl_filters(*searchurl*)

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

class koala.sourcing.digikey.**DigiKeyElnPart**(*dkpartno*, *ident=None*, *vendor=None*)

Bases: *koala.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*)

class koala.sourcing.digikey.**DigiKeyInvoice**(*vendor=None*, *inv_yaml=None*, *working_folder=None*)

Bases: *koala.sourcing.customs.CustomsInvoice*

_acquire_lines()

koala.sourcing.electronics module

Electronics Sourcing module documentation (sourcing.electronics)

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

koala.sourcing.electronics.**init_vendors**()

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

exception koala.sourcing.electronics.**SourcingException**

Bases: exceptions.Exception

koala.sourcing.electronics.**get_eff_acq_price**(*vsinfo*) `koala.sourcing.electronics.`**`get_sourcing_information`**(*ident, qty, avendors=[<koala.sourcing.digikey.VendorDigiKey object at 0x2ae42f07c7d0>, <koala.sourcing.csil.VendorCSIL object at 0x2ae42f0a46d0>, <koala.sourcing.pricelist.VendorPricelist object at 0x2ae42f0a4890>, <koala.sourcing.pricelist.VendorPricelist object at 0x2ae42f772ad0>, <koala.sourcing.pricelist.VendorPricelist object at 0x2ae42f718790>, <koala.sourcing.pricelist.VendorPricelist object at 0x2ae42f74b610>, <koala.sourcing.pricelist.VendorPricelist object at 0x2ae42f8561d0>], allvendors=False*)

`koala.sourcing.electronics.`**`get_vendor_by_name`**(*name*)

koala.sourcing.orders module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.sourcing.orders.`**`CompositeOrderElem`**(*order*, *ident*, *rqty*, *shortage*)

Bases: `object`

static **`get_eff_acq_price`**(*source*)

`ident`

`in_gsymlib`

`rqty`

`resqty`

`shortage`

`gl_compl_qty`

`sources`

`selsource`

`sorted_other_sources`

`other_sources`

`excess`(*vsinfo*)

`is_sourceable`

`order`()

class `koala.sourcing.orders.`**`CompositeOrder`**(*vendor_list=None*, *orderref=None*)

Bases: `object`

`_columns` **= [('Component Details', [('Ident', None), ('In gsymlib', None)]), ('Requirement', [('Required', 'Qty'), ('Reser**

`orderref`

`add`(*ident*, *rqty*, *shortage*, *orderref=None*)

`_get_headers`(*row*)

_render_vsinfo(*vsinfo*, *row*, *basereq=None*)

_render_line(*line*, *writer*, *include_others*)

dump_to_file(*fname*, *include_others=True*)

generate_orders(*path*)

rebalance()

collapse()

koala.sourcing.pricelist module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class koala.sourcing.pricelist.**VendorPricelist**(*name*, *dname*, *pclass*, *mappath=None*, *currency_code=None*, *currency_symbol=None*, *pricelistpath=None*)

Bases: `koala.sourcing.vendors.VendorBase`

_load_pricecsv(*fname*)

_generate_insert_idents()

_get_generator_idents(*pricegen*)

static _get_generator_values(*pricegen*)

TODO This function should be parcelled out to conventions :param gendata: :return:

search_vpnos(*ident*)

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

_pl_get_vpart_dict(*vpartno*)

get_optimal_pricing(*ident*, *rqty*)

class koala.sourcing.pricelist.**PricelistPart**(*vp_dict*, *ident*, *vendor*)

Bases: `koala.sourcing.vendors.VendorPartBase`

_get_prices(*vp_dict*)

get_price_qty(*qty*)

class koala.sourcing.pricelist.**AnalogDevicesInvoice**(*vendor=None*, *inv_yaml=None*, *working_folder=None*)

Bases: `koala.sourcing.customs.CustomsInvoice`

_acquire_lines()

koala.sourcing.vendors module

Vendors module documentation (`sourcing.vendors`)

class koala.sourcing.vendors.**VendorInvoiceLine**(*invoice*, *ident*, *vpno*, *unitp*, *qty*, *desc=None*)

Bases: `object`

desc

ident

vpno

unitprice

extendedprice

effectiveprice

qty

class `koala.sourcing.vendors.`**VendorInvoice**(*vendor*, *inv_no*, *inv_date*)

Bases: `object`

vendor_name

currency

inv_no

inv_date

linecount

lines

extendedtotal

effectivetotal

_acquire_lines()

class `koala.sourcing.vendors.`**VendorOrder**(*vendor*, *orderref*)

Bases: `object`

add(*line*)

lines

orderref

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

Bases: `object`

name

pclass

mappath

map

currency

get_vpnos(*ident*)

search_vpnos(*ident*)

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

get_optimal_pricing(*ident*, *rqty*)

add_order_additional_cost_component(*desc*, *percent*)

get_effective_price(*price*)

get_additional_costs(*price*)

order_baseprice

add_order_baseprice_component(*desc*, *value*)

add_to_order(*line*, *orderref=None*)

_dump_open_order(*path*)

_generate_purchase_order(*path*)

finalize_order(*path*)

class koala.sourcing.vendors.**VendorPrice**(*moq*, *price*, *currency_def*, *oqmultiple=1*)
Bases: object

moq

oqmultiple

unit_price

extended_price(*qty*)

class koala.sourcing.vendors.**VendorPartBase**(*ident*, *vendor*)
Bases: object

add_price(*price*)

vpno

vqtyavail

manufacturer

mpartno

vpartdesc

ident

pkgqty

abs_moq

get_price(*qty*)

class koala.sourcing.vendors.**VendorElnPartBase**(*ident*, *vendor*)
Bases: *koala.sourcing.vendors.VendorPartBase*

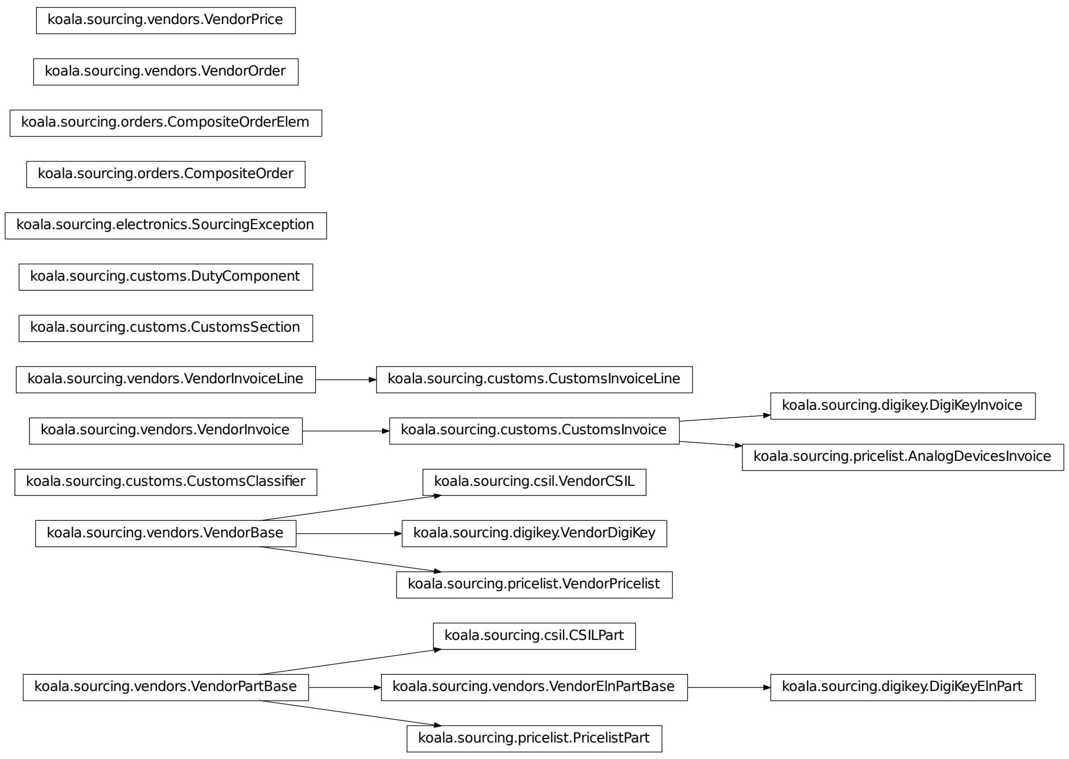
package

datasheet

Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

Inheritance Diagram


koala.sourcing.vendors.VendorPrice
koala.sourcing.vendors.VendorOrder
koala.sourcing.orders.CompositeOrderElem
koala.sourcing.orders.CompositeOrder
koala.sourcing.electronics.SourcingException
koala.sourcing.customs.DutyComponent
koala.sourcing.customs.CustomsSection
koala.sourcing.vendors.VendorInvoiceLine
koala.sourcing.customs.CustomsInvoiceLine
koala.sourcing.digikey.DigiKeyInvoice
koala.sourcing.vendors.VendorInvoice
koala.sourcing.customs.CustomsInvoice
koala.sourcing.pricelist.AnalogDevicesInvoice
koala.sourcing.customs.CustomsClassifier
koala.sourcing.csil.VendorCSIL
koala.sourcing.vendors.VendorBase
koala.sourcing.digikey.VendorDigiKey
koala.sourcing.pricelist.VendorPricelist
koala.sourcing.csil.CSILPart
koala.sourcing.vendors.VendorPartBase
koala.sourcing.vendors.VendorElnPartBase
koala.sourcing.digikey.DigiKeyElnPart
koala.sourcing.pricelist.PricelistPart

