The Journal of Open Source Software

xml2jupyter: Mapping parameters between XML and Jupyter widgets

Randy Heiland[1], **Daniel Mishler**[1], **Tyler Zhang**[1], **Eric Bower**[1], **and Paul Macklin**[1]

1 Intelligent Systems Engineering, Indiana University

DOI: 10.21105/joss.01408

Software

- Review
- Repository
- Archive

Submitted: 08 April 2019
Published: 01 July 2019

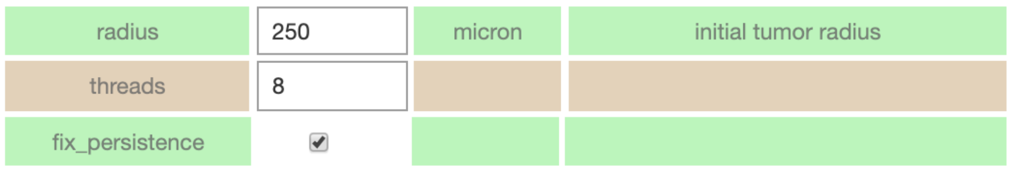
Summary

Jupyter Notebooks (Kluyver et al., 2016, Perkel (2018)) provide executable documents (in a variety of programming languages) that can be run in a web browser. When a notebook contains graphical widgets, it becomes an easy-to-use graphical user interface (GUI). Many scientific simulation packages use text-based configuration files to provide parameter values and run at the command line without a graphical interface. Manually editing these files to explore how different values affect a simulation can be burdensome for technical users, and impossible to use for those with other scientific backgrounds. xml2jupyter is a Python package that addresses these scientific bottlenecks. It provides a mapping between configuration files, formatted in the Extensible Markup Language (XML), and Jupyter widgets. Widgets are automatically generated from the XML file and these can, optionally, be incorporated into a larger GUI for a simulation package, and optionally hosted on cloud resources. Users modify parameter values via the widgets, and the values are written to the XML configuration file which is input to the simulation's command-line interface. xml2jupyter has been tested using PhysiCell (Ghaffarizadeh, Heiland, Friedman, Mumenthaler, & Macklin, 2018), an open source, agent-based simulator for biology, and it is being used by students for classroom and research projects. In addition, we use xml2jupyter to help create Jupyter GUIs for PhysiCell-related applications running on nanoHUB (Madhavan et al., 2013).

A PhysiCell configuration file defines model-specific `<user_parameters>` in XML. Each parameter element consists of its name with attributes, defining its data *type*, *units* (optional), *description* (optional), whether the widget should be *hidden* (optional), and the parameter's default value. The attributes will determine the appearance and behavior of the Jupyter widget. For numeric widgets (the most common type for PhysiCell), xml2jupyter will calculate a delta step size as a function of the default value and this step size will be used by the widget's graphical increment/decrement feature.

To illustrate, we show the following simple XML example, containing each of the four (currently) supported data types and the various attributes:

```
<PhysiCell_settings>
  <user_parameters>
    <radius type="double" units="micron"
        description="initial tumor radius">250.0
    </radius>
    <threads type="int">8</threads>
    <color type="string" hidden="true">red</color>
    <fix_persistence type="bool">True</fix_persistence>
  </user_parameters>
</PhysiCell_settings>
```



radius	250	micron	initial tumor radius
threads	8		
fix_persistence	✓		

Figure 1: Simple example of XML parameters as Jupyter widgets.

When we map this into Jupyter widgets, we obtain the rendered results in Figure 1. Notice the `color` parameter is not displayed since we specified it should be hidden in the XML. The name of the other parameters, their values, and attributes, if present, are displayed in rows (as disabled Jupyter button widgets). Using alternating row colors ("zebra stripes") helps visually match associated fields and avoid changing the wrong parameter value. For numeric widgets (type "int" or "double"), we compute a delta step value based on the magnitude (log) of the initial value. For example, the `radius` widget will have a step value of 10, whereas `threads` will have a step value of 1.

