HarNice!

The free, open-source electrical system design tool.

Disclaimer

- I wrote this in past tense, but it's **far** from done!
- This is my first-ever real coding project
  - I don't claim that my code is any good – just the thoughts!
- I've never really used another real system-engineering tool
  - Maybe I'm reinventing a wheel
  - Either way, it's a super fun project

What is an electrical system?

- Any collection of circuit boards, devices, other things, that connect to each other with wires or harnesses
- Requires engineering trades and decisions to design
- Examples:
  - Avionics system on a rocket or satellite
  - Vehicle HITL
  - Test stand
  - Concert sound system
  - Residential AC power distribution

What is **not** an electrical system?

- Physical electrical behavior
  - We take the assumption that any box has inputs and outputs and what it does to relate the two is irrelevant
- Network routing
  - Software configs are agnostic to the electrical systems

How do you design your system?

- The competition:
  - Zuken, E-plan, etc
  - SUPER expensive, cumbersome, training-intensive

Pen and paper vibes

- Alternatively, Vizio, Powerpoint, Excel, MS Paint
  - No metadata in your drawings
  - No enforceable design rule checks
  - Conflicting sources of truth
  - Required manual dependencies
  - No centralized revision control

Harnice **solves** this.

- There can and should always exist **one single source of truth** for any entire electrical system.
- On the assumption that the following are known, every artifact (harness drawing, BOM, etc) should be fully deterministic.
  1. Your system is fully defined
     - You know how many boxes you need and how you want them to be connected
  2. You have complete details about how each box works
     - Box connectors and channels are fully defined
  3. Every design standard you adhere to is expressible
     - Ex. “We always put yellow heat shrink if the harness contains a signal with this name”
     - Ex. “Every time a harness contains more than two receptacle connectors, switch the font size on the label”
  4. The physical routing of harnesses in 2D or 3D is defined somehow
     - Harnice doesn’t know how long a harness needs to be

How do you know what to design?

- Traditionally, the engineer **compiles** information either from a design guide, from industry knowledge, or other sources,

  ...manually...
- until their design is complete.

INSTEAD:

- Harnice encourages the user NOT to compile any information, and instead, explicitly document your design standards and rules in a machine-readable format.
- ***Let the machines work for you!***

High-level requirements

- Harnice shall have just one command which should update everything about a product
- Harnice shall allow users to input their design rules using a well-known human-readable language
- Harnice may only produce ONE deterministic solution to any product, and raise an error if multiple or zero solutions are found
- Harnice shall not operate on released parts

Flexibility requirements

- If you can define something as an “instance” with “attributes”, it should be able to end up in a harness build file.
- Harnice shall minimize the number of terms that represent physical goods (see prev. slide)

The solution

- You specify your system using Kicad or Altium
  - Components represent boxes
  - Nets represent harnesses
- Harnice runs a bunch of code on your netlist to determine harness requirements
  - Builds a channel map
  - Builds a System Instances List (list of every part and every connection for every harness)
- Harnice harness editor can look for a harness inside a system
  - Adds parts based on standard build rules
  - Exports build drawings and other artifacts instantaneously

a word on terminology…

Term	Description	Scope	Attributes	Examples
“System”	A complete overview of every relevant piece of hardware in your system	• Systems contain boxes and nets	• Which box part numbers and how many are you using • How each box is connected to each other	• Vehicle HITL • Test stand • Concert sound system • Residential AC power distribution
“Box”	A physical device that interfaces with other boxes via hardware electrical signals		• Part number • Connectors • Channels	• Pressure sensor • ADC • Light switch • MacBook
“Channel”	A set of signals	• Boxes have Channels • Channels exist in a centralized library	• Other compatible channels • Input, output, or both • Range, tolerance • Data format	• “Power in” • “Mic level signal” • FTT (4-20mA) • 500mHz 50ohm
“Signal”	An individual conductor going in or out of a box	• Channels are defined to contain Signals • Box Connectors must contain all Box Channels’ Signals	• Signal name	• +5V • Chassis • CS, MISO, MOSI
“Net”	A set of connectors	• Nets defined at the system-level • Nets are not harness part numbers	• Which connectors of which boxes are included in the net	• N$1
“Harness”	A physical device that exists to link boxes together	• Harnesses contain Circuits and mating connectors	• Harness part number • Contained parts	• Lightning cable • Your favorite octopus harness
“Circuit”	A connection between one signal and another	• Circuits contain contacts, conductors, etc that form an electrical path	• Name	• Channel 1 +5V

