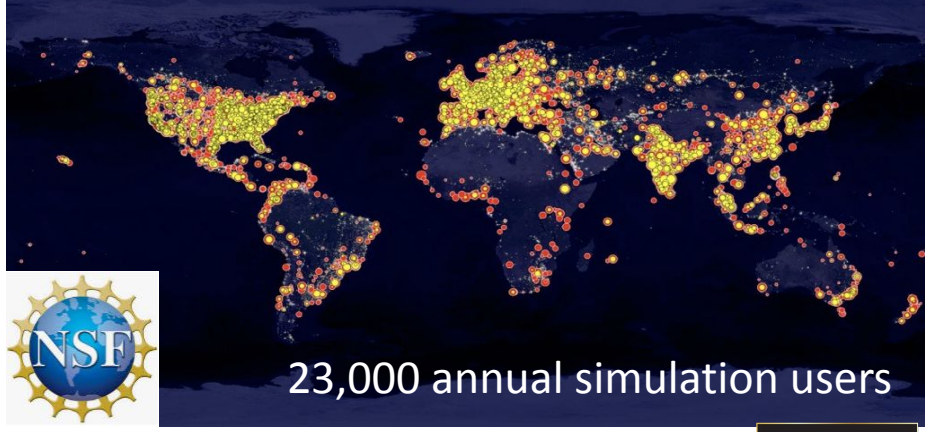


nanoHUB:
getting started guide to tool developers

Develop and publish tools in nanoHUB
Make your research reproducible and your workflows and data FAIR

Tanya Faltens, Daniel Mejia, Steven Clark, Juan Carlos Verduzco & Ale Strachan*
* strachan@purdue.edu
School of Materials Engineering &
Purdue University
West Lafayette, Indiana USA

Overview

1. Why publish tools & apps in nanoHUB?
   - Tools are publications (DOIs and indexed by Web of Science)
   - Share your work with your community (22,000+ annual sim users)
2. Various tool and app types
   - Apps, workflows, Jupyter notebooks, commercial codes, X11 GUIs
3. Sim2Ls, FAIR workflows and data
   - Develop and publish Sim2Ls
4. Developing Apps
   - Connecting Sim2Ls to Jupyter and Web Apps
5. Tool Publication process
   - Register, deploy, test, and publish
6. Development environment
   - A Unix development environment (Jupyter or Linux desktop)
7. Simulation and data as a service
   - Launching tools and querying the ResultsDB

Overview

- Connecting Sim2Ls to Jupyter
  - Jupyter Widgets
  - sim2lbuilder
  - Custom widgets

- Connecting Sim2Ls to Web Apps
  - nanoHUB end points / REST
  - nanohub-uidl to create Apps

Overview

- Connecting Sim2Ls to Jupyter
  - Jupyter Widgets
  - sim2lbuilder
  - Custom widgets

Jupyter Widgets

Jupyter widgets tutorial - https://github.com/jupyter-widgets/tutorial

- “A Python widget is an object that represents a control on the front end, like a slider. A single control can be displayed multiple times - they all represent the same python object”

```
import ipywidgets as widgets
```

- There are many widgets distributed with ipywidgets (core widgets)
(https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html)

```
widgets.Dropdown(
    options=['1', '2', '3'],
    value='2',
    description='Number:',
    disabled=False,
)
```

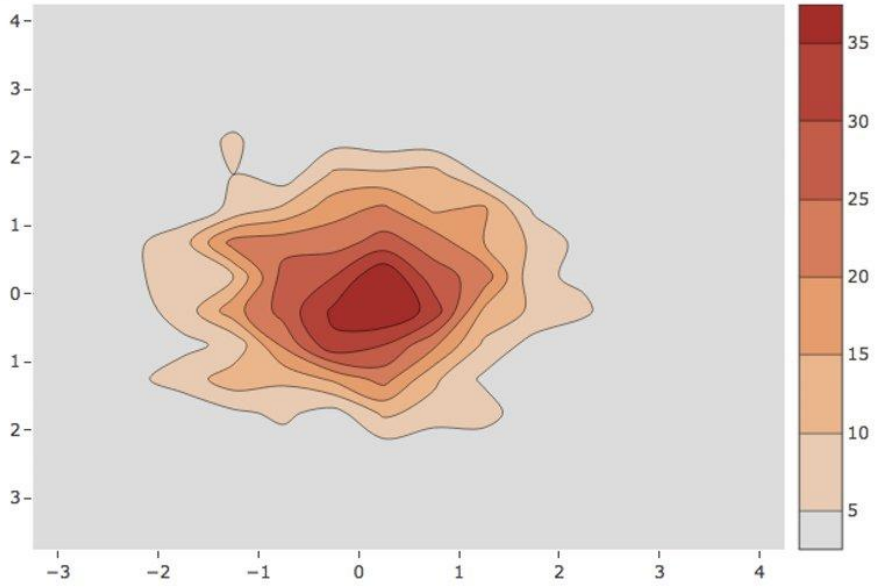
Jupyter Widgets - Community

https://jupyter.org/widgets

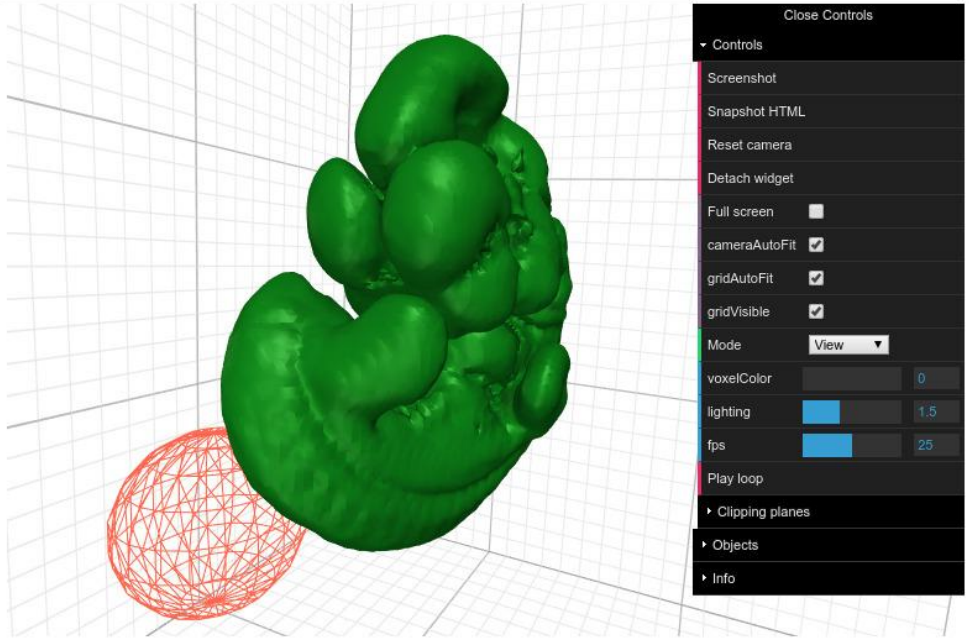
- Jupyter community has contribute multiple notebook extensions to help with data visualization, and GUI development.

```
import plotly.graph_objs as go
```

```
x = np.random.randn(1000)
y = np.random.randn(1000)

go.FigureWidget(
    data=[
        {'x': x, 'y': y, 'type': 'histogram2dcontour'}
    ]
)
```