3.1.10 koala.testing package

Subpackages

koala.testing.instruments package

This file is part of koala See the COPYING, README, and INSTALL files for more information

`koala.testing.instruments.`**`get_instrument_object`**(*instst*)

Submodules

RadioShack 2200087 DMM Interface Module (`koala.testing.instruments.RS2200087`) This module provides the instrument object for RadioShack's 2200087 Digital Multimeter with PC interface. It uses the `driver2200087` module to handle the communication with the instrument, while subclassing the Twisted protocol of that module to provide one which produces *`koala.utils.types.signalbase.SignalPoint`* and *`koala.utils.types.signalbase.SignalWave`* objects instead, which contain Type objects from the *`koala.utils.types`* module. This allows seamless integration with Koala's *`koala.testing`* module.

Usage example

```
>>> from crochet import setup
>>> setup()
>>> from koala.testing.instruments import get_instrument_object
>>> o = get_instrument_object('RS2200087')
>>> o.channel.get()
>>> o.channel.reset_wave()
>>> wave = o.channel.get_next_chunk()
>>> wave += o.channel.get_next_chunk()
```

Module Contents

Classes

InstrumentRS2200087()	This is the primary class provided by this module, and the object generated by *koa*
DMMInputChannel(parent, interface)	This class provides the bulk of the accessors to the instrument for downstream use.
KoalaProtocol2200087(port, buffer_size)	This subclasses the twisted protocol from `driver2200087.runner` which han
KoalaFactory2200087()	This factory produces protocols integrated with Koala unit classes, producing Signa
factory	Module's instance of the Protocol Factory.

Processors

rex_list	List of regular expressions, each of which matches the string for one type of data from `driver`
voltage_processor(m)	Processor that converts a regex match of a Voltage string from `driver2200087.serialDe`
resistance_processor(m)	Processor that converts a regex match of a Resistance string from `driver2200087.serial`
capacitance_processor(m)	Processor that converts a regex match of a Capacitance string from `driver2200087.seria`
frequency_processor(m)	Processor that converts a regex match of a Frequency string from `driver2200087.serial`
time_processor(m)	Processor that converts a regex match of a Time string from `driver2200087.serialDeco`
current_processor(m)	Processor that converts a regex match of a Current string from `driver2200087.serialDe`
power_processor(m)	Processor that converts a regex match of a PowerRatio string from `driver2200087.seria`
duty_processor(m)	Processor that converts a regex match of a DutyCycle string from `driver2200087.serial`
hfe_processor(m)	Processor that converts a regex match of a HFR string from `driver2200087.serialDeco`
continuity_processor(m)	Processor that converts a regex match of a Continuity string from `driver2200087.serial`

`koala.testing.instruments.RS2200087.`**`voltage_processor`**`(`*m*`)`

Processor that converts a regex match of a Voltage string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

Parameters **m** – `re.match` object from one of the Voltage regexs in *rex_list*

Returns String compatible with the *koala.utils.types.electromagnetic.Voltage* class and its subclasses

Return type str

`koala.testing.instruments.RS2200087.`**`resistance_processor`**`(`*m*`)`

Processor that converts a regex match of a Resistance string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

Parameters **m** – `re.match` object from one of the Resistance regexs in *rex_list*

Returns String compatible with the *koala.utils.types.electromagnetic.Resistance* class and its subclasses

Return type str

`koala.testing.instruments.RS2200087.`**`capacitance_processor`**`(m)`
Processor that converts a regex match of a Capacitance string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

Parameters **m** – `re.match` object from one of the Capacitance regexs in *rex_list*

Returns String compatible with the *koala.utils.types.electromagnetic.Capacitance* class and its subclasses

Return type str

`koala.testing.instruments.RS2200087.`**`frequency_processor`**`(m)`
Processor that converts a regex match of a Frequency string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

Parameters **m** – `re.match` object from one of the Frequency regexs in *rex_list*

Returns String compatible with the *koala.utils.types.time.Frequency* class and its subclasses

Return type str

`koala.testing.instruments.RS2200087.`**`time_processor`**`(m)`
Processor that converts a regex match of a Time string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

Parameters **m** – `re.match` object from one of the Time regexs in *rex_list*

Returns String compatible with the *koala.utils.types.time.TimeSpan* class and its subclasses

Return type str

`koala.testing.instruments.RS2200087.`**`current_processor`**`(m)`
Processor that converts a regex match of a Current string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

Parameters **m** – `re.match` object from one of the Current regexs in *rex_list*

Returns String compatible with the *koala.utils.types.electromagnetic.Current* class and its subclasses

Return type str

`koala.testing.instruments.RS2200087.`**`power_processor`**`(m)`
Processor that converts a regex match of a PowerRatio string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

Parameters **m** – `re.match` object from one of the PowerRatio regexs in *rex_list*

Returns String compatible with the *koala.utils.types.electromagnetic.PowerRatio* class and its subclasses

Return type str

`koala.testing.instruments.RS2200087.`**`hfe_processor`**`(m)`
Processor that converts a regex match of a HFR string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

Parameters **m** – `re.match` object from one of the HFE regexs in *rex_list*

> **Returns** String compatible with the *`koala.utils.types.electromagnetic.HFE`* class and its subclasses
>
> **Return type** str

`koala.testing.instruments.RS2200087.`**`duty_processor`**`(`*m*`)`

Processor that converts a regex match of a DutyCycle string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

> **Parameters** **`m`** – `re.match` object from one of the DutyCycle regexs in *`rex_list`*
>
> **Returns** String compatible with the *`koala.utils.types.electromagnetic.DutyCycle`* class and its subclasses
>
> **Return type** str

`koala.testing.instruments.RS2200087.`**`continuity_processor`**`(`*m*`)`

Processor that converts a regex match of a Continuity string from `driver2200087.serialDecoder` and returns a string compatible with Koala Unit Types

> **Parameters** **`m`** – `re.match` object from one of the Continuity regexs in *`rex_list`*
>
> **Returns** String compatible with the *`koala.utils.types.electromagnetic.Continuity`* class and its subclasses
>
> **Return type** str

`koala.testing.instruments.RS2200087.`**`rex_list`** **= [(<class 'koala.utils.types.thermodynamic.Temperature'>, <_sr**

List of regular expressions, each of which matches the string for one type of data from `driver2200087.serialDecoder`. Each element of the list is a *tuple*, containing the following elements:

0.Koala type class from *`koala.utils.types`* which is applicable to the data point.

1.The compiled regular expression.

2.The applicable processor, which acts on the match object to produce a string. If the processor is None, then the named match group 'string' is used to instantiate the appropriate unit class.

class `koala.testing.instruments.RS2200087.`**`KoalaProtocol2200087`**`(`*port*, *buffer_size*`)`

Bases: `driver2200087.runner.InstProtocol2200087`

This subclasses the twisted protocol from `driver2200087.runner` which handles serial communications with 2200087 multimeters. It produces *`koala.utils.types.signalbase.SignalPoint`* and *`koala.utils.types.signalbase.SignalWave`* objects instead of strings and deque objects, which contain Type objects from the *`koala.utils.types`* module.

This protocol exists and operates within the context of a twisted reactor. Applications themselves built on twisted should be able to simply import this protocol (or its factory).

Synchronous / non-twisted applications should directly use the `driver2200087.runner.InstInterface2200087` class instead, and pass this protocol's factory to modify its behavior.

> **Parameters**
>
> - **`port`** (*str*) – Port on which the device is connected. Default '/dev/ttyUSB0'.
> - **`buffer_size`** (*int*) – Length of the point buffer in the protocol.

`reset_buffer``(`*unitclass=<class 'koala.utils.types.unitbase.DummyUnit'>*`)`

Resets the point buffer to a new *`koala.utils.types.signalbase.SignalWave`* with the unitclass as specified by the parameter. Any data presently within it will be lost.

> **Parameters** **`unitclass`** – Class of Unit that the Wave points are composed of.

next_chunk()

Returns The next chunk of data points in the form of a SignalWave

Return type *koala.utils.types.signalbase.SignalWave*

static _get_point(*string*)

Processes a string returned by driver2200087.serialDecoder and converts it into a *koala.utils.types.signalbase.SignalPoint* instance composed of the correct Unit class.

