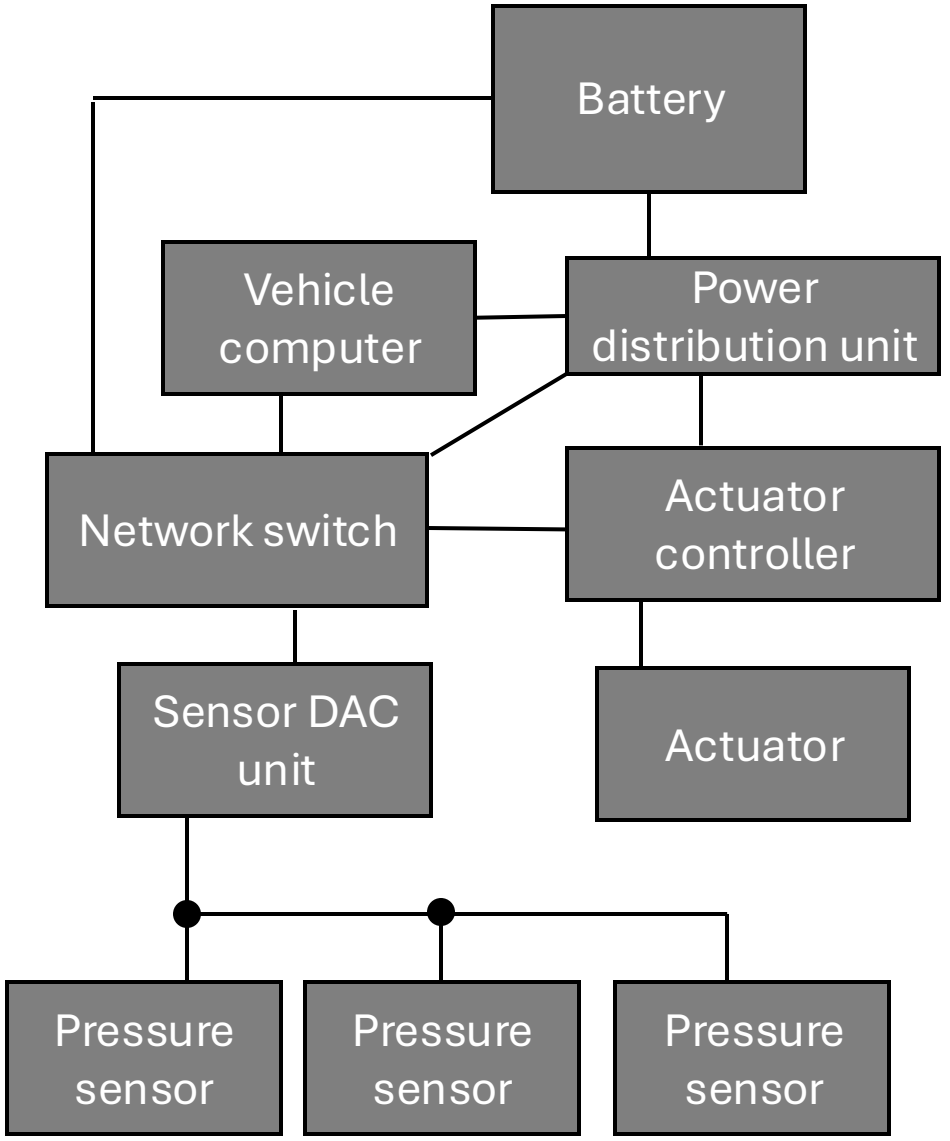
Har*nice!*

Your free, open-source electrical system CAD tool.

BACKGROUND

Define “Electrical System”

- An electrical system is any collection of circuit boards, electrical boxes, devices, etc, that connect to and interact with each other with wires or harnesses
- Examples:
  - Concert sound system
  - Control compute system for plane, car, rocket
  - Commercial power distribution system
  - iPhone
  - City power distribution network



What problem is Harnice trying to solve?

- There should always be **one single source of truth** that describes the devices in your system and how they are connected. Harnice helps designers keep track of it.

- Harnice provides a landing pad for designers to be able to define design rules that can build, validate, simulate, or perform any other action on their system design in a clear, concise, human-readable, and machine-readable format.

- Harnice helps designers produce any possible downstream data that can be derived from the system definition. Any system artifact will always be directly traceable back to the original source of truth.

Where does the Harnice scope end?

at this point

- We're not yet concerned about what's going on inside your devices
  - Only how they interact with the outside world
- Software configuration
  - Sure, you can write any rule you want into your Harnice system processor, but Harnice is intended to track hardware.
- How your electrical hardware interacts with non-electrical things
  - Don't care about size, weight, fluid type, mounting, physical, thermal, environmental compliance, etc.
- As-built configurations
  - We aren't yet offering complete ERP/MES functionality

DESIGN METHODOLOGY

How are you designing your system right now?

Existing "Smart" Software Packages

- Zuken, E-plan, Altium Harnessing, etc
- **SUPER** expensive
- Training-intensive
- Not user-friendly
- Not customizable

Existing Manual Options

- Literal paper and pen hand drawings
- Vizio, Powerpoint, Excel, MS Paint, WireViz make good-looking drawings but without metadata
- Design rule checks impossible
- Conflicting sources of truth

Harnice solves all of this!

Instead, let the machines work for **you!**

Without Harnice…

- Designers **compile, read, sort through** information either from a design guide, from industry knowledge, or other sources…

  …manually…

- while documenting the **results** of their studies, until their design is complete.
- Bad because there's inherently less traceability back to design intent, rampant broken links to sources of truth, results scale proportionally to design time

Instead, with Harnice…

- Designers are encouraged to spend their time documenting their **design inputs** (standards, rules, trades, processes), in a machine-readable format.
- So much better because python does the boring work while the designers put their brains to use!

Minimum viable inputs…

Harnice will operate on just the following…

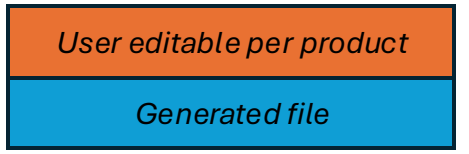
- Devices are fully defined
  - Every signal, channel, connector is accounted for in every device
  - Sufficient device electrical behavior is defined
- A block diagram is drawn by the designer
  - Each device is accounted for, and each harness connection is made linking connectors
- Your rules are qualitatively expressed
  - Channel mapping preferences
  - Part selection tree
  - Length, size, count, device attribute requirements
  - Any other rule you could possibly ever need
- Physical harness routing information can be imported or manually defined

… regard unlimited outputs

Given a system definition, the following can be produced automatically, predictably, effortlessly:

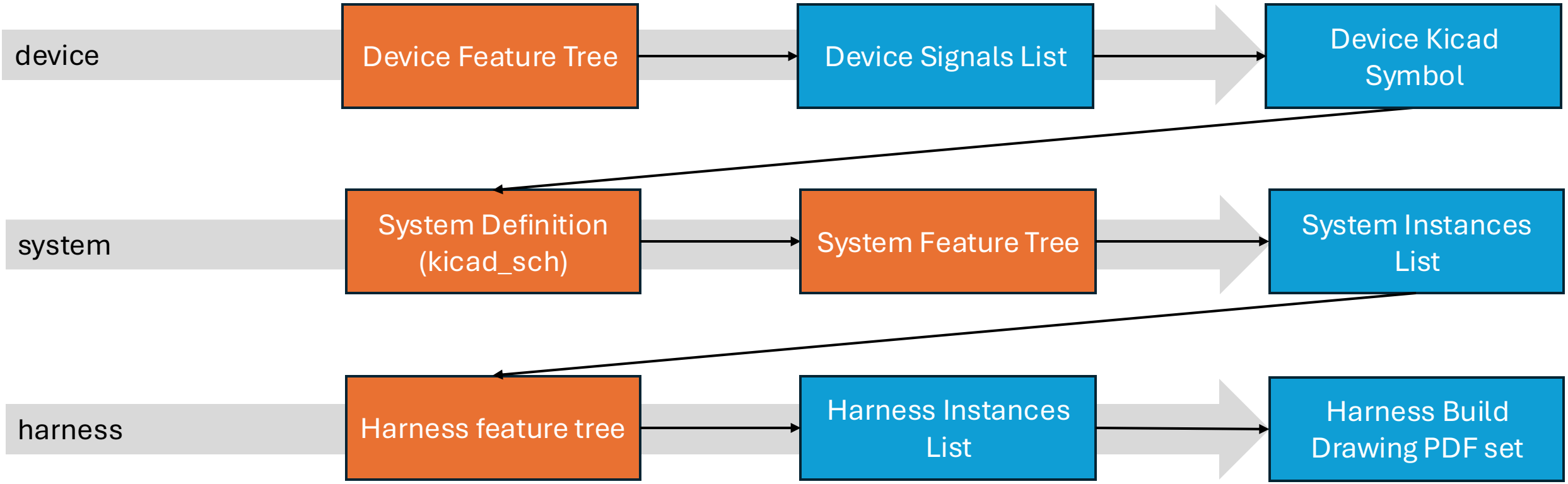
- Bill of materials of your entire system
  - (devices, harnesses, parts, cable lengths, bits of heat shrink, volume of solder, you name it)
  - Versions, release dependencies, part number per reference designator, etc
- Every harness build drawing, formboard, wirelist in PDF and CSV formats, all at once
  - Even stylistic choices can be automated into your system render
- Electrical behavior
  - Steady-state and even transient system responses can be simulated with just simple device and harness lump-element definitions
- Write your own!
  - Harnice makes it easy to write simple Python scripts that can tell you anything about your system.

THE SOLUTION

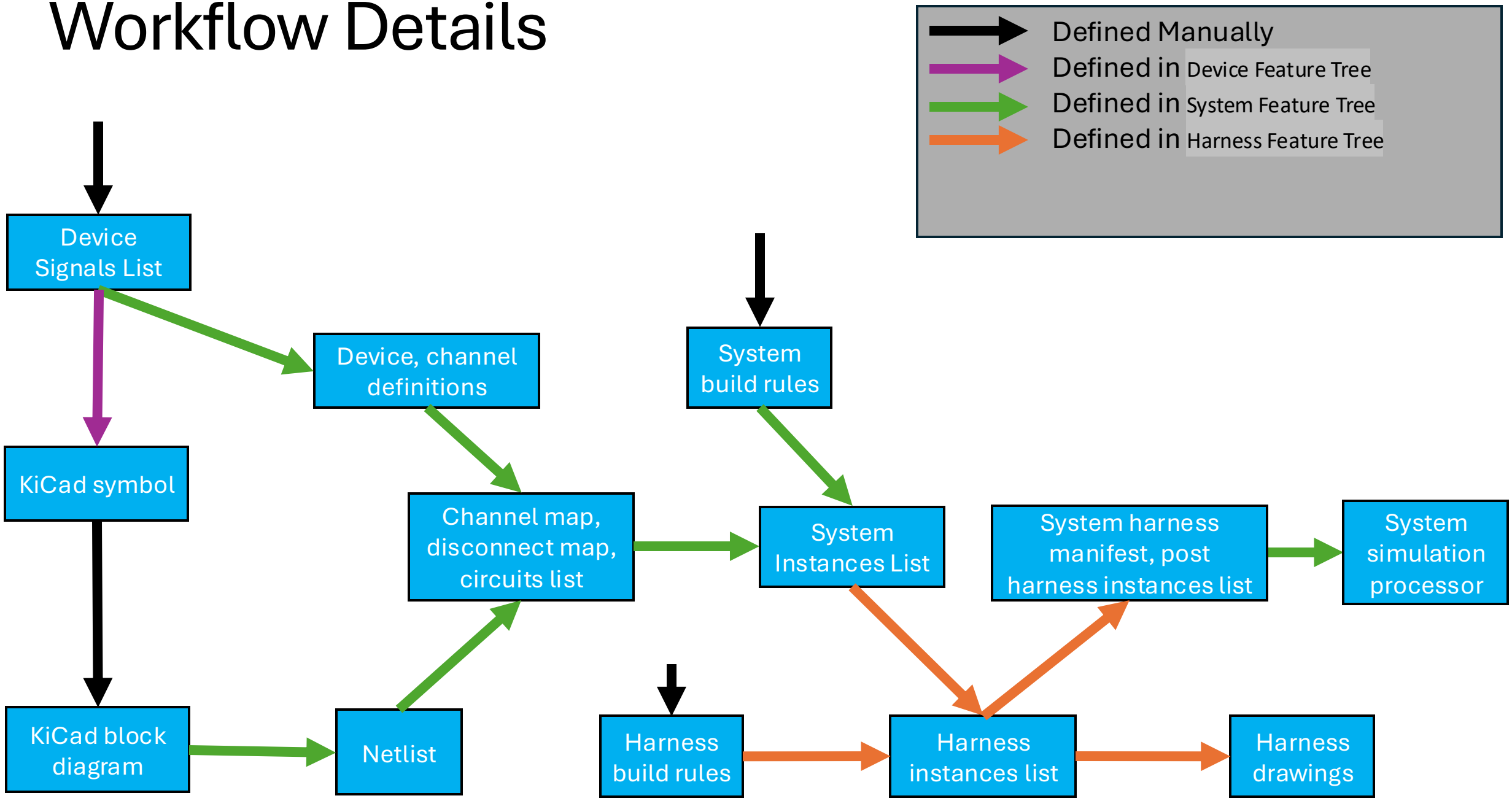


Workflow Overview

- When you “render” a product (run `harnice –r` from the command line or GUI button press), a long process is executed.
- During the process, existing information is queried, processed, and every output is regenerated.



