HarNice!

The free, open-source electrical system design tool.

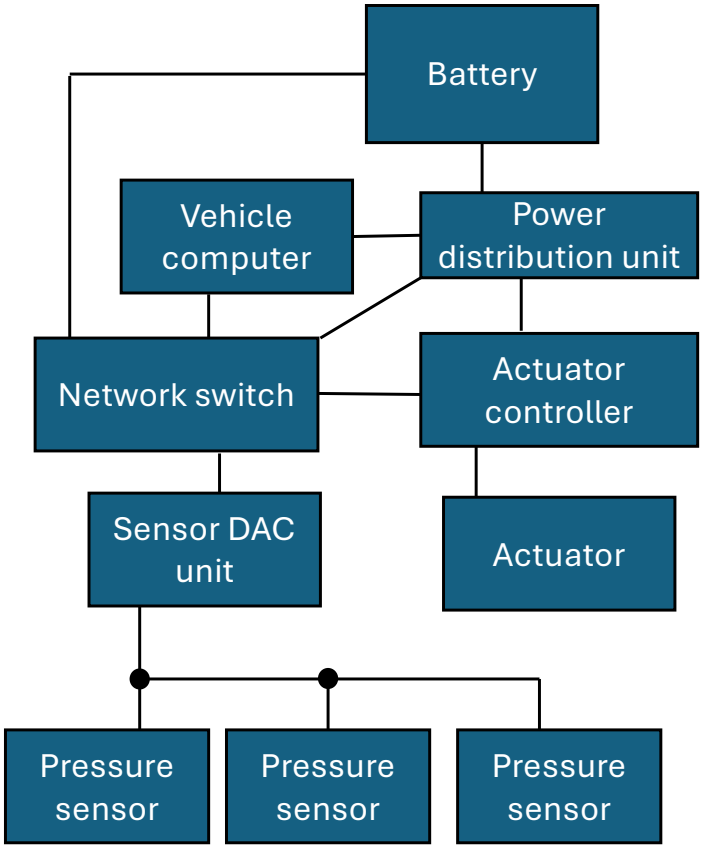
Disclaimer

- 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 implementation is any good – just judge my approach!
- I've never truly used another real systems engineering tool
  - Maybe I'm reinventing a wheel
  - Maybe we can steal all their business by making the first free and open source electrical systems engineering tool

DO NOT USE THIS SLIDESHOW FOR HOW-TO ADVICE. INFO AND TERMINOLOGY CHANGED QUCKLY WHILE I WAS DRAFTING THIS PRESENTATION.

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:
  - Concert sound system
  - Avionics system on a rocket or satellite
  - Commercial power distribution system
  - iPhone



What is **not** an electrical system?

(from the perspective of HarNice)

- Specifics of the electrical behavior going on inside a device
  - We take the assumption that any device has inputs and outputs and what it does to relate the two is not part of “system engineering”
  - It’s okay to simplify what a box does as long as it does not jeopardize your understanding of its external functionality
- Network routing
  - Software configs are agnostic to the electrical systems
- How your electrical hardware interacts with non-electrical things
  - We don’t care about g-loading, what kind of fluid your sensors are measuring, etc

How do you design your system right now?

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

Pen and paper vibes: hardly better than graph paper

- 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 devices you need and how you want them to be connected
  2. You have complete details about how each device works
     - Device 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 a harness can be imported or asserted
     - 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

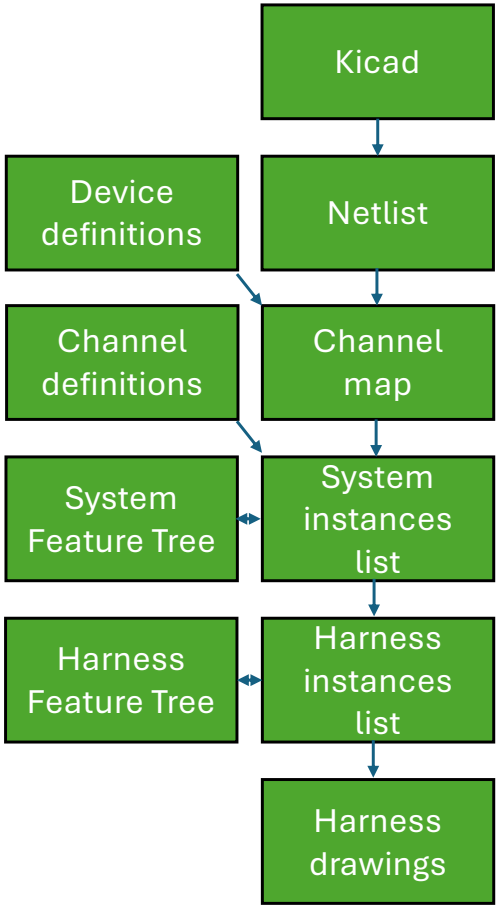
- Harnice shall have just one command which should update everything about a product
- Harnice shall allow users to input their design rules using a well-known human-readable language
- Harnice may only produce ONE deterministic solution to any product, and raise an error if multiple or zero solutions 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. The terms “connector” or “cable” don’t inherently mean anything

The solution

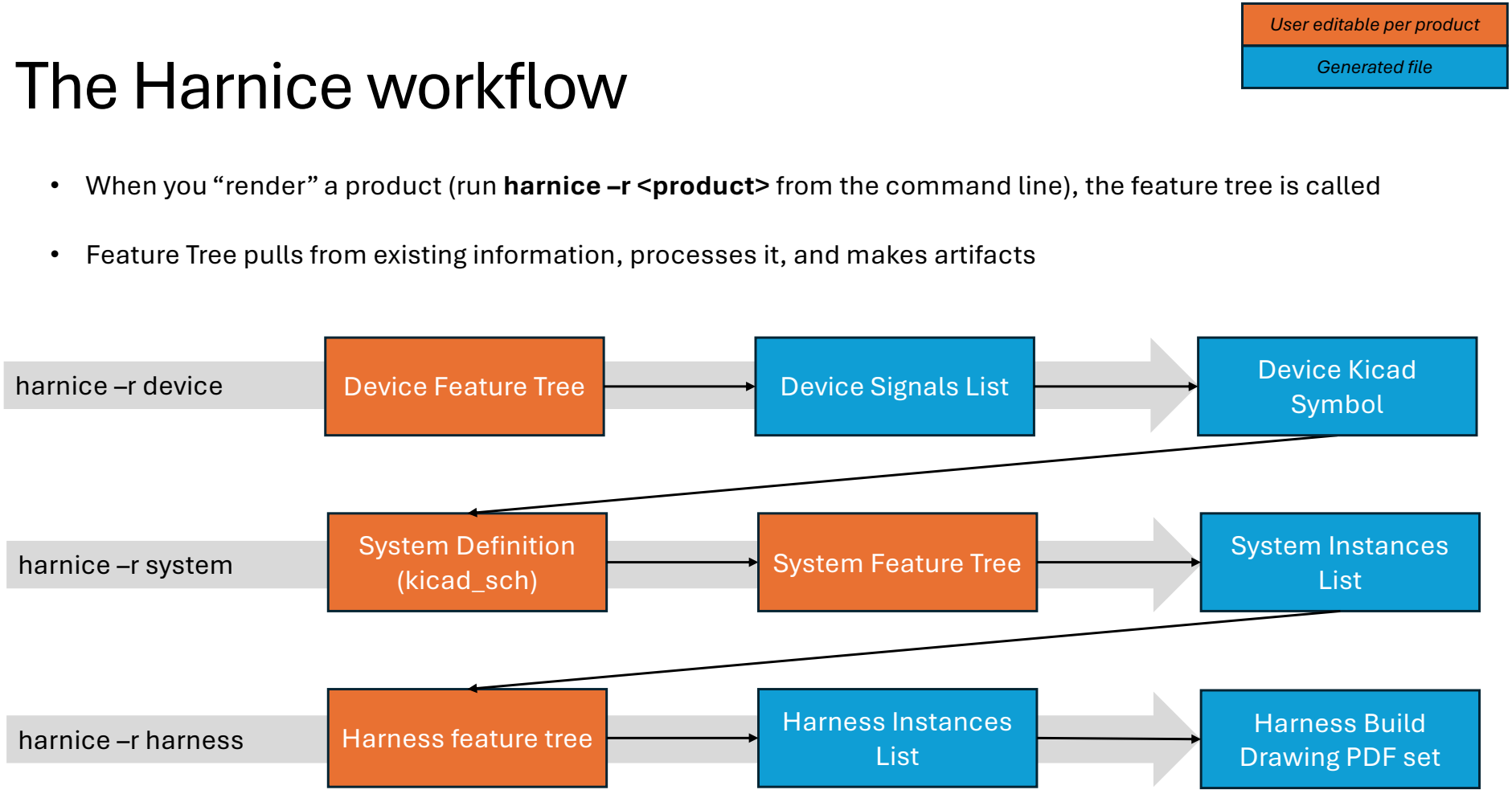
- You specify your system using Kicad or Altium
  - Components represent devices
  - Nets represent harnesses
- Harnice runs a bunch of code on your netlist to determine harness requirements
  - Builds a channel map
  - Builds a System Instances List (list of every part and every connection for every 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 devices and nets	• Which device part numbers and how many are you using • How each device is connected to each other	• Vehicle HITL • Test stand • Concert sound system • Residential AC power distribution
“Device”	A physical device that interfaces with other devices via hardware electrical signals		• Part number • Connectors • Channels	• Pressure sensor • ADC • Light switch • MacBook
“Channel”	A set of signals	• Devices have Channels • Channels exist in a centralized library • Channels are not allowed across multiple connectors yet	• 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 device	• Channels are defined to contain Signals • Device Connectors must contain all device 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 devices are included in the net	• N$1
“Harness”	A physical device that exists to link devices 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 Harnice workflow