Returns SignalPoint composed of the correct type and value from the string

Return type *koala.utils.types.signalbase.SignalPoint*

See also:

rex_list

frame_received(*frame*)

Re-implements the Base class's frame_received function, producing SignalPoints instead. When a signal point of a different type as the point buffer is encountered, it resets the point buffer and initializes it to the new type.

See also:

_get_point()

class koala.testing.instruments.RS2200087.**KoalaFactory2200087**

Bases: driver2200087.runner.InstFactory2200087

This factory produces protocols integrated with Koala unit classes, producing SignalPoints and SignalWaves instead of strings.

See the driver2200087.runner.InstFactory2200087 documentation for more detailed information.

buildProtocol(*port='/dev/ttyUSB0'*, *buffer_size=100*)

This function returns a KoalaProtocol2200087 instance, bound to the port specified by the param port.

Parameters

- **port** (*str*) – Serial port identifier to which the device is connected
- **buffer_size** (*int*) – Length of the point buffer in the protocol. Default 100.

koala.testing.instruments.RS2200087.**factory = <koala.testing.instruments.RS2200087.KoalaFactory2200087 ins**

Module's instance of the Protocol Factory. This should be used whenever the Protocol class is to be instantiated (as opposed to subclassed)

class koala.testing.instruments.RS2200087.**DMMInputChannel**(*parent*, *interface*)

Bases: *koala.testing.instrumentbase.InstrumentInputChannelBase*

This class provides the bulk of the accessors to the instrument for downstream use.

Parameters

- **parent** – The instrument of which this channel is a part of.
- **interface** – The interface to the instrument.

get(*unitclass=None*, *flush=True*)

Gets the latest data point in the channel's point buffer. By default, it also flushes any older points from the buffer. This behavior can be suppressed by passing flush=False.

Parameters

- **unitclass** – Unit class of which the data point is expected, or None if you don't care.

- **flush** – Whether or not older points should be flushed. Default True.

Returns Latest datapoint.

Return type *koala.utils.types.signalbase.SignalPoint*

get_next_chunk(*unitclass=None*)
Gets the next chunk of data from the instrument channel. Note that the chunk returned has already been removed from the channel's point buffer.

Parameters **unitclass** – Unit class of which the data point is expected, or None if you don't care.

Returns Wave containing all but the latest data point in the channel's point buffer.

Return type *koala.utils.types.signalbase.SignalWave*

reset_wave()
Resets the point / wave buffer in the channel

class koala.testing.instruments.RS2200087.**InstrumentRS2200087**
Bases: *koala.testing.instrumentbase.InstrumentBase*

This is the primary class provided by this module, and the object generated by *koala.testing.instruments.get_instrument_object()* is an instance of this class. All downstream Koala application code interfaces with this class, and through it, with *DMMInputChannel*.

The primary code access to this class is though it's *channel* property.

Instantiating this object results in the calling of *_detect()*, which prepares the object for use.

Variables **_dmm** – The underlying instrument interface object

_detect()
Creates and initializes the `driver2200087.runner.InstInterface2200087` object, which in turn creates the protocol object and establishes the connection to the device.

The instrument interface object created is stored in the object's `_dmm` instance variable.

Subsequently, a *DMMInputChannel* instance is instantiated and linked to the interface.

configure(*configuration*)

channel

Returns The IntrumentInputChannel object.

Return type *DMMInputChannel*

reset_waves()
Resets the point / wave buffers in the instrument's channels

koala.testing.tests package

Module contents This file is part of koala See the COPYING, README, and INSTALL files for more information

koala.testing.tests.**get_test_object**(*testst*)

Submodules

koala.testing.autotest module

This file is part of koala See the COPYING, README, and INSTALL files for more information

`koala.testing.autotest.`**`register_autotest`**`(`*ident*, *testsuite*`)`

koala.testing.instrumentbase module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.testing.instrumentbase.`**`InstrumentChannelBase`**`(`*parent*`)`
Bases: `object`

class `koala.testing.instrumentbase.`**`InstrumentInputChannelBase`**`(`*parent*`)`
Bases: *`koala.testing.instrumentbase.InstrumentChannelBase`*

`get``()`

class `koala.testing.instrumentbase.`**`InstrumentOutputChannelBase`**`(`*parent*`)`
Bases: *`koala.testing.instrumentbase.InstrumentChannelBase`*

`set``(`*signal*`)`

class `koala.testing.instrumentbase.`**`InstrumentBase`**`(`*channels=None*`)`
Bases: `object`

`_detect``()`

`configure``(`*configuration*`)`

`configurations`

`channels`

koala.testing.testbase module

This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.testing.testbase.`**`TestPrepBase`**`(`*parent*`)`
Bases: `object`

Object representing a preparatory step for a Test

`run_prep``()`

class `koala.testing.testbase.`**`TestMeasurementBase`**`(`*parent*`)`
Bases: `object`

Object representing a single measurement for a Test

`do_measurement``()`
This is an example measurement function. This should be overridden by the actual Test classes to perform the actual measurement, and this code can be used as a starting point.

The result of the measurement would typically be some composition of instances of `koala.utils.type.signalbase.SignalBase`.

`render``()`
This is an example render function. This should be overridden by the actual Test classes to render the actual result.

Rendering means encoding the test result into a JSON representation, which can later be dumped into a postgres database.

class `koala.testing.testbase.`**`RunnableTest`**
Bases: `object`

run_test()

commit_results()

class koala.testing.testbase.**TestBase**(*parent*)

Bases: *koala.testing.testbase.RunnableTest*

Object representing a full runnable Test of the same Measurement type

run_test()

commit_results()

class koala.testing.testbase.**TestSuiteBase**

Bases: *koala.testing.testbase.RunnableTest*

Object representing a full runnable Test Suite on a single entity

add_test(*test*)

run_test()

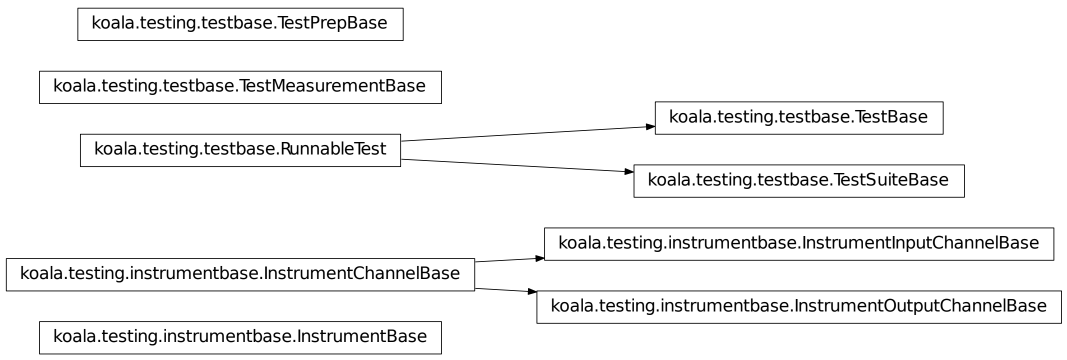
commit_results()

koala.testing.testrunner module

Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

Inheritance Diagram


koala.testing.testbase.TestPrepBase
koala.testing.testbase.TestMeasurementBase
koala.testing.testbase.RunnableTest
koala.testing.testbase.TestBase
koala.testing.testbase.TestSuiteBase
koala.testing.instrumentbase.InstrumentChannelBase
koala.testing.instrumentbase.InstrumentInputChannelBase
koala.testing.instrumentbase.InstrumentOutputChannelBase
koala.testing.instrumentbase.InstrumentBase

3.1.11 The Utils Package (`koala.utils`)

The `utils` package contains various Utils modules which provide functionality that is either :

- Logically independent of the main Koala functionality.
- Used by multiple Koala's packages, either due to generic nature of the functionality, or because it defines the structure that functionality should take within Koala.

These modules typically expose specific functionality of third-party or built-in Python modules and simplify access by restricting the ways in which these modules can be used. For instance, various options are preselected within the Utils module, meaning application code does not have to bother about it. This comes at the cost of reducing the flexibility available when using this functionality in application code.

The intent of the `koala.utils` package and it's modules is to attempt to create a consistent structure throughout the `koala` codebase. The `koala.utils` module, in effect, specifies the preferred way to do certain common things. Koala application code which makes use of the functionality of this module is expected to benefit in the form of consistency, relative ease in honoring user-provided parameters (across the codebase), and not need to deal with the underlying libraries and the wide range of usage options available for the most common cases.

It is expected that this structure will be critical to being able to easily adapt the Koala codebase to new environments and for making more flexibility accessible to administrators of Koala instances.

Many of the modules provided here can be re-implemented to use different underlying implementations, while preserving the interface they present to Koala application code.

Warning: It is possible that this approach will soon reach the point of diminishing returns.