The workflow

- Regardless of which product you're working on, when you call "render", harnice will run "featuretree.py"
- Feature Tree, notionally based on Solidworks, will sequentially build up the product's data structure based on "prebuilders" you can write and select
- Feature Tree will also call Artifact Builders, which do stuff like generate a BOM, make a PDF drawing, etc

Harnice products

	System	Harness	Box	Part	Title block	Flagnote
Inputs	Netlist exported from Kicad or Altium	System instances list, manual YAML or JSON harness definition	ICD.py		• Params.json	• Params.json
Data structure	Instances list	Instances list	Signals list			
Outputs			.Kicad_sym	• Attributes.json • Drawing.svg	• Attributes.json • Drawing.svg	• Drawing.svg

“Instances Lists” as data structures

- Harnesses and Systems use “Instances Lists”
  - A list of every single item, idea, note, part, instruction, circuit, literally anything that comprehensively describes how to build that harness or system
  - TSV (tab-separated-values, big spreadsheet)
  - Declined alternatives: STEP files, schematics, dictionaries not general, descriptive, or human readable enough

```
INSTANCES_LIST_COLUMNS = [
    'instance_name',
    'print_name',
    'bom_line_number',
    'mpn', #unique part identifier (manufacturer + part numbe
    'item_type', #connector, backshell, whatever
    'parent_instance', #general purpose reference
    'location_is_node_or_segment', #each instance is either b
    'cluster', #a group of co-located parts (connectors, back
    'circuit_id', #which signal this component is electricall
    'circuit_id_port', #the sequential id of this item in its
    'length', #derived from formboard definition, the length
    'diameter', #apparent diameter of a segment <---------- c
    'node_at_end_a', #derived from formboard definition
    'node_at_end_b', #derived from formboard definition
    'parent_csys_instance_name', #the other instance upon whi
    'parent_csys_outputcsys_name', #the specific output coord
    'translate_x', #derived from parent_csys and parent_csys_
    'translate_y', #derived from parent_csys and parent_csys_
    'rotate_csys', #derived from parent_csys and parent_csys_
    'absolute_rotation', #manual add, not nominally used unle
    'note_type',
    'note_number', #<--------- merge with parent_csys and imp
    'bubble_text',
    'note_text',
    'supplier',
    'lib_latest_rev',
    'lib_rev_used_here',
    'lib_modified_in_part',
    'lib_status',
    'lib_datemodified',
    'lib_datereleased',
    'lib_drawnby',
    'debug',
    'debug_cutoff'
]
```