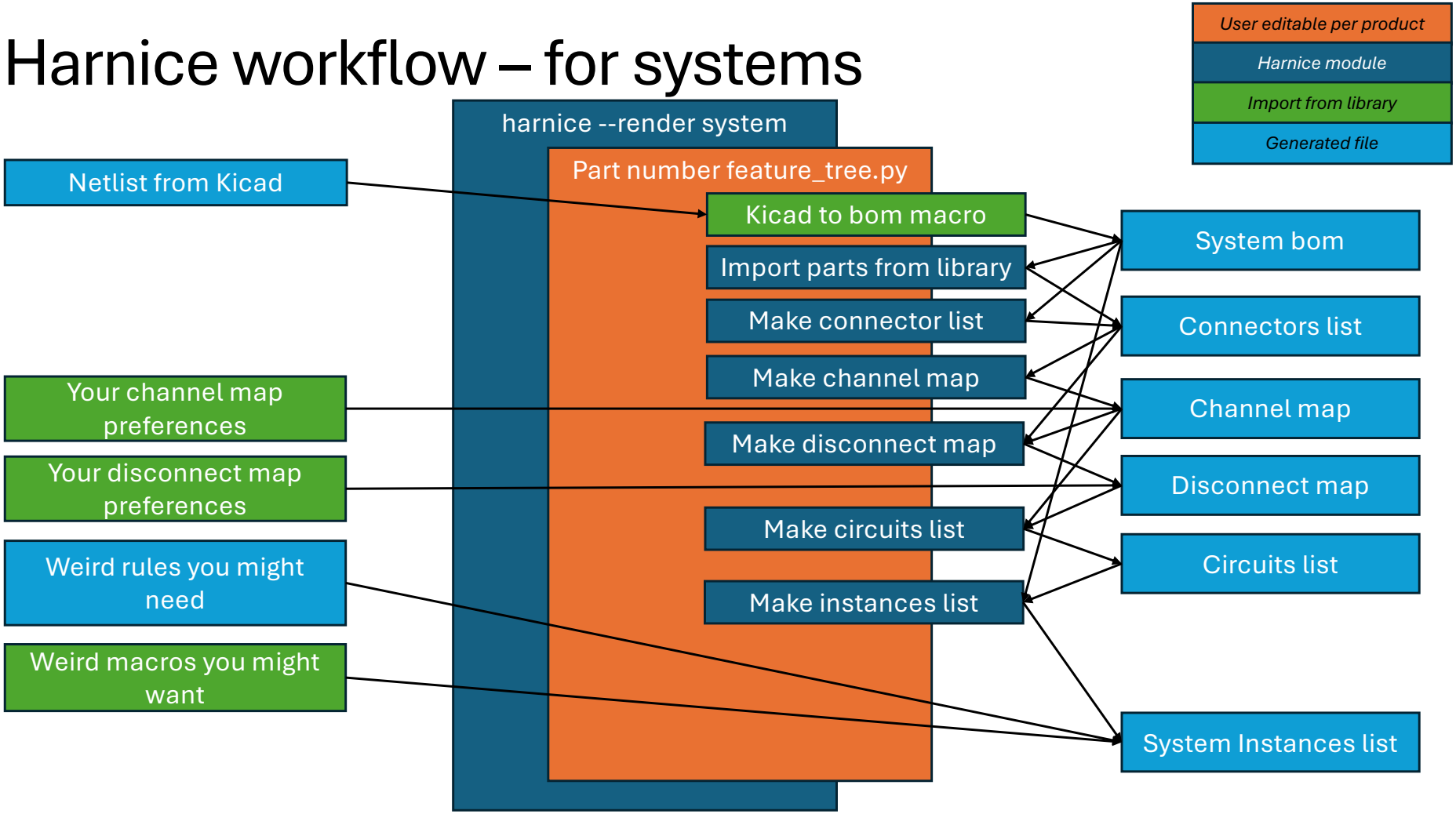
- When you “render” a product (run **harnice –r <product>** from the command line), the feature tree is called
- Feature Tree pulls from existing information, processes it, and makes artifacts

Harnice products

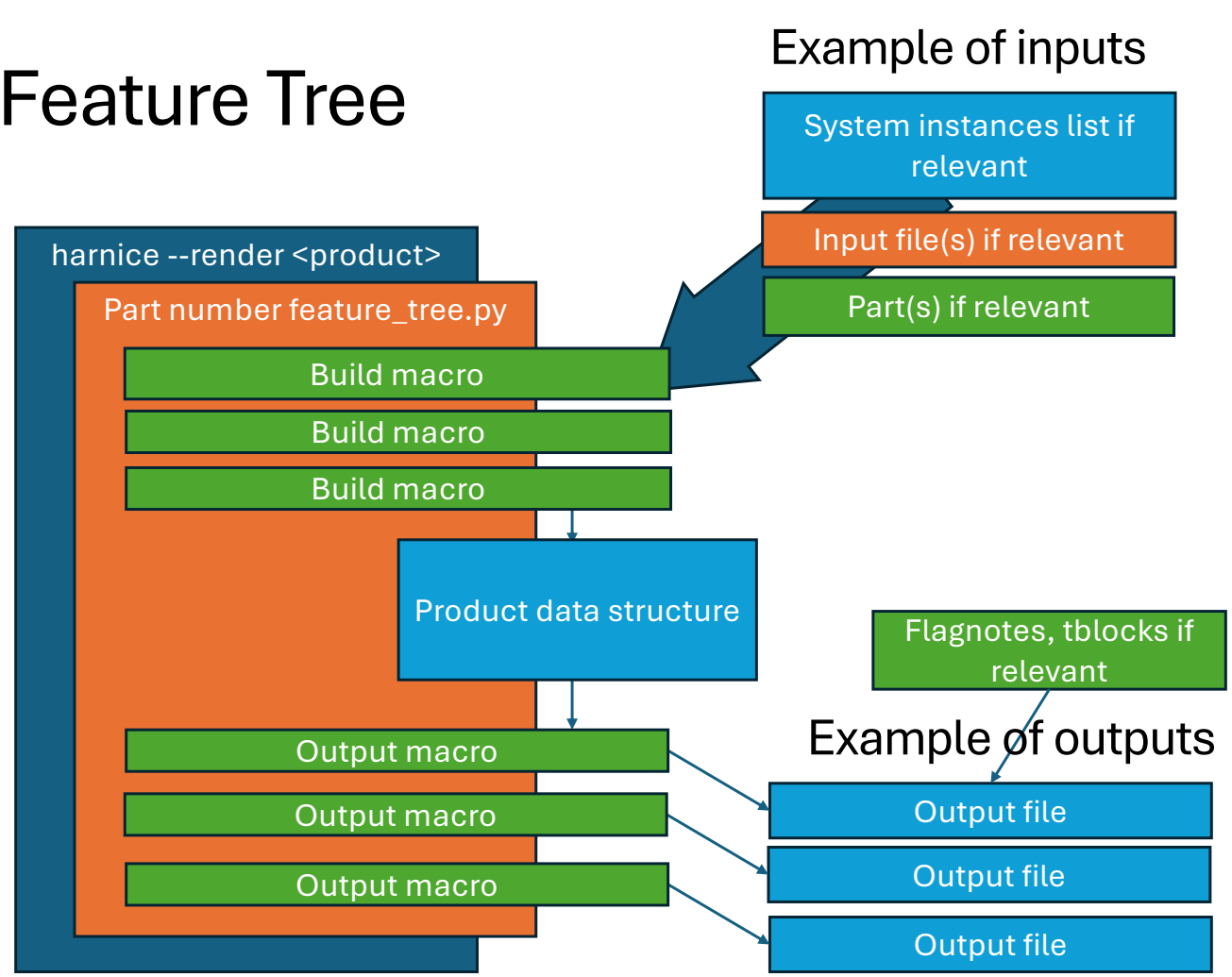
- When you “render” a product, inputs are converted to outputs