The Utility Modules

The Progressbar Package (`koala.utils.progressbar`)

This package produces animated progress bars on the terminal. It is based on the one written by Nadia Alramli in 2009, made available under the BSD License. This license is retained for this package (`koala.utils.progressbar`) and it's submodules. The changes to the original code are minimal.

Submodules

`progressbar.progressbar`

Draws an animated terminal progress bar.

```
>>> p = ProgressBar("blue")
>>> p.render(percentage, message)
```

class `koala.utils.progressbar.progressbar.`**`ProgressBar`**(*color=None*, *width=None*, *block='xe2x96x88'*, *empty=' '*)

Bases: `object`

Terminal progress bar class

Parameters

- **`color`** (*str*) – Color name (BLUE GREEN CYAN RED MAGENTA YELLOW WHITE BLACK)
- **`width`** (*numbers.Number*) – Bar width (optional)
- **`block`** (*char*) – Progress display character (default '')
- **`empty`** (*char*) – Bar display character (default ' ')

`TEMPLATE` = **'%(percent)-2s%% %(color)s%(progress)s%(normal)s%(empty)s %(message)s\n'**

`PADDING` = **7**

render (*percent*, *message=''*)
Print the progress bar.

Parameters

- **percent** (*int*) – The progress percentage (0-100)
- **message** (*str*) – Message string (optional)

clear ()
Clear all printed lines

progressbar.terminal

Terminal controller module

```
>>> print BG_BLUE + 'Text on blue background' + NORMAL
>>> print BLUE + UNDERLINE + 'Blue underlined text' + NORMAL
>>> print BLUE + BG_YELLOW + BOLD + 'text' + NORMAL
```

koala.utils.progressbar.terminal.**default** ()
Set the default attribute values

koala.utils.progressbar.terminal.**setup** ()
Set the terminal control strings

koala.utils.progressbar.terminal.**render** (*text*)
Helper function to apply controls easily

The Types Package (koala.utils.types)

The `koala.utils.types` package, like all the `koala.utils` modules, includes code that ideally resides outside of Koala itself, perhaps using some standard or third-party package. In the case of `koala.utils.types`, this package provides various modules and consequently classes to handle special data types - usually data structures and units. These use cases are more thoroughly and effectively handled by various third-party packages, and its existence in Koala is largely due to a combination of the *Not Invented Here* syndrome, as well as the overwhelming number of possible options available, none of which seem to be ideal drop in replacements for Koala core's requirements.

While you can use the units and data structures from `koala.utils.types` independent of the rest of Koala, you probably should not. The code itself should function, but it lacks the thoroughness of type-checking or the efficiency of more full fledged implementations. Even when using Koala units to handle data generated by one of the Koala submodules, I strongly recommend using a full-fledged unit package instead. If you'd like to use them anyway, well, you have been warned.

Converters from Koala types to other forms should be reasonably straightforward to write, and over time some these converters and/or re-implementations using an established types/units package will hopefully find their way into the classes defined within `koala.utils.types`.

The general principles of design used for the units defined within this package are as follows :

- If there is an obvious good-fit python package available to provide the units or data structures required, use that instead and simply proxy to that package within this module. Data-structures are relatively more complex beasts, so the remaining principles don't apply to them. They follow more of a *use your best judgement* type of development principle.
- Unit Type classes should, as far as possible, derive from `unitbase.UnitBase`.

- The fundamental nature of unit instantiation must remain stable and consistent within the various units defined. The underlying implementation isn't important as long as it provides the functionality required via the accessors Koala expects. More importantly, changing the underlying implementation should not change the way application code uses the units.
- In most cases, primary unit instantiation is using a string containing the value the object should hold as well as the unit. Units can also be instantiated using a `numbers.Number` instance, but application code should avoid doing this. This interface is provided largely to support arithmetic operations defined between units.
- For each 'simple' unit class, unless provided by an external library, the core information should be a number in the "canonical" unit of that class. The class should also store the original string.
- Wherever possible, the numbers stored should be `decimal.Decimal` instances and not floats. Number of significant digits should ideally be tracked, but presently is not.
- The unit classes should support the bare-minimum set of valid operations. As far as possible, the simple mathematical operations performed with two elements of the same unitclass should produce a valid result (if physically meaningful)
- The unit classes should not produce physically meaningless results. This is not something that's tested all that much, so it's likely corner cases have slipped through the cracks.
- Be unafraid to throw exceptions. If application code misuses the units, the application is asking to be penalized.

Warning: No effort is made to have a complete set of units, or even a complete set of units of the same dimensionality as the supported units. Addition of new types is done on a lazy basis, as are type conversions, inter-unit arithmetic, and other ranges. This policy is likely to change only if this module is pulled out of Koala into it's own units library, and maintaining that isn't something I'm likely to do alone.

Types

Currency Types (`koala.utils.types.currency`) The *`koala.utils.types.currency`* contains classes which allow for easy use and manipulation of currency values. The primary focus is on the primary use cases of Currencies within koala, i.e. :

- Handling foreign exchange conversions and exchange rates in application code without too much fuss.
- Handling currency arithmetic and comparisons.

This module uses a specific *Base Currency*, defined by `koala.utils.config.BASE_CURRENCY` and `koala.utils.config.BASE_CURRENCY_SYMBOL` and available as this module's *`native_currency_defn`* module variable. In case this module is to be used independent of Koala, at least those configuration options *must* be defined in *`koala.utils.config`*.

Module Contents

`native_currency_defn`	The native currency definition used by the module
`CurrencyDefinition`(code[, symbol, exchval])	Instances of this class define a currency.
`CurrencyValue`(val, currency_def)	Instances of this class define a specific currency value, or a certain sum of mor

See also:

`koala.utils.types`, for an overview applicable to most types defined in Koala.

Todo

The core numbers in this module need to switched to `decimal.Decimal`.

---

class `koala.utils.types.currency.`**`CurrencyDefinition`**(*code*, *symbol=None*, *exchval=None*)

Bases: `object`

Instances of this class define a currency.

The minimal requirement to define a currency is a *`code`*, which would usually be a standard internationally recognized currency code.

In addition to the *`code`*, a currency definition also includes an optional *`symbol`*, which is used to create string representations of currency values in that currency. In the absence of a *`symbol`*, the *`code`* is used in it's place.

Unless otherwise specified during the instantiation of the class, the exchange rate is obtained from internet services by the *`_get_exchval()`* method.

Parameters

- **`code`** – Standard currency code.
- **`symbol`** – Symbol to use to represent the currency. Optional.
- **`exchval`** – Exchange rate to use, if not automatic. Optional.

`code`

Returns The currency code.

Return type str

`symbol`

Returns The currency symbol, or code if no symbol.

Return type str

`exchval`

Returns The exchange rate

Return type float

static **`_get_exchval`**(*code*)

Obtains the exchange rate of the currency definition's *`code`* using the http://jsonrates.com JSON API. The native currency is used as the reference.

Parameters **`code`** (*str*) – The currency code for which the exchange rate is needed.

Returns The exchange rate of currency specified by code vs the native currency.

Return type float

Todo

Switch to currencyrates API instead.

`__eq__`(*other*)

Two instances of *`CurrencyDefinition`* will evaluate to be equal only when all three parameters of the instances are equal.

`koala.utils.types.currency.`**`native_currency_defn`** **= <koala.utils.types.currency.CurrencyDefinition object>**

The native currency definition used by the module

This definition uses the code contained in `koala.utils.config.BASE_CURRENCY` and symbol `koala.utils.config.BASE_CURRENCY_SYMBOL`. Application code should import this definition instead of creating new currency definitions whenever one is needed to represent a native currency value.

class `koala.utils.types.currency.`**`CurrencyValue`**(*val*, *currency_def*)

Bases: `object`

Instances of this class define a specific currency value, or a certain sum of money.

The *currency_def* can either be a *`CurrencyDefinition`* instance (recommended), or a string containing the code for the currency.

Parameters

- **`val`** – The numerical value.
- **`currency_def`** (*`CurrencyDefinition`* or str) – The currency definition within which the value is defined.

Note: Since the exchange rate is obtained at the instantiation of the *`CurrencyDefinition`*, using a string instead of a predefined *`CurrencyDefinition`* instance may result in instances of the same currency, but with different exchange rates.

Variables

- **`_currency_def`** – The currency definition of the source value of the instance.
- **`_val`** – The numerical value in the source currency of the instance.

Arithmetic Operations

`__add__`(other)	Addition of two *`CurrencyValue`* instances returns a *`CurrencyValue`* instance with the sum of the two oper
`__sub__`(other)	Subtraction of two *`CurrencyValue`* instances returns a *`CurrencyValue`* instance with the difference of the t
`__mul__`(other)	Multiplication of one *`CurrencyValue`* instance with a numerical type results in a *`CurrencyValue`* instance,
`__div__`(other)	Division of one *`CurrencyValue`* instance with a numerical type results in a *`CurrencyValue`* instance, whos
`__cmp__`(other)	The comparison of two *`CurrencyValue`* instances behaves identically to the comparison of the operands' *`nat`*

`native_value`

The numerical value of the currency value in the native currency, i.e., that defined by *`native_currency_defn`*.