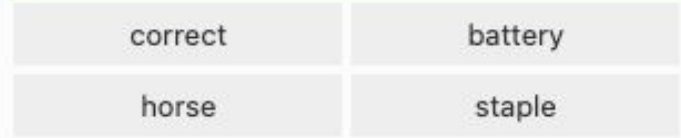

```
from vtkplotter import *

embedWindow(backend='k3d')

actor = load(datadir+'embryo.tif', threshold=40)
actor.color('green')
s = Sphere(r=2000, res=10).color('tomato').pos(8000,0,0).wireframe()

show(actor, s)
```



Jupyter Widgets - Layout

- Widgets can be grouped on different types of containers and layouts

```
from ipywidgets import Button, HBox, VBox

words = ['correct', 'horse', 'battery', 'staple']
items = [Button(description=w) for w in words]
left_box = VBox([items[0], items[1]])
right_box = VBox([items[2], items[3]])
HBox([left_box, right_box])
```




```
tab_contents = ['P0', 'P1', 'P2', 'P3', 'P4']
children = [widgets.Text(description=name) for name in tab_contents]
tab = widgets.Tab()
tab.children = children
tab.titles = [str(i) for i in range(len(children))]
tab
```

https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20Layout.html

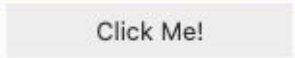
Jupyter Widgets - Events

- Changes on a widget (python object state) can be connected to python functions.
  - New values are passed as parameter to the function

```
int_range = widgets.IntSlider()
output2 = widgets.Output()

display(int_range, output2)

def on_value_change(change):
    with output2:
        print(change['new'])

int_range.observe(on_value_change, names='value')
```

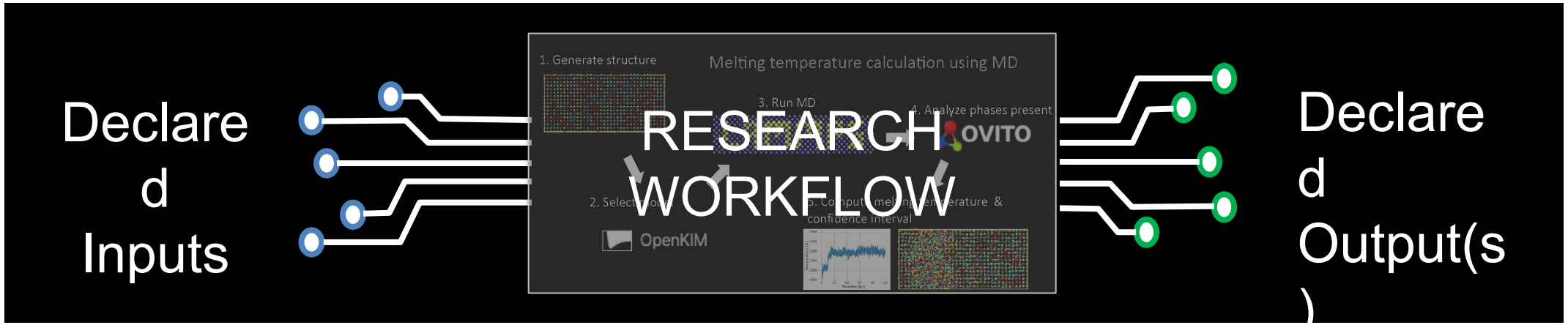
- Special widget “button” has an onclick event.

```
from IPython.display import display
button = widgets.Button(description="Click Me!")
output = widgets.Output()

display(button, output)

def on_button_clicked(b):
    with output:
        print("Button clicked.")

button.on_click(on_button_clicked)
```



What are Sim2Ls?

- Full end-to-end computational workflow
  - Input(s) → workflow → output(s)
  - Including all pre-processing and post-processing steps

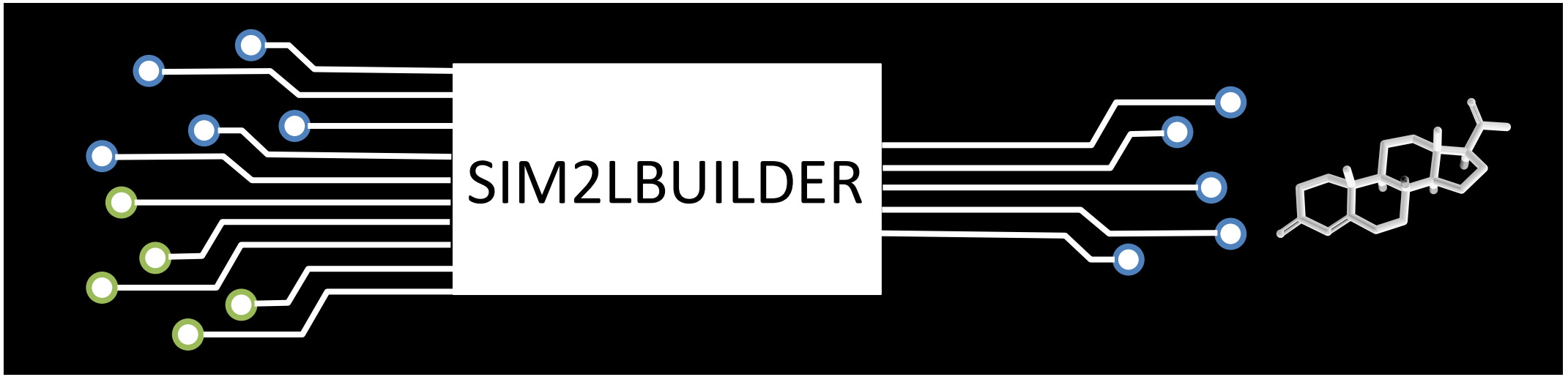


- Simulation as a service. Launch Sim2Ls from:
  - From a GUI or App
  - From AI/ML or high-throughput workflows
  - From inside nanoHUB or outside

Hunt M, Clark S, Mejia D, Desai S, Strachan A (2022) Sim2Ls: FAIR simulation workflows and data. PLoS ONE 17(3): e0264492. https://doi.org/10.1371/journal.pone.0264492

What is Sim2Lbuilder?

- GUI builder
  - Input(s) + output(s) → Jupyter widgets



- Creates default Layout for a Sim2L
  - Only need the simtool name

Sim2Lbuilder - Basics

- Loading libraries
  - searchForSimTool - to locate the simtool
  - getSimToolInputs - to extract inputs given a simtool path

```
1 from sim2lbuilder import WidgetConstructor, GetSimtoolDefaultSchema
2 from simtool import searchForSimTool, getSimToolInputs, Run
```

- Loading default schema
  - e.g. “meltingkim” sim2l

```
1 schema = GetSimtoolDefaultSchema("meltingkim")
2 schema
```