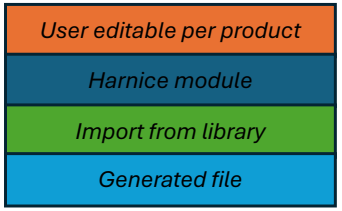
	System	Harness	Device	Part	Title block	Flagnote
Inputs (existing information)	Netlist exported from Kicad or Altium	System instances list, manual YAML or JSON harness definition	ICD.py		• Params.json	• Params.json
Generated outputs	Instances list.tsv	Instances list.tsv	Signals list.tsv .Kicad_sym	• Attributes.json • Drawing.svg	• Attributes.json • Drawing.svg	• Drawing.svg

Harnice workflow – for systems



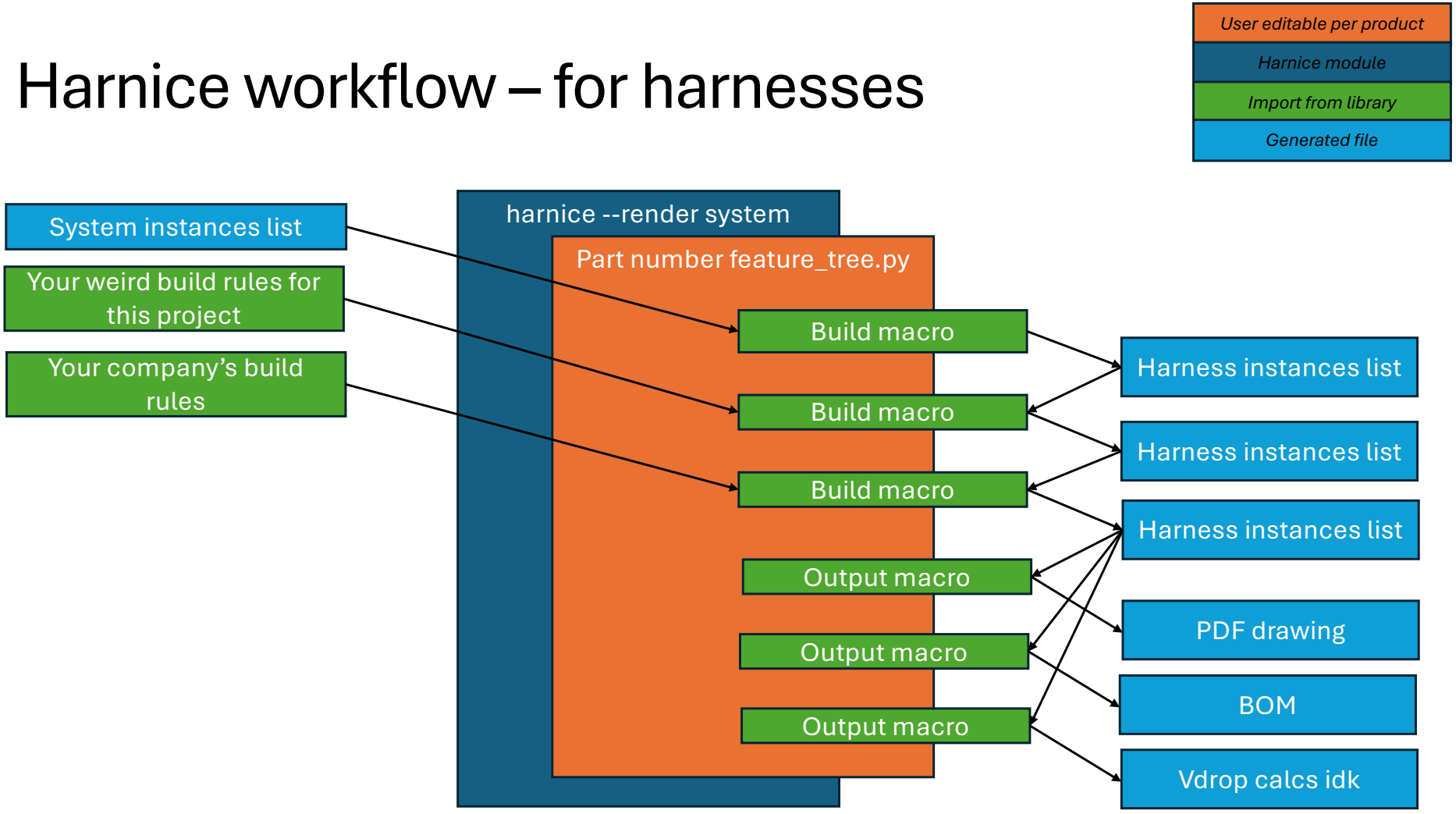
Feature Tree





- 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
- Feature Tree will also call macros, which do stuff like generate a BOM, make a PDF drawing, etc

Harnice workflow – for harnesses



System Feature Tree Example

```
from harnice import featuretree_utils, system_utils, instances_list

#========================================================================
#                    KICAD PROCESSING
#========================================================================
featuretree_utils.run_macro("kicad_pro_to_bom", "system_builder", "https://github.com/kenyonshutt/harnice-library-public")
system_utils.pull_devices_from_library()


#========================================================================
#                    CHANNEL MAPPING
#========================================================================
featuretree_utils.run_macro("kicad_pro_to_system_connector_list", "system_builder", "https://github.com/kenyonshutt/harnice-library-public")
system_utils.new_blank_channel_map()

#add manual channel map commands here. key=(from_device_refdes, from_device_channel_id)
#system_utils.map_and_record({from_key}, {to_key})

#map channels to other compatible channels by sorting alphabetically then mapping compatibles
featuretree_utils.run_macro("basic_channel_mapper", "system_builder", "https://github.com/kenyonshutt/harnice-library-public")

#if mapped channels must connect via disconnects, add the list of disconnects to the channel map
system_utils.find_shortest_disconnect_chain()

#map channels that must pass through disconnects to available channels inside disconnects
system_utils.new_blank_disconnect_map()

#add manual disconnect map commands here

#map channels passing through disconnects to available channels inside disconnects
featuretree_utils.run_macro("disconnect_mapper", "system_builder", "https://github.com/kenyonshutt/harnice-library-public")

#process channel and disconnect maps to make a list of every circuit in your system
system_utils.make_circuits_list()

#========================================================================
#                    INSTANCES LIST
#========================================================================
instances_list.make_new_list()
instances_list.add_connector_contact_nodes_and_circuits()

for instance in instances_list.read_instance_rows():
    if instance.get("item_type") == "Connector":
        if instance.get("mating_device_connector_mpn") == "XLR3M":
            instances_list.modify(instance.get("instance_name"),{
                "mpn":"NC3FXX"
            })
        elif instance.get("mating_device_connector_mpn") == "XLR3F":
            instances_list.modify(instance.get("instance_name"),{
                "mpn":"NC3MXX"
            })
```

Channel Types

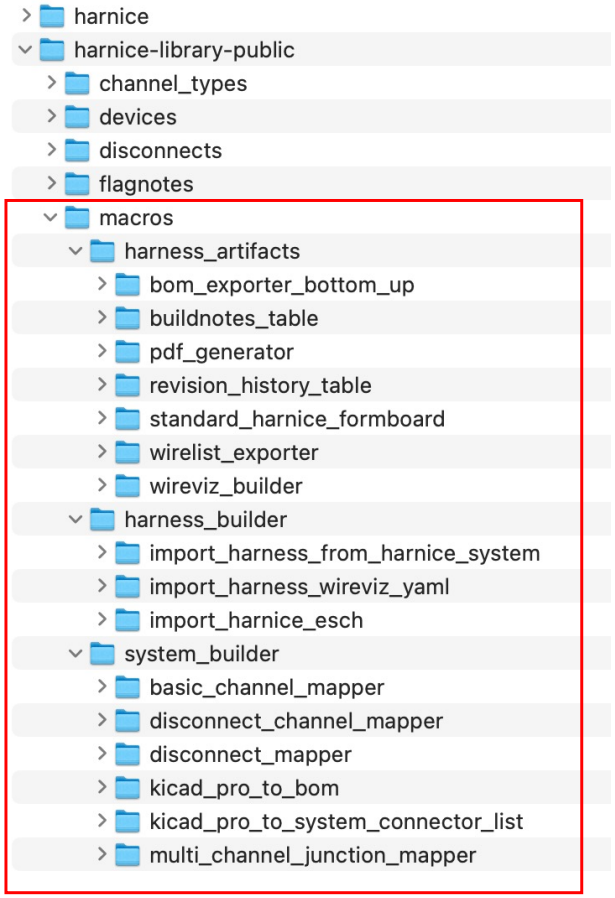
Refer to a channel type as

```
channel_type = (channel_type_id, repo)
```

Represents a type of channel that is compatible with other channels of similar types that can be mapped to each other.