Return type float

`native_string`

The string representation of the currency value in the native currency, i.e., that defined by *`native_currency_defn`*.

Return type str

`source_value`

The numerical value of the currency value in the source currency, i.e., that defined by *`source_currency`*.

Return type float

`source_string`

The string representation of the currency value in the source currency, i.e., that defined by *`source_currency`*.

Return type str

source_currency
The currency definition of the source currency, i.e, the instance variable `_currency_def`.

Return type *CurrencyDefinition*

__add__(*other*)
Addition of two *CurrencyValue* instances returns a *CurrencyValue* instance with the sum of the two operands, with currency conversion applied if necessary.

If the *source_currency* of the two operands are equal, the result uses the the same *source_currency*. If not, the result is uses the *native_currency_defn* as it's *source_currency*.

If the other operand is a numerical type and evaluates to 0, this object is simply returned unchanged.

Addition with all other Types / Classes is not supported.

Return type *CurrencyValue*

__mul__(*other*)
Multiplication of one *CurrencyValue* instance with a numerical type results in a *CurrencyValue* instance, whose value is is the currency type operand's value multiplied by the numerical operand's value.

The *source_currency* of the returned *CurrencyValue* is the same as that of the currency type operand.

Multiplication with all other Types / Classes is not supported.

Return type *CurrencyValue*

__div__(*other*)
Division of one *CurrencyValue* instance with a numerical type results in a *CurrencyValue* instance, whose value is is the currency type operand's value divided by the numerical operand's value.

The *source_currency* of the returned *CurrencyValue* is the same as that of the currency type operand. In this case, the first operand must be a *CurrencyValue*, and not the reverse.

Division of one *CurrencyValue* instance by another returns a numerical value, which is obtained by performing the division with the operands' *native_value*.

Division with all other Types / Classes is not supported.

Return type *CurrencyValue*

__sub__(*other*)
Subtraction of two *CurrencyValue* instances returns a *CurrencyValue* instance with the difference of the two operands, with currency conversion applied if necessary.

If *source_currency* of the two operands are equal, the result uses the the same *source_currency*. If not, the result is in the *native_currency_defn*.

If the other operand is a numerical type and evaluates to 0, this object is simply returned unchanged.

Subtraction with all other Types / Classes is not supported.

Return type *CurrencyValue*

__cmp__(*other*)
The comparison of two *CurrencyValue* instances behaves identically to the comparison of the operands' *native_value*.

Comparison with all other Types / Classes is not supported.

koala.utils.types.lengths module This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.utils.types.lengths.`**`Length`**(*lstr=None*, *length=None*, *defunit='mm'*)

Bases: `object`

`_parse_length`()

`decimal`

`_length`

`lstr`

Base Unit Types (`koala.utils.types.unitbase`) The Types provided in this module are not intended for direct use. Instead, they provide reusable primitives which can be sub-classed to provide functional Types.

Ideally, all Unit classes should derive from the *`UnitBase`* class provided here. In practice, only the newer Types follow this inheritance, while the older ones still need to be migrated to this form.

Module Contents

`UnitBase`(value, _dostr, _parse_func)	The base class for all *`koala.utils.types`* units.
`NumericalUnitBase`(value, _orders, _dostr, ...)	The base class for all *`koala.utils.types`* numerical units.
`DummyUnit`()	This class provides a type for placeholder objects.
`parse_none`(value)	A placeholder parse function which can be used if the Unit requires / supports r
`parse_percent`(value)	A parse function for use by the *`Percentage`* Type and its subclasses.
`Percentage`(value)	A base Unit class which provides support for Types that are essentially percenta

See also:

`koala.utils.types`, for an overview applicable to most types defined in Koala.

class `koala.utils.types.unitbase.`**`UnitBase`**(*value*, *_dostr*, *_parse_func*)

Bases: `object`

The base class for all *`koala.utils.types`* units.

When instantiated, the *value* param is processed as follows :

- `str` value is passed though *_parse_func*, and whatever it returns is stored.
- `numbers.Number` values are first converted into `decimal.Decimal` and then stored.
- All other *value* types are simply stored as is.

Parameters

- **`value`** – The core value to be stored.
- **`_dostr`** – The canonical unit / order string to use.
- **`_parse_func`** – The function used to parse string values into an actual value in the canonical unit.

Arithmetic Operations

This class supports no arithmetic operations.

value

Returns The core value of the Unit instance, in it's canonical unit.

__add__(*other*)

__mul__(*other*)

__div__(*other*)

__sub__(*other*)

__cmp__(*other*)

class koala.utils.types.unitbase.**NumericalUnitBase**(*value*, *_orders*, *_dostr*, *_parse_func*)
Bases: *koala.utils.types.unitbase.UnitBase*

The base class for all *koala.utils.types* numerical units.

Provides the patterns used by the various Numerical Units to provide their functionality. This class represents and implements the core ideas that remain valid across Units (for the most part). The various methods and functions implemented here establish the minimum required functionality and behaviour expected of all numerical units.

Specific numerical unit classes may override the methods present here to tweak the implementation and/or the interface as per the requirements of the quantity they represent, as long as they stay true to the spirit of the architecture.

Parameters

- **value** – The core value to be stored.
- **_orders** – The recognized orders / units for the Unit.
- **_dostr** – The canonical unit / order string to use.
- **_parse_func** – The function used to parse string values into an actual value in the canonical unit.

See also:

UnitBase

The *orders* Parameter

The *orders* parameter can either be a list of strings or a list of tuples.

- In case it is a *list* of *str*, it is assumed that each string represents a unit value 1000 times smaller than the next.
- In case it is a *list* of *tuple*, it is assumed that the first element of the tuple is the string representation of the order, and the second element is the multipicative factor relative to the default order string.
- In both cases, note that first order within which the unit value's representation lies between 1 and 1000 is used to produce the unit's string representation. As such, you should place higher priority or more 'standard' units towards the beginning of the list.

Arithmetic Operations

__add__(other)	Addition of two Unit class instances of the same type returns a Unit class instance of the same type, with the sum

Table 3.7 – continued from previous

`__sub__`(other)	Subtraction of two Unit class instances of the same type returns a Unit class instance of the same type, with the di
`__mul__`(other)	Multiplication of one Unit type class instance with a numerical type results in a Unit type class instance of the sam
`__div__`(other)	Division of one Unit type class instance with a numerical type results in a Unit type class instance of the same typ
`__cmp__`(other)	The comparison of two Unit type class instances of the same type behaves identically to the comparison of the op

`__add__`(*other*)

Addition of two Unit class instances of the same type returns a Unit class instance of the same type, with the sum of the two operands as it's value.

If the other operand is a numerical type and evaluates to 0, this object is simply returned unchanged.

Addition with all other Types / Classes is not supported.

`__mul__`(*other*)

Multiplication of one Unit type class instance with a numerical type results in a Unit type class instance of the same type, whose value is the Unit type operand's value multiplied by the numerical operand's value.

Multiplication with all other Types / Classes is not supported.

`__div__`(*other*)

Division of one Unit type class instance with a numerical type results in a Unit type class instance of the same type, whose value is the Unit type operand's value divided by the numerical operand's value.

In this case, the first operand must be a Unit type class instance, and not the reverse.

Division of one Unit type class instance by another of the same type returns a numerical value, which is obtained by performing the division with the operands' value.

Division with all other Types / Classes is not supported.

`__sub__`(*other*)

Subtraction of two Unit class instances of the same type returns a Unit class instance of the same type, with the difference of the two operands as it's value.

If the other operand is a numerical type and evaluates to 0, this object is simply returned unchanged.

Subtraction with all other Types / Classes is not supported.

`__cmp__`(*other*)

The comparison of two Unit type class instances of the same type behaves identically to the comparison of the operands' values.

Comparison with all other Types / Classes is not supported.

class `koala.utils.types.unitbase.`**`DummyUnit`**

Bases: *`koala.utils.types.unitbase.UnitBase`*

This class provides a type for placeholder objects. The original use case is for handling wave boundaries in streaming protocols.

Does not support any arithmetic operations.

`__add__`(*other*)

`__mul__`(*other*)

`__div__`(*other*)

`__sub__`(*other*)

`__cmp__`(*other*)

`koala.utils.types.unitbase.`**`parse_none`**`(`*value*`)`
A placeholder parse function which can be used if the Unit requires / supports no parsing.

`koala.utils.types.unitbase.`**`parse_percent`**`(`*value*`)`
A parse function for use by the *`Percentage`* Type and its subclasses.

class `koala.utils.types.unitbase.`**`Percentage`**`(`*value*`)`
Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

A base Unit class which provides support for Types that are essentially percentages.

