



23,000 annual simulation users



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

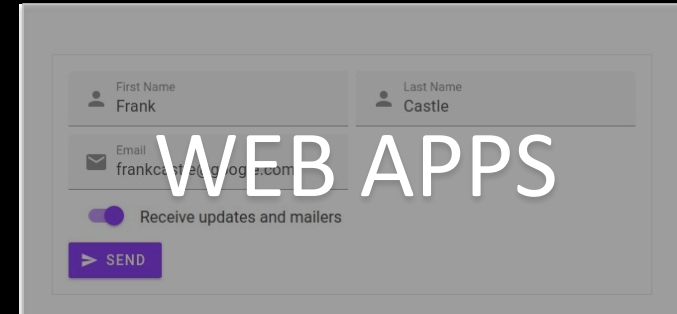
Overview

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

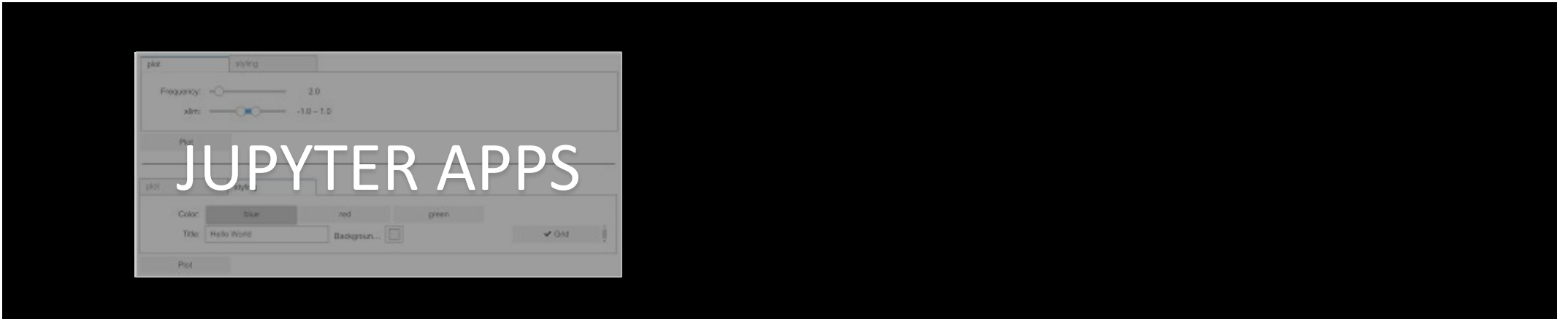
- Connecting Sim2Ls to Jupyter
 - Jupyter Widgets
 - sim2lbuilder
 - Custom widgets



- 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,  
)
```

Number:

2



Jupyter Widgets - Community

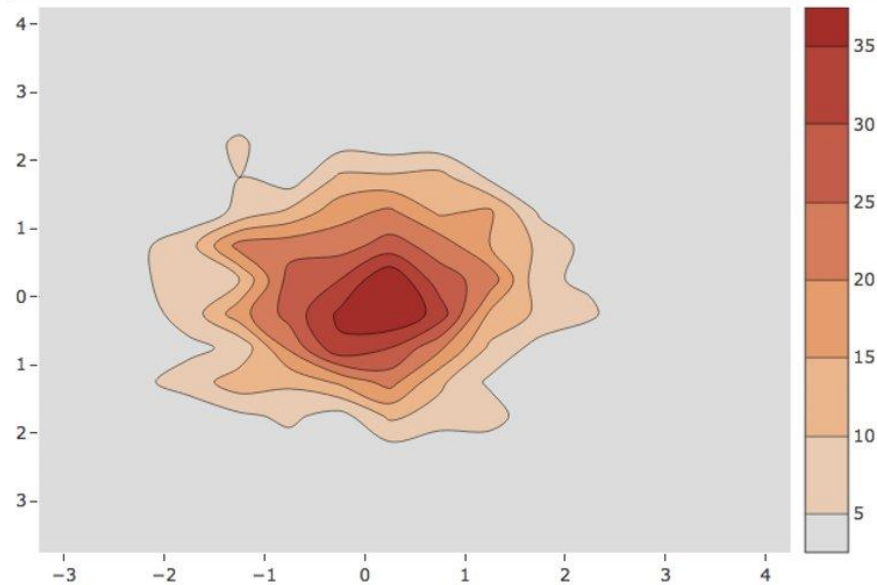
<https://jupyter.org/widgets>

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

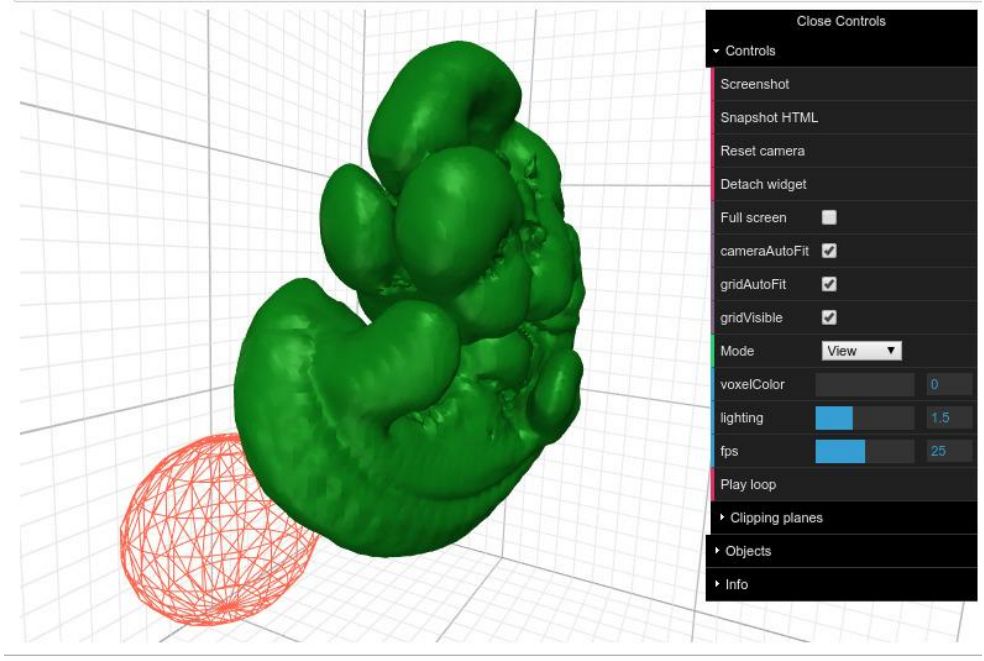
```
import plotly.graph_objs as go
```