For a more realistic example, consider the `config_biorobots.xml` configuration file (found in the `config_samples` directory). The XML elements in the `<user_parameters>` block include the (optional) *description* attribute which briefly describes the parameter and is displayed in another widget. To demonstrate xml2jupyter on this XML file, one would: 1) clone or download the repository, 2) copy the XML configuration file to the root directory, and 3) run the `xml2jupyter.py` script, providing the XML file as an argument.

```
$ cp config_samples/config_biorobots.xml .
$ python xml2jupyter.py config_biorobots.xml
```

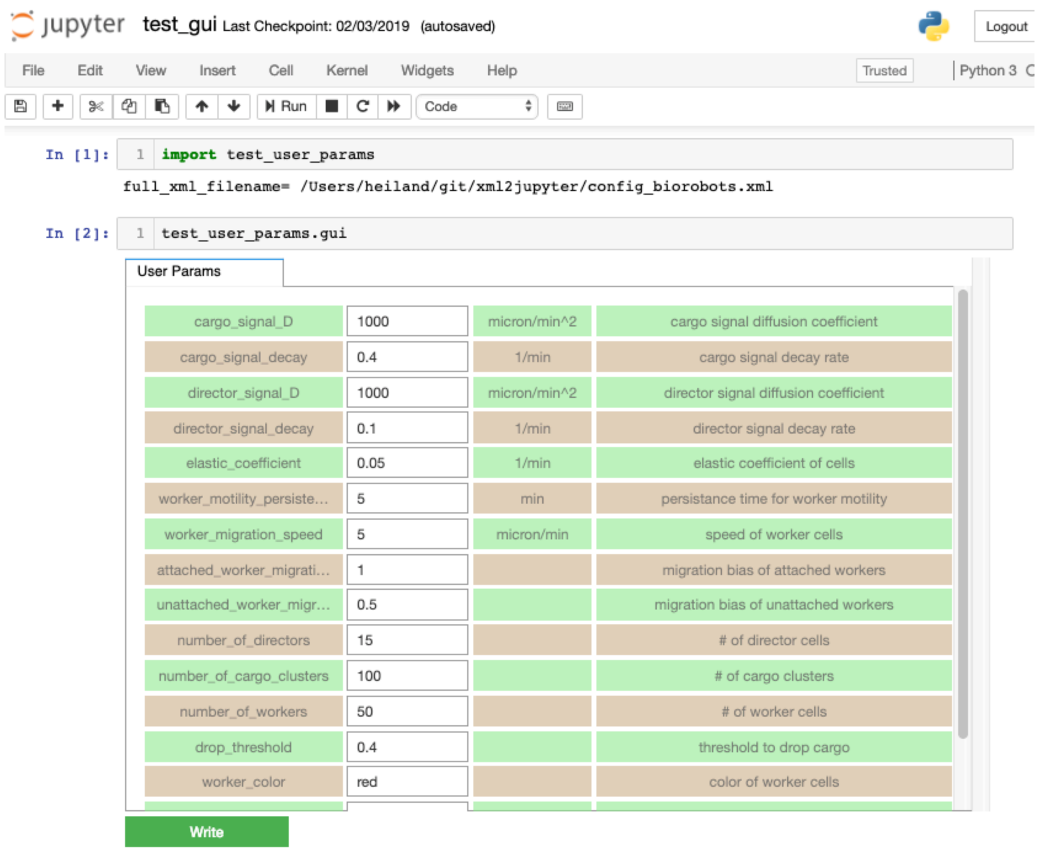
The `xml2jupyter.py` script parses the XML and generates a Python module, `user_params.py`, containing the Jupyter widgets, together with methods to populate their values from the XML and write their values back to the XML. To "validate" the widgets were generated correctly, one could, minimally, open `user_params.py` in an editor and inspect it. But to actually see the widgets rendered in a notebook, we provide a simple test:

```
$ python xml2jupyter.py config_biorobots.xml test_user_params.py
$ jupyter notebook test_gui.ipynb
```

This should display a minimal notebook in your browser and, after selecting `Run all` in the `Cell` menu, you should see the notebook shown in Figure 2.