The contribution this base class makes is to be able to parse percentage strings so that the Descendant class need not.

Only the standard *`NumericalUnitBase`* Arithmetic is supported by this class at this time.

koala.utils.types.electromagnetic module This file is part of koala See the COPYING, README, and INSTALL files for more information

`koala.utils.types.electromagnetic.`**`parse_resistance`**`(`*value*`)`

`koala.utils.types.electromagnetic.`**`parse_capacitance`**`(`*value*`)`

`koala.utils.types.electromagnetic.`**`parse_voltage`**`(`*value*`)`

`koala.utils.types.electromagnetic.`**`parse_current`**`(`*value*`)`

`koala.utils.types.electromagnetic.`**`parse_dbm`**`(`*value*`)`

`koala.utils.types.electromagnetic.`**`parse_hfe`**`(`*value*`)`

class `koala.utils.types.electromagnetic.`**`Resistance`**`(`*value*`)`
Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

class `koala.utils.types.electromagnetic.`**`Capacitance`**`(`*value*`)`
Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

class `koala.utils.types.electromagnetic.`**`Voltage`**`(`*value*`)`
Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

class `koala.utils.types.electromagnetic.`**`VoltageAC`**`(`*value, _orders, _dostr, _parse_func*`)`
Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

class `koala.utils.types.electromagnetic.`**`VoltageDC`**`(`*value*`)`
Bases: *`koala.utils.types.electromagnetic.Voltage`*

class `koala.utils.types.electromagnetic.`**`DiodeVoltageDC`**`(`*value*`)`
Bases: *`koala.utils.types.electromagnetic.VoltageDC`*

class `koala.utils.types.electromagnetic.`**`Current`**`(`*value*`)`
Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

class `koala.utils.types.electromagnetic.`**`CurrentAC`**`(`*value*`)`
Bases: *`koala.utils.types.electromagnetic.Current`*

class `koala.utils.types.electromagnetic.`**`CurrentDC`**`(`*value*`)`
Bases: *`koala.utils.types.electromagnetic.Current`*

class `koala.utils.types.electromagnetic.`**`PowerRatio`**`(`*value*`)`
Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

class `koala.utils.types.electromagnetic.`**`HFE`**`(`*value*`)`
Bases: *`koala.utils.types.unitbase.UnitBase`*

class `koala.utils.types.electromagnetic.`**`Continuity`**(*value*)
 Bases: *`koala.utils.types.unitbase.UnitBase`*

class `koala.utils.types.electromagnetic.`**`DutyCycle`**(*value*)
 Bases: *`koala.utils.types.unitbase.Percentage`*

koala.utils.types.thermodynamic module

`koala.utils.types.thermodynamic.`**`parse_temperature`**(*value*)

class `koala.utils.types.thermodynamic.`**`Temperature`**(*value*)
 Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

koala.utils.types.time module This file is part of koala See the COPYING, README, and INSTALL files for more information

`koala.utils.types.time.`**`parse_frequency`**(*string*)

class `koala.utils.types.time.`**`Frequency`**(*value*)
 Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

class `koala.utils.types.time.`**`TimeSpan`**(*value*)
 Bases: *`koala.utils.types.unitbase.NumericalUnitBase`*

 `timedelta`

class `koala.utils.types.time.`**`TimeStamp`**(*year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None*)
 Bases: `arrow.arrow.Arrow`

class `koala.utils.types.time.`**`TimeDelta`**(*years=0, months=0, days=0, hours=0, minutes=0, seconds=0, microseconds=0*)
 Bases: `object`

Data Structures / Composite Types

koala.utils.types.signalbase module

class `koala.utils.types.signalbase.`**`SignalBase`**(*unitclass*)
 Bases: `object`

 `unitclass`

class `koala.utils.types.signalbase.`**`SignalErrorBar`**(*quantization=None, noise_floor=None, error_pc=None*)
 Bases: `object`

class `koala.utils.types.signalbase.`**`SignalPoint`**(*unitclass, value, ts=None*)
 Bases: *`koala.utils.types.signalbase.SignalBase`*

 `error_bar`

class `koala.utils.types.signalbase.`**`SignalWave`**(*unitclass, points=None, spacing=None, ts0=None, interpolation='piecewise_linear', buffer_size=None, use_point_ts=True*)
 Bases: *`koala.utils.types.signalbase.SignalBase`*

 `last_timestamp`

 `points`

 `add_point`(*point, ts=None*)

 `clear`()

popleft(**args*, ***kwargs*)

pop(**args*, ***kwargs*)

append(**args*, ***kwargs*)

appendleft(**args*, ***kwargs*)

extend(**args*, ***kwargs*)

extendleft(**args*, ***kwargs*)

koala.utils.types.cartesian module This file is part of koala See the COPYING, README, and INSTALL files for more information

class `koala.utils.types.cartesian.`**CartesianPoint**(*x*, *y*)
Bases: `object`

unit = 'mm'

static _norm_repr(*v*)

class `koala.utils.types.cartesian.`**CartesianLineSegment**(*p1*, *p2*)
Bases: `object`

x(*t*)

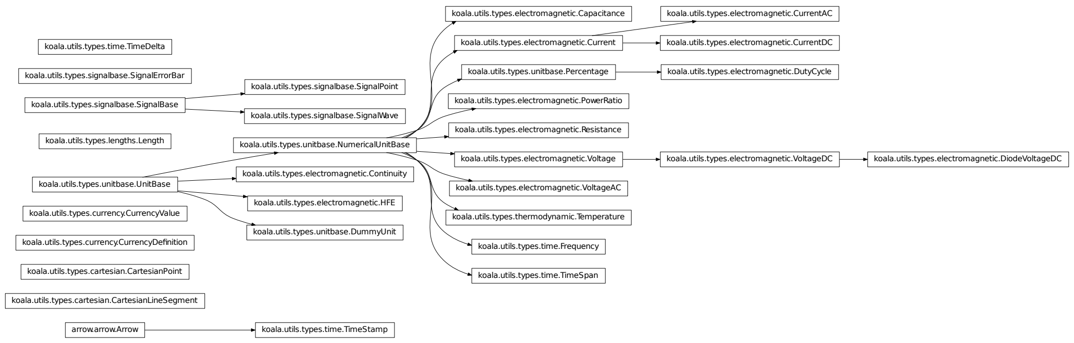
y(*t*)

t_x(*x*)

t_y(*y*)

length()

Inheritance Diagram


koala.utils.types.time.TimeDelta
koala.utils.types.signalbase.SignalErrorBar
koala.utils.types.signalbase.SignalBase
koala.utils.types.signalbase.SignalPoint
koala.utils.types.signalbase.SignalWave
koala.utils.types.lengths.Length
koala.utils.types.unitbase.NumericalUnitBase
koala.utils.types.unitbase.UnitBase
koala.utils.types.electromagnetic.Continuity
koala.utils.types.electromagnetic.HFE
koala.utils.types.unitbase.DummyUnit
koala.utils.types.currency.CurrencyValue
koala.utils.types.currency.CurrencyDefinition
koala.utils.types.cartesian.CartesianPoint
koala.utils.types.cartesian.CartesianLineSegment
arrow.arrow.Arrow
koala.utils.types.time.TimeStamp
koala.utils.types.electromagnetic.Capacitance
koala.utils.types.electromagnetic.Current
koala.utils.types.unitbase.Percentage
koala.utils.types.electromagnetic.PowerRatio
koala.utils.types.electromagnetic.Resistance
koala.utils.types.electromagnetic.Voltage
koala.utils.types.electromagnetic.VoltageAC
koala.utils.types.thermodynamic.Temperature
koala.utils.types.time.Frequency
koala.utils.types.time.TimeSpan
koala.utils.types.electromagnetic.CurrentAC
koala.utils.types.electromagnetic.CurrentDC
koala.utils.types.electromagnetic.DutyCycle
koala.utils.types.electromagnetic.VoltageDC
koala.utils.types.electromagnetic.DiodeVoltageDC

The Colors Module (`koala.utils.colors`)

This module provides RGB color palettes, which can be used by application code for rendering things.

The color palette available is *`tableau20`*, a palette from Tableau.

For more information on how this is intended to be used and the original source of the code, see How to make beautiful data visualizations in Python with matplotlib, written by Randal S. Olson.

Note: This module is released under the MIT license. See the included *LICENSE.txt* for information about *koala*'s licensing structure.

koala.utils.colors.**tableau20 = [(0.12156862745098039, 0.4666666666666667, 0.7058823529411765), (0.682352941176**

The *tableau20* color palette [(R,G,B)]

The Config Module (`koala.utils.config`)

This module provides the core configuration for Koala.

The Koala configuration file is itself a Python file, and is imported from it's default location.