Harness Instances List example

0020250720-H01-rev2-instances_list

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	instance_nai	print_name	bom_line_nu	mpn	item_type	parent_insta	location_is_r	cluster	circuit_id	circuit_id_po	length	diameter	node_at_end	node_at_end	parent_csys_	parent_csys_	translate_x	translate_y	rotate_csys	absolute_rot	note_type	note_numbe	bubble_text	note_text	supplier	lib_latest_re
2	origin				Node		Node																			
3	circuit1				Circuit																					
4	X1	X1	1	D38999_26Z	Connector		Node	X1							X1.node	origin	0	0	0						public	1
5	X1.node				Node		Node	X1							origin	origin	0	0	195	195						
6	X1.cavity7	7			Connector ci	X1	Node	X1	circuit1	0																
7	circuit1.contact0		2	TXPA20	Contact	X1.cavity7	Node	X1	circuit1	1															public	
8	C1		3	test	Cable		Segment				8															
9	C1.conducto	red			Conductor	C1	Segment		circuit1	2	8															
10	X2	X2	4	D38999_26Z	Connector		Node	X2							X2.bs	connector	3	0	0						public	1
11	X2.node				Node		Node	X2							origin	origin	0	2.08	165	165						
12	X2.cavity1	1			Connector ci	X2	Node	X2	circuit1	3																
13	circuit2				Circuit																					
14	X1.cavity6	6			Connector ci	X1	Node	X1	circuit2	0																
15	C1.conducto	black			Conductor	C1	Segment		circuit2	1	8															
16	X2.cavity2	2			Connector ci	X2	Node	X2	circuit2	2																
17	circuit3				Circuit																					
18	X1.cavity8	8			Connector ci	X1	Node	X1	circuit3	0																
19	C1.conducto	green			Conductor	C1	Segment		circuit3	1	8															
20	X2.cavity4	4			Connector ci	X2	Node	X2	circuit3	2																
21	circuit4				Circuit																					
22	X1.cavity9	9			Connector ci	X1	Node	X1	circuit4	0																
23	C3		3	test	Cable		Segment				24															
24	C3.conducto	1			Conductor	C3	Segment		circuit4	1	24															
25	X4	X4	4	D38999_26Z	Connector		Node	X4							X4.bs	connector	3	0	0						public	1
26	X4.node				Node		Node	X4							origin	origin	0.98	-4.03	205	205						
27	X4.cavity1	1			Connector ci	X4	Node	X4	circuit4	2																
28	circuit5				Circuit																					
29	X1.cavity10	10			Connector ci	X1	Node	X1	circuit5	0																
30	C3.conducto	2			Conductor	C3	Segment		circuit5	1	24															
31	X4.cavity2	2			Connector ci	X4	Node	X4	circuit5	2																
32	circuit6				Circuit																					
33	X1.cavity11	11			Connector ci	X1	Node	X1	circuit6	0																
34	C3.conducto	shield			Conductor	C3	Segment		circuit6	1	24															
35	X4.cavity3	3			Connector ci	X4	Node	X4	circuit6	2																
36	circuit7				Circuit																					
37	X1.cavity12	12			Connector ci	X1	Node	X1	circuit7	0																
38	C4		3	test	Cable		Segment				20															
39	C4.conducto	1			Conductor	C4	Segment		circuit7	1	20															
40	X500	X500	4	D38999_26Z	Connector		Node	X500							X500.bs	connector	3	0	0						public	1
41	X500.node				Node		Node	X500							origin	origin	19.72	2.08	15	15						
42	X500.cavity1	1			Connector ci	X500	Node	X500	circuit7	2																
43	circuit8				Circuit																					
44	X1.cavity13	13			Connector ci	X1	Node	X1	circuit8	0																
45	C4.conducto	2			Conductor	C4	Segment		circuit8	1	20															
46	X500.cavity2	2			Connector ci	X500	Node	X500	circuit8	2																
47	circuit9				Circuit																					
48	X1.cavity14	14			Connector ci	X1	Node	X1	circuit9	0																
49	C4.conducto	shield			Conductor	C4	Segment		circuit9	1	20															
50	X500.cavity3	3			Connector ci	X500	Node	X500	circuit9	2																

Artifact builders

- Artifact builders will scour the Instances List or other artifact outputs and make other things out of it
  - BOM
  - Formboard arrangement
  - PDF drawing sheet
  - Analysis calcs
  - Write your own!

```
#=====================================================================
#                    CONSTRUCT HARNESS ARTIFACTS
#=====================================================================
scales = {
    "A": 1,
    "B": 0.25
}
featuretree.runartifactbuilder("bom_exporter_bottom_up", "public", artifact_id="bom1")
featuretree.runartifactbuilder("standard_harnice_formboard", "public", artifact_id="formboard1", scale=scales.get("A"))
featuretree.runartifactbuilder("standard_harnice_formboard", "public", artifact_id="formboard2", scale=scales.get("B"))
featuretree.runartifactbuilder("wirelist_exporter", "public", artifact_id="wirelist1")
featuretree.runartifactbuilder("revision_history_table", "public", artifact_id="revhistory1")
featuretree.runartifactbuilder("buildnotes_table", "public", artifact_id="buildnotestable1")
featuretree.runartifactbuilder("pdf_generator", "public", artifact_id="drawing1", scales=scales)

featuretree.copy_pdfs_to_cwd()
```

Prebuilders

- Prebuilders are pre-written “macros” that actively add or modify lines on an instances list.
  - Can read information from the instances list
  - Can read information from other support files
- Examples
  - featuretree.runprebuilder(“wireviz_yaml_prebuilder”, “public”)
    - Reads a wireviz YAML (another commonly used harness design format)
  - featuretree.runprebuilder(“add_yellow_htshrk_to_plugs”, “kenyonshutt”)
    - You can write any rule or set of rules you want in Python, save it to your library, and call it from a harness feature tree.
    - This one, for example, might scour the instances list:
      - for plug in instances_list:
        - if item_type==plug:
          - instances_list.add(heatshrink, to cable near plug)