PhysiCell Jupyter GUI

Our ultimate goal is to generate a fully functional GUI for PhysiCell users. xml2jupyter provides one important piece of this - dynamically generating widgets for custom user parameters for a model. By adding additional Python modules to provide additional components (tabs) of the GUI that are common to all PhysiCell models, a user can configure, run, and visualize output from a simulation. Two tabs that provide visualization of output files are shown in Figure 3 with results from the *biorobots* simulation. Note that some of the required modules are not available in the Python standard library, e.g., Matplotlib (Hunter, 2007) and SciPy (Oliphant, 2007). We provide instructions for installing these additional dependencies in the repository README.


jupyter test_gui Last Checkpoint: 02/03/2019 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
User Params
Write

```
In [1]: import test_user_params
full_xml_filename= /Users/heiland/git/xml2jupyter/config_biorobots.xml

In [2]: test_user_params.gui
```

cargo_signal_D	1000	micron/min^2	cargo signal diffusion coefficient
cargo_signal_decay	0.4	1/min	cargo signal decay rate
director_signal_D	1000	micron/min^2	director signal diffusion coefficient
director_signal_decay	0.1	1/min	director signal decay rate
elastic_coefficient	0.05	1/min	elastic coefficient of cells
worker_motility_persiste…	5	min	persistance time for worker motility
worker_migration_speed	5	micron/min	speed of worker cells
attached_worker_migrati…	1		migration bias of attached workers
unattached_worker_migr…	0.5		migration bias of unattached workers
number_of_directors	15		# of director cells
number_of_cargo_clusters	100		# of cargo clusters
number_of_workers	50		# of worker cells
drop_threshold	0.4		threshold to drop cargo
worker_color	red		color of worker cells

Figure 2: The biorobots parameters rendered as Jupyter widgets.

