E7EPD Database

Release 0.6.0

Electro707

Dec 19, 2022

CONTENTS

This is a electronics database specification, allowing you to store both parts and projects.

This project as a whole includes 3 components:

- E7EPD Database Specification: This is a document which describes the specification for storing part and project to the database.
- e7epd Python Package: This is the only officially supported method of creating the database and interacting with it.
- CLI Utility: This is a Python command line front-end for interacting with the e7epd Python Package.

The database specification will be separately revisioned from the CLI utility and Python package

CHAPTER

ONE

TABLE OF CONTENTS:

1.1 Database CLI Application

While the database can be operated by any external program, a CLI utility exist for interfacing with the database.

1.1.1 Requirements

To run this CLI application, you will need `python>3.7`. On top of that, other than the built-in packages you will need the following packages as well:

- rich
- questionary
- engineering_notation
- SQLAlchemy
- alembic

1.1.2 Usage

Coming Soon!

1.1.3 Digikey Barcode Scanning

The CLI application allows for scanning and inputting a Digikey 2D barcode that are included in a part's bag. To utilize this feature, my fork of *digikey-api* must be installed, which can be done with:

```
pip install git+https://github.com/Electro707/digikey-api.
↪git@8549f42a1853c9d371c3fb1b0b8d780d405174d8
```

Initially, you need to setup the DigikeyAPI with you client secret and client ID in the CLI for your Digikey Developer application which is done thru the main menu under the *'Digikey API Settings'* option. This application must have the *'Barcode'* and *'Product Information'* APIs enabled in Digikey's API settings. For more information on Digikey's APIs, see https://developer.digikey.com/get_started.

When asked for a manufacturer part number in the CLI application, you can scan a Digikey 2D barcode instead of typing in a manufacturer number. The input to the CLI application must be the scanned data.

1.2 API Reference

1.2.1 Python Documentation

class e7epd.**E7EPD**(*db_conn: Engine*)

class **ConfigTable**(*session: Session*, *engine: Engine*)

get_info(*key: str*) → Optional[str]

store_info(*key: str*, *value: str*)

get_db_version() → Optional[str]

store_current_db_version()

class **GenericItem**(*session: Session*)

This is a generic part constructor, which other components (like Resistors) will base off of

create_part(*part_info: ComponentTypeVar*)

Function to create a part for the given info

Parameters

part_info (GenericItem) – The part item class to add to the database

get_all_parts() → Iterable[ComponentTypeVar]

Get all parts in the database

Returns

Raises this if the manufacturer part number does not exist in the database

Return type

EmptyInDatabase

is_database_empty()

Checks if the database is empty

Returns

True if the database is empty, False if not

commit()

Commits any changes done to part(s)

rollback()

Roll back changes done to part(s)

class **GenericComponent**(*session: Session*)

get_all_mfr_part_numb_in_db() → list

Get all manufaturer part numbers as a list

Returns

A list of all manufacturer part numbers in the database

get_sorted_parts(*search_filter: list*)

get_part_by_mfr_part_numb(*mfr_part_numb: str*) → ComponentTypeVar

Function that returns parts parameters by part ID

Parameters

mfr_part_numb (*str*) – The part's manufacturer part number

Returns

The part's item class

Raises
EmptyInDatabase – If the SQL ID does not exist in the database

delete_part_by_mfr_number(*mfr_part_numb: str*)

Function to delete a part by the manufacturer part number

Parameters
mfr_part_numb – The manufacturer part number of the part to delete

Raises
EmptyInDatabase – Raises this exception if there is no part in the database with that manufacturer part number

check_if_already_in_db(*part_info: ComponentTypeVar*)

Checks if a given part is already in the database Currently it's done through checking for a match with the manufacturer part number

Parameters
part_info (GenericItem) – A part item class

Raises
- **UserWarning** – This should NEVER be triggered unless something went terribly wrong or if you manually edited the database. If the former, please create an bug entry on the project's Github page with the traceback
- *InputException* – If the manufacturer part number is None, this will get raised

check_if_already_in_db_by_manuf(*mfr_part_numb: str) -> (None*, *<class 'int'>*)

Function that checks if a part is already in the database. A generic part just goes by manufacturer part number, otherwise a component-specific callback needs to be added (to look up generic parts for example without a manufacturer part number)

Parameters
mfr_part_numb (*str*) – The manufacturer part number to look for.