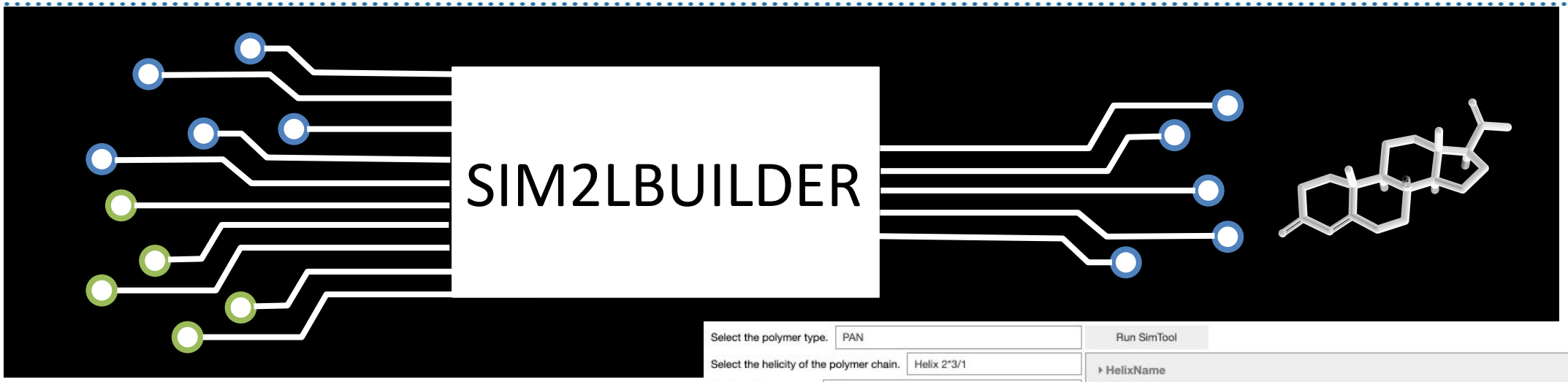
```
{'inputs': {'material': {'type': 'Text',
   'description': 'Element to be simulated',
   'value': 'Ni'},
  'crystal_structure': {'type': 'Text',
   'description': 'Crystal structure for initial conditions',
   'value': 'fcc'},
  'lattice_parameter': {'type': 'BoundedFloatText',
   'description': 'Lattice parameter for initial conditions',
   'units': <Unit('angstrom')>,
   'min': 2.0,
   'max': 10.0,
   'value': 3.5203},
  'Tsolid': {'type': 'BoundedFloatText',
   'description': 'Initial temperature assigned to the solid region',
   'units': <Unit('kelvin')>,
   'min': 1,
   'max': 5000,
   'value': 800},
  'Tliquid': {'type': 'BoundedFloatText',
```

Sim2Lbuilder – Basic App

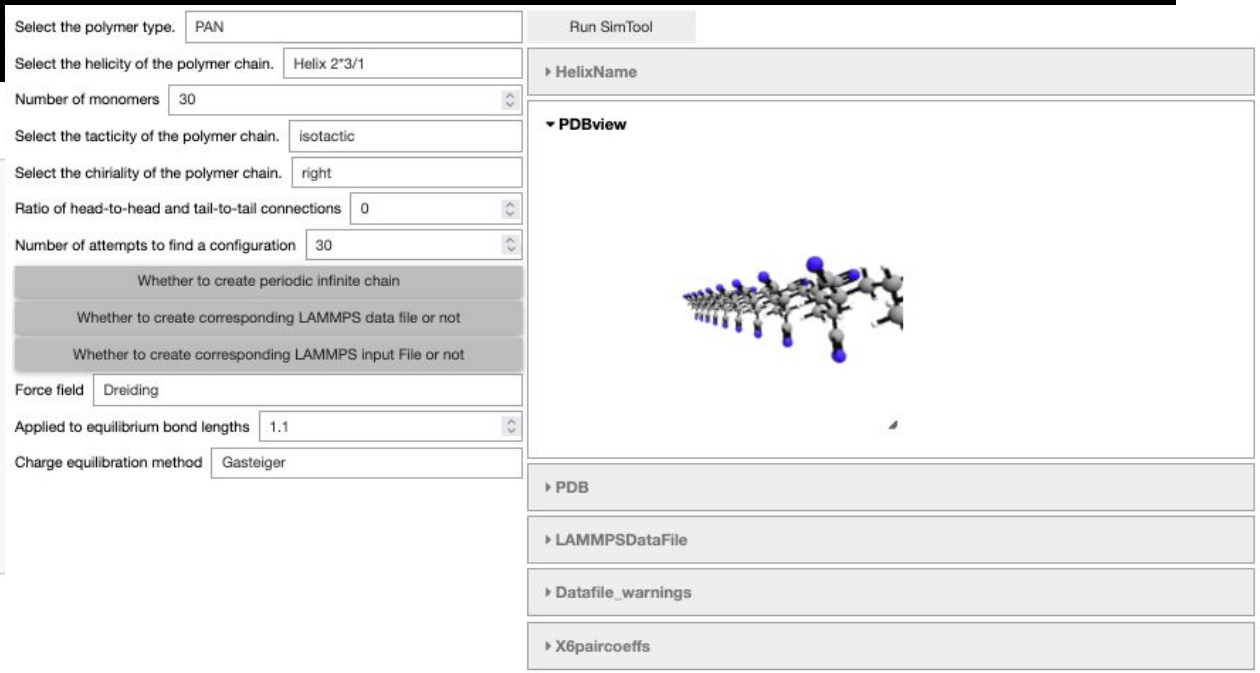
- Create Run Function and assemble the App
  - Create a python function, widget with inputs and outputs as parameter
  - Connect the function to RunSimTool method
  - Assemble the widget
  - Display the widget

```
1  def RunSimTool(widget, *kargs):
2      import simtool
3      stl = simtool.searchForSimTool("mdsandbox")
4      inputs = simtool.getSimToolInputs(stl)
5      for i,w in widget.inputs.items():
6          inputs[i].value = w.value
7      r = simtool.Run(stl, inputs)
8      for outk, out in widget.outputs.items():
9          with out:
10             print(r.read(outk))
11 s = WidgetConstructor(schema)
12 s.RunSimTool = RunSimTool
13 s.assemble()
14 s
```

What is Sim2Lbuilder?




```
from sim2lbuilder import WidgetConstructor, GetSimtoolDefaultSchema
from simtool import searchForSimTool, getSimToolInputs, Run
SIMTOOL_NAME = "meltingkim"
schema = GetSimtoolDefaultSchema(SIMTOOL_NAME)
def RunSimTool(widget, tool=SIMTOOL_NAME):
    stl =searchForSimTool(tool)
    inputs =getSimToolInputs(stl)
    for i,w in widget.inputs.items():
        inputs[i].value = w.value
    r =Run(stl, inputs)
    for outk, out in widget.outputs.items():
        with out:
            print(r.read(outk))
s = WidgetConstructor(schema)
s.RunSimTool = RunSimTool
s.display()

```



Sim2Lbuilder – Modifying default schema

- Schema can be customized to use different types of widgets as inputs or outputs, and change widgets parameters

```
schema["inputs"]["cell_replication_number"]["type"] = "IntText"
schema["inputs"]["config_seed"]["type"] = "IntText"
schema["inputs"]["config_seed"]["value"] = 1
schema["inputs"]["composition_Nb"]["value"] = 0.25
schema["inputs"]["composition_Mo"]["value"] = 0.25
schema["inputs"]["composition_Ta"]["value"] = 0.25
schema["inputs"]["composition_W"]["value"] = 0.25
schema["inputs"]["Tinitial"]["value"] = 1200
schema["inputs"]["Tequil"]["value"] = 1200
schema["inputs"]["Pequil"]["value"] = 1.01
schema["inputs"]["velocity_seed"]["value"] = 123456
schema["inputs"]["velocity_seed"]["type"] = "IntText"
schema["inputs"]["config_seed"]["value"] = 1
schema["inputs"]["cell_replication_number"]["value"] = 4
schema["inputs"]["thermalization_time"]["value"] = 2000
schema["inputs"]["thermalization_time"]["type"] = "IntText"
```