```
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])
```

correct	battery
horse	staple

```
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
```

0	1	2	3	4
P3 <input type="text"/>				

<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)

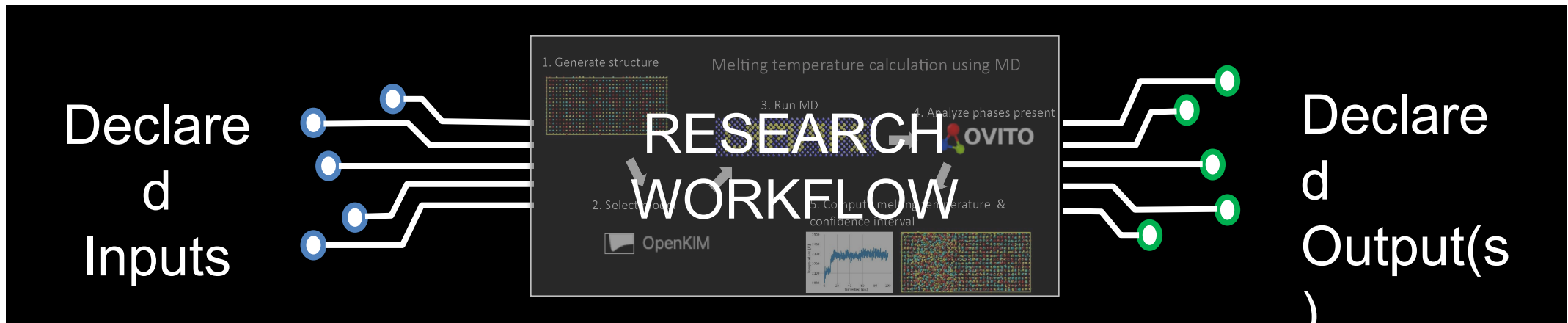
def on_button_clicked(b):
    with output:
        print("Button clicked.")

button.on_click(on_button_clicked)
```

Click Me!

What are Sim2Ls?

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

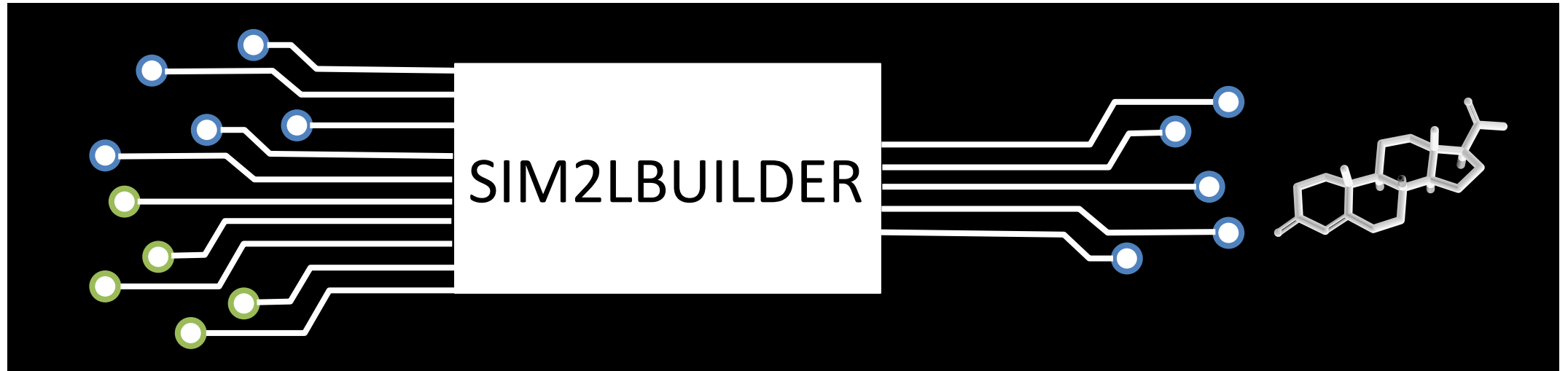


- Simulation as a service. Launch Sim2Ls from:
 - From a GUI or App
 - From AI/ML or high-throughput workflows
 - From inside nanoHUB or outside