Returns
None if the part doesn't exist in the database, The SQL ID if it does

Raises
- **UserWarning** – This should NEVER be triggered unless something went terribly wrong or if you manually edited the database. If the former, please create an bug entry on the project's Github page with the traceback
- *InputException* – If the manufacturer part number is None, this will get raised

class PCBs(*session: Session*)

get_all_boardnames() → list

get_revision_per_boardname(*board_name: str*) → list

get_by_boardname_and_rev(*board_name: str*, *rev: str*) → ComponentTypeVar

class Resistance(*session: Session*)

static print_formatted_from_spec(*spec_dict: dict*) → str

class Capacitors(*session: Session*)

class Inductors(*session: Session*)

class ICs(*session: Session*)

class Diodes(*session: Session*)

class Crystals(*session: Session*)

class MOSFETs(*session: Session*)

class BJTs(*session: Session*)

class Connectors(*session: Session*)

class LEDs(*session: Session*)

class Fuses(*session: Session*)

class Buttons(*session: Session*)

class MiscComps(*session: Session*)

components

A helper dictionary containing all components (PCBs are not included)

close()

Commits to the database and closes the database connection. Call this when exiting your program

save()

Saves any changes done to the database

get_component_by_table_name(*table_name: str*)

check_if_already_in_db_by_manuf(*mfr_part_numb: str*) -> *(None*, <*class 'int'*>)

Checks if a manufacturer part number is already in the database for all component types

Parameters

mfr_part_numb – The manufacturer part number to look for

Returns: A tuple, the first index beingn the SQL ID, the second being the component GenericPart class of the part

get_all_mfr_part_numb_in_db()

Gets all stored manufacturer part numbers in the database

Returns: A list containing all manufacturer part number

get_all_component_part_number() → dict

Gets all component's manufacturer part number in the database NOTE: This excludes PCBs as it is not a component

Returns: A dictionary of lists containing all part numbers per component

wipe_database()

Wipes the component databases

update_database()

Updates the database to the most recent revision

is_latest_database() → bool

Returns whether the database is matched with the latest rev :returns: True if the database is the latest, False if not :rtype: bool

backup_db()

Backs up the database under a new backup file

exception e7epd.InputException(*message*)

Exception that gets raised on any input error

exception e7epd.**EmptyInDatabase**

Exception that gets raised when there is no parts in the database

exception e7epd.**NegativeStock**(*amount_to_make_zero*)

Exception that gets raised when the removal of stock will result in a negative stock, which is physically impossible (you're always welcome to prove me wrong)

amount_to_make_zero

How many parts to make the part's stock zero.

Type
int

1.2.2 Autofill Helpers

This wrapper module also comes with some autofill helpers. All are in the dictionary `autofill_helpers_list`. Here are the current autofill helpers available:

- `ic_manufacturers`: IC manufacturers like TI and Cypress
- `ic_types`: The IC type, like Microcontroller and ADC
- `capacitor_types`: Capacitor types like Ceramic and Electrolytic
- `diode_type`: Diode Types
- `passive_manufacturers`: Passives (resistors, capacitors, etc) manufacturers
- `passive_packages`: Packages for passives
- `ic_packages`: Packages for ICs
- `mosfet_types`: MOSFET types (N-Channel, P-Channel)
- `bjt_types`: BTJ Type (NPN, PNP)
- `fuse_types`: Fuse Type (Slow Blow, PTC, etc)
- `led_types`: LED "Type" (Red, RGB, Addressable, etc)

1.3 E7EPD Database Specification

Rev 0.5

1.3.1 Specification Notes

Components

All components, when specified, will be based of the *GenericPart* table spec that contains common columns. Note that the *GenericPart* table doesn't exist by itself but is used in this document as columns that every other table should have

Different Tables Per Components

To allow for the parameterization of parts, and due to SQL's column nature where it's usually unchanged, each specific component type (like resistors, IC, etc) will have its own SQL table.