Macros

Macros I currently have written in the public library



- A macro is a chunk of Python that has access to your project files or any other Python-capable function
- When you call featuretree_utils.run_macro(), it will import the macro from a library and run it in your script
- Some macros are designed to be used to build systems, build harnesses, or export contents “artifacts” from a harness instances list

Build Macros

- Intended to add or modify lines on an instances list based on a standard set of rules or instructions
  - Can read information from the instances list
  - Can read information from other support files
- Examples
  - featuretree.runmacro(“import_wireviz_yaml”, “public”)
    - Reads a wireviz YAML (another commonly used harness design format)
  - featuretree. runmacro(“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)

Signals Lists

- Signals Lists are the primary way Harnice stores information about devices
- Source of truth for devices or disconnects
- Exhaustive list of every electrical connection going into or out of the device or disconnect

Device signals list (for Shure SM58 microphone)

channel_id	signal	connector_name	cavity	connector_mpn	channel_type
out1	pos	out1	2	XLR3M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
out1	neg	out1	3	XLR3M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
out1-shield	chassis	out1	1	XLR3M	(5, 'https://github.com/kenyonshutt/harnice-library-public')

Disconnect signals list (for a bespoke audio use case)

channel_id	signal	A_cavity	B_cavity	A_connector_mpn	A_channel_type	B_connector_mpn	B_channel_type
ch0	pos	24	24	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch0	neg	12	12	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch0-shield	chassis	25	25	DB25F	(5, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(5, 'https://github.com/kenyonshutt/harnice-library-public')
ch1	pos	10	10	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch1	neg	23	23	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch1-shield	chassis	11	11	DB25F	(5, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(5, 'https://github.com/kenyonshutt/harnice-library-public')
ch2	pos	21	21	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch2	neg	9	9	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch2-shield	chassis	22	22	DB25F	(5, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(5, 'https://github.com/kenyonshutt/harnice-library-public')
ch3	pos	7	7	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch3	neg	20	20	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch3-shield	chassis	8	8	DB25F	(5, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(5, 'https://github.com/kenyonshutt/harnice-library-public')
ch4	pos	18	18	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch4	neg	6	6	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch4-shield	chassis	19	19	DB25F	(5, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(5, 'https://github.com/kenyonshutt/harnice-library-public')
ch5	pos	4	4	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch5	neg	17	17	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch5-shield	chassis	5	5	DB25F	(5, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(5, 'https://github.com/kenyonshutt/harnice-library-public')
ch6	pos	15	15	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch6	neg	3	3	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch6-shield	chassis	16	16	DB25F	(5, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(5, 'https://github.com/kenyonshutt/harnice-library-public')
ch7	pos	1	1	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch7	neg	14	14	DB25F	(1, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
ch7-shield	chassis	2	2	DB25F	(5, 'https://github.com/kenyonshutt/harnice-library-public')	DB25M	(5, 'https://github.com/kenyonshutt/harnice-library-public')

Rules about Signals Lists

- Every combination of `(channel_id, signal)` must be unique within the signals list
  - You can't have two "ch1, pos" signals on the same device
- Every signal of a `channel_type` must be present in the Signals List
  - If your `channel_type` requires "positive" and "negative", you have to have both in your signals list
- Every signal in the Signals List must be present in the `channel_type`
  - If you need to add signals that aren't already part of your pre-defined channel_type, you'll need to define a new channel_type.
- You can't put signals of the same channel on different connectors
  - While this may sound convenient, it breaks a lot of internal assumptions Harnice is making on the back end about how to map channels.
  - If you need to do this, I recommend defining one channel type per signal, then write a macro for mapping the channels to their respective destinations.
- "A" and "B" channels of the same disconnect must be compatible with each other

Signals List Fields

`signals_list.write_signal()`

Device signals list

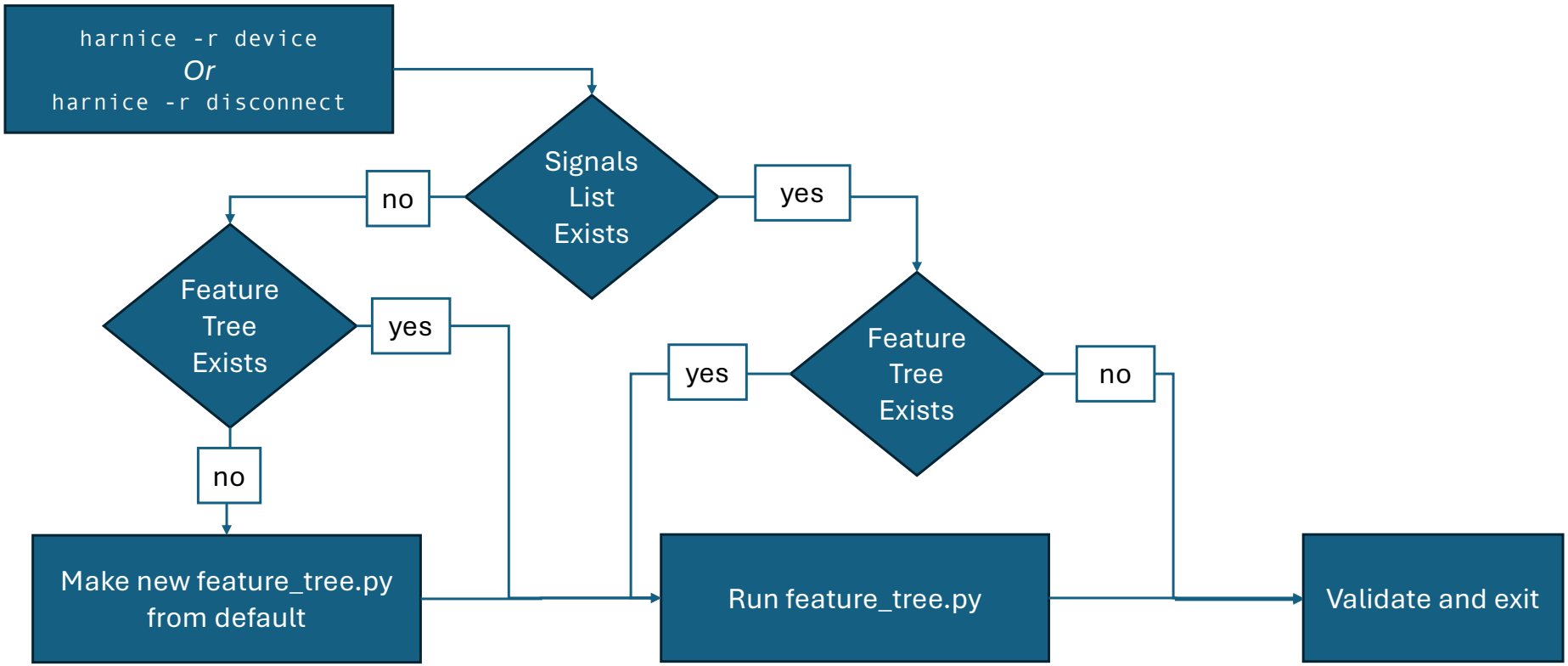
channel_id	signal	connector_name	cavity	connector_mpn	channel_type
Unique name of the channel that this signal is part of	Electrical signal “positive”, “TX”, etc	Name of the connector	Cavity name of where this signal lands on your device	Be specific, this may directly end up in your BOM later	Intended purpose, mate, and map compatibility is contained here

Disconnect signals list

channel_id	signal	A_cavity	B_cavity	A_connector_mpn	A_channel_type	B_connector_mpn	B_channel_type
Unique name of the channel that this signal is part of	Electrical signal “positive”, “TX”, etc	Cavity name of where this signal lands on the “A” side of this disconnector	Usually the same as “A” cavity if the connectors are compatible	Be specific, this may directly end up in your BOM later	Intended purpose, mate, and map compatibility is contained here		Not necessarily the same as “A_channel_type” because one side of disconnect is usually “input” and the other is “output”

Signals List Feature Tree

You can use a script to easily write long Signals Lists. Rely on basic Python to avoid typing manually.



KiCad Integration

- When you render a device Signals List, it’ll make a KiCad schematic symbol in the parent directory

- KiCad Symbol will contain ports that match the set of connectors that you’ve specified in Signals List

- Render will not affect placement or graphic design of your symbol, just port count and symbol attributes

Signals List Lightweight Render

- If you need to build a KiCad block diagram quickly, without specifying channels and signals, you can with --lightweight.