Prebuilders

```python
#=============== ASSIGN PRINT NAMES TO CONNECTORS #==============
for instance in instances_list.read_instance_rows():
    instance_name = instance.get("instance_name")
    if instance_name == "X1":
        instances_list.modify(instance_name,{
            "print_name": "P1"
        })
    elif instance_name == "X2":
        instances_list.modify(instance_name,{
            "print_name": "P2"
        })
    elif instance_name == "X3":
        instances_list.modify(instance_name,{
            "print_name": "P3"
        })
    elif instance_name == "X4":
        instances_list.modify(instance_name,{
            "print_name": "J1"
        })
    elif instance_name == "X500":
        instances_list.modify(instance_name,{
            "print_name": "J2"
        })
    elif instance.get("item_type") == "Connector":
        raise ValueError(f"Connector {instance.get("instance_name")} defined but does not have a print name assigned.")


#=============== ASSIGN BACKSHELLS AND ACCESSORIES TO CONNECTORS #==============
for instance in instances_list.read_instance_rows():
    instance_name = instance.get("instance_name")
    if instance.get("item_type") == "Connector":
        mpn = instance.get("mpn")
        if re.fullmatch(r"D38999_26ZA.+", mpn):
            if instance.get("print_name") not in ["P3", "J1"]:
                instances_list.add(f"{instance_name}.bs",{
                    "mpn": "M85049-88_9Z03",
                    "print_name": f"{instances_list.attribute_of(instance_name, "print_name")}.bs",
                    "bom_line_number": "True",
                    "supplier": "public",
                    "item_type": "Backshell",
                    "parent_instance": instance.get("instance_name"),
                    "cluster": instance_name,
                    "location_is_node_or_segment": "Node"
                })
```

Harnice workflow – for systems

Harnice workflow – for harnesses

Libraries

- Parts, prebuilders, artifact builders, etc, should be re-used and referenced for any future use
- Users are heavily encouraged to contribute to the public git repo for your cots parts
  - **harnice-library-public**
- However, you can define paths to as many library repos as you want.
  - **my-company-harnice-library**
  - **my-super-top-secret-harnice-library**
- You don’t even have to use git to control it!

Product version control

- To “render” a product (harness, part, etc) with Harnice, the CLI will force you to operate in a “rev folder”
- Revision data always stored in revision_history.tsv
- Revisions can be referenced elsewhere, ex in pdf_generator

pn	desc	rev	status	releaseticket	datestarted	datemodified	datereleased	drawnby	checkedby	revisionupdates	affectedinstances
0020250720-H01	HARNESS, DOES A, FOR B	1	OUTDATED		7/22/25	8/10/25		K SHUTT	W CANAVAN	INITIAL RELEASE	
0020250720-H01	HARNESS, DOES A, FOR B	2			8/10/25	8/10/25		K SHUTT	D RAIGOSA	CHANGED NOTE FOR CONNECTOR	X1

```
Mac:0020250720-H01 kenyonshutt$ harnice -r harness
Thanks for using Harnice!
This is a part folder (0020250720-H01).
Please `cd` into one of its revision subfolders (e.g. `0020250720-H01-rev1`) and rerun.
Mac:0020250720-H01 kenyonshutt$ cd 0020250720-H01-rev1
Mac:0020250720-H01-rev1 kenyonshutt$ harnice -r harness
Thanks for using Harnice!
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.13/bin/harnice", line 8, in <module>
    sys.exit(main())
             ~~~~^^
  File "/Users/kenyonshutt/Documents/GitHub/harnice/src/harnice/cli.py", line 28, in main
    render.harness()
    ~~~~~~~~~~~~~~^^
  File "/Users/kenyonshutt/Documents/GitHub/harnice/src/harnice/commands/render.py", line 26, in harness
    fileio.verify_revision_structure()
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^
  File "/Users/kenyonshutt/Documents/GitHub/harnice/src/harnice/fileio.py", line 342, in verify_revision_structure
    raise RuntimeError(f"Revision {rev} status is not clear. Harnice will only let you render revs with a blank status.")
RuntimeError: Revision 1 status is not clear. Harnice will only let you render revs with a blank status.
Mac:0020250720-H01-rev1 kenyonshutt$ cd ..
Mac:0020250720-H01 kenyonshutt$ cd 0020250720-H01-rev2
Mac:0020250720-H01-rev2 kenyonshutt$ harnice -r harness
Thanks for using Harnice!
Working on PN: 0020250720-H01, Rev: 2

Importing parts from library
ITEM NAME                    STATUS
X1                           library up to date (rev1)
X2                           library up to date (rev1)
X4                           library up to date (rev1)
X500                         library up to date (rev1)
X3                           library up to date (rev1)
X2.bs                        library up to date (rev1)
X4.bs                        library up to date (rev1)
X500.bs                      library up to date (rev1)
X3.bs                        library up to date (rev1)
-Origin node: 'X1.node'
Harnice: harness 0020250720-H01 rendered successfully!

Mac:0020250720-H01-rev2 kenyonshutt$
```

5	6	7	8	9	10	

REVISION	UPDATE	STATUS	DRAWN BY	CHECKED BY	STARTED	MODIFIED
1	INITIAL RELEASE	OUTDATED	K SHUTT	W CANAVAN	7/22/25	8/10/25
2	CHANGED NOTE FOR CONNECTOR		K SHUTT	D RAIGOSA	8/10/25	8/10/25

A

B

Library flexibility

All parts in a library are version controlled per previous slide...

revision_history.tsv to track all revisions of a product

When importing an item from library, you can request different versions or overwrite imported libraries flexibly...