1.3.2 Table Spec

GenericPart Table

Name	SQL Type	Required?	Description
id	INTEGER PRIMARY KEY	YES	Each row in the database will contain an unique SQL ID
stock	INT	YES	The number of parts in stock
mfr_part_numb	VARCHAR	YES	The manufacturer part number, used to distinguish each part from another
manufacturer	VARCHAR		The manufacturer of the component
package	VARCHAR	YES	The part's physical package
storage	VARCHAR		The part's storage location
comments	TEXT		Comments about the part
datasheet	TEXT		The datasheet of the part

Resistor Table

Append the *GenericPart* Table to this table. Table Name: `resistance`

Name	SQL Type	Required?	Description
resistance	FLOAT	YES	The resistor's resistance
tolerance	FLOAT		The resistor's tolerance as a float (so a 5% resistor will be stored as 5)
power	FLOAT		The resistor's power rating in W

Capacitor Table

Append the *GenericPart* Table to this table. Table Name: `capacitor`

Name	SQL Type	Required?	Description
capacitance	FLOAT	YES	The capacitor's capacitance
tolerance	FLOAT		The capacitor's tolerance as a float (so a 5% capacitor will be stored as 5)
voltage	FLOAT		The capacitor's maximum voltage rating
temp_coef	VARCHAR		The capacitor's temperature coefficient
cap_type	VARCHAR		The capacitor types, which should only be 'electrolytic', 'ceramic', 'tantalum', 'film'. If a type is not listed, you can enter a custom type, just make sure that it's consistent for different parts (also create an Issue on the Github page so we can all have it :)

Inductor Table

Append the *GenericPart* Table to this table. Table Name: `inductor`

Name	SQL Type	Required?	Description
inductance	FLOAT	YES	The inductance of the inductor
tolerance	FLOAT		The inductor's tolerance as a float (so a 5% inductor will be stored as 5)
max_current	FLOAT		The inductor's maximum current

Diode Table

Append the *GenericPart* Table to this table. Table Name: `diode`

Name	SQL Type	Required?	Description
diode_type	VARCHAR	YES	Diode Type (Regular, Zener, Schottky, etc)
max_current	FLOAT		Max/Peak Current
average_current	FLOAT		Average Current Rating
max_rv	FLOAT		Max reverse voltage

IC Table

Append the *GenericPart* Table to this table. Table Name: `ic`

Name	SQL Type	Required?	Description
ic_type	VARCHAR	YES	The IC type, for example microcontroller, ADC, comparator, etc.

Crystal Table

Append the *GenericPart* Table to this table. Table Name: `crystal`

Name	SQL Type	Required?	Description
frequency	FLOAT	YES	The frequency of the crystal
load_c	FLOAT		The load capacitance (in pF) of the crystal
esr	FLOAT		The ECR (in Ohms) of the crystal
stability_ppm	FLOAT		The stability (in ppm) of the crystal

MOSFET Table

Append the *GenericPart* Table to this table. Table Name: `mosfet`

Name	SQL Type	Required?	Description
mosfet_type	VARCHAR	YES	The MOSFET type (N-Channel or P-Channel)
vdss	FLOAT		The max Drain-Source voltage of the MOSFET
vgss	FLOAT		The max Gate-Source voltage of the MOSFET
vgs_th	FLOAT		The Gate-Source threshold voltage of the MOSFET
i_d	FLOAT		The max continuous drain current of the MOSFET
i_d_pulse	FLOAT		The max pulsed/peak drain current of the MOSFET

BJT Table

Append the *GenericPart* Table to this table. Table Name: `bjt`

Name	SQL Type	Required?	Description
bjt_type	VARCHAR	YES	The BJT type (NPN or PNP)
vcbo	FLOAT		The max Collector-Base voltage of the BJT
vceo	FLOAT		The max Collector-Emitter voltage of the BJT
vebo	FLOAT		The max Emitter-Base voltage of the BJT
i_c	FLOAT		The max continuous collector current of the BJT
i_c_peak	FLOAT		The max pulsed/peak collector current of the BJT

Connector Table

Append the *GenericPart* Table to this table. Table Name: `connector`

Name	SQL Type	Required?	Description
conn_type	VARCHAR	YES	The connector type (Banana, Rect. Header, Test point, etc)

LED Table

Append the *GenericPart* Table to this table. Table Name: `led`

Name	SQL Type	Required?	Description
led_type	VARCHAR	YES	The LED's color (Red, Blue, RGB, etc)
vf	FLOAT		The LED's forward voltage
max_i	FLOAT		The LED's maximum forward current