Workflow Details



The Approach

Harnice is a python package that processes netlists and turns them into outputs.

- Specify your system definition using Kicad or Altium to make a block diagram
  - 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

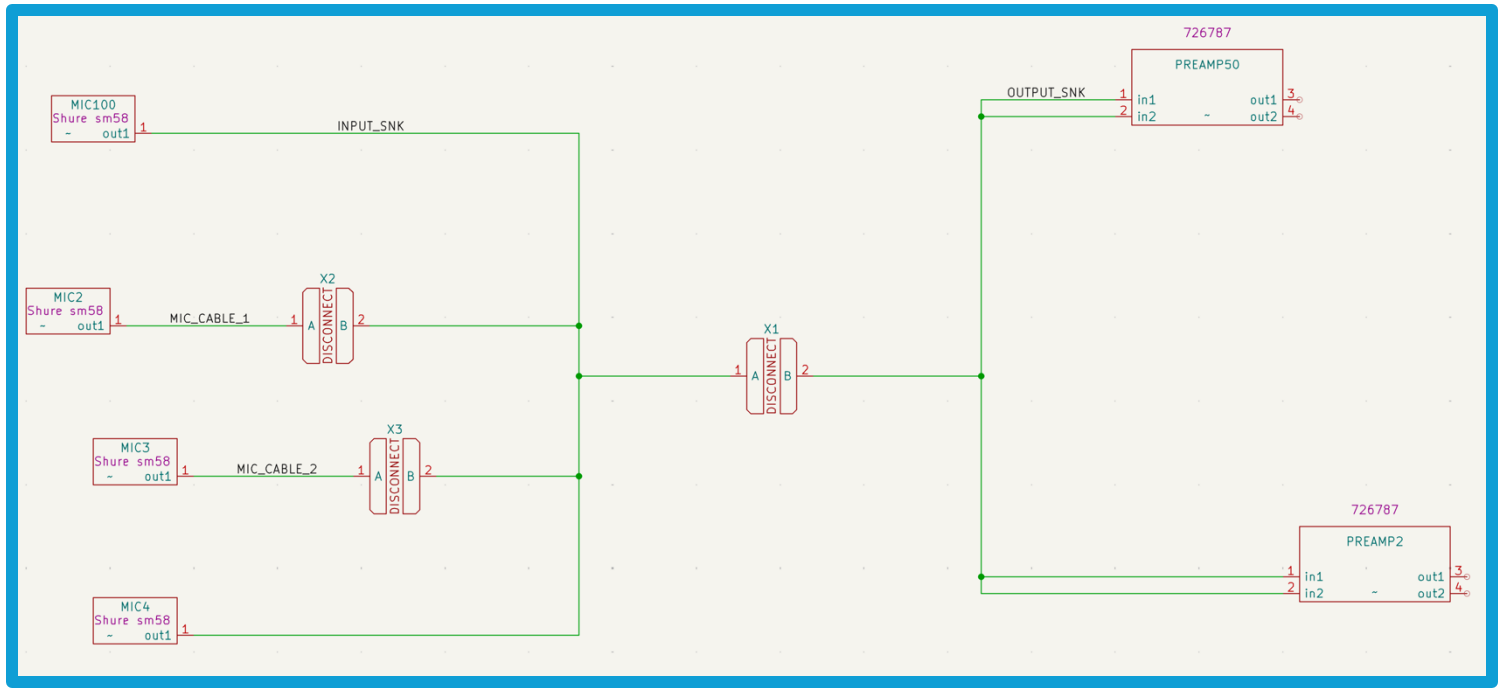
Defining your Devices

channel_id	signal	connector_name	cavity	connector_mpn	channel_type
in1	pos	in1	2	XLR3F	(1, 'https://github.com/kenyonshutt/harnice-library-public')
in1	neg	in1	3	XLR3F	(1, 'https://github.com/kenyonshutt/harnice-library-public')
in1-shield	chassis	in1	1	XLR3F	(5, 'https://github.com/kenyonshutt/harnice-library-public')
in2	pos	in2	2	XLR3F	(1, 'https://github.com/kenyonshutt/harnice-library-public')
in2	neg	in2	3	XLR3F	(1, 'https://github.com/kenyonshutt/harnice-library-public')
in2-shield	chassis	in2	1	XLR3F	(5, 'https://github.com/kenyonshutt/harnice-library-public')
out1	pos	out1	2	XLR3M	(4, 'https://github.com/kenyonshutt/harnice-library-public')
out1	neg	out1	3	XLR3M	(4, 'https://github.com/kenyonshutt/harnice-library-public')
out1-shield	chassis	out1	1	XLR3M	(5, 'https://github.com/kenyonshutt/harnice-library-public')
out2	pos	out2	2	XLR3M	(4, 'https://github.com/kenyonshutt/harnice-library-public')
out2	neg	out2	3	XLR3M	(4, 'https://github.com/kenyonshutt/harnice-library-public')
out2-shield	chassis	out2	1	XLR3M	(5, 'https://github.com/kenyonshutt/harnice-library-public')

- Primary data structure of a device is a TSV called “signals_list”.
- Generated from a python script that you define
- Produces a KiCad symbol that you can import into your project for a block diagram.
- *Future work: will contain a .kicad_esch file that will allow you to define device electrical behavior.*

```python
ch_type_ids = {
    "in": (1, "https://github.com/kenyonshutt/harnice-library-public"),
    "out": (4, "https://github.com/kenyonshutt/harnice-library-public"),
    "chassis": (5, "https://github.com/kenyonshutt/harnice-library-public"),
}

xlr_pinout = {"pos": 2, "neg": 3, "chassis": 1}

connector_mpns = {"XLR3F": ["in1", "in2"], "XLR3M": ["out1", "out2"]}


def mpn_for_connector(connector_name):
    for mpn, conn_list in connector_mpns.items():
        if connector_name in conn_list:
            return mpn
    return None


signals_list.new()

for connector_name in ["in1", "in2", "out1", "out2"]:
    if connector_name.startswith("in"):
        channel_type = ch_type_ids["in"]
    elif connector_name.startswith("out"):
        channel_type = ch_type_ids["out"]
    else:
        continue

    channel_name = connector_name
    connector_mpn = mpn_for_connector(connector_name)

    for signal in chtype.signals(channel_type):
        signals_list.append(
            channel_id=channel_name,
            signal=signal,
            connector_name=connector_name,
            cavity=xlr_pinout.get(signal),
            channel_type=channel_type,
            connector_mpn=connector_mpn,
        )

    # Add shield row
    signals_list.append(
        channel_id=f"{channel_name}-shield",
        signal="chassis",
        connector_name=connector_name,
        cavity=xlr_pinout.get("chassis"),
        channel_type=ch_type_ids["chassis"],
        connector_mpn=connector_mpn,
    )
```

Defining your Block Diagram



- Device symbols can be added to your KiCad schematic.
- KiCad wires can be drawn that represent entire harnesses.
- KiCad is agnostic to the individual conductors, channels, or signals of a harness, just that there are certain connectors that are connected to each other.

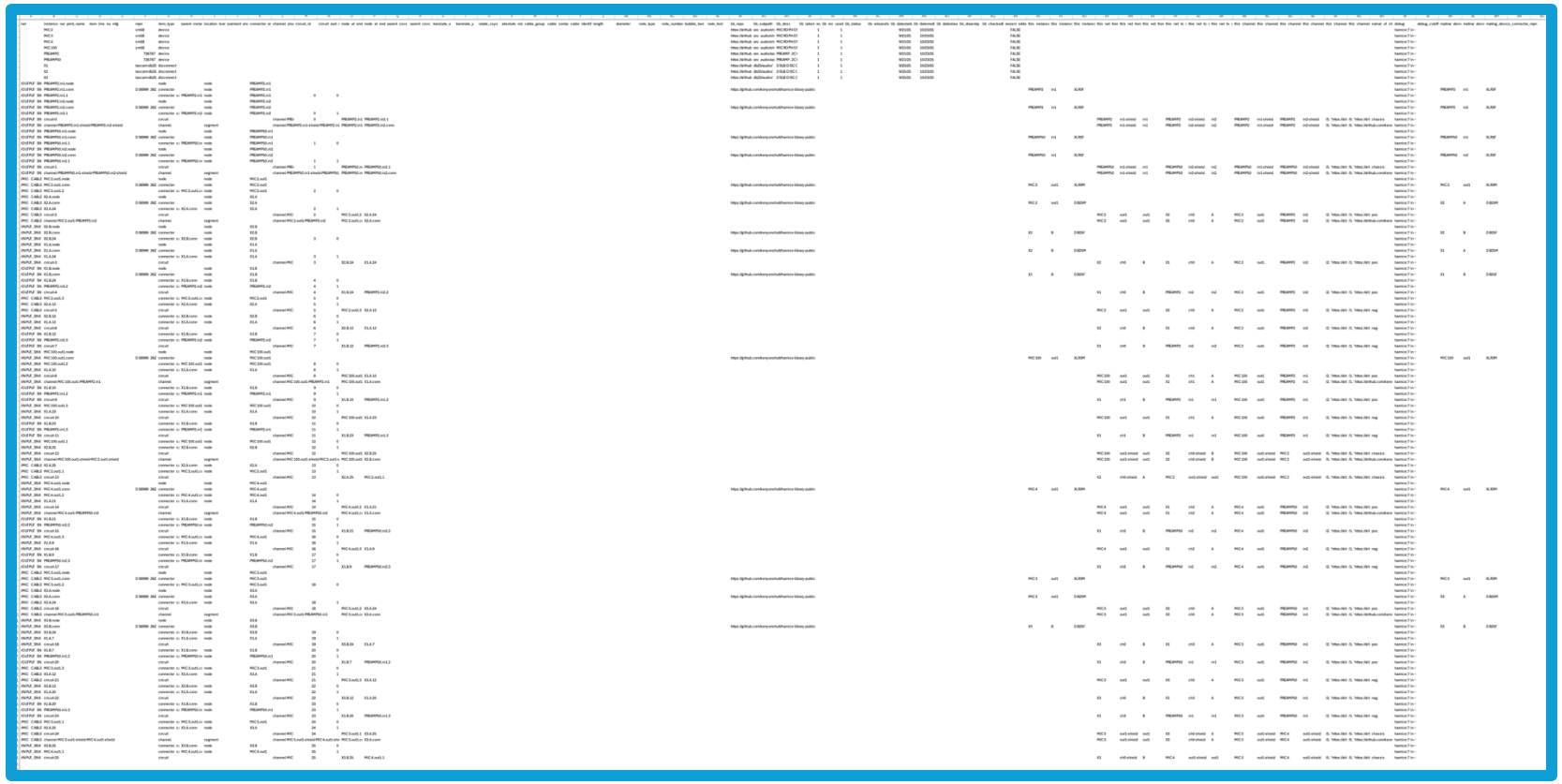
Rendering your System

```
#=========================================================================
#                   KICAD PROCESSING
#=========================================================================
feature_tree_utils.run_macro("kicad_sch_to_pdf", "system_artifacts", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="blockdiagram-1")
feature_tree_utils.run_macro("kicad_pro_to_bom", "system_builder", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="bom-1")

#=========================================================================
#                   COLLECT AND PULL DEVICES FROM LIBRARY
#=========================================================================
system_utils.make_instances_from_bom()

#=========================================================================
#                   CHANNEL MAPPING
#=========================================================================
feature_tree_utils.run_macro("kicad_pro_to_system_connector_list", "system_builder", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="system-connector-list-1")
manifest.new()
channel_map.new()

#add manual channel map commands here. key=(from_device_refdes, from_device_channel_id)
#channel_map.map(("MIC3", "out1"), ("PREAMP1", "in2"))

#map channels to other compatible channels by sorting alphabetically then mapping compatibles
feature_tree_utils.run_macro("basic_channel_mapper", "system_builder", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="channel-mapper-1")

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

#map channels that must pass through disconnects to available channels inside disconnects
disconnect_map.new()

#add manual disconnect map commands here
#disconnect_map.already_assigned_disconnects_set_append(('X1', 'ch0'))

#map channels passing through disconnects to available channels inside disconnects
feature_tree_utils.run_macro("disconnect_mapper", "system_builder", "https://github.com/kenyonshutt/harnice-library-public", artifact_id="disconnect-mapper-1")

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

#=========================================================================
#                   INSTANCES LIST
#=========================================================================
system_utils.make_instances_for_connectors_cavities_nodes_channels_circuits()

#assign mating connectors
for instance in fileio.read_tsv("instances list"):
    if instance.get("item_type") == "connector":
        if instance.get("this_instance_mating_device_connector_mpn") == "XLR3M":
            instances_list.modify(instance.get("instance_name"),{
                "mpn":"D38999_26ZA98PN",
                "lib_repo":"https://github.com/kenyonshutt/harnice-library-public"
```

- When a python file (feature_tree.py) is called, the KiCad netlist is exported, channels are mapped, and a system instances list is generated.