Library will be imported new at every Render for traceability purposes

User-editable copy (instances_list tracks “modified” from imported library)

Different instances, even with the same MPN, are imported separately

All imported items work the same way

Show me what you can do!

- Here's an output from artifact_builder called **"pdf_generator".**
- It compiled the outputs of other artifact_builders like "**harnice standard formboard builder**", "**bom to svg**", "**buildnotes to svg**"
- It knows the scale, the location, the buildnotes you need, what your parts look like, the revision history, all from the information on the Instances List

- Another page shows a 1:1 scale view on a 8.5”x11” sheet.
- Connectors and backshells shown here are just rectangles, but could easily be photos or CAD screenshots.
- Flagnote positions are not well-defined, but can be configured as needed.

- Pages can be edited with your favorite SVG editor...

- ***Of course…***
- All output artifacts are perfectly in sync with the Instances List

- Every part can have as many child coordinate systems as you need
- These can be referenced in the instances list
- Used for things like flagnotes, child parts, or other related items
- Can be used to place parts on a formboard drawing, calculate distances, etc

```
{
    "csys_parent_prefs": [
        ".node"
    ],
    "tooling_info": {
        "tools": {}
    },
    "build_notes": {},
    "csys_children": {
        "connector": {
            "x": 2,
            "y": 1.5,
            "angle": 0,
            "rotation": 90
        },
        "flagnote-1": {
            "angle": 0,
            "distance": 2,
            "rotation": 0
        },
        "flagnote-leader-1": {
            "angle": 0,
            "distance": 1,
            "rotation": 0
        },
        "flagnote-2": {
            "angle": 15,
            "distance": 2,
            "rotation": 0
        },
        "flagnote-leader-2": {
            "angle": 15,
            "distance": 1,
            "rotation": 0
        },
        "flagnote-3": {
            "angle": -15,
            "distance": 2,
            "rotation": 0
        },
        "flagnote-leader-3": {
            "angle": -15,
            "distance": 1,
            "rotation": 0
        },
```

Defining a physical layout

- For now, formboard_definition.tsv allows user to input length and angle preferences
- These sync with cables and requirements from instances_list
- Eventually, I'll make a GUI

	A	B	C	D	E	F	G
1	segment_id	node_at_end	node_at_end	length	angle	diameter	
2	segment_a	X1.node	node1	4	15	0.1	
3	segment_b	node1	X2.node	4	165	0.1	
4	segment_c	node1	node2	8	0	0.5	
5	segment_d	node2	X4.node	12	205	0.1	
6	segment_e	node2	node3	4	0	0.6	
7	segment_f	node3	X500.node	4	15	0.1	
8	segment_g	node3	X3.node	4	345	0.1	
9							
10							
11							

Interesting problems with **Harness Builder**

- Revision control
- Mapping harness reference designators to harness part numbers

Interesting problems with **Harness Builder**

- Mapping nets to harness part numbers
  - There's no guarantee that the user finds it useful to make a build part number out of every net. Assigning nets to harness part numbers is an engineering judgement call that Harnice will not solve for the engineer.
  - A system will produce a set of nets
  - For each net you find useful to make into a harness, make a part number first, and direct that part number prebuilder towards that net
- Example
  - Your system contains a bunch of COTS harnesses (cat6 ethernet cables)
  - Harnice system will render each instance as a separate net
  - There's no need to make a separate part number or harness drawing for each of those nets

Interesting problems with **System Builder**

- Channel maps
  - Just because you have a net that contains multiple connectors, that is not enough information to define which channel connects to where
  - If you have a net, you know the signals and channels of the net's connectors, and you know the compatible channel list, Harnice can produce a list

Interesting problems with **System Builder**

- Transmission line effects
  - Given that

Interesting problems with **System Builder**

- Harness disconnects

Things I learned about coding

- Git
- Fileio.path()

Infinite possibilities

Where are we as of today?

Where I want to take this