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

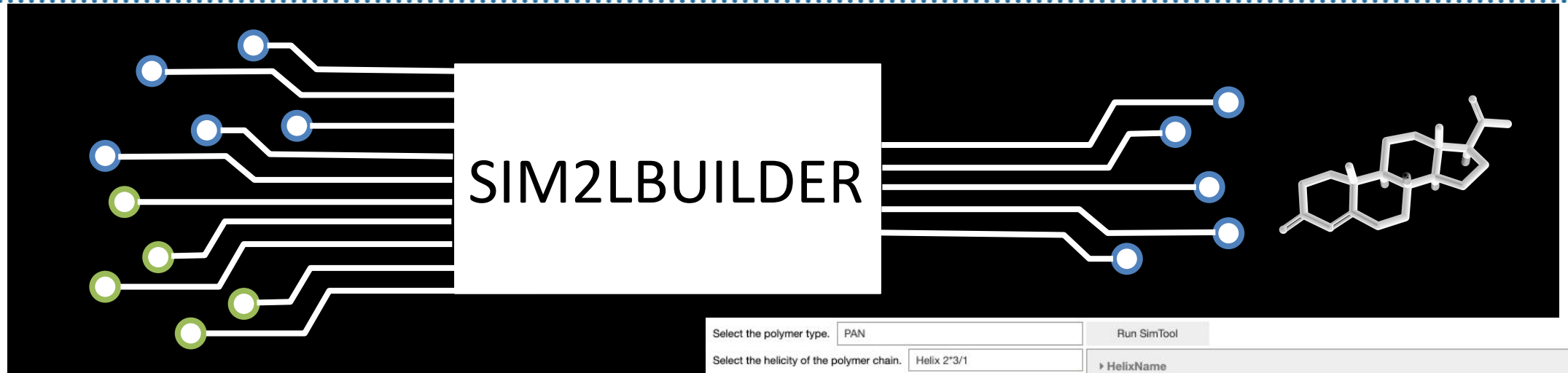
```
1 schema = GetSimtoolDefaultSchema("meltingkim")
2 schema
{'inputs': {'material': {'type': 'Text',
  'description': 'Element to be simulated',
  'value': 'Ni'},
  'crystal_structure': {'type': 'Text',
  'description': 'Crystal structure for initial conditions',
  'value': 'fcc'},
  'lattice_parameter': {'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',
  'description': 'Initial temperature assigned to the liquid region'}}
```

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?



```
1 from sim2lbuilder import WidgetConstructor, GetSimtoolDefaultSchema
2 from simtool import searchForSimTool, getSimToolInputs, Run
3 SIMTOOL_NAME = "meltingkim"
4 schema = GetSimtoolDefaultSchema(SIMTOOL_NAME)
5 def RunSimTool(widget, tool=SIMTOOL_NAME):
6     stl = searchForSimTool(tool)
7     inputs = getSimToolInputs(stl)
8     for i,w in widget.inputs.items():
9         inputs[i].value = w.value
10    r = Run(stl, inputs)
11    for outk, out in widget.outputs.items():
12        with out:
13            print(r.read(outk))
14    s = WidgetConstructor(schema)
15    s.RunSimTool = RunSimTool
16    s.display()
17
```

Select the polymer type. PAN	Run SimTool
Select the helicity of the polymer chain. Helix 2*3/1	▶ HelixName
Number of monomers 30	▼ PDBview
Select the tacticity of the polymer chain. isotactic	
Select the chirality of the polymer chain. right	
Ratio of head-to-head and tail-to-tail connections 0	▶ PDB
Number of attempts to find a configuration 30	▶ LAMMPSDataFile
Whether to create periodic infinite chain	▶ Datafile_warnings
Whether to create corresponding LAMMPS data file or not	▶ X6paircoeffs
Whether to create corresponding LAMMPS input File or not	
Force field Dreiding	
Applied to equilibrium bond lengths 1.1	
Charge equilibration method Gasteiger	