- Each system has its own feature tree, and it acts as the primary collector for your design rules.

System Instances List



- An instances list is the complete source of truth for a system.
- It’s a TSV database that records every possible item and its attributes that may belong to your system.
- This file is where every downstream output pulls data from.

Harness Feature Tree

```
#=========================================================================
#                  build_macro SCRIPTING
#=========================================================================
feature_tree_utils.run_macro(
    "import_harness_from_harnice_system",
    "harness_builder",
    "https://github.com/kenyonshutt/harnice-library-public",
    "harness-from-system-1",
    system_pn_rev=["0000250810-S01","rev7"],
    path_to_system_rev=os.path.join("/Users/kenyonshutt/The Dropbox/Kenyon Shutt/Projects/2024-11-26-harnice/dev-systems/0000250810-S01/", "0000250810-S01-rev7"),
    target_net="/OUTPUT_SNK",
    manifest_nets=["/OUTPUT_SNK"]
)

#=========================================================================
#                 HARNESS BUILD RULES
#=========================================================================

#=========================================================================
#                  IMPORT PARTS FROM LIBRARY
#=========================================================================
for instance in fileio.read_tsv("instances list"):
    if instance.get("item_type") in ["connector", "backshell"]:
        if instance.get("instance_name") not in ["X100"]:
            if instance.get("mpn") not in ["TXPA20"]:
                library_utils.pull(instance)

#=========================================================================
#                  LOCATE PARTS PER COORDINATE SYSTEMS
#=========================================================================
for instance in fileio.read_tsv("instances list"):
    parent_csys = None
    parent_csys_outputcsys_name = None

    if instance.get("item_type") == "connector":
        parent_csys = instances_list.instance_in_connector_group_with_item_type(instance.get("connector_group"), "backshell")
        parent_csys_outputcsys_name = "connector"
        if parent_csys == 0:
            parent_csys = instances_list.instance_in_connector_group_with_item_type(instance.get("connector_group"), "node")
            parent_csys_outputcsys_name = "origin"

    elif instance.get("item_type") == "backshell":
        parent_csys = instances_list.instance_in_connector_group_with_item_type(instance.get("connector_group"), "node")
        parent_csys_outputcsys_name = "origin"
    else:
        continue

    instances_list.modify(instance.get("instance_name"), {
        "parent_csys_instance_name": parent_csys.get("instance_name"),
        "parent_csys_outputcsys_name": parent_csys_outputcsys_name
    })
```

- When a python file (`feature_tree.py`) is called, a System Instances List is queried, and every instance that's related to the associated KiCad net is brought in.
- Similar to systems, each harness has its own feature tree, and it acts as the primary collector for your design rules.
- It is also in charge of producing outputs, like build drawings, formboard drawings, or wirelists. More on those later.

DEFINITION OF TERMS

Harnice's strict requirement on flexibility

- Major departure from existing eCAD packages:
  - **YOU CAN PUT ANYTHING* ANYWHERE.**

- As long as a thing can be defined as an "instance", it can end up in your electrical system. Harnice doesn't care if you need string, nails, plumbing fittings, raccoons, cables, or connectors.

- You write your own instance definitions, your own item_types, your own vocabulary, your own relationships.

The following slides define the exceptions to the above. We couldn't get away with everything.

General Terminology

NEEDS CONTENT

Sorted alphabetically ->

channel_ map	circuits_list	disconnect _map	instances_list	library_ history	manifes		post_ harness_ instances_list	rev_ history	signals_ list
A list of channels on devices within merged_nets that are either mapped to other channels or are unmapped	A list of every individual electrical connection that must be present in your system or harness to satisfy your channel and disconnect maps	A list of every available channel on a every disc and chann may or not pas through it	A list of ev phy es a system or harness	g the most recent render of the current product	ce signator to part number(s), and may contain other information indexed to the reference designator	A file where users can manually define flagnotes	SHOULD WE CALL THIS CONNECTOR S LIST?	A list of every physical or notional thing, drawing element, or concept that includes instances added at the harness level, that represents a system	A record of every revision of a part, and its release status	A list of circuits that appear on connectors and are part of channels that belong to a device

Needs content

Types of Products

These are things you can “render” by calling `harnice -r` or `harnice --render` from within a partnumber/revision directory, but when done so, fundamentally different things happen.

Sorted alphabetically ->

cable	channel_type	device	disconnect	flagnote	harness	part	system	tblock
Something you can purchase by the spool or lot, contains conductor(s) that can be assigned to circuits inside a harness. Usually shows up in a bill of materials.	Uniquely identifiable set of signals that allow electrical intent to be documented and later referenced	Item you can buy that has signals, connectors, channels. Exhibits electrical behavior. The fundamental “block” element in a block diagram.	Set of two electrical connectors that has a predefined pinout, connector selection, and set of channels that can host circuits.	A bubble shape on a drawing that usually points to something via a leader arrow.	A set of circuits that satisfies a channel map. Also contains instructions about how to be built from a set of selected parts.	Something you can purchase per each or per unit. May have a 1:1 drawing that can be located on a formboard drawing. usually shows up in a bill of materials.	A collection of devices and harnesses that satisfies a set of functionality requirements for some external purpose.	A page SVG, usually with your name or company logo, that makes your drawings look professional.

Types of Lists

These are commonly used data structures in Harnice. They're almost all TSV's (tab-separated values). Functions related to these lists can be called by `list_name.command()`

Sorted alphabetically ->

channel_ map	circuits_list	disconnect _map	instances_list	library_ history	manifest	manual_ flagnotes_list	netlist	post_ harness_ instances_list	rev_ history	signals_ list
A list of channels on devices within merged_nets that are either mapped to other channels or are unmapped	A list of every individual electrical connection that must be present in your system or harness to satisfy your channel and disconnect maps	A list of every available channel on a every disconnect, and every channel that may or may not pass through it	A list of every physical or notional thing, drawing element, or concept that defines a system or harness	A report of what was imported during the most recent render of the current product	A table that relates reference designator to part number(s), and may contain other information indexed to the reference designator	A file where users can manually define flagnotes	SHOULD WE CALL THIS CONNECTOR S LIST?	A list of every physical or notional thing, drawing element, or concept that includes instances added at the harness level, that represents a system	A record of every revision of a part, and its release status	A list of circuits that appear on connectors and are part of channels that belong to a device

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)

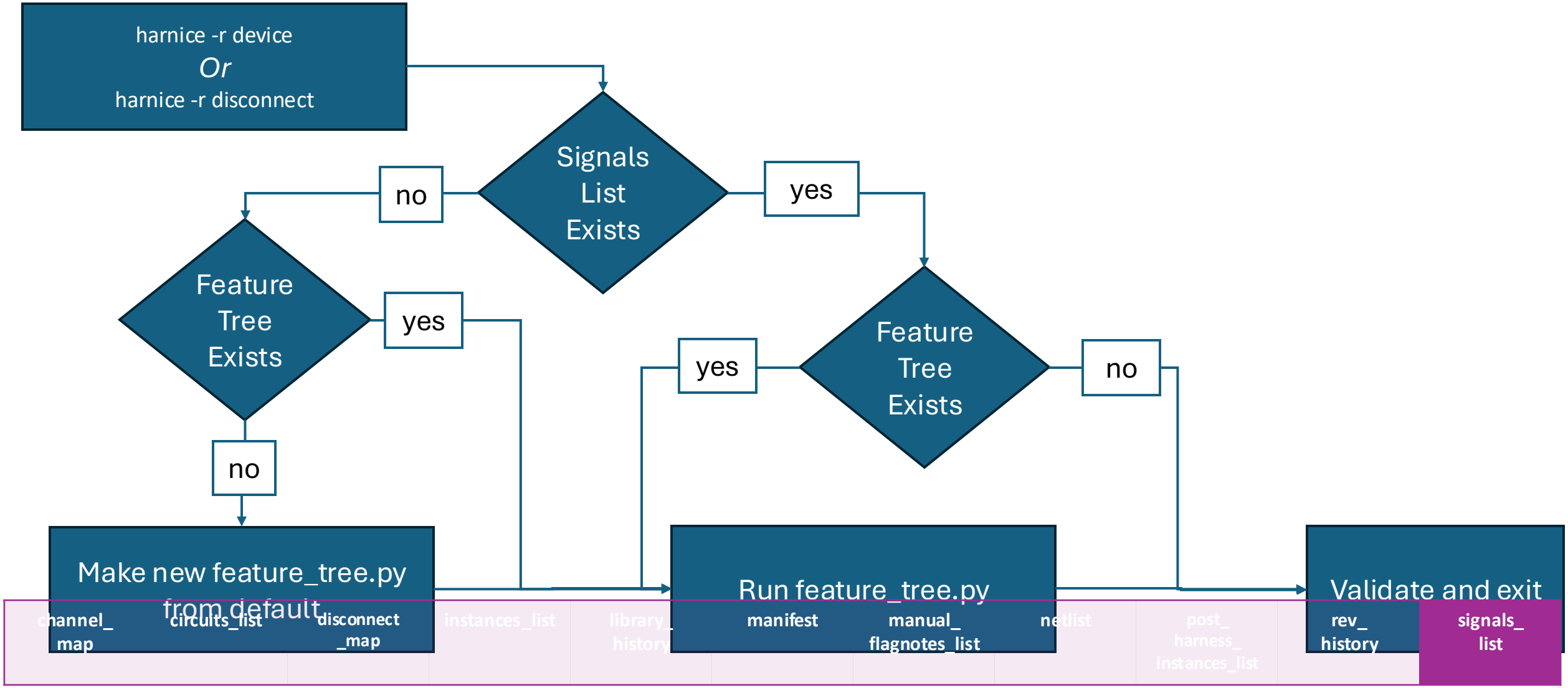
channel_id	signal	connector_name	cavity	connector_mpn	channel_type
out1	pos	out1	2	XLR3M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
out1	neg	out1	3	XLR3M	(2, 'https://github.com/kenyonshutt/harnice-library-public')
out1-shield	chassis	out1	1	XLR3M	(5, 'https://github.com/kenyonshutt/harnice-library-public')

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')

Signals List Feature Tree

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



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”

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

channel_ map	circuits_list	disconnect _map	instances_list	library_ history	manifest	manual_ flagnotes_list	netlist	post_ harness_ instances_list	rev_ history	signals_ list