Fuse Table

Append the *GenericPart* Table to this table. Table Name: `fuse`

Name	SQL Type	Required?	Description
fuse_type	VARCHAR	YES	The fuse type (Glass, PTC, etc)
max_v	FLOAT		The fuse's max voltage
max_i	FLOAT		The fuse's absolute maximum current
trip_i	FLOAT		The fuse's trip current
hold_i	FLOAT		The fuse's hold current

Button/Switch Table

Append the *GenericPart* Table to this table. Table Name: `button`

Name	SQL Type	Required?	Description
bt_type	VARCHAR	YES	The button/switch type (Tactile, Rocker, etc)
circuit_t	VARCHAR		The button/switch's configuration (SPDT, SPST-NO, etc)
max_v	FLOAT		The button/switch's max voltage
max_i	FLOAT		The button/switch's absolute maximum current

Misc Table

This table is exactly the same as the *GenericPart* Table. Table Name: `misc_c`

1.3.3 PCBs

Each PCB will have parts associated with it. This should allow the user application to determine if it's possible to build up a board given the current component's stock.

PCB Table

Table Name: `pcb`

Name	SQL Type	Required?	Description
id	INTEGER PRIMARY KEY	YES	Each row in the database will contain an unique SQL ID
stock	INT	YES	The number of parts in stock
board_name	VARCHAR	YES	The board's name. Can also be thought of as the project's name
rev	VARCHAR	YES	The pcb's revision
sub_rev	VARCHAR		The pcb's sub-revision
comments	TEXT		Comments about the part
parts	JSON	YES	A JSON list containing all of the parts used for this project

Parts JSON List

The parts JSON is a list of dictionaries containing the all parts used for a particular board.

The dictionaries in this list is formatted as follows for a component:

Key	Value Type	Description
comp_type	string	The component type (resistor, bjt, etc) which corresponds to the part's table name
part	dict	A dictionary describing the part
qty	int	The quantity of this part used in this board
alternatives	list	A list of alternative parts that can be used, each part being the same format as the part key above. This list can be left as an empty array.

The part key above is a dictionary containing a set of filter key-value pairs that narrows down a part. For example, for a part with the manufacturer part number of "PART123", the part dict would be .. code-block:

```
{
    mfr_part_numb: PART123
}
```

As the manufacturer part number is unique to each part, this filter would only find a single part. With a resistor for example, where a specific part does not matter, the part dict would look something like .. code-block:

```
{
    resistance: 1000
    power: >0.125
    package: 0805
}
```

The > prefix in *power: >0.125* indicates that the power value must be greater than 1/8W, and anything above that is fine as well.

1.4 Changelog

The database specification will be separately tracked from the Python DB wrapper and CLI tool

1.4.1 E7EPD Database Specification

- **v0.05:**
  - Initial Release
- **v0.1:**
  - Removed the Microcontroller table
  - Added an all-encompassing IC table
  - Added a PCB Table
  - Clean up some mismatch between the wrapper and this spec
- **v0.2:**
  - Added Inductor table
  - Added Diode table
- **v0.3:**
  - Added a `storage` key to all parts
  - Merged `user_comments` and `part_comments` to just one `comments` column
  - Removed `power` for the capacitor table
  - Updated type of the `comments` column to `TEXT`
  - **Added a spec for**
    - Crystals
    - MOSFETs
    - BJTs
    - Connectors
    - LEDs
    - Fuses
    - Switches/Buttons
    - Misc/Others
- **v0.4:**
  - Added datasheet column for all components
  - Removal of `project_name` from the PCB table and replaced it `board_name`
  - Added a `parts` JSON list for the PCB table, allowing parts to be cross-referenced per board

- **v0.5:**
  - Changed the SQL primary key for all components to `mfr_part_numb`, and removed the `id` column
  - Added a `user` column for all components
  - Removed `sub_rev` column from `PCB` table