This module performs the import using the *koala.utils.fs.import_()* function and the `imp` module, following which all the recognized *and* expected configuration options are imported from the instance configuration module into this namespace.

Todo

The present implementation is fairly silly and tedious. A better implementation is necessary.

Todo

The actual configuration options themselves need to be documented.

The Database Utils Module (`koala.utils.db`)

This module provides utilities to deal with Koala's Database. While the actual functionality is provided by the `sqlalchemy` package, the contents of this utility module simplify and specify the application code's interaction with `sqlalchemy`

Module Contents

koala.utils.db.**init_db_engine**()

Initializes the database engine and binds it to the Database URI defined by the *koala.utils.config* module.

This function is called within the module and an engine is readily available in the module variable *koala.utils.db.engine*. Application code should not have to create a new engine for the normal use cases.

koala.utils.db.**engine = Engine(postgresql://koala:***@localhost:5432/koala_qtpl)**

The `sqlalchemy.Engine` object

koala.utils.db.**Session = sessionmaker(class_='Session',autoflush=True, bind=Engine(postgresql://koala:***@localhos**

The `sqlalchemy.sessionmaker` bound to the database engine

koala.utils.db.**get_session**(**args*, ***kwds*)

Application executable code will typically only have to interact with this `contextmanager`. It should use this to create a database session, perform its tasks, whatever they may be, within this contex, and then exit the context.

koala.utils.db.**with_db**(*func*)

class `koala.utils.db.`**`BaseMixin`**

Bases: `object`

This Mixin can / should be used (by inheriting from) by all Model classes defined by application code. It defines the `__tablename__` attribute of the Model class to the name of the class and creates a Primary Key Column named id in the table for the Model.

`id` = Column(None, Integer(), table=None, primary_key=True, nullable=False)

`koala.utils.db.`**`DeclBase`**

The `sqlalchemy` declarative base for all Model defined by / in Koala

alias of `Base`

class `koala.utils.db.`**`TimestampMixin`**

Bases: `object`

This Mixin can be used by any Models which require a timestamp to be created. It adds a column named `created_at`, which defauts to the time at which the object is instantiated.

`created_at` = Column(None, DateTime(), table=None, default=ColumnDefault(datetime.datetime(2015, 8, 4, 18, 4, 42, 7

`koala.utils.db.`**`commit_metadata`**`()`

This function commits all metadata to the table. This function should be run after importing **all** the Model classes, and it will create the tables in the database.

The Filesystem Utils Module (`koala.utils.fs`)

This module provides utilities to deal with filesystems. For the most part, this module basically proxies specific requests to various other third-party or python libraries.

Module Contents

`TEMPDIR`	The path to the temporary directory which all application code can in
`fsutils_cleanup`()	Called when the python interpreter is shutting down.
`zipdir`(path, zfpath)	Creates a zip file at `zfpath` containing all the files in `path`.
`Crumb`(name, path)	A named tuple definition for a Crumb of a Breadcrumb.
`get_path_breadcrumbs`(path[, base, rootst])	Given a certain filesystem `path` and an optional `base`, this function
`get_folder_mtime`(folder)	Given the path to a certain filesystem `folder`, this function returns a
`get_file_mtime`(f)	Given the path to a certain filesystem `file`, this function returns a `d`
`VersionedOutputFile`(pathname[, numSavedVersions])	Create a new output file.
`import_`(fpath)	Imports the file specified by the `fpath` parameter using the `imp` pyt

`koala.utils.fs.`**`TEMPDIR`** **= '/tmp/tmpkbdXNj'**

The path to the temporary directory which all application code can import, and create whatever temporary files it needs within it.

This directory will be removed by Koala at clean application exit or by the Operating System as per it's policies.

Every execution of koala in a separate process owns it's own temporary directory.

See also:

`fsutils_cleanup()`

`koala.utils.fs.`**`zipdir`**`(`*path*, *zfpath*`)`

Creates a zip file at `zfpath` containing all the files in `path`. This function is simple wrapper around python's `zipfile` module.

Parameters

- **`path`** – Path of the source folder, which is to be added to the zip file.
- **`zfpath`** – Path of the zip file to create.

Returns The path of the created zip file.

class `koala.utils.fs.`**`Crumb`**`(`*name*, *path*`)`

Bases: `tuple`

A named tuple definition for a Crumb of a Breadcrumb. This can be used to construct breadcrumb navigation by application code. While it resides in the *`koala.utils.fs`* module, the same type should be used for other forms of breadcrumbs as well. It is placed here due to its proximity to `os.path`.

`_asdict``()`

Return a new OrderedDict which maps field names to their values

`_fields` **= ('name', 'path')**

classmethod **`_make`**`(`*iterable*, *new=<built-in method __new__ of type object at 0x90aa40>*, *len=<built-in function len>*`)`

Make a new Crumb object from a sequence or iterable

`_replace``(`*_self*, ***kwds*`)`

Return a new Crumb object replacing specified fields with new values

`name`

Alias for field number 0

`path`

Alias for field number 1

`koala.utils.fs.`**`get_path_breadcrumbs`**`(`*path*, *base=None*, *rootst='Root'*`)`

Given a certain filesystem `path` and an optional `base`, this function returns a list of *`Crumb`* objects, forming the breadcrumbs to that path from the base. The value of `rootst` is prepended to the list, providing a means to insert a title indicating the base of the breadcrumb list.

Parameters

- **`path`** (*str*) – The path to the target, compatible with `os.path`
- **`base`** (*str*) – The path of the base, compatible with `os.path`. Optional.
- **`rootst`** (*str*) – The string for the root breadcrumb.

Returns The breadcrumbs.

Return type `list` of *`Crumb`*

`koala.utils.fs.`**`get_folder_mtime`**`(`*folder*`)`

Given the path to a certain filesystem `folder`, this function returns a `datetime.datetime` instance representing the time of the latest change of any file contained within the folder.

Parameters **`folder`** (*str*) – The path to the `folder`, compatible with `os.path`

Returns The time of the latest change within the `folder`

Return type `datetime.datetime`

See also:

`get_file_mtime()`

`koala.utils.fs.`**`get_file_mtime`**(*f*)

Given the path to a certain filesystem `file`, this function returns a `datetime.datetime` instance representing the time of the latest change of that file.

Parameters **`f`** (*str*) – The path to the `file`, compatible with `os.path`

Returns The time of the latest change within the `folder`

Return type `datetime.datetime`

See also:

`get_folder_mtime()`

class `koala.utils.fs.`**`VersionedOutputFile`**(*pathname*, *numSavedVersions=3*)

Create a new output file.

Parameters

- **`pathname`** – The name of the file to [over]write.
- **`numSavedVersions`** – 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 `_numSavedVersions` versions are retained.

`koala.utils.fs.`**`fsutils_cleanup`**()

Called when the python interpreter is shutting down. Cleans up all *koala.utils.fs* related objects and other artifacts created by the module.

Performs the following tasks:

- Removes the *`TEMPDIR`*

`koala.utils.fs.`**`import_`**(*fpath*)

Imports the file specified by the `fpath` parameter using the `imp` python module and returns the loaded module.

Parameters **`fpath`** – Path of the python module to import.

Returns Module object of the imported python module.

The LibreOffice Utils Module (`koala.utils.libreoffice`)

This module provides utilities to deal with LibreOffice files. Functionality is implemented lazily, so this doesn't really do much.

Presently, whatever it does do, it does by relying on external python code, executed in a shell by `subprocess`. This is required since the default LibreOffice install I have only supports a python 3 API.

For details about how it actually works, look at the documentation of the `sofficehelpers` package. The scripts used from that package are :

- `sofficehelpers.ssconverter` for converting spreadsheets to CSV files.

Todo

Publish the `sofficehelpers` package somewhere, or find a pre-existing alternative. If there is a functionally sufficient alternative, it will probably be better.

`koala.utils.libreoffice.`**`open_files`** **= []**

A list of all files currently open which the module is responsible for.

class `koala.utils.libreoffice.`**`XLFile`**(*filepath*)

Bases: `object`

This class allows reading data from LibreOffice supported Spreadsheet files, by converting each sheet into a CSV file, which can then be read by application code. Note that this is a one way conversion only. The original spreadsheet file is not written to.

When the object is instantiated, a copy of the file is made in the temporary folder by *`_make_copy()`*, which is then converted into one CSV file per sheet in the original spreadsheet file by *`_make_csv_files()`*. It then adds itself to the modules *`open_files`* list.

Parameters **`filepath`** – The path of the spreadsheet to be opened.

Variables

- **`fpath`** – The path to the original spreadsheet file.
- **`fname`** – The file name of the original spreadsheet file.
- **`_csv_list`** – The list of CSV files generated from the spreadsheet, one per sheet.

`get_csv_path`(*sheetname*)

Gets the path to the CSV file that was created at object initialization corresponding to the sheetname provided.

Parameters **`sheetname`** – Name of the Spreadsheet Sheet.