Signals List Python Commands

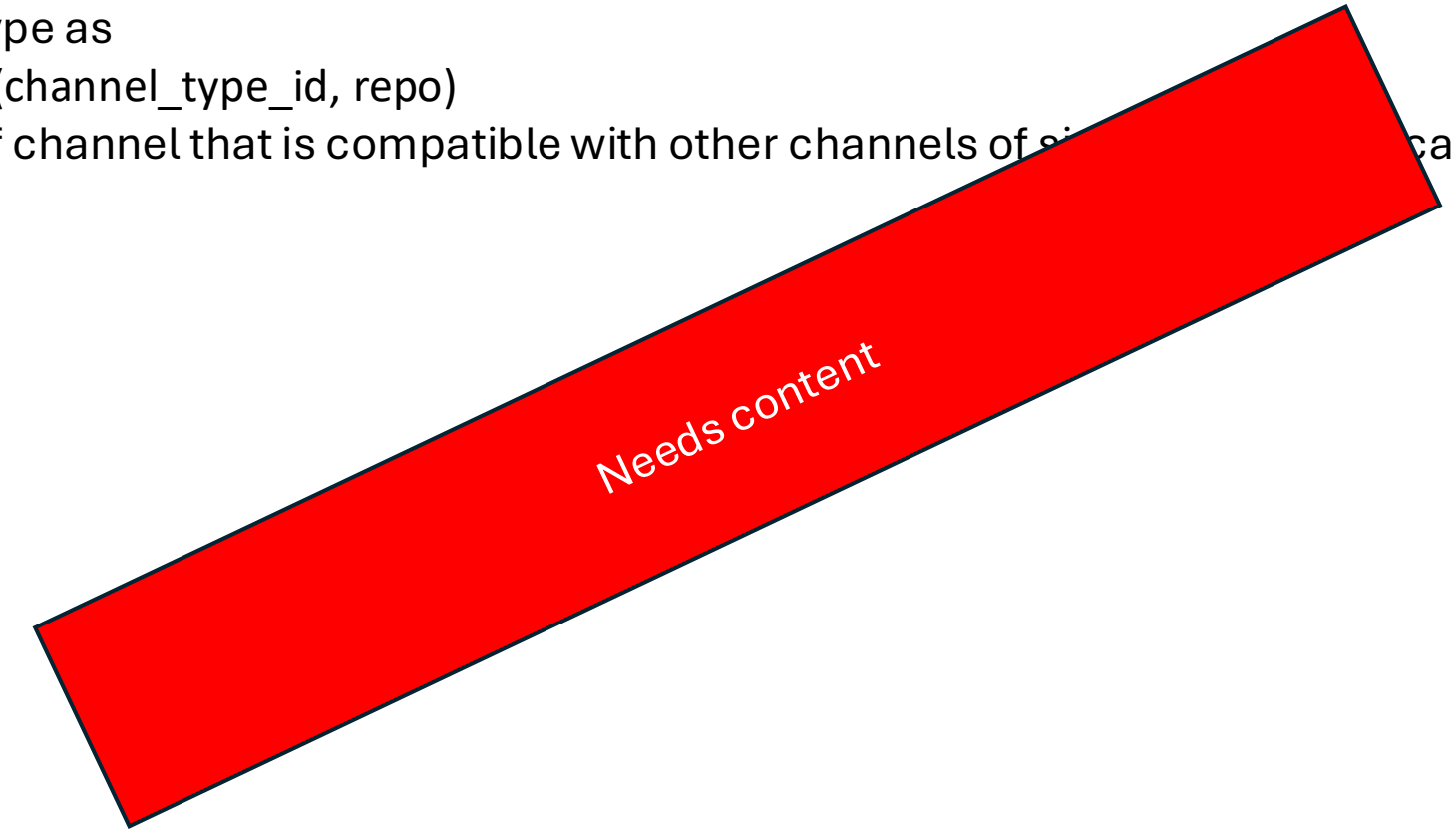
Command	Args	Returns	Purpose
signals_list.new_list()			Create a new list
signals_list.write_signal()	See earlier slides on what's required and why		Add a signal to a signals list
signals_list.signals_of_channel_type()	Channel type in touple form (channel_type_id, repo) *One arg, in parenthesis*	List 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

Channel Types

Refer to a channel type as

channel_type = (channel_type_id, repo)

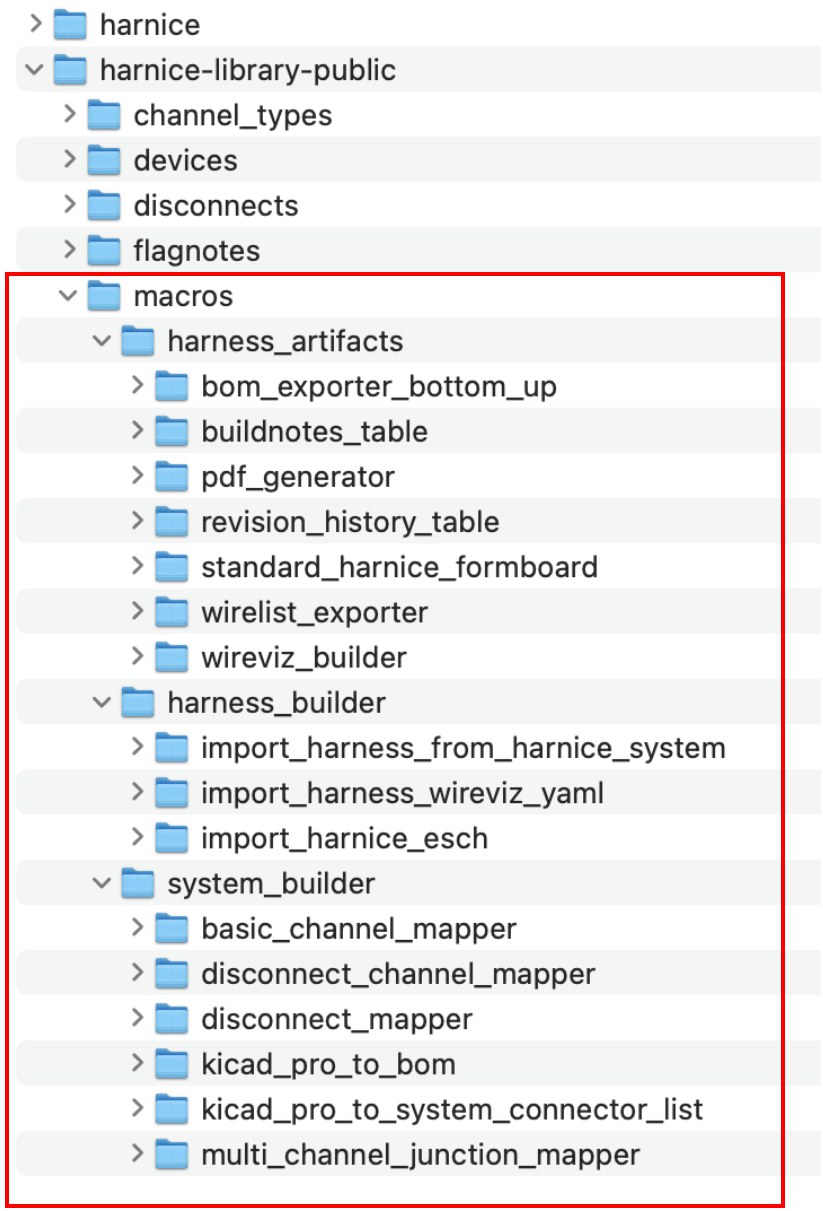
Represents a type of channel that is compatible with other channels of si can be mapped to each other.



MACROS

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)

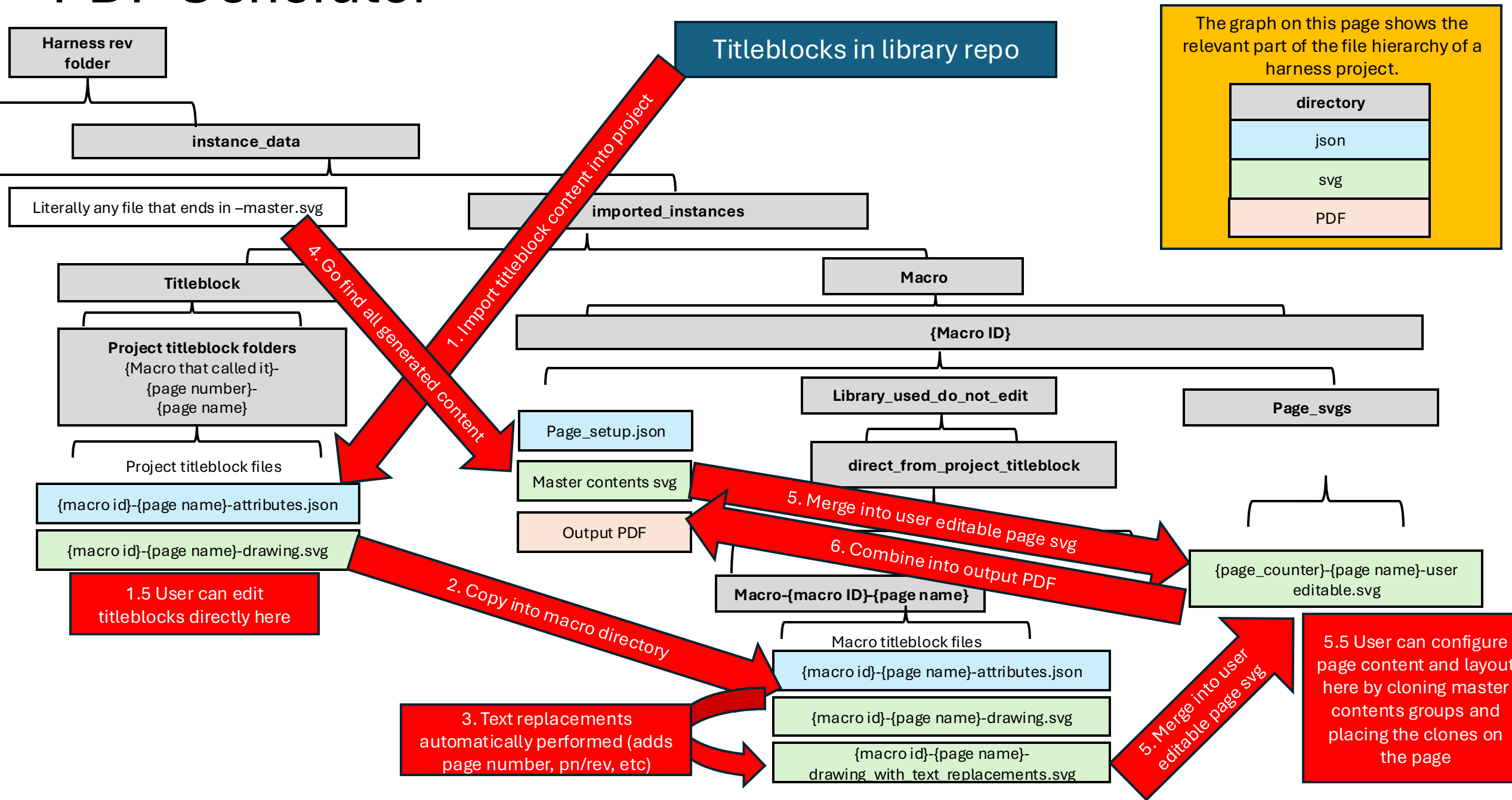
How to Generate Outputs

- 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()
```

PDF Generator

This is the notional flow of SVG content when you run the PDF generator macro.



FILE STRUCTURE

Centralized file structure

- fileio.py contains functions that keep your files organized and easy to reference when they're needed.
- Products and macros are all supposed to have a function file_structure() that represents the file structure of the contents of that product on top of the current directory (rev folder).
- Each value that contains a dict represents a directory, and each value that contains a string represents a file.
- Keys are used for files to be more human readable.
- You can add args as needed to describe more complicated structures.