Sim2Lbuilder – Modifying default schema **Images**

- Schema can be customized to use different types of widgets as inputs or outputs, and change widgets parameters

SCHEMA

```
{
  …
  'outputs': {
    'output1' : { 'type': 'Image' },
    'output2' : { 'type': 'Output' },
    …
  },
  …
}
```

RUNSIMTOOL FUNCTION

```
file = open("nanohub.png", "rb")
image = file.read()
widget.outputs["output1"].value = image

from IPython.display import Image
widget.outputs["output2"].clear_output()
with widget.outputs["output2"]:
  display(Image(url= "nanohub.png", width=200))
```

Sim2Lbuilder – Modifying default schema **2D PLOTS**

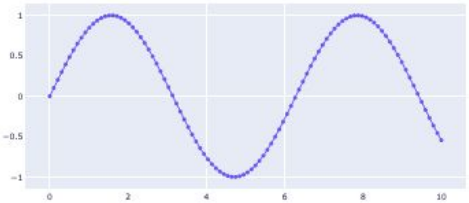
- Schema can be customized to use different types of widgets as inputs or outputs, and change widgets parameters

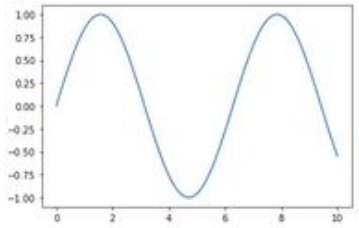
SCHEMA

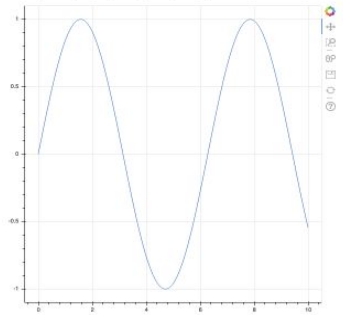
```
{
…
  'outputs': {
    'output1' : {
      'type': 'FigureWidget',
      'module' : 'plotly.graph_objects'
    },
    'output2' : { 'type': 'Output' },
    'output3' : { 'type': 'Output' },
    …
  }
  …
}
```

RUNSIMTOOL FUNCTION

```
x = np.linspace(0, 10, 100)
y = np.sin(x)
widget.outputs["output1"].add_trace(go.Scatter(x=x, y=y)

with widget.outputs["output2"]:
  fig = plt.figure()
  plt.plot(x, y)
  plt.show(fig)

with widget.outputs["output3"]:
  graph = figure()
  graph.line(x, y)
  show(graph)
  bokeh.io.output_notebook()
```







Sim2Lbuilder – Modifying default schema **3D PLOTS**

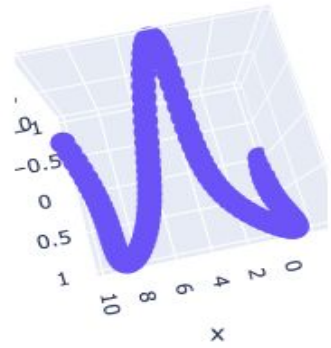
- Schema can be customized to use different types of widgets as inputs or outputs, and change widgets parameters

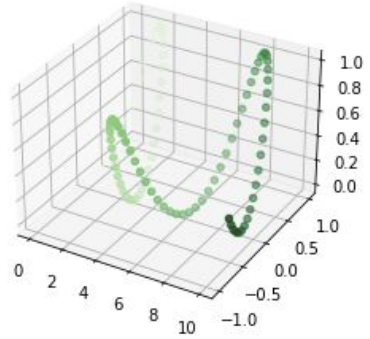
SCHEMA

```
{
…
  'outputs': {
    'output1' : {
      'type': 'FigureWidget',
      'module' : 'plotly.graph_objects'
    },
    'output2' : { 'type': 'Output' },
    …
  }
  …
}
```

RUNSIMTOOL FUNCTION

```
x = np.linspace(0, 10, 100)
y = np.sin(x)
z = y*np.sin(x)

widget.outputs["output1"].add_trace(go.Scatter3d(x=x, y=y, z=z,
mode='lines+markers'))

with widget.outputs["output2"]:
        fig = plt.figure()
        ax = plt.axes(projection='3d')
        ax.scatter3D(x, y, z, c=x, cmap='Greens');
        plt.show(fig)
```





Sim2Lbuilder – Modifying default schema **SURFACES**

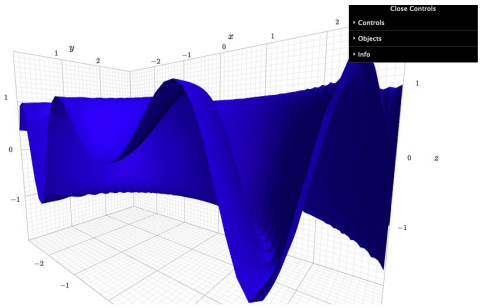
- Schema can be customized to use different types of widgets as inputs or outputs, and change widgets parameters

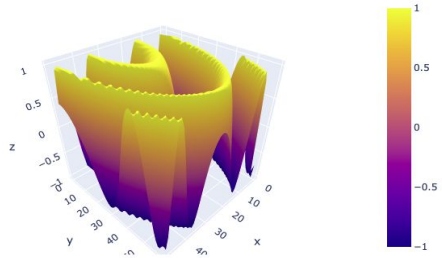
SCHEMA

```
{
…
  'outputs': {
    'output1' : {
      'type': 'FigureWidget',
      'module' : 'plotly.graph_objects'
    },
    'output2' : {
      'type': 'plot',
      'module' : 'k3d'
    },
    …
  }
  …
}
```

RUNSIMTOOL FUNCTION

```
x = np.linspace(xmin, xmax, Nx, dtype=np.float32
y = np.linspace(ymin, ymax, Ny, dtype=np.float32
x, y = np.meshgrid(x, y)
f = np.sin(x ** 2 + y ** 2)

widget.outputs["output1"].data = []
widget.outputs["output1"].add_trace(go.Surface(z=f))
widget.outputs["output2"].layout.flex= '1'

widget.outputs["output2"].objects=[]
widget.outputs["output2"].__iadd__(k3d.surface(f, xmin=xmin, xmax=xmax,
ymin=ymin, ymax=ymax));
widget.outputs["output2"].layout.flex= '1'
```





Sim2Lbuilder – Modifying default schema **MOLECULES**

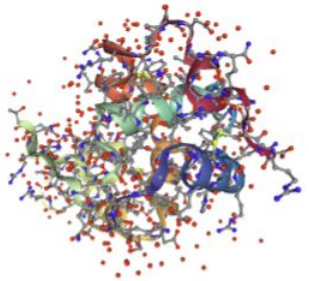
- Schema can be customized to use different types of widgets as inputs or outputs, and change widgets parameters

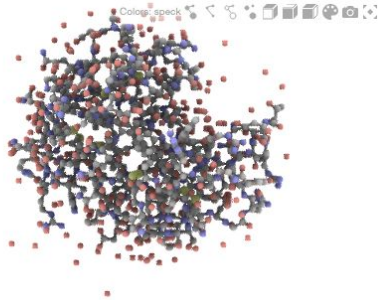
SCHEMA

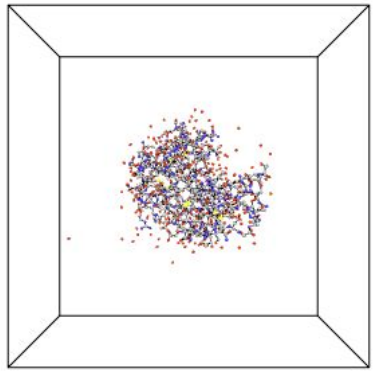
```
{
…
  'outputs': {
    'output1' : {
      'type': 'NGLWidget',
      'module' : 'nglview'
    },
    'output2' : {
      'type': 'Speck',
      'module' : ipyspeck.speck'
    },
    'output3' : {
      'type': Output,
    }
    …
  }
  …
}
```