Sim2Lbuilder – Modifying default schema

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

```
1 schema["inputs"]["cell_replication_number"]["type"] = "IntText"
2 schema["inputs"]["config_seed"]["type"] = "IntText"
3 schema["inputs"]["config_seed"]["value"] = 1
4 schema["inputs"]["composition_Nb"]["value"] = 0.25
5 schema["inputs"]["composition_Mo"]["value"] = 0.25
6 schema["inputs"]["composition-Ta"]["value"] = 0.25
7 schema["inputs"]["composition_W"]["value"] = 0.25 |
8 schema["inputs"]["Tinitial"]["value"] = 1200
9 schema["inputs"]["Tequil"]["value"] = 1200
10 schema["inputs"]["Pequil"]["value"] = 1.01
11 schema["inputs"]["velocity_seed"]["value"] = 123456
12 schema["inputs"]["velocity_seed"]["type"] = "IntText"
13 schema["inputs"]["config_seed"]["value"] = 1
14 schema["inputs"]["cell_replication_number"]["value"] = 4
15 schema["inputs"]["thermalization_time"]["value"] = 2000
16 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

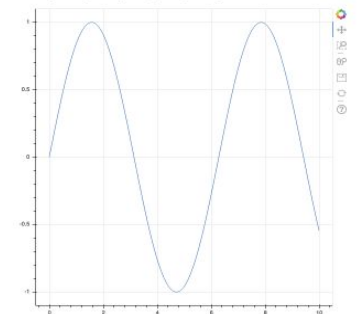
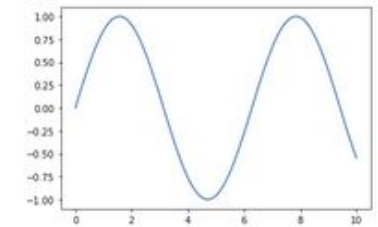
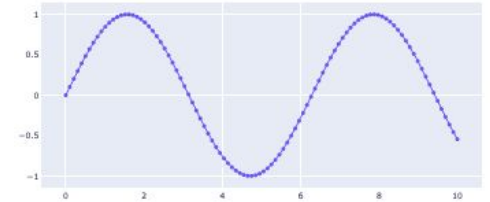
```
{  
...  
  'outputs': {  
    'output1': {  
      'type': 'FigureWidget',  
      'module': 'plotly.graph_objects'  
    },  
    'output2': { 'type': 'Output' },  
    'output3': { 'type': 'Output' },  
    ...  
  }  
  ...  
}
```

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

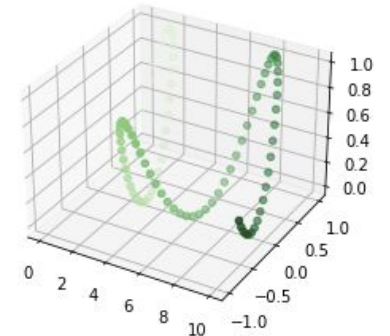
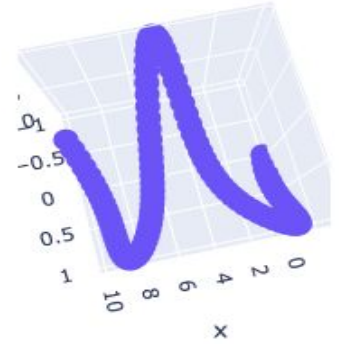
```
{  
...  
'outputs': {  
  'output1': {  
    'type': 'FigureWidget',  
    'module': 'plotly.graph_objects'  
  },  
  'output2': { 'type': 'Output' },  
...  
}
```

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

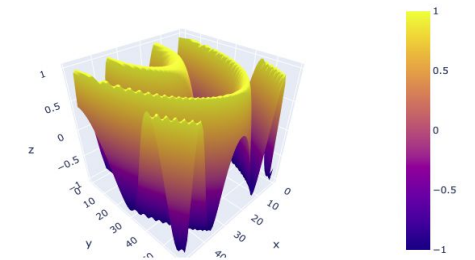
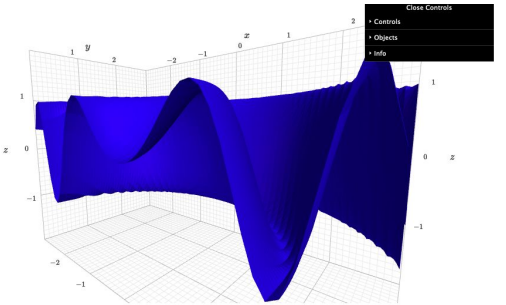
```
{
...
'outputs': {
  'output1': {
    'type': 'FigureWidget',
    'module': 'plotly.graph_objects'
  },
  'output2': {
    'type': 'plot',
    'module': 'k3d'
  },
...
}
...
}
```

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',
  }
...
}
```

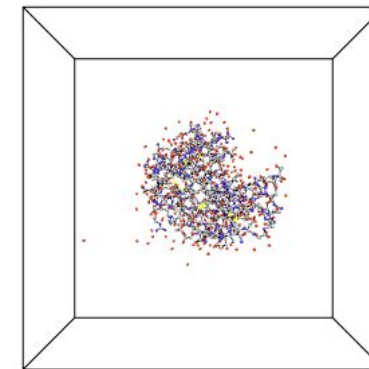
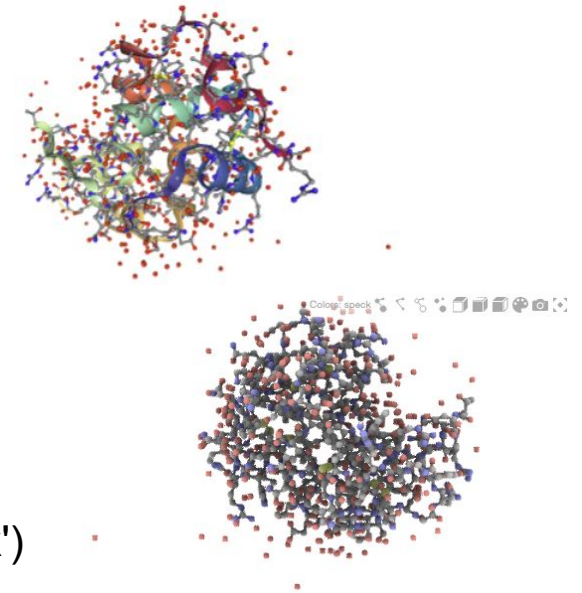
RUNSIMTOOL FUNCTION

```
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**

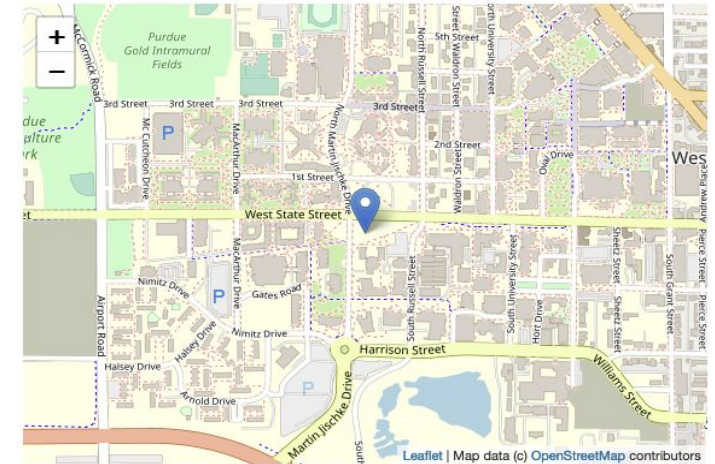
- Schema can be customized to use different types of widgets as inputs or outputs, and change widgets parameters

SCHEMA

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

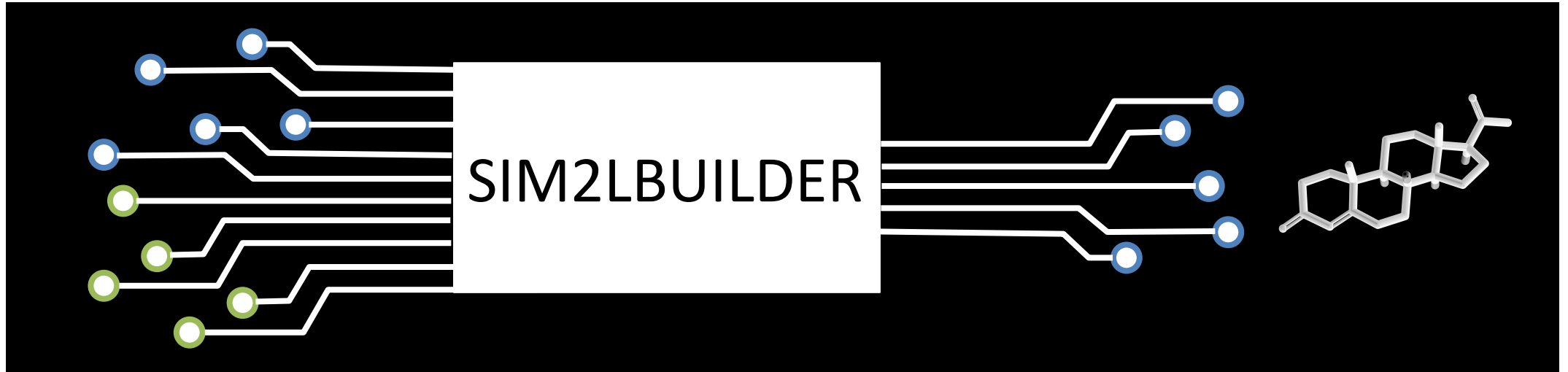
RUNSIMTOOL FUNCTION