```
def file_structure(item_type=None, instance_name=None):
    return {
        f"{fileio.partnumber('pn-rev')}-feature_tree.py": "feature tree",
        f"{fileio.partnumber('pn-rev')}-instances_list.tsv": "instances list",
        f"{fileio.partnumber('pn-rev')}-formboard_graph_definition.png": "formboard graph definition png",
        "instance_data": {
            "imported_instances": {
                item_type: {instance_name: {"library_used_do_not_edit": {}}}
            },
            "generated_instances_do_not_edit": {},
        },
        "interactive_files": {
            f"{fileio.partnumber('pn-rev')}.formboard_graph_definition.tsv": "formboard graph definition",
            f"{fileio.partnumber('pn-rev')}.flagnotes.tsv": "flagnotes manual",
        },
    }
```

How to query file structures

- When a product is rendered, that product's file structure is automatically loaded into fileio.

- You can reference files in that product by calling `fileio.path("file key")` from your script.
- You can reference directories in that product by calling `fileio.dirpath("dirname")` from your script.

- If you want to reference files that might not be defined in your product's file structure (you're running a bespoke macro with some weird content), you can temporarily pass a filepath dictionary into the function: `fileio.path("file key", structure_dict={})`

- The same fileio functions can be called to regard different results depending on what structure_dict your current product, project, macro, etc has defined.

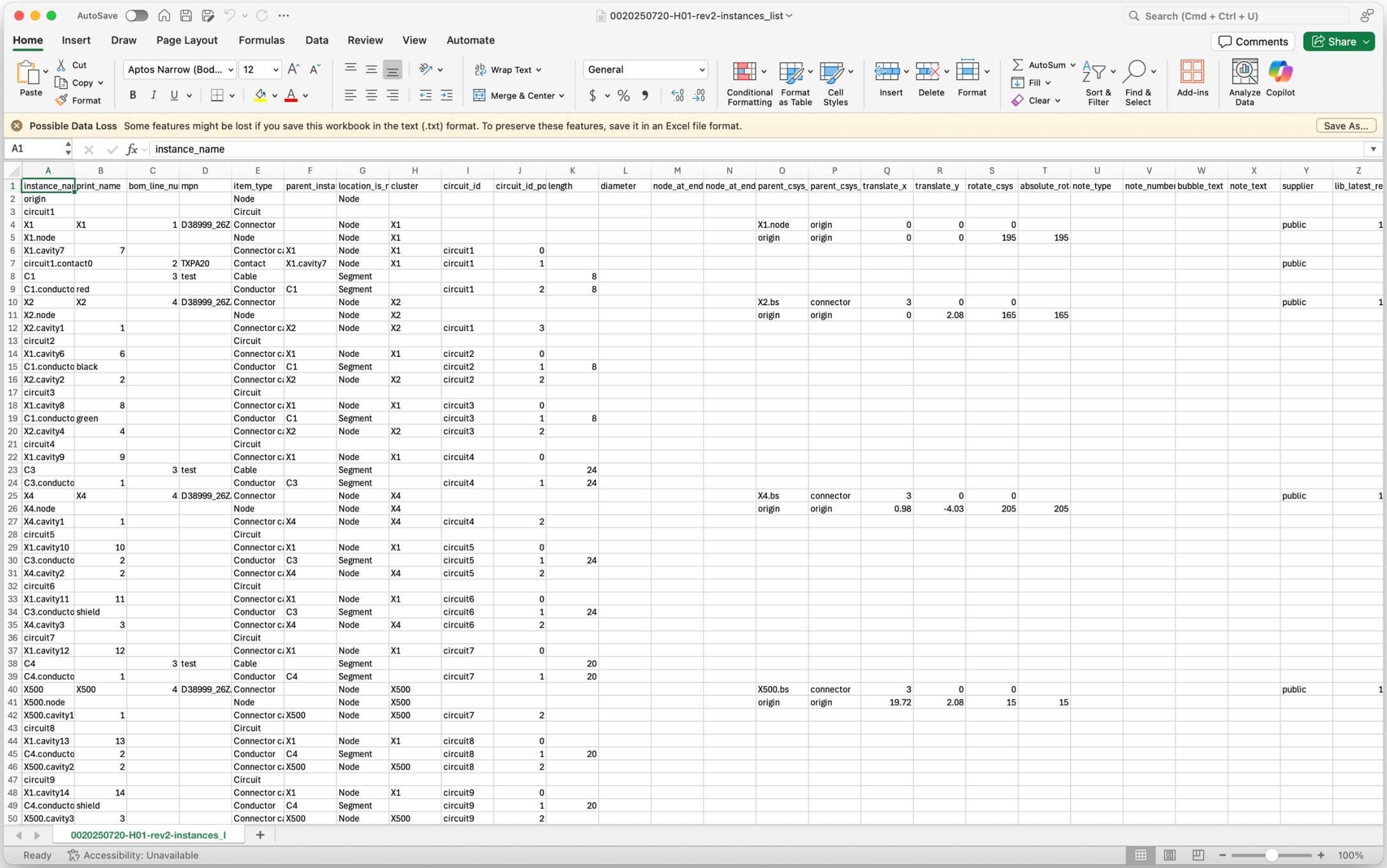
Fileio python commands

Command	Args	Returns	Purpose
fileio.set_file_structure()	File structure dict		Set the default fileio.path() and fileio.dirpath() lookups. Call at the beginning of each product render
fileio.file_structure()		File structure dict	You shouldn't need this, use path() or dirpath() instead
fileio.part_directory()		Directory of the part you're working in	Anything
fileio.rev_directory()		Directory of the rev of the part you're working in	Anything
fileio.partnumber()	Wanted format *# given a part number "pppppp-revR"* *# "pn-rev" returns "pppppp-revR"* *# "pn" returns "pppppp"* *# "rev" returns "revR"* *# "R" returns "R"*	Part number, rev strings of the current product you're working on in various formats	Anything
fileio.silentremove()	Filepath		Remove a file or a directory and all its contents if it exists
fileio.path()	Filename key Optional: alternative file structure dict	Path to the file referenced by key	Any time a file needs to be referenced directly
fileio.dirpath()	Name of directory Optional: alternative file structure dict	Path to the folder referenced by key	Any time a directory needs to be referenced directly
fileio.verify_revision_structure()			Ensures you're working in a directory recognized as a "rev" directory (must be inside a "part" directory)
fileio.today()		today's date in a common format	
fileio.get_path_to_project()	traceable_key	Looks up traceable key in a local git-ignored lookup table for the local path to a project	Referencing projects that's not the current project (ex. pulling harness data out of a system)
fileio.read_tsv()	path to file, optional delimeter (default \t for tab-separated-values)	data of a csv with delimeter as a list of dictionaries	Standardize the way we're reading TSV's. TSV's are preferred to CSV's because they let you use commas in your data which is useful
fileio.newrev()			Works with the CLI to make a new revision file in the part directory. Updates the names of all the content from old rev number to new rev number

Data structure: “instances_list.tsv”

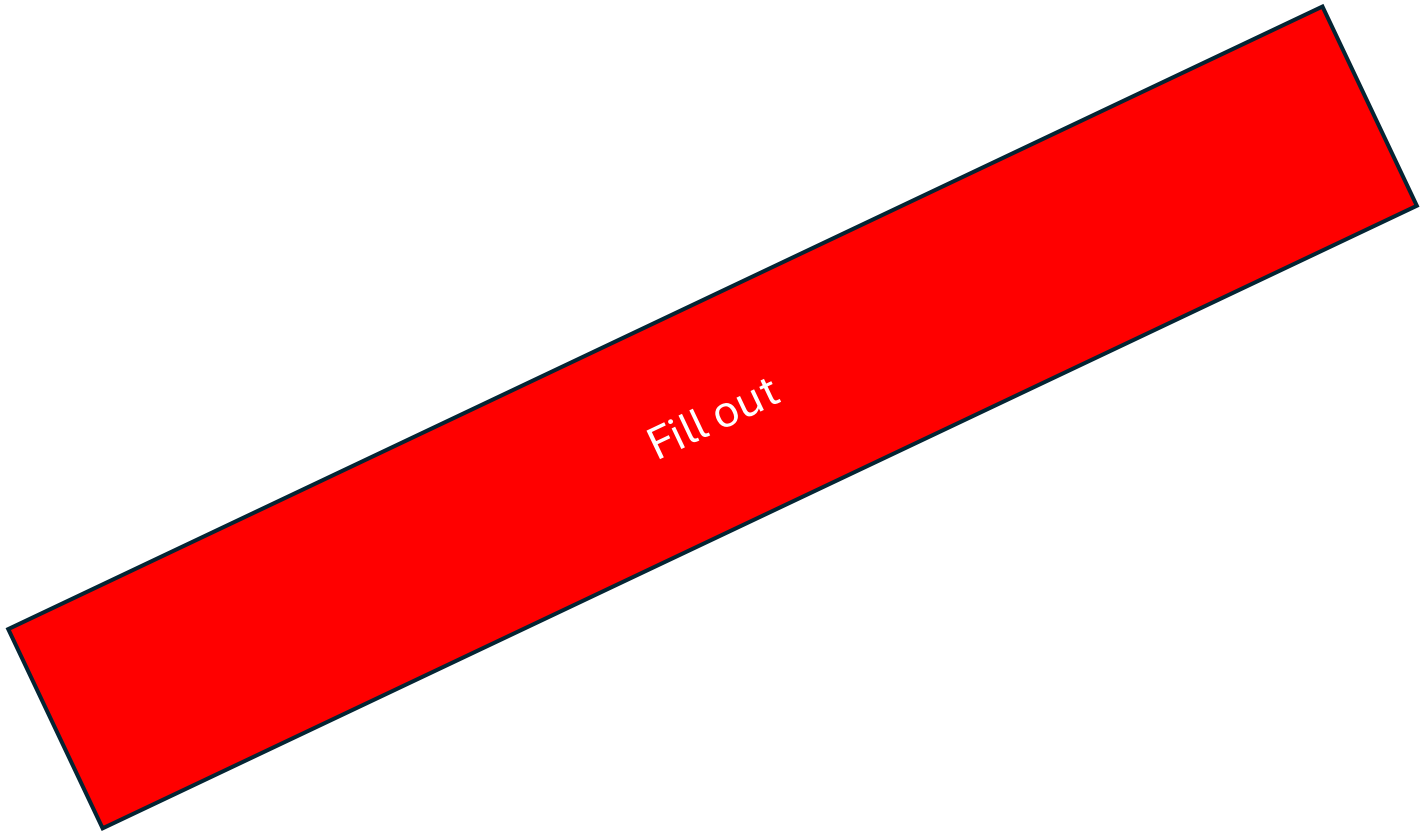
- A list of every single item, idea, note, part, instruction, circuit, literally anything that comprehensively describes how to build that harness or system
- TSV (tab-separated-values, big spreadsheet)
- Declined alternatives: STEP files, schematics, dictionaries not general, descriptive, or human readable 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'
]
```