- Run `harnice -l device` or `harnice --lightweight device` from your device directory.

- This will follow the same flowchart as --render, but truncate the validation process.

- You will not be able to map channels with --lightweight

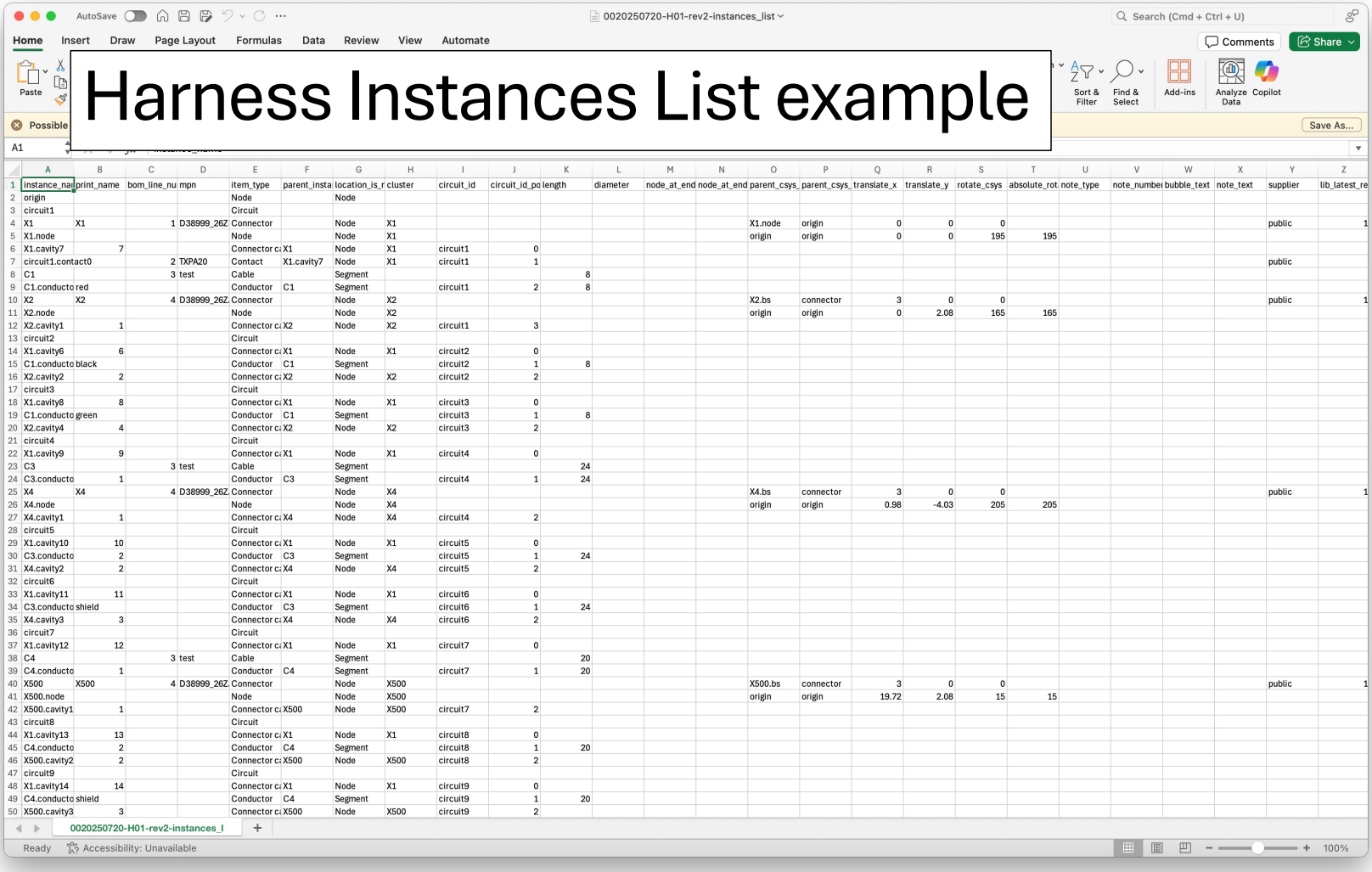
Signals List Python Commands

Command	Args	Returns	Purpose
`signals_list.new_list()`			Create a new list
`signals_list.write_signal()`	See earlier slides on what's required and why		Add a signal to a signals list
`signals_list.signals_of_channel_type()`	Channel type in touple form (channel_type_id, repo) *One arg, in parenthesis*	List of all the signals of that channel type	Use to validate that you've specified all the necessary channels when constructing a signals list
`signals_list.compatible_channel_types()`	Channel type in touple form (channel_type_id, repo)	List of touples of all other compatible channel types	See if a channel in question is compatible with another channel
`signals_list.cavity_of_signal()`	1. channel_id (what channel are you looking for) 2. signal (what signal are you looking for) 3. path_to_signals_list (where are you looking for it at)	Cavity	Quickly find which cavity a signal of a channel lands on in a specific device
`signals_list.connector_name_of_channel()`	1. channel_id (what channel are you looking for) 2. path_to_signals_list (where are you looking for it at)	Connector name	Quickly find which connector a channel exists on in a specific device
`signals_list.path_of_channel_type()`	Channel type in touple form (channel_type_id, repo)	Local path to a channel_types spreadsheet	Given a channel_type touple, find the local path to its source-of-truth channel types list
`signals_list.parse_channel_type()`	Channel type in touple form (channel_type_id, repo)	list of size 2: [int channel type id,	Given a channel_type touple, split it into parsed values that are easy to interact with

Data structure: “instances_list.tsv”