1.4.2 Database Python DB Wrapper

- **v0.05:**
  - Initial Release
- **v0.2:**
  - Updated spec for Database Rev 0.2
  - Changed main class name from `EEData` to `E7EPD`
  - Allowing user-given `sqlite3` connections
  - Added some autocomplete lists for some part's info like IC manufacturers and capacitor types
  - Better documentation
  - Added `` `wipe_database` `` function
  - Added a way for the backend to store configurations about itself
  - Added a key to check the database specification the database is under and the `E7EPD` class
- **v0.3:**
  - Switched to sqlalchemy for handing SQL
  - As the wrapper input includes a sqlalchemy engine, any sql type that sqlalchemy supports should be supported
  - Updated spec for Database Rev 0.3
  - Added the first migration from Database Rev 0.2 to 0.3 with `alembic`
  - Added a `PCB` class from database spec (didn't for 0.2)
  - Added autofill helpers for the new component types
  - Re-factored backend spec and display_as lists
- **v0.4:**
  - Updated spec for Database Rev 0.4
  - Added migrations from database rev 0.3 to 0.4
  - More autofill helpers
  - Removed the `update_part` function as the variable returned by `get_part_by_id` or `get_part_by_mfr_part_numb` already has a database link, so modifying that with a `commit` command will make the changes
  - Added a `commit` and `rollback` function for when modifying a returned part linked to the database
  - Re-made the `backup_db` function to actually work. Dumps content as a JSON file
- **v0.4.1:**
  - No changes

- **v0.5:**
  - Added an overall *E7EPD* helper function to get all manufacturer part number
  - Separated the `PCB` class into it's own compared to components with some different function calls
  - Added a `User` column per component
  - New `print_formatted_from_spec` function for resistances, to print out for example "A 5k resistor with a 5% tollerance"
  - `get_sorted_parts()` component class function now allows operators like ">" and "<"
  - **Removed the following component class functions:**
    - `append_stock_by_manufacturer_part_number`
    - `remove_stock_by_manufacturer_part_number`
    - `get_part_by_id`
  - Added a typing hinting for every component class instead of just `GenericComponent`
  - Seperated the `GenericPart` class into `GenericItem` and `GenericComponent`, `GenericComponent` having functions more specific to components (like things related to the manufacturer part number)
- **v0.5.1:**
  - Removed the *EmptyInDatabase* return exception from *get_all_mfr_part_numb_in_db()*, instead just returns an empty list
- **v0.6.0:**
  - No changes

1.4.3 CLI

- **v0.1:**
  - Initial Release
- **v0.2:**
  - Added initial setup for user to set the `sqlite3` database file
  - Added option to enter values as a percentage (so for example 1/4 for 0.25)
  - Added autocomplete for part's values like capacitor type, if they exist in the database wrapper
  - Added autocomplete hinting when a manufacturer part number is asked
  - Added option to remove and append stock to a part
  - Moved around options so that there is an "initial screen" before choosing components
  - Added a check for the database revision on startup
- **v0.3:**
  - Updated for the new Wrapper 0.3 database argument
  - Changed options so it's easier to add a new part
  - Allowing option for a mySQL database
  - Allowing option to add multiple databases
  - Allowing option to select which database to connect to

- **v0.4:**
  - Added ability to scan a Digikey barcode for the manufacturer part number
  - Added ability to edit a part's properties
- **v0.4.1:**
  - Fixed bug with an empty database where you could not enter a manufacturer part number
  - Added more safety checks and exception handling around the user input function
  - Changed the way the Digikey API got loaded and set as to not be intrusive
  - Added docs for the Digikey barcode scanning feature
  - Fixed the *Digikey API Settings* option in the main menu to allow changing the Client ID and Client Secret
  - Added a __main__.py file to allow execution of *e7epd* as a Python module with *python -m e7epd*
- **v0.5:**
  - Added more docs to existing functions to make it easier to interpret
  - Made it so creating an existing part would prompt to instead add the part to the existing stock
  - Added better messages about stock when adding or removing stock
  - Added PCBs as an option to add them
  - Added a menu to search the current inventory against a PCB to see if one can be built
  - Allowing for usage of operators like `>`, `>=`, `<`, `<=` when filtering the database
  - Added new `Seach Parts` menu option (which is the same as `Individual Components View -> Print parts in DB`
- **v0.5.1:**
  - Fixed bug with a new empty database when looking up the current list of parts to use as a typehint
  - Fixed inputting a negative number in removing or adding stock
  - Fixed exception when there are no PCBs in the database
- **v0.6.0:**
  - Allowing for MySQL or PostgreSQL options for database
  - Better CLI handling if one does not want to upgrade database
  - Better docs
- **TODOs:**
  - Add option to import BOM file/CSV file
  - Add ability to "interact" with the PCB table
  - Add cross-coerelation between a PCB's parts and parts in the database

PYTHON MODULE INDEX

INDEX

P

R

S

U

W