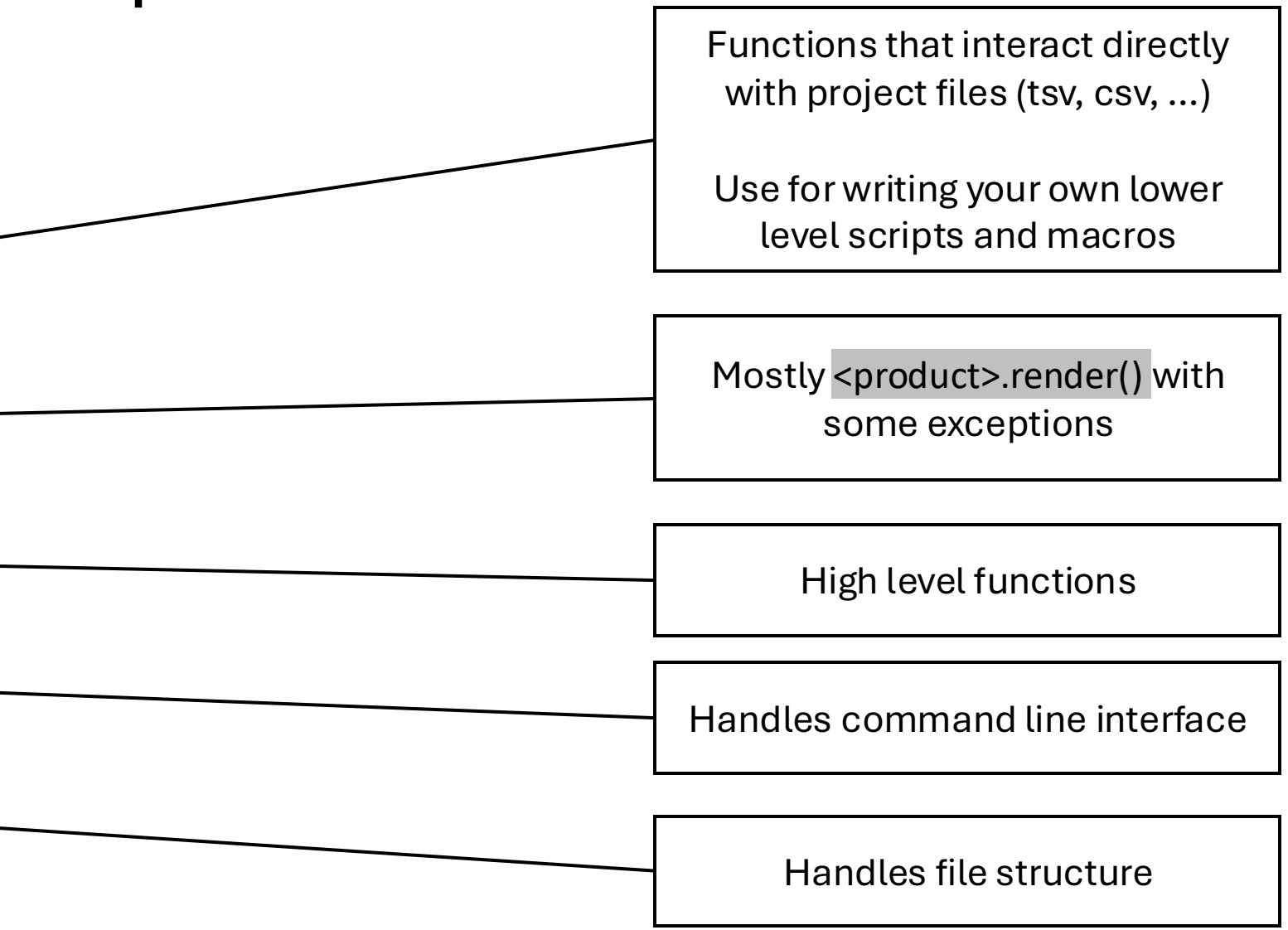
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	instance_na	print_name	bom_line_nu	mpn	item_type	parent_insta	location_is_r	cluster	circuit_id	circuit_id_po	length	diameter	node_at_end	node_at_end	parent_csys_	parent_csys_	translate_x	translate_y	rotate_csys	absolute_rot	note_type	note_numbe	bubble_text	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 c	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 c	X2	Node	X2	circuit1	3																
13	circuit2				Circuit																					
14	X1.cavity6	6			Connector c	X1	Node	X1	circuit2	0																
15	C1.conducto	black			Conductor	C1	Segment		circuit2	1	8															
16	X2.cavity2	2			Connector c	X2	Node	X2	circuit2	2																
17	circuit3				Circuit																					
18	X1.cavity8	8			Connector c	X1	Node	X1	circuit3	0																
19	C1.conducto	green			Conductor	C1	Segment		circuit3	1	8															
20	X2.cavity4	4			Connector c	X2	Node	X2	circuit3	2																
21	circuit4				Circuit																					
22	X1.cavity9	9			Connector c	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 c	X4	Node	X4	circuit4	2																
28	circuit5				Circuit																					
29	X1.cavity10	10			Connector c	X1	Node	X1	circuit5	0																
30	C3.conducto	2			Conductor	C3	Segment		circuit5	1	24															
31	X4.cavity2	2			Connector c	X4	Node	X4	circuit5	2																
32	circuit6				Circuit																					
33	X1.cavity11	11			Connector c	X1	Node	X1	circuit6	0																
34	C3.conducto	shield			Conductor	C3	Segment		circuit6	1	24															
35	X4.cavity3	3			Connector c	X4	Node	X4	circuit6	2																
36	circuit7				Circuit																					
37	X1.cavity12	12			Connector c	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 c	X500	Node	X500	circuit7	2																
43	circuit8				Circuit																					
44	X1.cavity13	13			Connector c	X1	Node	X1	circuit8	0																
45	C4.conducto	2			Conductor	C4	Segment		circuit8	1	20															
46	X500.cavity2	2			Connector c	X500	Node	X500	circuit8	2																
47	circuit9				Circuit																					
48	X1.cavity14	14			Connector c	X1	Node	X1	circuit9	0																
49	C4.conducto	shield			Conductor	C4	Segment		circuit9	1	20															
50	X500.cavity3	3			Connector c	X500	Node	X500	circuit9	2																

Local Paths vs Traceable URLs



Harnice Code Repo Structure

- Harnice
  - Lists
  - Products
  - Utils
  - cli.py
  - fileio.py



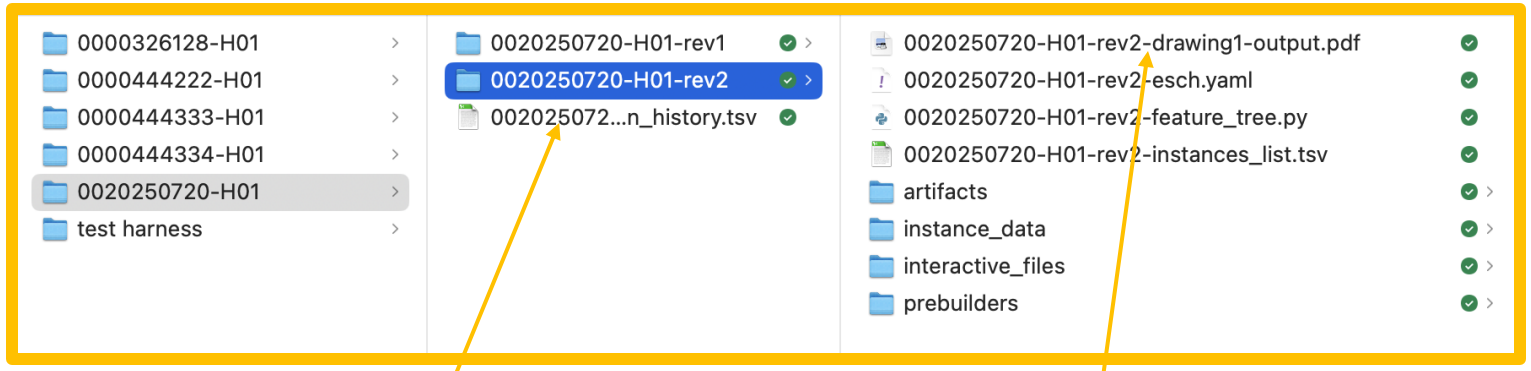
Libraries

- 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!

REVISION CONTROL

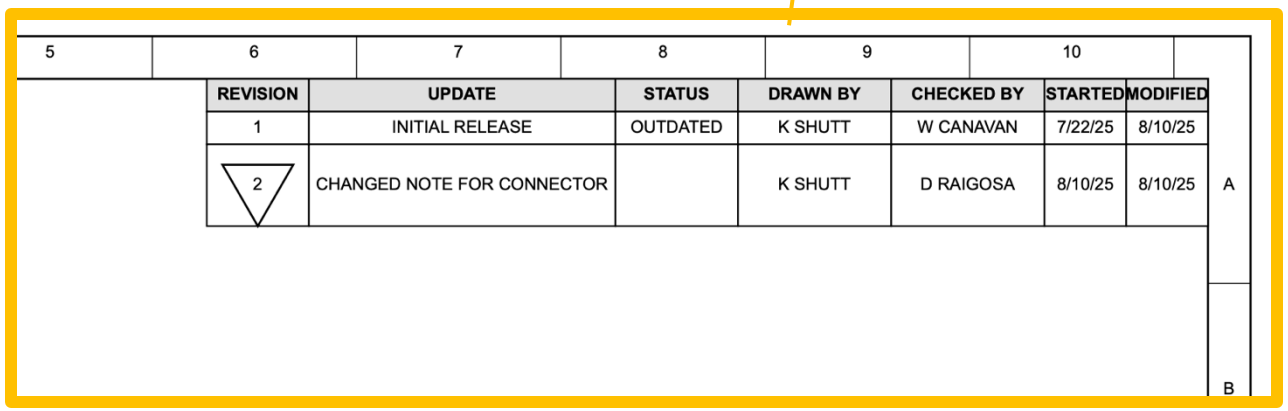
Product version control

- 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

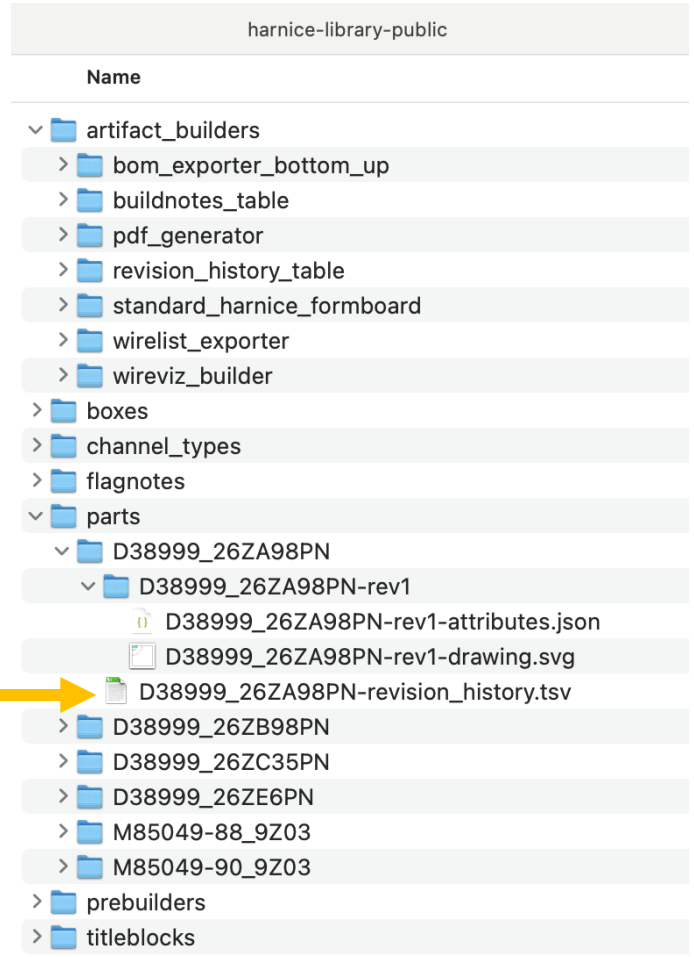
```
Mac:0020250720-H01 kenyonshutt$ harnice -r harness
Thanks for using Harnice!
This is a part folder (0020250720-H01).
Please `cd` into one of its revision subfolders (e.g. `0020250720-H01-rev1`) and rerun.
Mac:0020250720-H01 kenyonshutt$ cd 0020250720-H01-rev1
Mac:0020250720-H01-rev1 kenyonshutt$ 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$
```



LIBRARIES

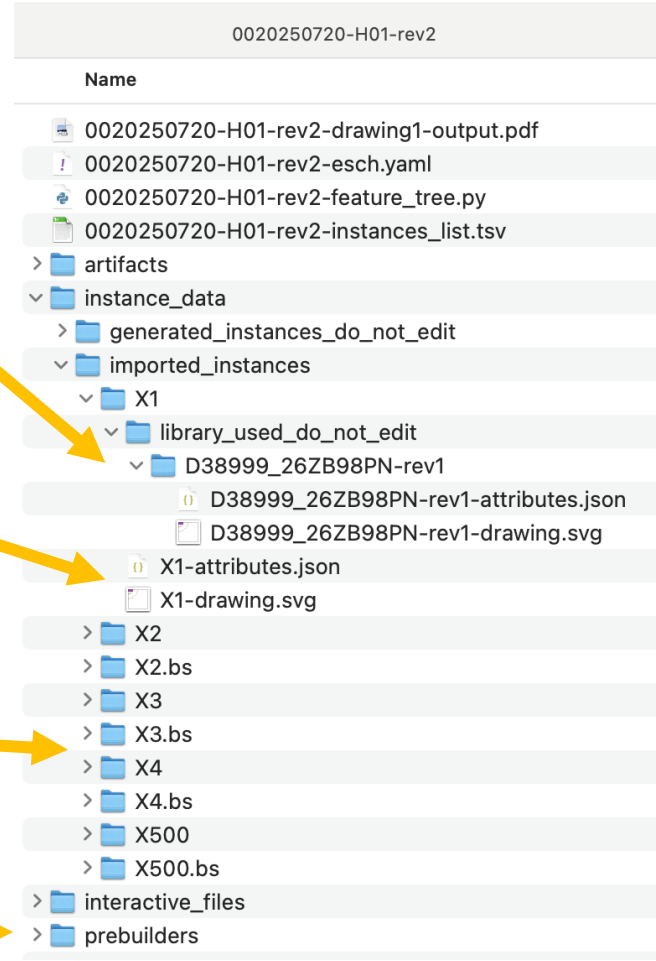
Library flexibility

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)