- 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'
]
```



	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 ca	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 ca	X2	Node	X2	circuit1	3																
13	circuit2				Circuit																					
14	X1.cavity6	6			Connector ca	X1	Node	X1	circuit2	0																
15	C1.conducto	black			Conductor	C1	Segment		circuit2	1	8															
16	X2.cavity2	2			Connector ca	X2	Node	X2	circuit2	2																
17	circuit3				Circuit																					
18	X1.cavity8	8			Connector ca	X1	Node	X1	circuit3	0																
19	C1.conducto	green			Conductor	C1	Segment		circuit3	1	8															
20	X2.cavity4	4			Connector ca	X2	Node	X2	circuit3	2																
21	circuit4				Circuit																					
22	X1.cavity9	9			Connector ca	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 ca	X4	Node	X4	circuit4	2																
28	circuit5				Circuit																					
29	X1.cavity10	10			Connector ca	X1	Node	X1	circuit5	0																
30	C3.conducto	2			Conductor	C3	Segment		circuit5	1	24															
31	X4.cavity2	2			Connector ca	X4	Node	X4	circuit5	2																
32	circuit6				Circuit																					
33	X1.cavity11	11			Connector ca	X1	Node	X1	circuit6	0																
34	C3.conducto	shield			Conductor	C3	Segment		circuit6	1	24															
35	X4.cavity3	3			Connector ca	X4	Node	X4	circuit6	2																
36	circuit7				Circuit																					
37	X1.cavity12	12			Connector ca	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 ca	X500	Node	X500	circuit7	2																
43	circuit8				Circuit																					
44	X1.cavity13	13			Connector ca	X1	Node	X1	circuit8	0																
45	C4.conducto	2			Conductor	C4	Segment		circuit8	1	20															
46	X500.cavity2	2			Connector ca	X500	Node	X500	circuit8	2																
47	circuit9				Circuit																					
48	X1.cavity14	14			Connector ca	X1	Node	X1	circuit9	0																
49	C4.conducto	shield			Conductor	C4	Segment		circuit9	1	20															
50	X500.cavity3	3			Connector ca	X500	Node	X500	circuit9	2																

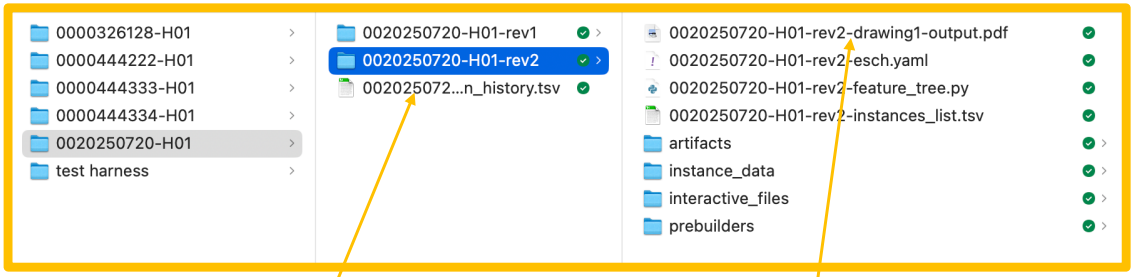
Local Paths vs Traceable URLs

Libraries

- Parts, macros, titleblocks, flagnotes, 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 if you don’t want!

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
- Harnice will not render a revision if there’s data in the “status” field, i.e. “released” or “outdated”
- Revision information 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

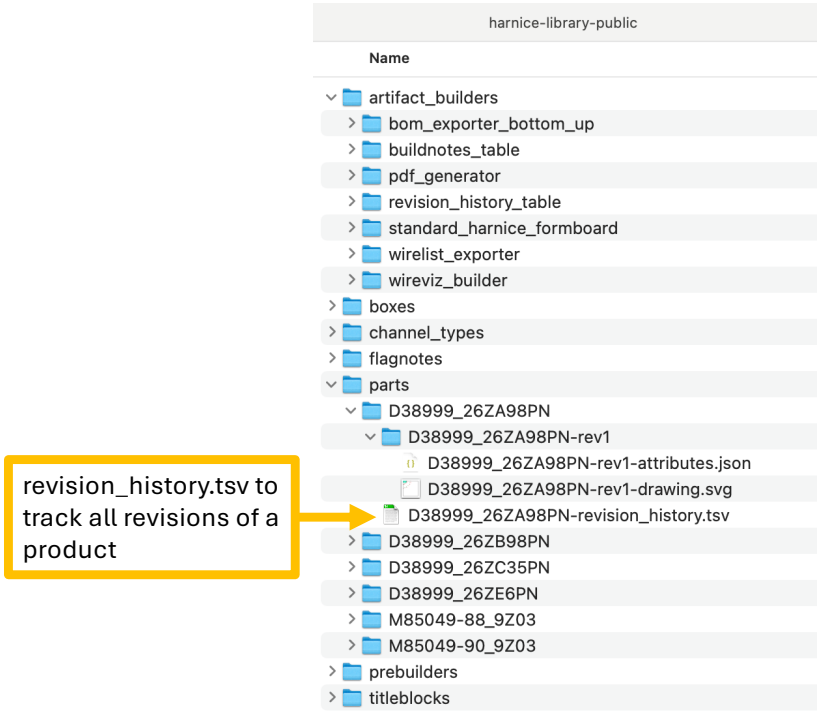
How to Make Outputs!

- Output Macros 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!

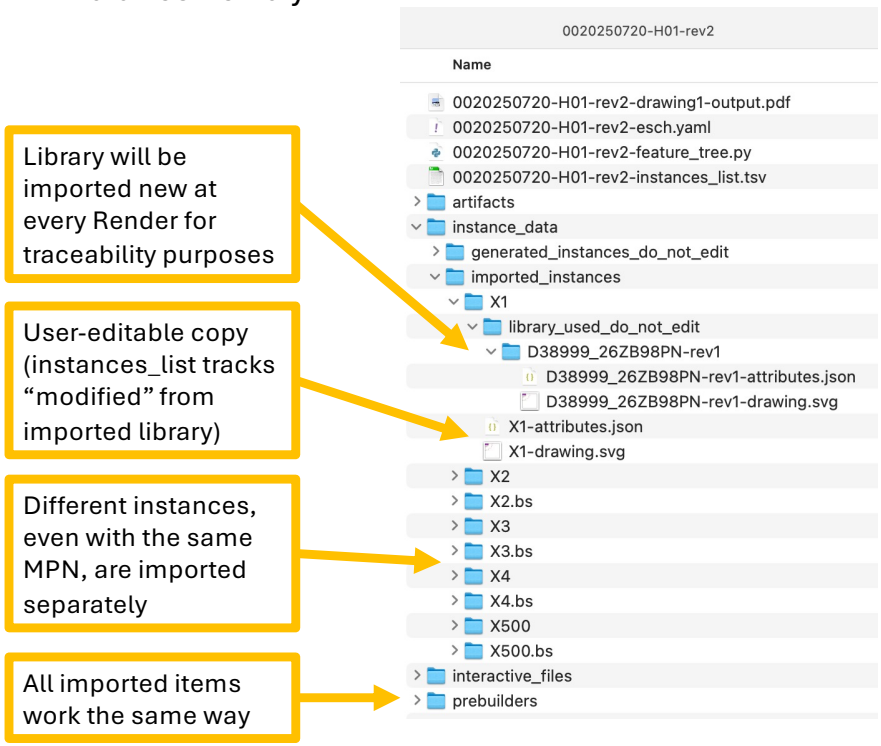
```
#======================================================================
#                  CONSTRUCT HARNESS ARTIFACTS
#======================================================================
scales = {
    "A": 1
}

featuretree_utils.run_macro("bom_exporter_bottom_up", "harness_artifacts", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="bom1")
featuretree_utils.run_macro("standard_harnice_formboard", "harness_artifacts", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="formboard1", scale=scales.get("A"))
featuretree_utils.run_macro("wirelist_exporter", "harness_artifacts", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="wirelist1")
featuretree_utils.run_macro("revision_history_table", "harness_artifacts", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="revhistory1")
featuretree_utils.run_macro("buildnotes_table", "harness_artifacts", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="buildnotestable1")
featuretree_utils.run_macro("pdf_generator", "harness_artifacts", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="drawing1", scales=scales)