RUNSIMTOOL FUNCTION

```
obConversion = openbabel.OBConversion()
obConversion.SetInAndOutFormats("pdb", "xyz")
mol = openbabel.OBMol()
obConversion.ReadFile(mol, "1hel.pdb")
co2 = obConversion.WriteString(mol)

widget.outputs["output1"].layout.width= '100%'
widget.outputs["output1"].add_component('1hel.pdb')
widget.outputs["output1"].add_representation('ball+stick')

widget.outputs["output2"].data=co2

widget.outputs['output3'].clear_output()
with widget.outputs["output3"]:
    imolecule.draw("1hel.pdb")
```







Sim2Lbuilder – Modifying default schema **MAPS**

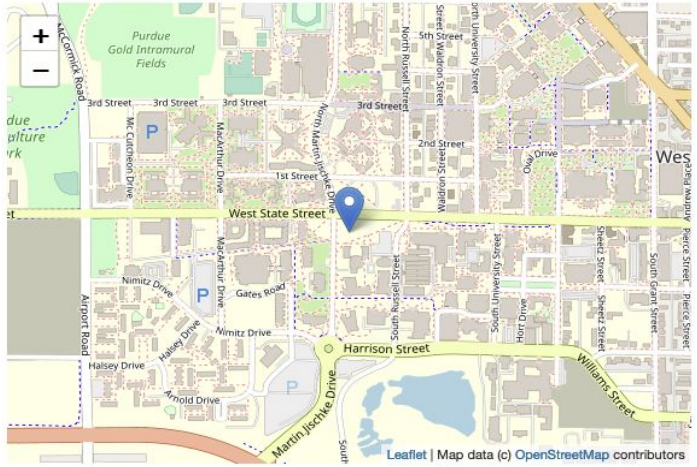
- Schema can be customized to use different types of widgets as inputs or outputs, and change widgets parameters

SCHEMA

```
{
…
  'outputs': {
    'output1' : {
      'type': 'Map',
      'module' : 'ipyleaflet'
    }
    …
  }
  …
}
```

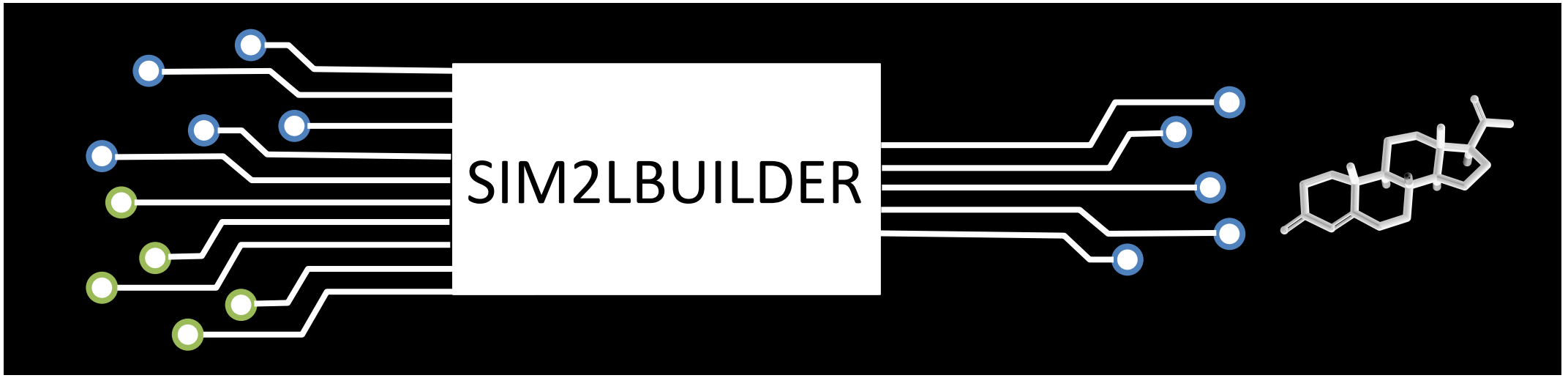
RUNSIMTOOL FUNCTION

```
widget.outputs["output1"].center=(40.4237, -86.9212)
widget.outputs["output1"].zoom=15
marker = ipyleaflet.Marker(
  location=(40.4237, -86.9212),
  draggable=False,
  title="Purdue University"
)
widget.outputs["output1"].add_layer(marker);
```



Sim2lbuilder - WidgetConstructor

- Widget can be exported as python code, imported later




```
s = WidgetConstructor(schema, format="file", widget_name="Widget0")
s.RunSimTool = RunSimTool
s.assemble()
…
import Widget0
Widget0.Widget0()
```

Overview

- Connecting Sim2Ls to Web Apps
  - nanoHUB end points / REST
  - nanohub-uidl to create Apps

nanoHUB REST/API

- The nanoHUB web API uses standard HTTP GET and POST methods
- In order to use the nanoHUB web API you need a nanoHUB account.
- To host apps outside nanoHUB, you need to register an app
  - https://nanohub.org/developer/api/applications/new.
- Python
  - requests library (https://pypi.org/project/requests/)
- Javascript
  - axios library (https://axios-http.com)

Authentication (OAuth2)

OAuth2 is a protocol that lets external applications request authorization to private details in a user's account without getting their password. This is preferred over Basic Authentication because tokens can be limited to specific types of data, and can be revoked by users at any time.

nanoHUB REST/API Getting a token

```python
import requests
auth_data = {
    'client_id': 'XXX',# XXX Get this info when you create a web app
    'client_secret': 'XXX',# XXX Get this info when you create a web app
    'grant_type': 'password',
    'username': 'XXX',# XXX Your nanoHUB username
    'password': 'XXX',# XXX Your nanoHUB password
}
auth_json = requests.post(
    'https://nanohub.org/api/developer/oauth/token',
    data=auth_data
)
if auth_json:
    print(auth_json.json())
else:
    print(auth_json.status_code)
```

User-Password

```python
import requests
import os
auth_data = {
    'grant_type' : 'tool',
}
with open(os.environ["SESSIONDIR"]+"/resources") as file:
    lines = [line.split(" ", 1) for line in file.readlines()]
    properties = {line[0].strip(): line[1].strip() for line in lines if len(line)==2}
    auth_data["sessiontoken"] = properties["session_token"]
    auth_data["sessionnum"] = properties["sessionid"]
auth_json = requests.post(
    'https://nanohub.org/api/developer/oauth/token',
    data=auth_data
)
if auth_json:
    print(auth_json.json())
else:
    print(auth_json.status_code)
```