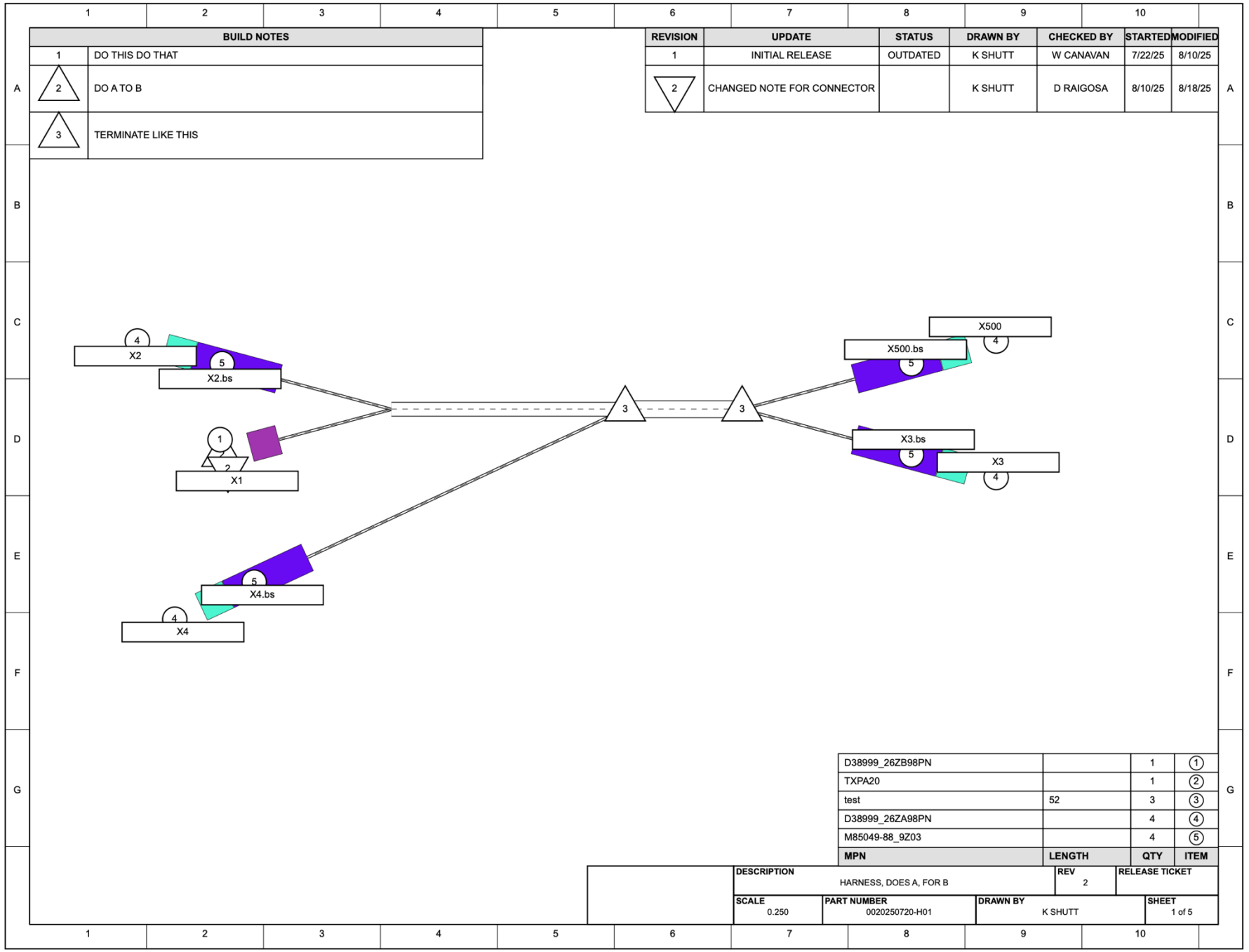
Different instances, even with the same MPN, are imported separately

All imported items work the same way

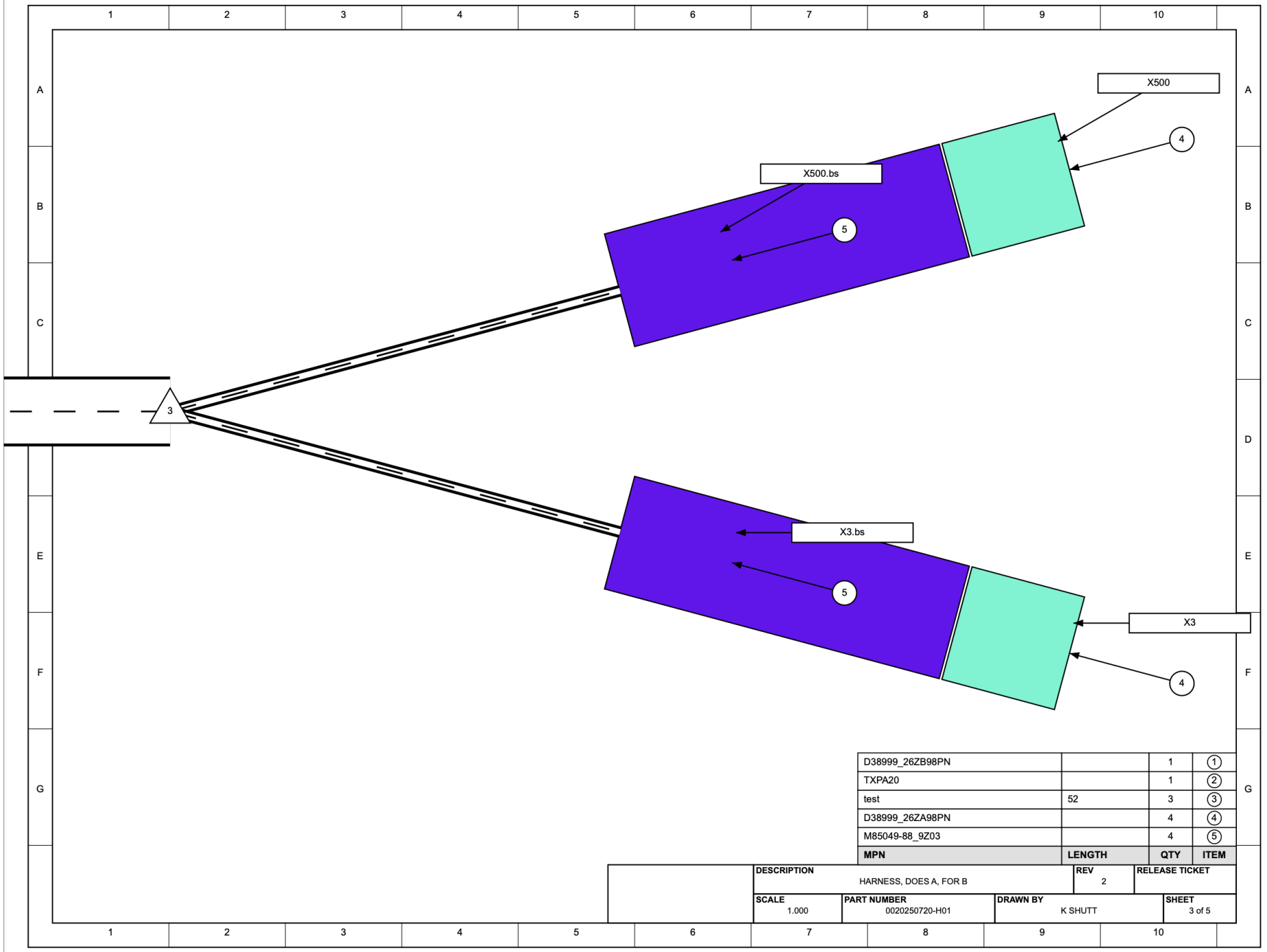
EXAMPLES

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.



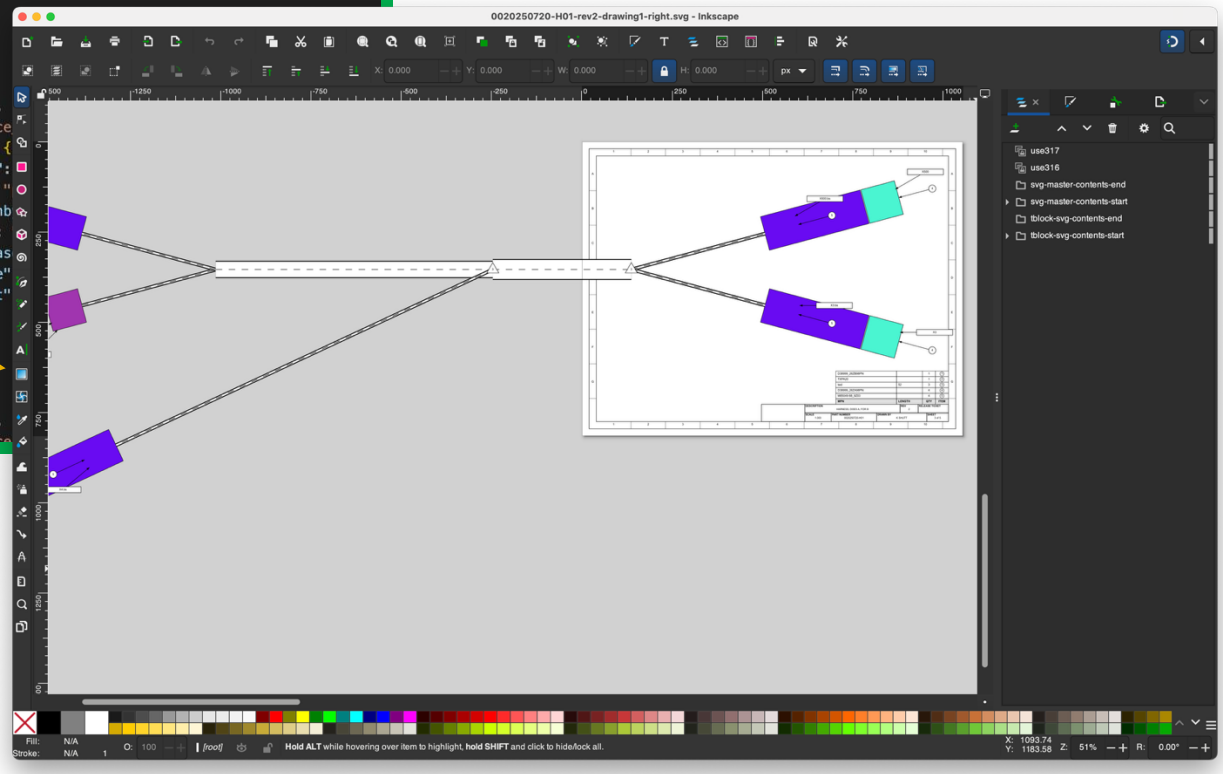
- Pages can be edited with your favorite SVG editor...