```
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 \(https://pypi.org/project/requests/\)](https://pypi.org/project/requests/)
- Javascript
 - [axios library \(https://axios-http.com\)](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

```
1 import requests
2 auth_data = {
3     'client_id': 'XXX', # XXX Get this info when you create a web app
4     'client_secret': 'XXX', # XXX Get this info when you create a web app
5     'grant_type': 'password',
6     'username': 'XXX', # XXX Your nanoHUB username
7     'password': 'XXX', # XXX Your nanoHUB password
8 }
9 auth_json = requests.post(
10     'https://nanohub.org/api/developer/oauth/token',
11     data=auth_data
12 )
13 if auth_json:
14     print(auth_json.json())
15 else:
16     print(auth_json.status_code)
```

User-Password

```
1 import requests
2 import os
3 auth_data = {
4     'grant_type': 'tool',
5 }
6 with open(os.environ["SESSIONDIR"]+"/resources") as file:
7     lines = [line.split(" ", 1) for line in file.readlines()]
8     properties = {line[0].strip(): line[1].strip() for line in lines if len(line)==2}
9     auth_data["sessiontoken"] = properties["session_token"]
10    auth_data["sessionnum"] = properties["sessionid"]
11 auth_json = requests.post(
12     'https://nanohub.org/api/developer/oauth/token',
13     data=auth_data
14 )
15 if auth_json:
16     print(auth_json.json())
17 else:
18     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

```
1 import sys
2 import nanohub.remote as nr
3 import nanohub.uidl
4 import warnings
5 warnings.filterwarnings("ignore")
6 import os
7 auth_data = {
8     'grant_type' : 'tool',
9 }
10 with open(os.environ["SESSIONDIR"]+"/resources") as file:
11     lines = [line.split(" ", 1) for line in file.readlines()]
12     properties = {line[0].strip(): line[1].strip() for line in lines if len(line)==2}
13     auth_data["sessiontoken"] = properties["session_token"]
14     auth_data["sessionnum"] = properties["sessionid"]
```

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

```
1 SIMTOOLNAME = "caecipher"
2 schema = tool.getSchema(SIMTOOLNAME)
3 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>

```
1 import sys
2 import json
3 import nanohub.uidl.teleport as t
4 from nanohub.uidl.simtool import SimtoolBuilder
5 from nanohub.uidl.material import MaterialBuilder
6 from nanohub.uidl.material import MaterialContent, MaterialLabContent
7 from nanohub.uidl.plotly import PlotlyBuilder
8 from nanohub.uidl.app import AppBuilder
9
10 from nanohub.uidl.nanohub import Auth
11 TOOLNAME = schema["name"]# "st4pnjunction"
12 REVISION = schema["revision"]# 6
13 STATE_LOADER_STATUS = "loader_status"
14 STATE_LOADER_OPEN = "loader_open"
15 STATE_LOGIN_OPEN = "login_open"
16 STATE_ERROR_STATUS = "error_status"
17 STATE_ERROR_OPEN = "error_open"
18 STATE_ALERT_STATUS = "alert_status"
19 STATE_ALERT_OPEN = "alert_open"
```