Current session

nanohub-remote

- Nanohub-remote is simple sdk to interact with nanoHUB REST/API
  - https://nanohub.org/resources/nhremote for more examples

```
import sys
import nanohub.remote as nr
import nanohub.uidl
import warnings
warnings.filterwarnings("ignore")
import os
auth_data = {
    'grant_type' : 'tool',
}
with open(os.environ["SESSIONDIR"]+"/resources") as file:
    lines = [line.split(" ", 1) for line in file.readlines()]
    properties = {line[0].strip(): line[1].strip() for line in lines if len(line)==2}
    auth_data["sessiontoken"] = properties["session_token"]
    auth_data["sessionnum"] = properties["sessionid"]
```

```
tool = nr.Sim2l(auth_data)
```

```
SIMTOOLNAME = "caecipher"
schema = tool.getSchema(SIMTOOLNAME)
params = tool.getToolParameters(SIMTOOLNAME)
```

Get schema and parameters for a sim2l

nanohub-uidl

- Nanohub-uidl is a library to create javascript code based on a json schema, inspired on the concepts of teleporthq
  - https://teleporthq.io/repl

```
import sys
import json
import nanohub.uidl.teleport as t
from nanohub.uidl.simtool import SimtoolBuilder
from nanohub.uidl.material import MaterialBuilder
from nanohub.uidl.material import MaterialContent, MaterialLabContent
from nanohub.uidl.plotly import PlotlyBuilder
from nanohub.uidl.app import AppBuilder

from nanohub.uidl.nanohub import Auth
TOOLNAME = schema["name"]# "st4pnjunction"
REVISION = schema["revision"]# 6
STATE_LOADER_STATUS = "loader_status"
STATE_LOADER_OPEN = "loader_open"
STATE_LOGIN_OPEN = "login_open"
STATE_ERROR_STATUS = "error_status"
STATE_ERROR_OPEN = "error_open"
STATE_ALERT_STATUS = "alert_status"
STATE_ALERT_OPEN = "alert_open"
```

Defining App state variables

nanohub-uidl / Project

- A projects is a collection of components, javascript libraries are loaded by default from public Content Delivery Network (CDN) repositories, this can be changed

```
Project = t.TeleportProject("PNToy Lab")
Component = Project.root

```

```
Project.libraries['react'] = 'https://nanohub.org/js_apps/react.production.min'
Project.libraries['react-dom'] = 'https://nanohub.org/js_apps/react-dom.production.min'
Project.libraries['material-ui'] = 'https://nanohub.org/js_apps/material-ui.production.mi
Project.libraries['materiallab-ui'] = 'https://nanohub.org/js_apps/material-ui-lab.produc
Project.libraries['math'] = 'https://cdnjs.cloudflare.com/ajax/libs/mathjs/6.6.1/math.min
Project.libraries['axios'] = 'https://nanohub.org/js_apps/axios.min'
Project.libraries['localforage'] = 'https://nanohub.org/js_apps/localforage.min'
Project.libraries['prop-types'] = 'https://nanohub.org/js_apps/prop-types.min'
```

Create main project

nanohub-uidl / Auxiliary widgets

- Add “loaders” to include visual feedback to user

Creating Error

```
ErrorMessage = SimtoolBuilder.Error(
    Component,
    error_status = STATE_ERROR_STATUS,
    error_open = STATE_ERROR_OPEN,
    is_open = False
)
```

Creating ALERT Mesage

```
AlertMessage = SimtoolBuilder.Error(
    Component,
    error_status = STATE_ALERT_STATUS,
    error_open = STATE_ALERT_OPEN,
    is_open = False,
    title = "Message"
)
AlertMessage.content.children[0].content.style['backgroundColor'] = '#FFF380'
```

Creating loader

```
Loader = SimtoolBuilder.Loader(
    Component,
    loader_status = STATE_LOADER_STATUS,
    loader_open = STATE_LOADER_OPEN,
    is_open = False
)
```

Other components

nanohub-uidl / sim2l schema

- The javascript app would need the sim2l schema to submit runs

```
SimtoolBuilder.buildSchema(
    Project,
    Component,
    url = "https://nanohub.org/api/dbexplorer/simtools",
    toolname = TOOLNAME,
    revision = REVISION
)
Component.addPropVariable(
    "onLoadSchema",
    {
        "type" : "func",
        'defaultValue' :
        '(e)=>{e.setState({"'+ STATE_LOADER_OPEN +'":false})}'
    }
)
```

nanohub-uidl / App authentication

- The web app can use two types of authentication:
- Auth.Login() to require user and password
- Auth.Session() reuse the current session (published on nanoHUB)

```
from secrets import IDCLIENT, SECRET
auth_data['client_id'] = IDCLIENT
auth_data['client_secret'] = SECRET
# to get client_id and client_secret, create a web application (https://nanohub.org/devel

Login, CLogin = Auth.Login(
    Project,
    Component,
    client_id = auth_data['client_id'],
    client_secret = auth_data['client_secret'],
    url = "https://nanohub.org/api/developer/oauth/token",
    open_state = STATE_LOGIN_OPEN
)
Login.content.events["onAuth"] = [
    { "type": "stateChange", "modifies": STATE_LOGIN_OPEN, "newState": False},
    { "type": "propCall2", "calls": "buildSchema", "args": ['self'] }
]
```

```
Login, CLogin = Auth.Session(
    Project,
    Component,
    sessiontoken = auth_data["sessiontoken"],
    sessionnum = auth_data["sessionnum"],
    url = "https://nanohub.org/api/developer/oauth/token",
)
Login.content.events["onError"]=[
    { "type": "stateChange", "modifies": STATE_ERROR_OPEN, "newState": True},
    { "type": "stateChange", "modifies": STATE_ERROR_STATUS, "newState": '$e'},
]
Login.content.events["onAuth"] = [
    { "type": "propCall2", "calls": "buildSchema", "args": ['self'] }
]

```

nanohub-uidl / Settings + Layout

- Settings and layout can be extracted from the sim2l parameters
- IDs not included in the layout would be not visible in the app

```
SETTINGS = {
    'values': {
        'type': 'String',
        'default_value': 'Animal Jumps Merrily',
        'label': 'text',
        'description': 'lowercase string that is going to be encoded'
    }, 'shift_input': {
        'type': 'Integer',
        'default_value': 24,
        'units': None,
        'min': 1,
        'max': 50,
        'label': 'Shifting',
        'description': 'integer that determines the shift amount for encoding'
    }, 'cores' : {
        "type": "Integer",
        "default_value": 1,
        "min" : 1,
        "max" : 1,
        "units" : "",
        "description" : "Number of cores in in the venue",
        "label" : "Number of cores",
    }, "cutoff" : {
}
```

```
LAYOUT = {
    'input': {
        'type': 'group',
        'id': '',
        'label': 'Inputs',
        'enable': None,
        'layout': 'vertical',
        'children': [
            {
                'type': 'text',
                'id': 'values',
                'enable': None
            },{
                'type': 'number',
                'id': 'shift_input',
                'enable': None
            }
        ]
    }
}
```

```
SETTINGS = {}
for option, value in PARAMS.items():
    if isinstance(value,nr.params.Number):
        SETTINGS[option] = {
            "type": "Number",
            "default_value": value.default,
            "units" : value.units,
            "min" : value.min,
            "max" : value.max,
            "label" : value.label,
            "description" : value.description,
        }
    elif isinstance(value,nr.params.Integer):
        SETTINGS[option] = {
            "type": "Integer",
            "default_value": value.default,
            "units" : value.units,
            "min" : value.min,
            "max" : value.max,
            "label" : value.label,
            "description" : value.description,
        }
    elif isinstance(value,nr.params.String):
        SETTINGS[option] = {
            "type": "String",
            "default_value": value.default,
            "label" : value.label,
            "description" : value.description,
        }
    elif isinstance(value,nr.params.Choice):
        SETTINGS[option] = {
            "type": "Select",
            "default_value": value.default,
            "options": {k : k for k in value.options},
            "units" : value.units,
            "label" : value.label,
            "description" : value.description,
        }
    elif isinstance(value,nr.params.Boolean):
        SETTINGS[option] = {
            "type": "Boolean",
            "default_value": (value.default == "yes"),
            "description" : value.description,
            "label" : value.label,
        }
```