0020250720-H01-rev1

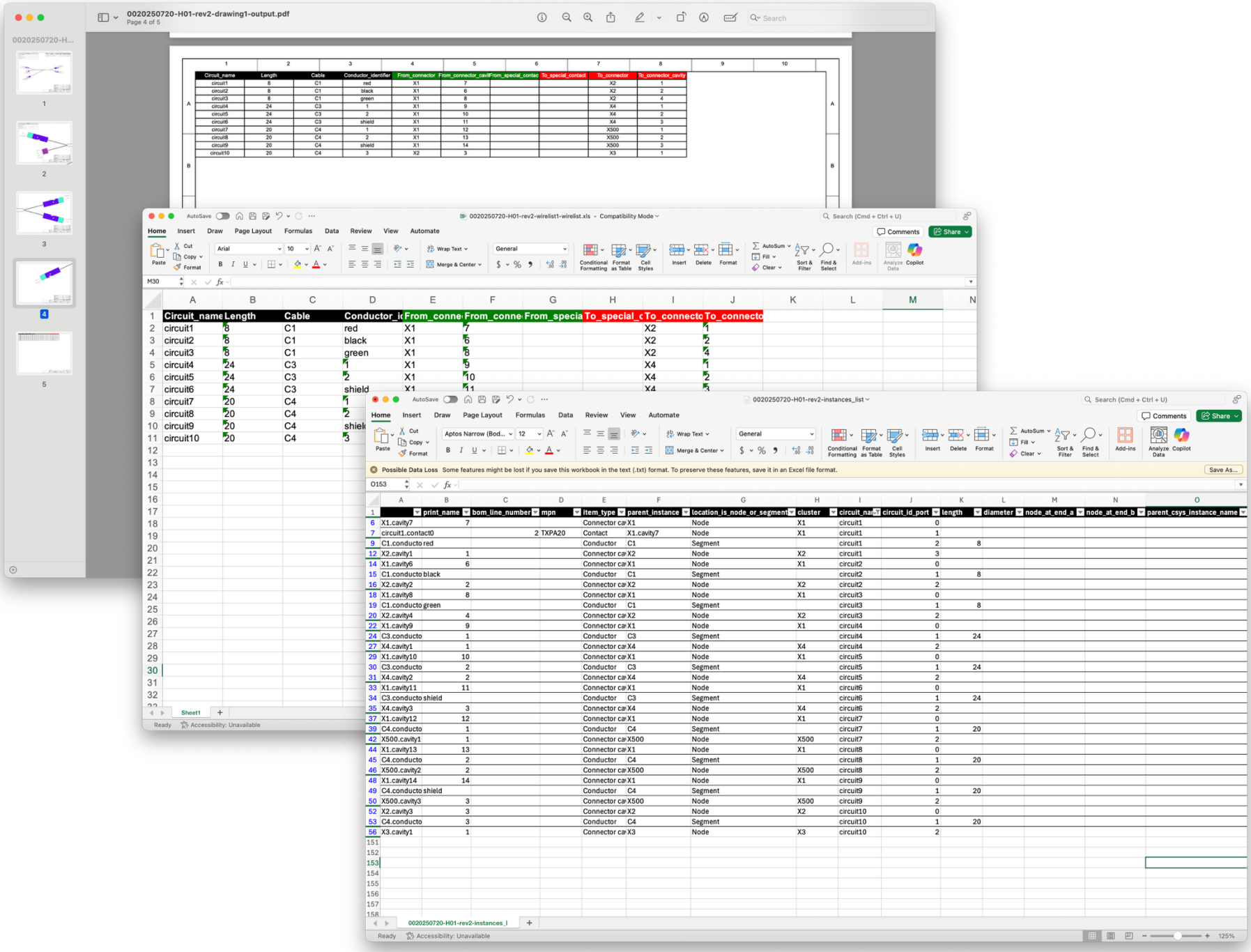
Name

- 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

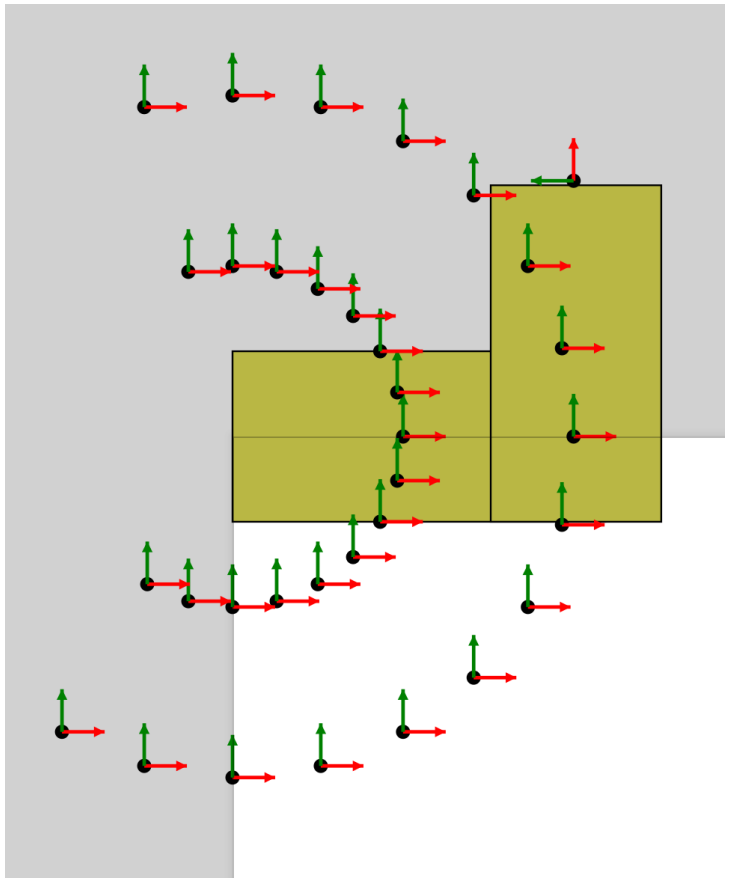

```
{
    "pages": [
        {
            "name": "overall",
            "supplier": "public",
            "titleblock": "harnice_tblock-11x8.5",
            "text_replacements": {
                "tblock-key-desc": "pull_from_revision_history(desc)",
                "tblock-key-pn": "pull_from_revision_history(pn)",
                "tblock-key-drawnby": "pull_from_revision_history(drawnby)",
                "tblock-key-rev": "pull_from_revision_history(rev)",
                "tblock-key-releaseticket": "",
                "tblock-key-scale": "B",
                "tblock-key-sheet": "autosheet"
            },
            "show_items": []
        },
        {
            "name": "left",
            "supplier": "public",
            "titleblock": "harnice_tblock-11x8.5",
            "text_replacements": {
                "tblock-key-desc": "pull_from_revision_history(desc)",
                "tblock-key-pn": "pull_from_revision_history(pn)",
                "tblock-key-drawnby": "pull_from_revision_history(drawnby)",
                "tblock-key-rev": "pull_from_revision_history(rev)",
                "tblock-key-releaseticket": "",
                "tblock-key-scale": "A",
                "tblock-key-sheet": "autosheet"
            },
            "show_items": []
        },
        {
            "name": "right",
            "supplier": "public",
            "titleblock": "harnice
            "text_replacements": {
                "tblock-key-desc":
                "tblock-key-pn": "
                "tblock-key-drawnb
                "tblock-key-rev":
                "tblock-key-releas
                "tblock-key-scale"
                "tblock-key-sheet"
            },
            "show_items": []
        },
        {
            "name": "wirelist",
            "supplier": "public",
```

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



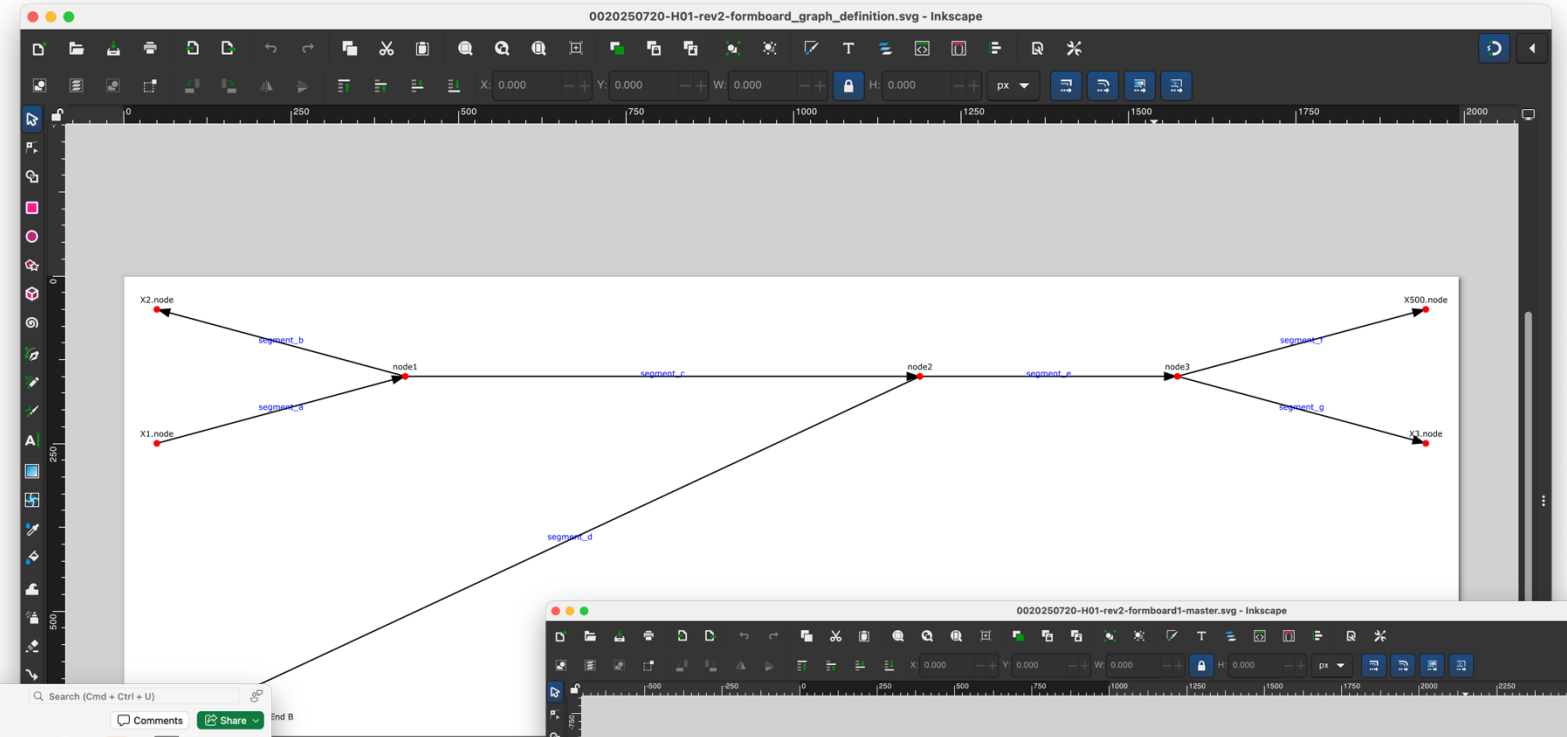
- Every part can have as many child coordinate systems as you need
- These can be referenced in the instances list
- Used for things like flagnotes, child parts, or other related items
- Can be used to place parts on a formboard drawing, calculate distances, etc

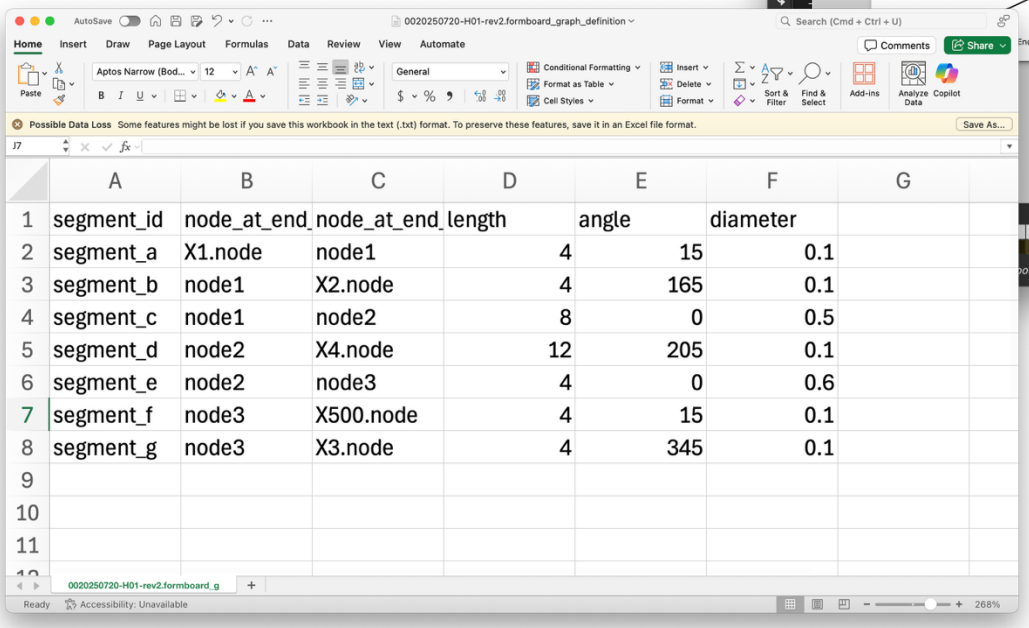



```
{
    "csys_parent_prefs": [
        ".node"
    ],
    "tooling_info": {
        "tools": {}
    },
    "build_notes": {},
    "csys_children": {
        "connector": {
            "x": 2,
            "y": 1.5,
            "angle": 0,
            "rotation": 90
        },
        "flagnote-1": {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