Defining App state variables

nanohub-uid1 / Project

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

```
1 Project = t.TeleportProject("PNToy Lab")
2 Component = Project.root
3
```

```
1 Project.libraries['react'] = 'https://nanohub.org/js_apps/react.production.min'
2 Project.libraries['react-dom'] = 'https://nanohub.org/js_apps/react-dom.production.min'
3 Project.libraries['material-ui'] = 'https://nanohub.org/js_apps/material-ui.production.min'
4 Project.libraries['materiallab-ui'] = 'https://nanohub.org/js_apps/material-ui-lab.production.min'
5 Project.libraries['math'] = 'https://cdnjs.cloudflare.com/ajax/libs/mathjs/6.6.1/math.min'
6 Project.libraries['axios'] = 'https://nanohub.org/js_apps/axios.min'
7 Project.libraries['localforage'] = 'https://nanohub.org/js_apps/localforage.min'
8 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

```
1 ErrorMessage = SimtoolBuilder.Error(  
2     Component,  
3     error_status = STATE_ERROR_STATUS,  
4     error_open = STATE_ERROR_OPEN,  
5     is_open = False  
6 )
```

Creating ALERT Message

```
1 AlertMessage = SimtoolBuilder.Error(  
2     Component,  
3     error_status = STATE_ALERT_STATUS,  
4     error_open = STATE_ALERT_OPEN,  
5     is_open = False,  
6     title = "Message"  
7 )  
8 AlertMessage.content.children[0].content.style['backgroundColor'] = '#FFF380'
```

Creating loader

```
1 Loader = SimtoolBuilder.Loader(  
2     Component,  
3     loader_status = STATE_LOADER_STATUS,  
4     loader_open = STATE_LOADER_OPEN,  
5     is_open = False  
6 )
```

Other components

nanohub-uid1 / sim2l schema

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

```
1 SimtoolBuilder.buildSchema(  
2   Project,  
3   Component,  
4   url = "https://nanohub.org/api/dbexplorer/simtools",  
5   toolname = TOOLNAME,  
6   revision = REVISION  
7 )  
8 Component.addPropVariable(  
9   "onLoadSchema",  
10  {  
11    "type" : "func",  
12    'defaultValue' :  
13    '(e)=>{e.setState({"'+ STATE_LOADER_OPEN +'":false})}'  
14  }  
15 )
```

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)

```
1 from secrets import IDCLIENT, SECRET
2 auth_data['client_id'] = IDCLIENT
3 auth_data['client_secret'] = SECRET
4 # to get client_id and client_secret, create a web application (https://nanohub.org/develop)
5
6 Login, CLogin = Auth.Login(
7     Project,
8     Component,
9     client_id = auth_data['client_id'],
10    client_secret = auth_data['client_secret'],
11    url = "https://nanohub.org/api/developer/oauth/token",
12    open_state = STATE_LOGIN_OPEN
13 )
14 Login.content.events["onAuth"] = [
15     { "type": "stateChange", "modifies": STATE_LOGIN_OPEN, "newState": False},
16     { "type": "propCall2", "calls": "buildSchema", "args": ['self'] }
17 ]
```

```
1 Login, CLogin = Auth.Session(
2     Project,
3     Component,
4     sessiontoken = auth_data["sessiontoken"],
5     sessionnum = auth_data["sessionnum"],
6     url = "https://nanohub.org/api/developer/oauth/token",
7 )
8 Login.content.events["onError"]=[
9     { "type": "stateChange", "modifies": STATE_ERROR_OPEN, "newState": True},
10    { "type": "stateChange", "modifies": STATE_ERROR_STATUS, "newState": '$e'},
11 ]
12 Login.content.events["onAuth"] = [
13     { "type": "propCall2", "calls": "buildSchema", "args": ['self'] }
14 ]
15 ]
```

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

```
1 SETTINGS = {
2   'values': {
3     'type': 'String',
4     'default_value': 'Animal Jumps Merrily',
5     'label': 'text',
6     'description': 'lowercase string that is going to be encoded'
7   }, 'shift_input': {
8     'type': 'Integer',
9     'default_value': 24,
10    'units': None,
11    'min': 1,
12    'max': 50,
13    'label': 'Shifting',
14    'description': 'integer that determines the shift amount for encoding'
15  }, 'cores': {
16    'type': "Integer",
17    'default_value': 1,
18    'min' : 1,
19    'max' : 1,
20    'units' : "",
21    'description' : "Number of cores in in the venue",
22    'label' : "Number of cores",
23  }, "cutoff" : {
24
25
26
27
28
29
30
31
32
33
34
35
36
37 }
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```
1 SETTINGS = {}
2 for option, value in PARAMS.items():
3   if isinstance(value, nr.params.Number):
4     SETTINGS[option] = {
5       "type": "Number",
6       "default_value": value.default,
7       "units" : value.units,
8       "min" : value.min,
9       "max" : value.max,
10      "label" : value.label,
11      "description" : value.description,
12    }
13  elif isinstance(value, nr.params.Integer):
14    SETTINGS[option] = {
15      "type": "Integer",
16      "default_value": value.default,
17      "units" : value.units,
18      "min" : value.min,
19      "max" : value.max,
20      "label" : value.label,
21      "description" : value.description,
22    }
23  elif isinstance(value, nr.params.String):
24    SETTINGS[option] = {
25      "type": "String",
26      "default_value": value.default,
27      "label" : value.label,
28      "description" : value.description,
29    }
30  elif isinstance(value, nr.params.Choice):
31    SETTINGS[option] = {
32      "type": "Select",
33      "default_value": value.default,
34      "options": {k : k for k in value.options},
35      "units" : value.units,
36      "label" : value.label,
37      "description" : value.description,
38    }
39  elif isinstance(value, nr.params.Boolean):
40    SETTINGS[option] = {
41      "type": "Boolean",
42      "default_value": (value.default == "yes"),
43      "description" : value.description,
44      "label" : value.label,
45    }
46  |
```

nanohub-uid1 / 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-uid1 / Settings component

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