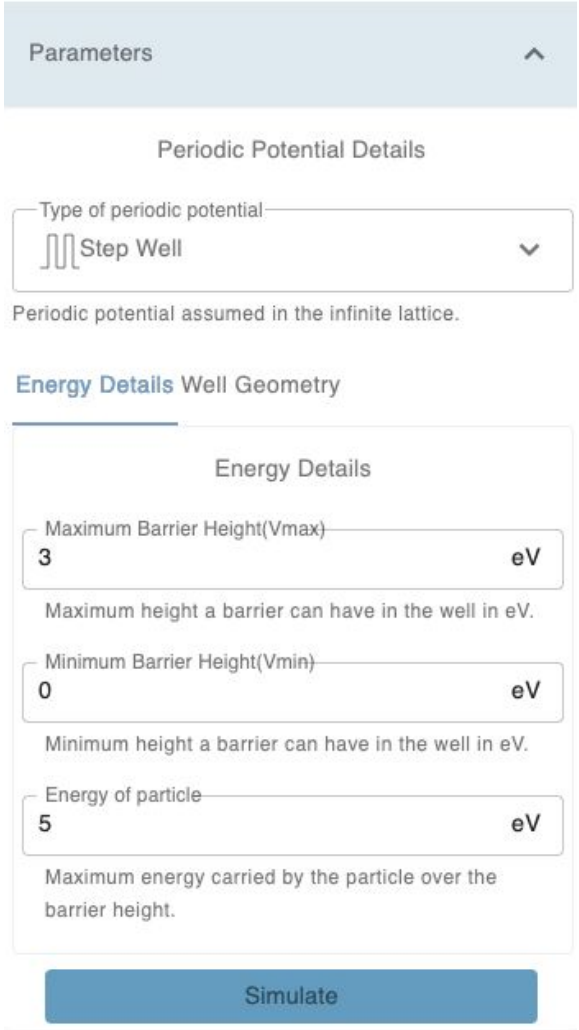
nanohub-uidl / Settings + Layout types

- Available setting types:
  - IconList, ButtonList, Select, IntegerAsString, Integer, Number, NumberAsString, String, Boolean , IntSwitch, DictionaryAsString, StringListAsString, NumberListAsString, IntListAsString
  - More types of widgets can be added modifying the schema or by creating additional components
- Available layout group:
  - Tab, group, container

nanohub-uidl / Settings component

- Settings and layout would be translated as a react component calling endpoint to run sim2l and request “outputs”

```
url_sim = "https://nanohub.org/api/dbexplorer/simtools"
AppSettings = AppBuilder.Settings(
    Project,
    Component,
    SETTINGS,
    url=url_sim,
    toolname = TOOLNAME,
    revision = REVISION,
    layout = LAYOUT['input'],
    outputs = ['Cipher', 'Repeated Chars', 'Repeated Chars Count'],
    runSimulation = "simtool"
)
AppSettings.content.events["onError"]=[
    { "type": "stateChange", "modifies": STATE_LOADER_OPEN,"newState": False},
    { "type": "stateChange", "modifies": STATE_ERROR_OPEN, "newState": True},
    { "type": "stateChange", "modifies": STATE_ERROR_STATUS, "newState": '$e'}
]
AppSettings.content.events["click"]=[
    { "type": "stateChange", "modifies": STATE_LOADER_OPEN,"newState": True}
]

AppSettings.content.events["submit"] = [
    { "type": "stateChange", "modifies": "parameters","newState": '$e.target.value'}
]

AppSettings.content.events["onStatusChange"]=[
    { "type": "stateChange", "modifies": STATE_LOADER_STATUS,"newState": "$e.target.value'
]
```



nanohub-uidl / Getting Results

- App can callback a javascript function after successfully getting results from the sim2l.
- hashkey is the identifier to recover results from browser datastore

```
eol = "\n"
js = ""
js += " (self, hashkey) => {" + eol
js += "  CacheStore.getItem(hashkey).then((value)=>{" + eol
js += "    var jsonOutput = JSON.parse(value);" + eol
js += "    var message = 'Cypher phrase : ' + jsonOutput['Cipher'] + '\\n'" + eol
js += "    message += 'Repeated Chars : ' + jsonOutput['Repeated Chars'] + '\\n'" + eol
js += "    self.setState({'alert_status':message});" + eol
js += "  });" + eol
js += "}" + eol
Component.addPropVariable("loadBaseOutput", {"type":"func", "defaultValue": js})
```

```
AppSettings.content.events["onSuccess"]=[
    { "type": "stateChange", "modifies": STATE_LOADER_OPEN,"newState": False },
    { "type": "stateChange", "modifies": STATE_ERROR_OPEN, "newState": False},
    { "type": "stateChange", "modifies": STATE_ERROR_STATUS, "newState": ''},
    {
        "type": "stateChange",
        "modifies": STATE_ALERT_OPEN,
        "newState": True,
        "callbacks" : [
          {
            "type": "propCall2",
            "calls": "loadBaseOutput",
            "args": ['self', 'arguments[1]']
          }
        ]
    }
]
```

nanohub-uidl / Getting Results / Plots

- refreshViews, loadXY and loadSequence allows developers to visualize and customize plots

```
SimtoolBuilder.loadXY(
    Project,
    Component,
    #cache_store = TOOLNAME + "_" + str(REVISION)
)
SimtoolBuilder.loadSequence(
    Project,
    Component,
    #cache_store = TOOLNAME + "_" + str(REVISION)
)

Project.root.addStateVariable("visualization", {"type":"object", "defaultValue": {
    'function':'loadSequence',
    'dataset' : ['Ec', 'Ev', 'Ei'],
    'layout' : {},
    'parameters' : {'position':'position','function':'function'}
}})

RESULTS = {
    "bands" : {
        'title' : 'Energy Band Diagram',
        'action' : { "type": "stateChange", "modifies": "visualization","newState": {
            'function':'loadSequence',
            'dataset' : ['Ec', 'Ev', 'Ei'],
            'layout' : {'title':'Energy Band Diagram', 'yaxis': { 'title' : 'Energy (eV)'
            'parameters' : {'position':'position','function':'function'},
            'normalize' : True,
            'start_trace' : 0
        },
            "callbacks" : onRefreshViews
        }
    },
```

loadXY: creates a single plot using sim2l outputs defined in the ”dataset” parameters, it expects sim2l output to be a dictionary for X and Y lists

loadSequence: creates multiple plots similarly to loadXY, , it expects sim2l output to be a dictionary, each key should contain a dictionary for X and Y lists

layout: customize plotly layout (https://plotly.com/javascript/reference/layout/)

parameters: describes the key for X values (position) and Y values (function) in the javascript object