featuretree_utils.copy_pdfs_to_cwd()
```

Library flexibility

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

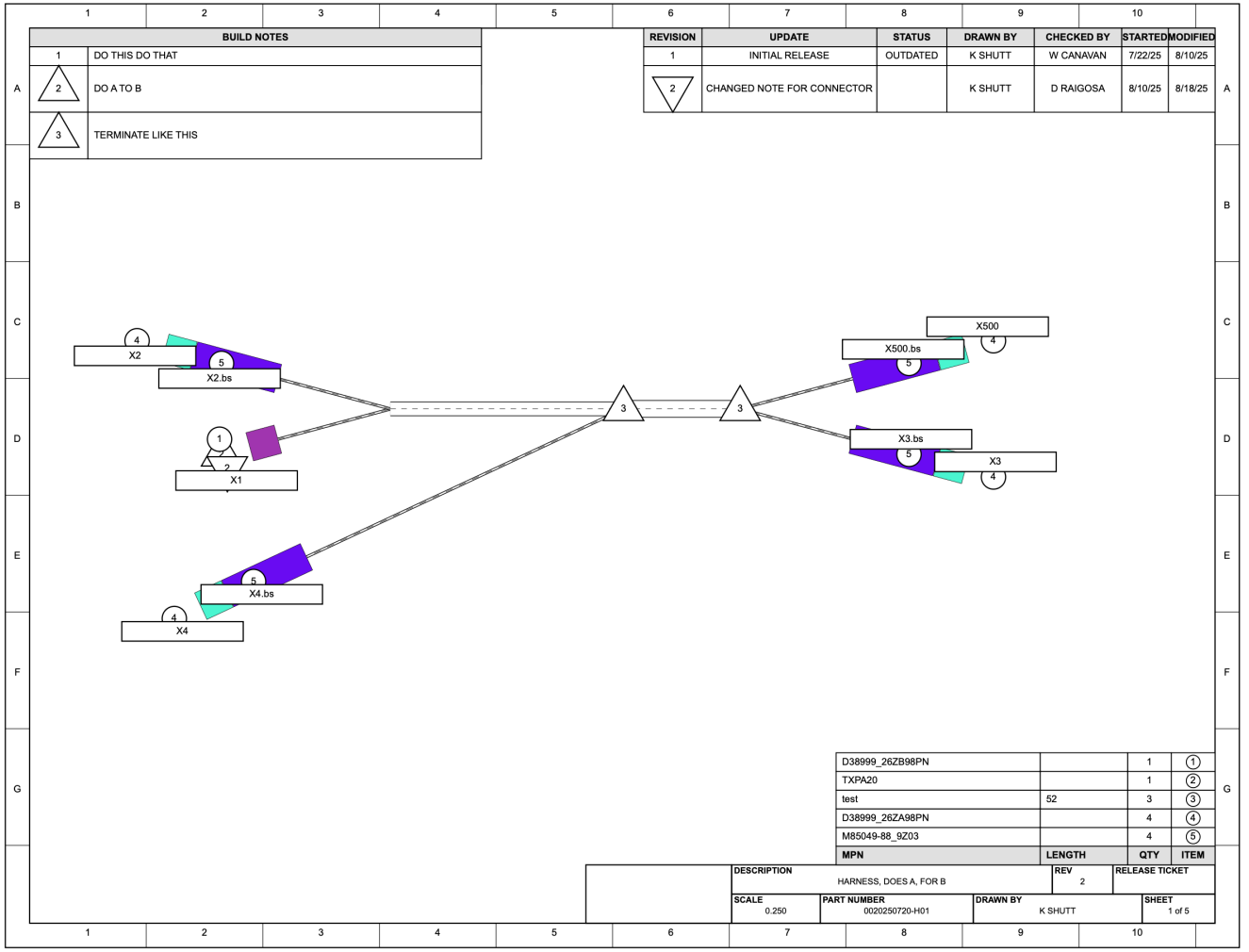


When importing an item from library, you can request different versions or overwrite imported libraries flexibly…

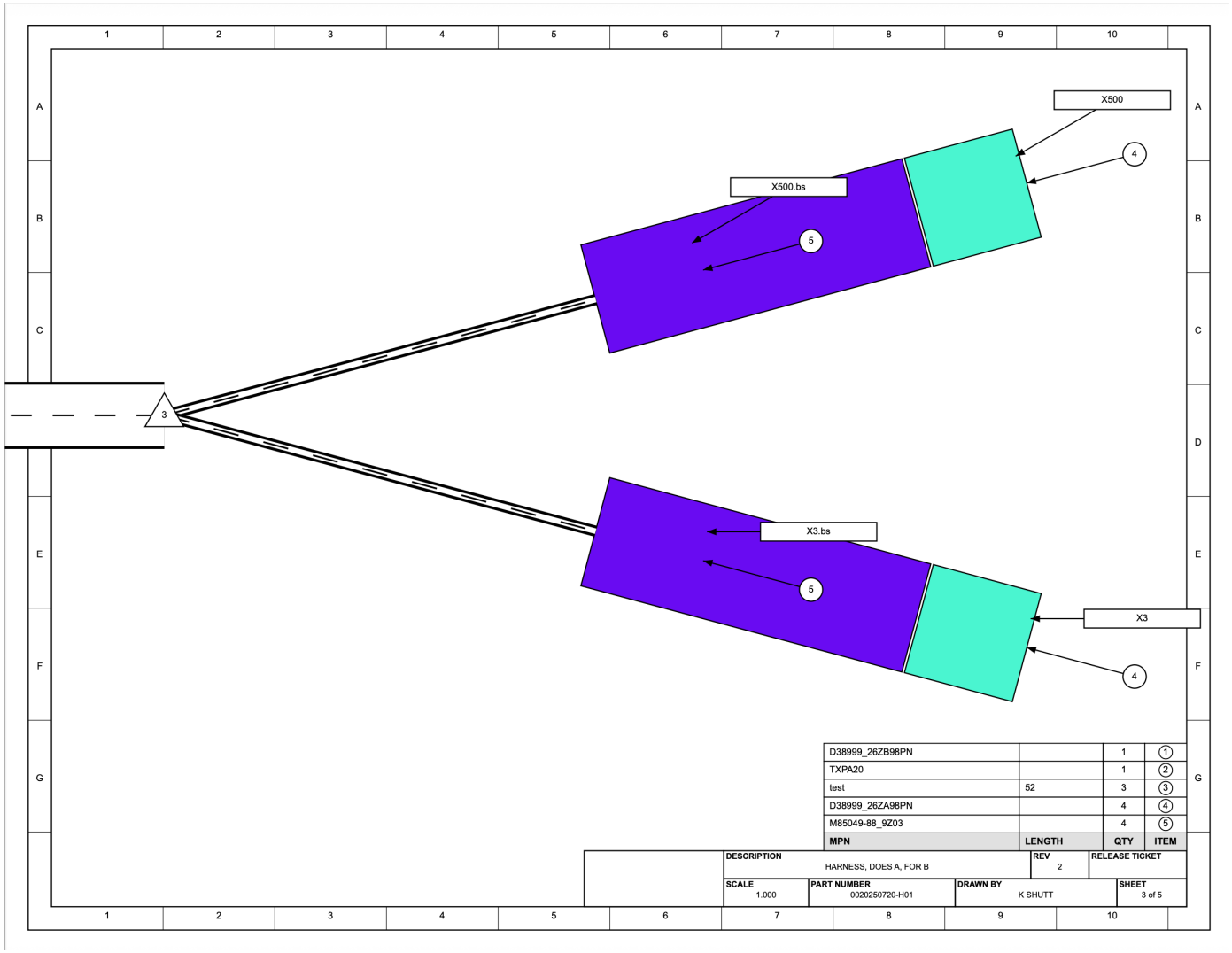


Show me what you can do!

- Here's an output from a macro called **"pdf_generator".**
- It compiled the outputs of other output macros 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.

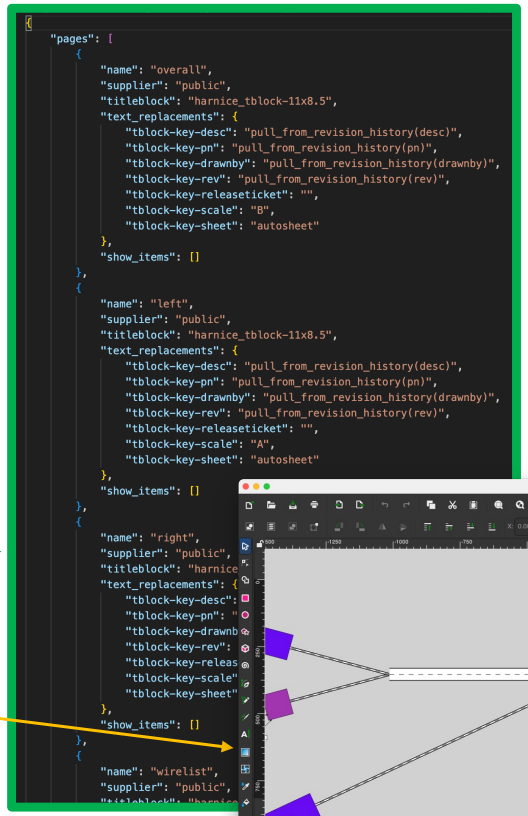


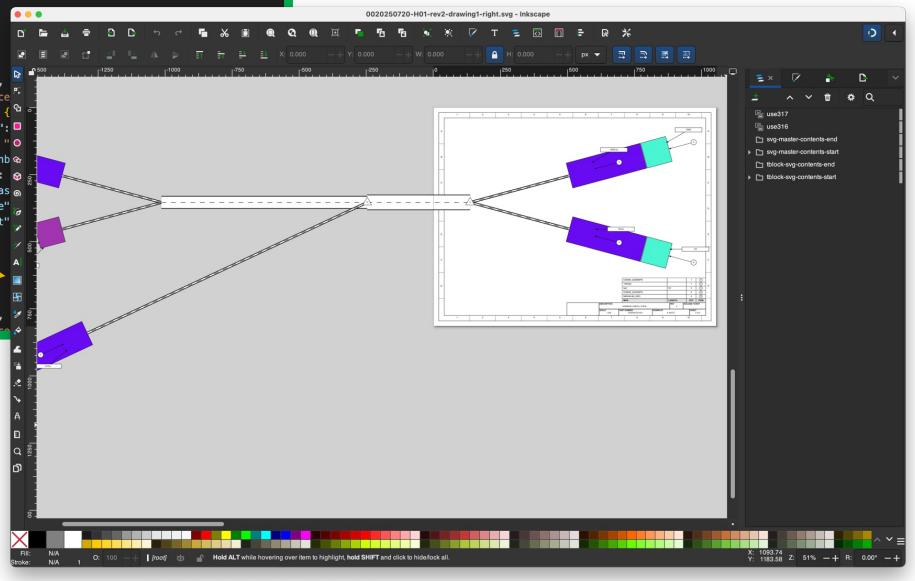
- Pages can be edited with your favorite SVG editor...

0020250720-H01-rev1

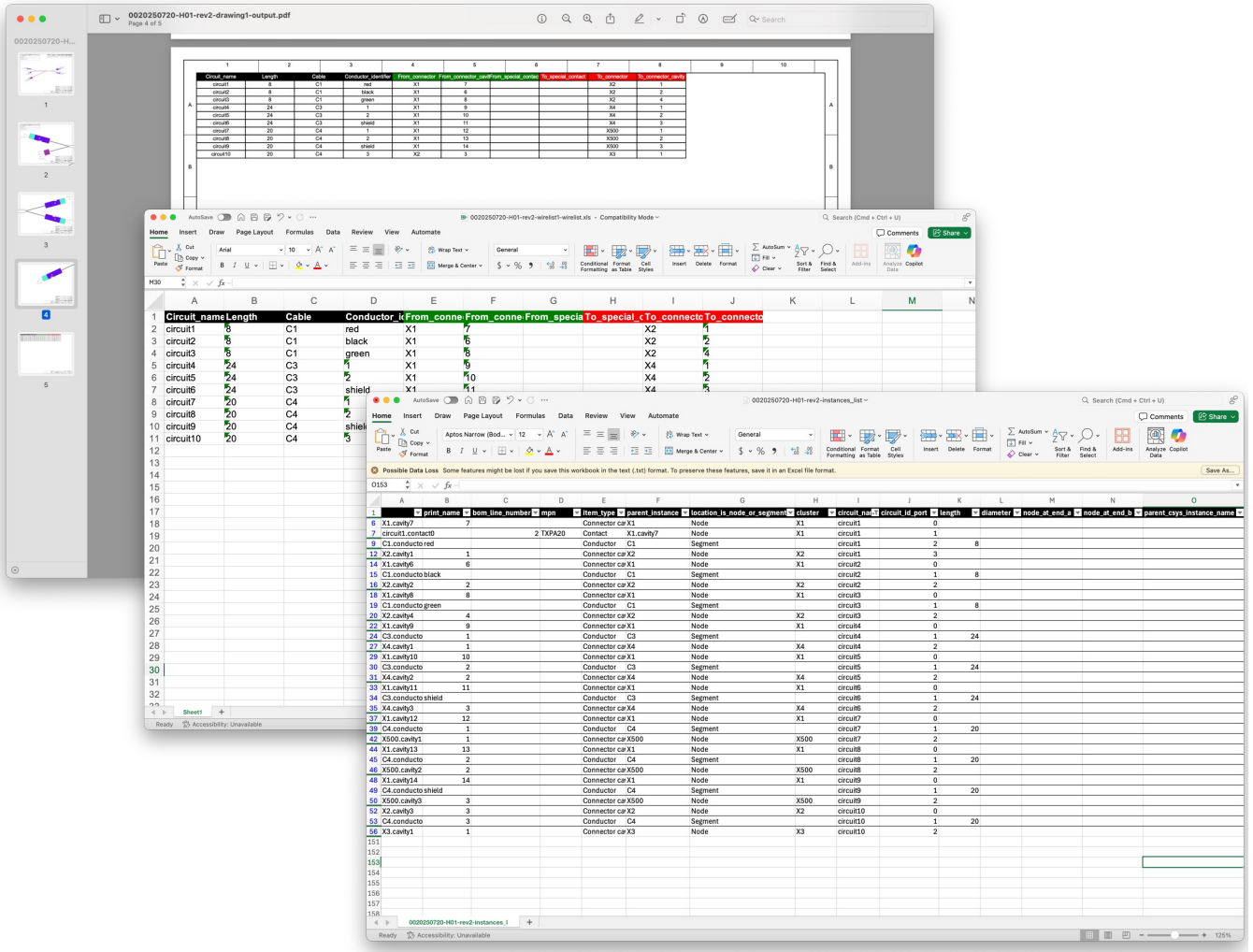
Name

- 0020250720-H01-rev1-drawing1-output.pdf
- 0020250720-H01-rev1-esch.yaml
- 0020250720-H01-rev1-feature_tree.py
- 0020250720-H01-rev1-instances_list.tsv
- artifacts
  - 0020250720-H01-rev1-formboard_graph_definition.svg
  - bom_exporter_bottom_up-bom1
  - buildnotes_table-buildnotestable1
  - pdf_generator-drawing1
    - 0020250720-H01-rev1-drawing1-mastercontents.svg
    - 0020250720-H01-rev1-drawing1-output.pdf
    - 0020250720-H01-rev1-page_setup.json
    - library_used_do_not_edit
    - page_svgs
      - 0020250720-H01-rev1-drawing1-left.svg