```
1 url_sim = "https://nanohub.org/api/dbexplorer/simtools"
2 AppSettings = AppBuilder.Settings(
3   Project,
4   Component,
5   SETTINGS,
6   url=url_sim,
7   toolname = TOOLNAME,
8   revision = REVISION,
9   layout = LAYOUT['input'],
10  outputs = ['Cipher', 'Repeated Chars', 'Repeated Chars Count'],
11  runSimulation = "simtool"
12 )
13 AppSettings.content.events["onError"]=[
14   { "type": "stateChange", "modifies": STATE_LOADER_OPEN, "newState": False},
15   { "type": "stateChange", "modifies": STATE_ERROR_OPEN, "newState": True},
16   { "type": "stateChange", "modifies": STATE_ERROR_STATUS, "newState": '$e'}
17 ]
18 AppSettings.content.events["click"]=[
19   { "type": "stateChange", "modifies": STATE_LOADER_OPEN, "newState": True}
20 ]
21
22 AppSettings.content.events["submit"] = [
23   { "type": "stateChange", "modifies": "parameters", "newState": '$e.target.value'}
24 ]
25
26 AppSettings.content.events["onStatusChange"]=[
27   { "type": "stateChange", "modifies": STATE_LOADER_STATUS, "newState": '$e.target.value'}
28 ]
```

Parameters ^

Periodic Potential Details

Type of periodic potential
 Step Well v

Periodic potential assumed in the infinite lattice.

Energy Details Well Geometry

Energy Details

Maximum Barrier Height(Vmax)
3 eV
Maximum height a barrier can have in the well in eV.

Minimum Barrier Height(Vmin)
0 eV
Minimum height a barrier can have in the well in eV.

Energy of particle
5 eV
Maximum energy carried by the particle over the barrier height.

Simulate

nanohub-uid1 / 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

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

```
1 AppSettings.content.events["onSuccess"]=[
2   { "type": "stateChange", "modifies": STATE_LOADER_OPEN, "newState": False },
3   { "type": "stateChange", "modifies": STATE_ERROR_OPEN, "newState": False},
4   { "type": "stateChange", "modifies": STATE_ERROR_STATUS, "newState": ''},
5   {
6     "type": "stateChange",
7     "modifies": STATE_ALERT_OPEN,
8     "newState": True,
9     "callbacks" : [
10      {
11        "type": "propCall2",
12        "calls": "loadBaseOutput",
13        "args": ['self', 'arguments[1]']
14      }
15    ]
16  }
17 ]
```

nanohub-uid1 / Getting Results / Plots

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

```
1 SimtoolBuilder.loadXY(  
2   Project,  
3   Component,  
4   #cache_store = TOOLNAME + "_" + str(REVISION)  
5 )  
6 SimtoolBuilder.loadSequence(  
7   Project,  
8   Component,  
9   #cache_store = TOOLNAME + "_" + str(REVISION)  
10 )  
11  
12 Project.root.addStateVariable("visualization", {"type":"object", "defaultValue": {  
13   'function':'loadSequence',  
14   'dataset' : ['Ec', 'Ev', 'Ei'],  
15   'layout' : {},  
16   'parameters' : {'position':'position','function':'function'}  
17 }}})  
18  
19 RESULTS = {  
20   "bands" : {  
21     'title' : 'Energy Band Diagram',  
22     'action' : { "type": "stateChange", "modifies": "visualization","newState": {  
23       'function':'loadSequence',  
24       'dataset' : ['Ec', 'Ev', 'Ei'],  
25       'layout' : {'title':'Energy Band Diagram', 'yaxis': { 'title' : 'Energy (eV)'  
26       'parameters' : {'position':'position','function':'function'},  
27       'normalize' : True,  
28       'start_trace' : 0  
29     },  
30     "callbacks" : onRefreshViews  
31   }  
32 },
```

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

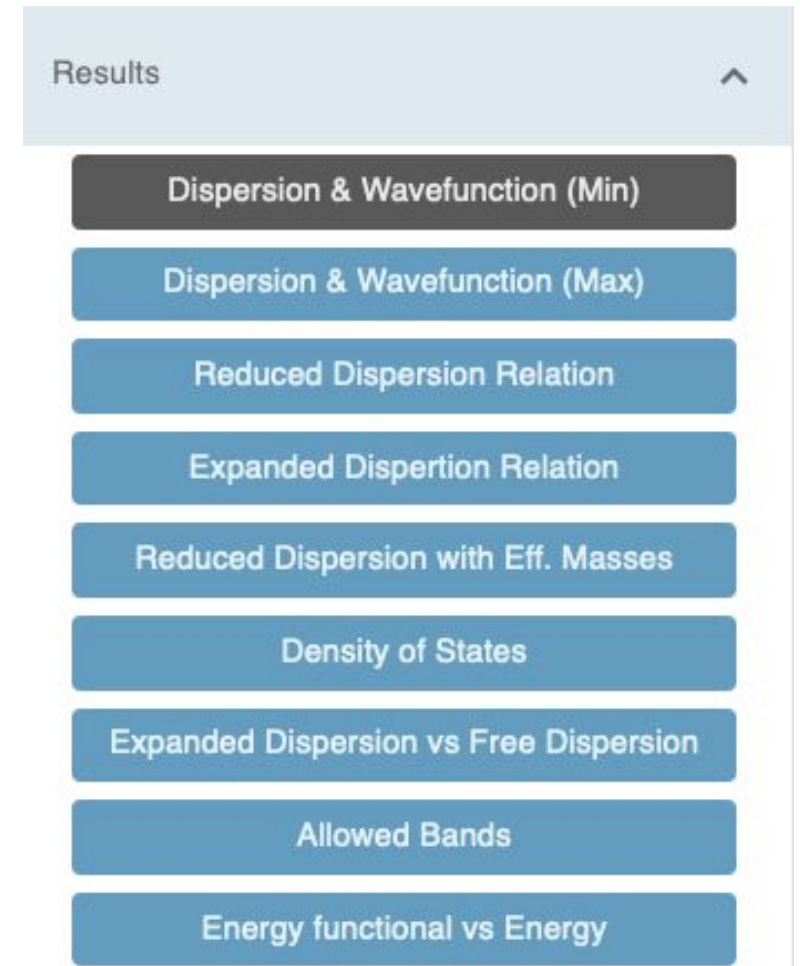
start_trace: sequence starts on the index of the trace

nanohub-uid1 / Getting Results / Plots

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