normalize: sequence shares the same axis for all plots

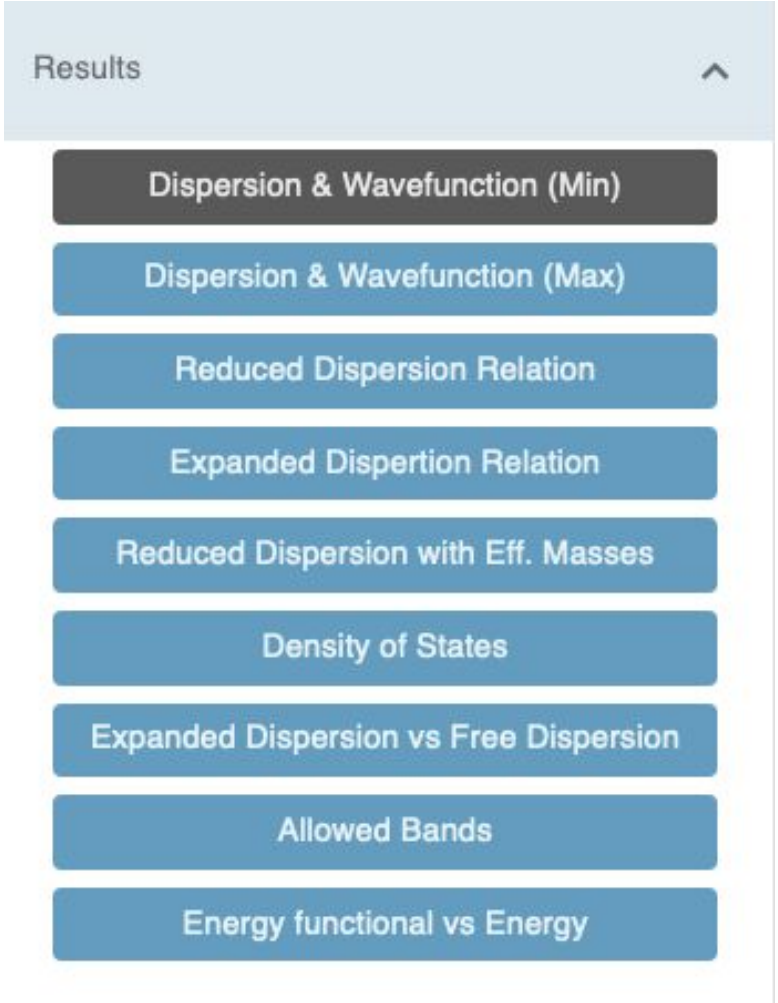
start_trace: sequence starts on the index of the trace

nanohub-uidl / Getting Results / Plots

- Each Key in the dictionary would be represented as a button on the Results component

```
    "carrier" : {
        'title' : 'Excess Carrier Density',
        'action' : {
            "type": "stateChange",
            "modifies": "visualization",
            "newState": {
                'function':'loadSequence',
                'dataset' : ['Excess Electron Density', 'Excess Hole Density'],
                'layout' : {'title':'Excess Carrier Density', 'yaxis': { 'type' : 'log',
                'parameters' : {'position':'position','function':'function'},
                'normalize' : True,
                'start_trace' : 1
            },
            "callbacks" : onRefreshViews
        }
    },
}
```

```
PNTOYResults = AppBuilder.Results(
    Component,
    results = RESULTS,
    onClick = [{ "type": "stateChange", "modifies": STATE_LOADER_OPEN,"newState": True }]
    onLoad = [
        { "type": "stateChange", "modifies": STATE_LOADER_OPEN,"newState": False }
    ],
)
```



nanohub-uidl / Assembling the app

- Customize the App changing the Theme colors
- Add images to customize the AppBar
- buildReact translate the schema to a React App

```
ThemeProvider = MaterialBuilder.ThemeProvider( Component, MaterialBuilder.DefaultTheme(
    primary_color = '#699FBB',
    secondary_color = '#f1f1f1',
    primary_bg = '#FFFFFF',
    secondary_bg = '#dbeaf0',
    default_button = 'rgba(255, 255, 255, 0.87)',
    primary_button = 'rgba(255, 255, 255, 0.87)',
    secondary_button = 'rgba(0, 0, 0, 0.87)',
    default_button_bg = 'rgb(63, 162, 192)',
    primary_button_bg = 'rgba(0, 0, 0, 0.65)',
    secondary_button_bg = 'rgba(0, 0, 0, 0.12)',
))
AppBar = MaterialBuilder.AppBar(
    title="Caesar Cipher Tool"
)
logo = t.TeleportElement(t.TeleportContent(elemen
logo.content.attrs["width"] = "120"
logo.content.attrs["src"] = "https://nanohub.org/
AppBar.content.children[0].addContent(logo)

```

```
Gridv = t.TeleportElement(MaterialContent(elementType="Grid"))
Gridv.content.attrs["container"] = True
Gridv.content.attrs["direction"] = "column"
Gridv.addContent(AppBar)
Gridv.addContent(AppSettings)

ThemeProvider.addContent(Gridv)
ThemeProvider.addContent(Loader)
ThemeProvider.addContent(ErrorMessage)
ThemeProvider.addContent(AlertMessage)
ThemeProvider.addContent(Login)

Component.addNode(ThemeProvider)
Project.buildReact( TOOLNAME+"PROD"+".html");
```

nanohub-uidl / Publishing the App

- Customize the App changing the Theme colors (ThemeProvider)
- Add images to customize the Application top header (AppBar)
- buildReact translates the created schema as a React App

```
ThemeProvider = MaterialBuilder.ThemeProvider( Component, MaterialBuilder.DefaultTheme(
    primary_color = '#699FBB',
    secondary_color = '#f1f1f1',
    primary_bg = '#FFFFFF',
    secondary_bg = '#dbeaf0',
    default_button = 'rgba(255, 255, 255, 0.87)',
    primary_button = 'rgba(255, 255, 255, 0.87)',
    secondary_button = 'rgba(0, 0, 0, 0.87)',
    default_button_bg = 'rgb(63, 162, 192)',
    primary_button_bg = 'rgba(0, 0, 0, 0.65)',
    secondary_button_bg = 'rgba(0, 0, 0, 0.12)',
))
AppBar = MaterialBuilder.AppBar(
    title="Caesar Cipher Tool"
)
logo = t.TeleportElement(t.TeleportContent(elementT
logo.content.attrs["width"] = "120"
logo.content.attrs["src"] = "https://nanohub.org/ap
AppBar.content.children[0].addContent(logo)

```

```
Gridv = t.TeleportElement(MaterialContent(elementType="Grid"))
Gridv.content.attrs["container"] = True
Gridv.content.attrs["direction"] = "column"
Gridv.addContent(AppBar)
Gridv.addContent(AppSettings)

ThemeProvider.addContent(Gridv)
ThemeProvider.addContent(Loader)
ThemeProvider.addContent(ErrorMessage)
ThemeProvider.addContent(AlertMessage)
ThemeProvider.addContent(Login)

Component.addNode(ThemeProvider)
Project.buildReact( TOOLNAME+"PROD"+".html");
```

More examples

- https://github.com/denphi/nanohub-uidl/tree/master/examples/tools
- https://nanohub.org/tools/pnjunctionapp
- https://nanohub.org/js_apps/kronig_penney.html

Overview

- Connecting Sim2Ls to Jupyter
  - Jupyter Widgets
  - sim2lbuilder
  - Custom widgets

- Connecting Sim2Ls to Web Apps
  - nanoHUB end points / REST
  - nanohub-uidl to create Apps