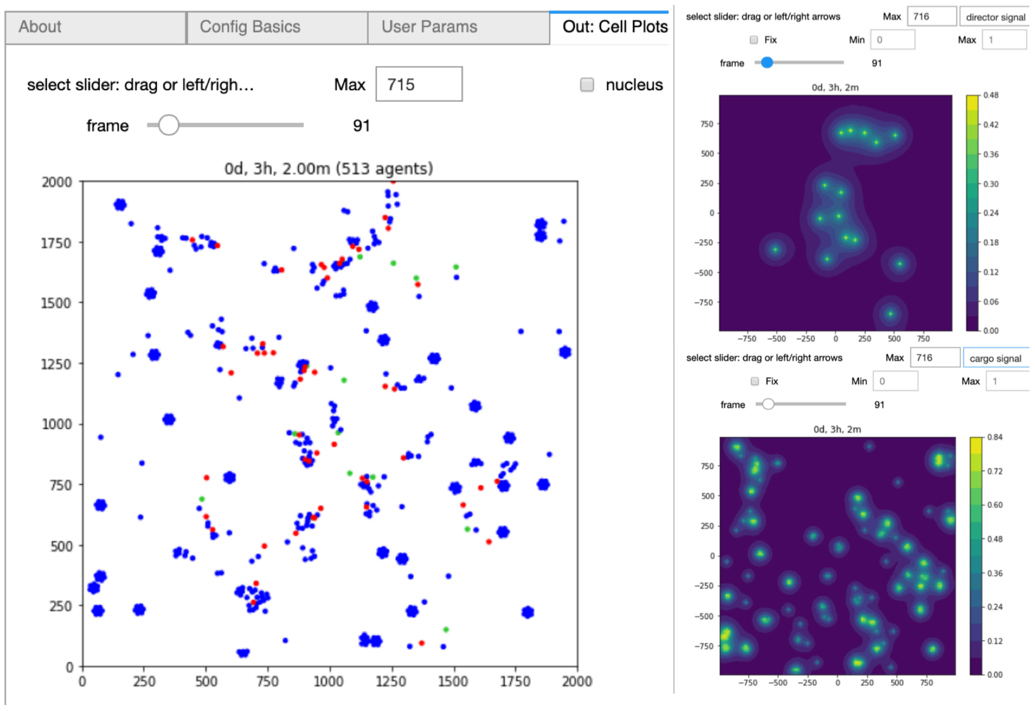
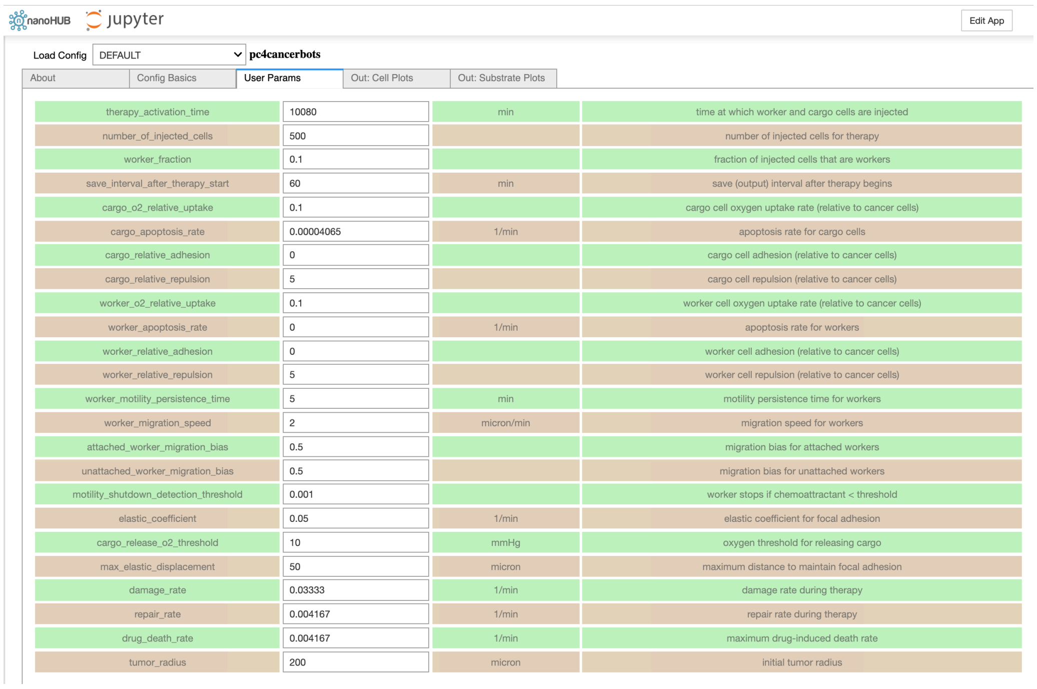


Figure 3: Plotting the biorobots (cells; left) and signals (substrates; right).



Load Config DEFAULT pc4cancerbots

therapy_activation_time	10080	min	time at which worker and cargo cells are injected
number_of_injected_cells	500		number of injected cells for therapy
worker_fraction	0.1		fraction of injected cells that are workers
save_interval_after_therapy_start	60	min	save (output) interval after therapy begins
cargo_o2_relative_uptake	0.1		cargo cell oxygen uptake rate (relative to cancer cells)
cargo_apoptosis_rate	0.00004065	1/min	apoptosis rate for cargo cells
cargo_relative_adhesion	0		cargo cell adhesion (relative to cancer cells)
cargo_relative_repulsion	5		cargo cell repulsion (relative to cancer cells)
worker_o2_relative_uptake	0.1		worker cell oxygen uptake rate (relative to cancer cells)
worker_apoptosis_rate	0	1/min	apoptosis rate for workers
worker_relative_adhesion	0		worker cell adhesion (relative to cancer cells)
worker_relative_repulsion	5		worker cell repulsion (relative to cancer cells)
worker_motility_persistence_time	5	min	motility persistence time for workers
worker_migration_speed	2	micron/min	migration speed for workers
attached_worker_migration_bias	0.5		migration bias for attached workers
unattached_worker_migration_bias	0.5		migration bias for unattached workers
motility_shutdown_detection_threshold	0.001		worker stops if chemoattractant < threshold
elastic_coefficient	0.05	1/min	elastic coefficient for focal adhesion
cargo_release_o2_threshold	10	mmHg	oxygen threshold for releasing cargo
max_elastic_displacement	50	micron	maximum distance to maintain focal adhesion
damage_rate	0.03333	1/min	damage rate during therapy
repair_rate	0.004167	1/min	repair rate during therapy
drug_death_rate	0.004167	1/min	maximum drug-induced death rate
tumor_radius	200	micron	initial tumor radius