```
133     "carrier" : {
134         'title' : 'Excess Carrier Density',
135         'action' : {
136             "type": "stateChange",
137             "modifies": "visualization",
138             "newState": {
139                 'function':'loadSequence',
140                 'dataset' : ['Excess Electron Density', 'Excess Hole Density'],
141                 'layout' : {'title':'Excess Carrier Density', 'yaxis': { 'type' : 'log',
142                 'parameters' : {'position':'position','function':'function'},
143                 'normalize' : True,
144                 'start_trace' : 1
145             },
146             "callbacks" : onRefreshViews
147         },
148     },
149 }
150
151
```

```
1 PNT0YResults = AppBuilder.Results(
2     Component,
3     results = RESULTS,
4     onClick = [{ "type": "stateChange", "modifies": STATE_LOADER_OPEN,"newState": True }]
5     onLoad = [
6         { "type": "stateChange", "modifies": STATE_LOADER_OPEN,"newState": False }
7     ],
8 )
```



nanohub-uid1 / Assembling the app

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

```
1 ThemeProvider = MaterialBuilder.ThemeProvider( Component, MaterialBuilder.DefaultTheme(  
2   primary_color = '#699FBB',  
3   secondary_color = '#f1f1f1',  
4   primary_bg = '#FFFFFF',  
5   secondary_bg = '#d9eaf0',  
6   default_button = 'rgba(255, 255, 255, 0.87)',  
7   primary_button = 'rgba(255, 255, 255, 0.87)',  
8   secondary_button = 'rgba(0, 0, 0, 0.87)',  
9   default_button_bg = 'rgb(63, 162, 192)',  
10  primary_button_bg = 'rgba(0, 0, 0, 0.65)',  
11  secondary_button_bg = 'rgba(0, 0, 0, 0.12)',  
12 ))  
13 AppBar = MaterialBuilder.AppBar(  
14   title="Caesar Cipher Tool"  
15 )  
16 logo = t.TeleportElement(t.TeleportContent(element  
17 logo.content.attrs["width"] = "120"  
18 logo.content.attrs["src"] = "https://nanohub.org/  
19 AppBar.content.children[0].addContent(logo)  
20
```

```
1 Gridv = t.TeleportElement(MaterialContent(elementType="Grid"))  
2 Gridv.content.attrs["container"] = True  
3 Gridv.content.attrs["direction"] = "column"  
4 Gridv.addContent(AppBar)  
5 Gridv.addContent(AppSettings)  
6  
7 ThemeProvider.addContent(Gridv)  
8 ThemeProvider.addContent(Loader)  
9 ThemeProvider.addContent(ErrorMessage)  
10 ThemeProvider.addContent(AlertMessage)  
11 ThemeProvider.addContent(Login)  
12  
13 Component.addNode(ThemeProvider)  
14 Project.buildReact( TOOLNAME+"PROD"+"html");
```

nanohub-uid1 / 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

```
1 ThemeProvider = MaterialBuilder.ThemeProvider( Component, MaterialBuilder.DefaultTheme(  
2   primary_color = '#699FBB',  
3   secondary_color = '#f1f1f1',  
4   primary_bg = '#FFFFFF',  
5   secondary_bg = '#dbeaf0',  
6   default_button = 'rgba(255, 255, 255, 0.87)',  
7   primary_button = 'rgba(255, 255, 255, 0.87)',  
8   secondary_button = 'rgba(0, 0, 0, 0.87)',  
9   default_button_bg = 'rgb(63, 162, 192)',  
10  primary_button_bg = 'rgba(0, 0, 0, 0.65)',  
11  secondary_button_bg = 'rgba(0, 0, 0, 0.12)',  
12 ))  
13 AppBar = MaterialBuilder.AppBar(  
14   title="Caesar Cipher Tool"  
15 )  
16 logo = t.TeleportElement(t.TeleportContent(elementT  
17 logo.content.attrs["width"] = "120"  
18 logo.content.attrs["src"] = "https://nanohub.org/ap  
19 AppBar.content.children[0].addContent(logo)  
20
```

```
1 Gridv = t.TeleportElement(MaterialContent(elementType="Grid"))  
2 Gridv.content.attrs["container"] = True  
3 Gridv.content.attrs["direction"] = "column"  
4 Gridv.addContent(AppBar)  
5 Gridv.addContent(AppSettings)  
6  
7 ThemeProvider.addContent(Gridv)  
8 ThemeProvider.addContent(Loader)  
9 ThemeProvider.addContent(ErrorMessage)  
10 ThemeProvider.addContent(AlertMessage)  
11 ThemeProvider.addContent(Login)  
12  
13 Component.addNode(ThemeProvider)  
14 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