      - 0020250720-H01-rev1-drawing1-overall.svg
      - 0020250720-H01-rev1-drawing1-right.svg
      - 0020250720-H01-rev1-drawing1-wirelist.svg
    - pdf_generator.py
    - tblock_svgs
  - revision_history_table-revhistory1
  - standard_harnice_formboard-formboard1
  - standard_harnice_formboard-formboard2
  - standard_harnice_formboard-formboard3
  - wirelist_exporter-wirelist1
- instance_data
- interactive_files
- prebuilders

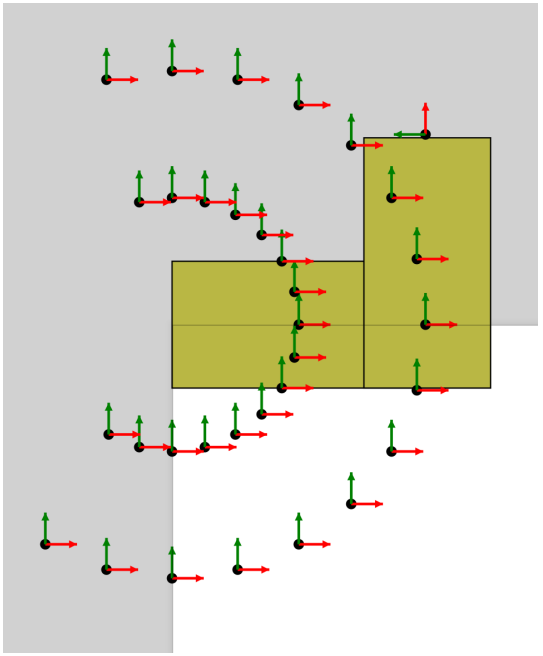




- ***Of course…***
- All output artifacts are perfectly in sync with the Instances List



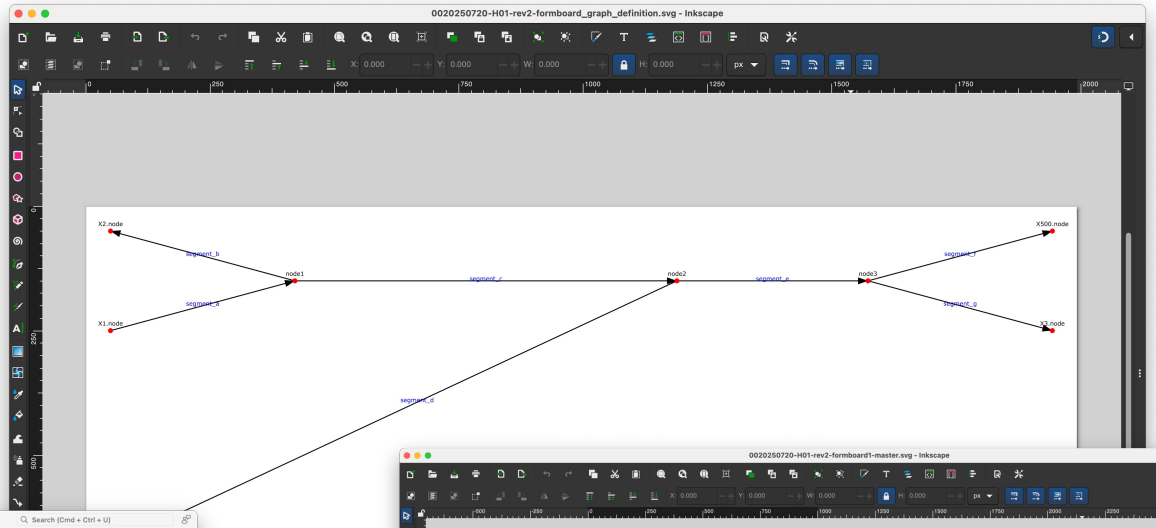
- Every part can have as many child coordinate systems as you need
- These can be referenced in the instances list
- Used for things like flagnotes, child parts, or other related items
- Can be used to place parts on a formboard drawing, calculate distances, etc

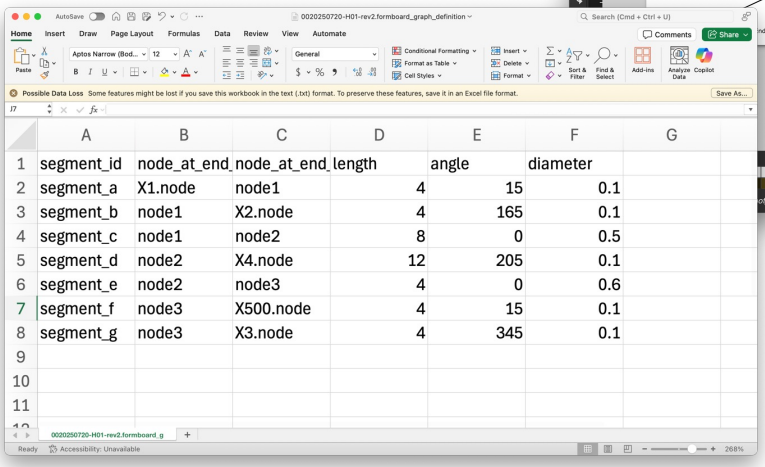



```
{
    "csys_parent_prefs": [
        ".node"
    ],
    "tooling_info": {
        "tools": {}
    },
    "build_notes": {},
    "csys_children": {
        "connector": {
            "x": 2,
            "y": 1.5,
            "angle": 0,
            "rotation": 90
        },
        "flagnote-1": {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