Figure 4: The cancer biorobots parameters in a nanoHUB Jupyter application.

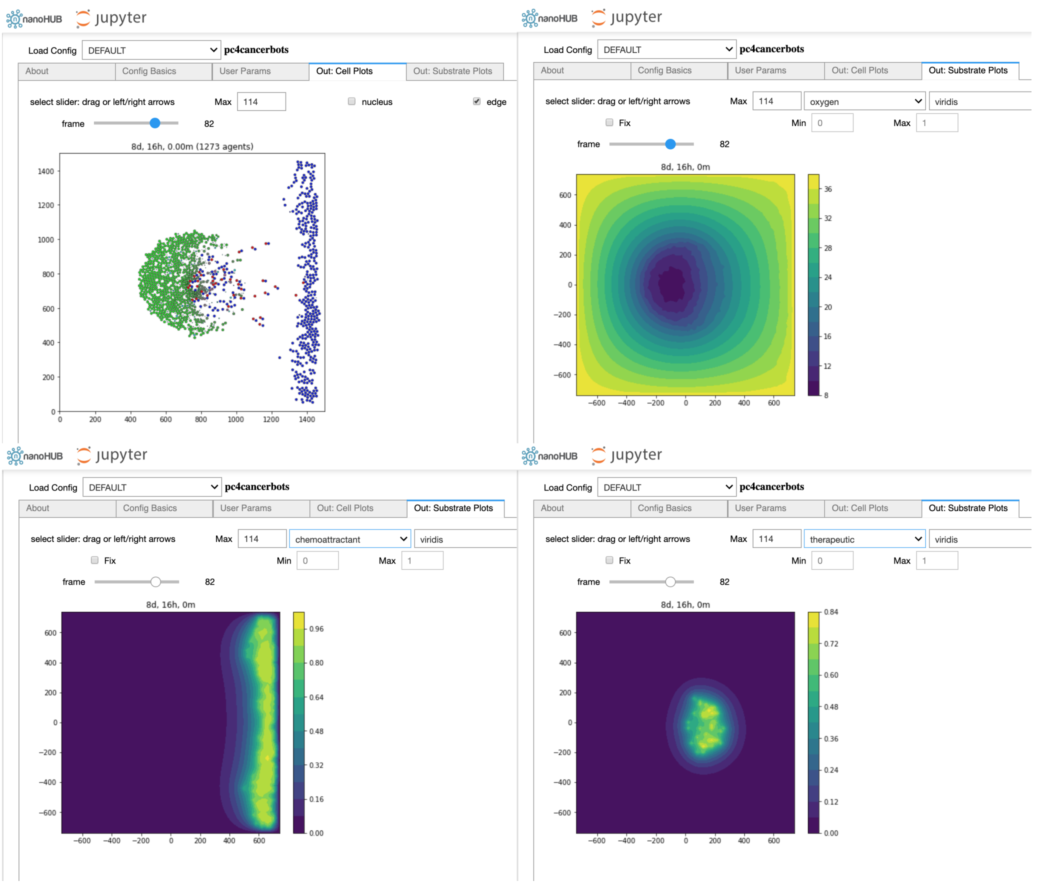
Extensions and Discussion

We hope others will be inspired to extend the core idea of this project to other text-based configuration files. XML is only one of several data-interchange formats and, although we created this tool for XML-based configurations based on needs to create GUIs for PhysiCell projects, the approach should be more broadly applicable to these other formats. And while the additional Python modules that provide visualization are also tailored to PhysiCell output, they can serve as templates for other scientific applications whose input and output file formats provide similar functionality.