`_make_copy`()

Makes a copy of the original file in the temporary directory

`_make_csv_files`()

Converts a SpreadSheet file supported by LibreOffice into a set of CSV files, one per sheet, using the `sofficehelpers.ssconverter` script. The output is parsed by *`_parse_sscout()`*.

`_parse_sscout`(*string*)

Parses the output of the `sofficehelpers.ssconverter` script, extracting the Sheet names and corresponding CSV file names from the output.

`close`()

Closes the object and removes it from the *`open_files`* list. Also runs *`_clean()`* to remove all the files it created.

`_clean`()

Removes all temporary files created during the object instantiation.

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

Gets the *`XLFile`* object for the spreadsheet at `fpath`, *opening* the file and creating the object if it does not already exist.

Parameters **`fpath`** – Path of the spreadsheet file to open.

Returns The opened *`XLFile`* instance.

`koala.utils.libreoffice.`**`ooutils_cleanup`**`()`

Called when the python interpreter is shutting down. Cleans up all *koala.utils.libreoffice* related objects and other artifacts created by the module.

Performs the following tasks:

- Closes all *`open_files`*

The Log Utils Module (`koala.utils.log`)

This module provides utilities to deal with logging systems. The intent of having this module instead of using `logging` directly is to allow the injection of various default parameters and options into all the loggers used from a central place, instead of littering them throughout the modules.

At present, this module does nothing that is overly useful, except for being able to set the default log level for all modules simultaneously.

Usage Example

```
>>> from koala.utils import log
>>> logger = log.get_logger(__name__, log.DEFAULT)
```

`koala.utils.log.`**`DEFAULT`** **= 20**

The default log level for all loggers created through this module, unless otherwise specified at the time of instantiation.

`koala.utils.log.`**`init`**`()`

`koala.utils.log.`**`get_logger`**`(`*name*, *level*`)`

Get a logger with the specified `name` and an optional `level`.

The levels from the python `logging` module can be used directly. For convenience, these levels are imported into this module's namespace as well, along with the *`DEFAULT`* level this module provides.

See python `logging` documentation for information about log levels.

Parameters

- **`name`** (*str*) – The name of the logger
- **`level`** (*int*) – Log level of the logger to be used. Default : *`DEFAULT`*.

Returns The logger instance

The PDF Utils Module (`koala.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.

koala.utils.pdf.**merge_pdf**(*pdflist*, *outfilepath*, *remove_sources=False*)

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

koala.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.

The Persistent State Utils Module (koala.utils.state)

This module uses `dataset` to provide a means to store persistent state information. Note that this means of storing persistent information is under review, since most of the present usages will benefit from full database integration. This will likely be replaced by integrating it into the application database using the interfaces in `koala.utils.db` instead.

The primary storage is an sqlite3 database, located within the INSTANCE_ROOT. This storage is accessible by application code by importing this module's `state_ds` module variable, and the documentation of `dataset` should be used interact directly with this object.

koala.utils.state.**state_ds = <Database(sqlite:////home/chintal/.koala/db/state.db)>**

A `dataset.Database` instance, attached to the sqlite3 database located at `db/state.db` under the instance root, defined by :data:`utils.config.INSTANCE_ROOT`

The WWW Utils Module (koala.utils.www)

This module provides utilities to deal with the internet. All application code should access the internet through this module, since this where support for proxies and caching is implemented.

Main Provided Methods

strencode(string)	This function converts unicode strings to ASCII, using python's `str.encode()`, replacing any unicode char
urlopen(url)	Opens a url specified by the `url` parameter.
get_soup(url)	Gets a `bs4` parsed soup for the `url` specified by the parameter.

This module uses the following configuration values from `koala.utils.config`:

Basic Settings

- `koala.utils.config.ENABLE_REDIRECT_CACHING` Whether or not redirect caching should be used.
- `koala.utils.config.TRY_REPLICATOR_CACHE_FIRST` Whether or not a replicator cache should be used.

Redirect caching speeds up network accesses by saving `301` and `302` redirects, and not needing to get the correct URL on a second access. This redirect cache is stored as a pickled object in the `INSTANCE_CACHE` folder. The effect of this caching is far more apparent when a replicator cache is also used.

Network Proxy Settings

- `koala.utils.config.NETWORK_PROXY_TYPE`
- `koala.utils.config.NETWORK_PROXY_IP`
- `koala.utils.config.NETWORK_PROXY_PORT`
- `koala.utils.config.NETWORK_PROXY_USER`
- `koala.utils.config.NETWORK_PROXY_PASS`

Replicator Cache Settings

The replicator cache is intended to be a `http-replicator` instance, to be used to cache the web pages that are accessed locally. If `TRY_REPLICATOR_CACHE_FIRST` is False, the replicator isn't actually going to be hit.

- `koala.utils.config.REPLICATOR_PROXY_TYPE`
- `koala.utils.config.REPLICATOR_PROXY_IP`
- `koala.utils.config.REPLICATOR_PROXY_PORT`
- `koala.utils.config.REPLICATOR_PROXY_USER`
- `koala.utils.config.REPLICATOR_PROXY_PASS`

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

This function converts unicode strings to ASCII, using python's `str.encode()`, replacing any unicode characters present in the string. Unicode characters which Koala expects to see in web content related to it are specifically replaced first with ASCII characters or character sequences which reasonably reproduce the original meanings.

Parameters **`string`** – unicode string to be encoded.

Returns ASCII version of the string.

`koala.utils.www.`**`dump_redirect_cache`**`()`

Called during python interpreter shutdown, this function dumps the redirect cache to the file system.

class `koala.utils.www.`**`CachingRedirectHandler`**

Bases: `urllib2.HTTPRedirectHandler`

This handler modifies the behavior of `urllib2.HTTPRedirectHandler`, resulting in a HTTP `301` or `302` status to be included in the `result`.

When this handler is attached to a `urllib2` opener, if the opening of the URL resulted in a redirect via HTTP `301` or `302`, this is reported along with the result. This information can be used by the opener to maintain a redirect cache.

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

Wraps the `urllib2.HTTPRedirectHandler.http_error_301()` handler, setting the `result.status` to `301` in case a http `301` error is encountered.

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

Wraps the `urllib2.HTTPRedirectHandler.http_error_302()` handler, setting the `result.status` to `302` in case a http `302` error is encountered.

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

Tests an opener obtained using `urllib2.build_opener()` by attempting to open Google's homepage. This is used to test internet connectivity.

`koala.utils.www.`**`_create_opener`**`()`

Creates an opener for the internet.

It also attaches the *`CachingRedirectHandler`* to the opener and sets its User-agent to `Mozilla/5.0`.

If the Network Proxy settings are set and recognized, it creates the opener and attaches the proxy_handler to it. The opener is tested and returned if the test passes.

If the test fails an opener without the proxy settings is created instead and is returned instead.

`koala.utils.www.`**`_create_replicator_opener`**`()`

Creates an opener for the replicator.

It also attaches the *`CachingRedirectHandler`* to the opener and sets its User-agent to `Mozilla/5.0`.

If the Network Proxy settings are set and recognized, it creates the opener and attaches the proxy_handler to it, and is returned.

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

Opens a url specified by the `url` parameter.

This function handles :

- Redirect caching, if enabled.
- Trying the replicator first, if enabled.
- Retries upto 5 times if it encounters a http `500` error.

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

Gets a `bs4` parsed soup for the `url` specified by the parameter. The `lxml` parser is used.

3.2 Submodules

3.2.1 koala.main module

This file is part of koala See the COPYING, README, and INSTALL files for more information

3.2.2 koala.runserver module

This file is part of koala See the COPYING, README, and INSTALL files for more information

3.3 Module contents

This file is part of koala See the COPYING, README, and INSTALL files for more information

Part III

Development TODOs

Todo

The core numbers in this module need to switched to `decimal.Decimal`.

(The original entry is located in /media/ldata/code/koala/koala/utils/types/currency.py:docstring of koala.utils.types.currency, line 28.)

Todo

Switch to currencyrates API instead.

(The original entry is located in /media/ldata/code/koala/koala/utils/types/currency.py:docstring of koala.utils.types.currency.CurrencyDefinition._get_exchval, line 10.)

Todo

The present implementation is fairly silly and tedious. A better implementation is necessary.

(The original entry is located in /media/ldata/code/koala/koala/utils/config.py:docstring of koala.utils.config, line 14.)

Todo

The actual configuration options themselves need to be documented.

(The original entry is located in /media/ldata/code/koala/koala/utils/config.py:docstring of koala.utils.config, line 17.)

Todo

Publish the `sofficehelpers` package somewhere, or find a pre-existing alternative. If there is a functionally sufficient alternative, it will probably be better.

(The original entry is located in /media/ldata/code/koala/koala/utils/libreoffice.py:docstring of koala.utils.libreoffice, line 16.)

- genindex
- modindex
- search

Symbols