xml2jupyter has helped us port PhysiCell-related Jupyter tools to nanoHUB, a scientific cloud for nanoscience education and research that includes running interactive simulations in a browser. For example, Figure 4 shows the xml2jupyter-generated *User Params* tab in our `pc4cancerbots` tool running on nanoHUB. Figure 5 shows the cells (upper-left) and three different substrate plots for this same tool. This particular model and simulation is described in this video.

Other PhysiCell-related nanoHUB tools that have been created using xml2jupyter include `pc4heterogen`, `pcISA`, and `pc4cancerimmune`. Readers can create a free account on nanoHUB and run these simulations for themselves. We encourage students to use xml2jupyter to create their own nanoHUB tools of PhysiCell models that 1) can be run and evaluated by the instructor, 2) can be shared with others, and 3) become part of a student's living portfolio. (Another repository, https://github.com/rheiland/tool4nanobio, provides instructions and scripts to help generate a full GUI from an existing PhysiCell model.)

We welcome suggestions and contributions to xml2jupyter. For example, currently, we arrange the generated parameter widgets vertically, one row per parameter. This is an appropriate layout for an educational setting. But if a GUI will be used by researchers who are already familiar with the parameters, it may be preferable to generate a more compact layout of widgets, e.g., in a matrix with only the parameter names and values. Moreover, it may be useful to provide additional control over styling and placement by a separate style.xml or



Load Config DEFAULT pc4cancerbots

About | Config Basics | User Params | Out: Cell Plots | Out: Substrate Plots

select slider: drag or left/right arrows Max 114 ☐ nucleus ☑ edge

frame 82

Load Config DEFAULT pc4cancerbots

About | Config Basics | User Params | Out: Cell Plots | Out: Substrate Plots

select slider: drag or left/right arrows Max 114 oxygen viridis

☐ Fix Min 0 Max 1

frame 82

Load Config DEFAULT pc4cancerbots

About | Config Basics | User Params | Out: Cell Plots | Out: Substrate Plots

select slider: drag or left/right arrows Max 114 chemoattractant viridis

☐ Fix Min 0 Max 1

frame 82

Load Config DEFAULT pc4cancerbots

About | Config Basics | User Params | Out: Cell Plots | Out: Substrate Plots

select slider: drag or left/right arrows Max 114 therapeutic viridis

☐ Fix Min 0 Max 1

frame 82

Figure 5: The cancer biorobots Jupyter notebook on nanoHUB.

similar file, or by an external cascading style sheet. We will explore these options in the future, with the aim of separating as much GUI specification and styling from the original scientific application as possible. Such a decoupling would make it easier for scientific developers to continue refining their scientific codes without worrying about impact on the GUI, and without undue encumbrance by non-scientific annotations.

Also, we currently provide just 2-D visualizations of (spatial) data. In the near future, we will provide visualizations of 3-D models and welcome suggestions from the community.

Acknowledgements

We thank the National Science Foundation (1720625) and the National Cancer Institute (U01-CA232137-01) for generous support. Undergraduate and graduate students in the Intelligent Systems Engineering department at Indiana University provided internal testing, and students and researchers within the NSF nanoMFG (1720701) group generously provided external testing. All of their feedback resulted in considerable improvements to this project. Finally, we thank our collaborators at Purdue University, especially Martin Hunt and Steve Clark, who provided technical support with nanoHUB and Jupyter.

References

Ghaffarizadeh, A., Heiland, R., Friedman, S. H., Mumenthaler, S. M., & Macklin, P. (2018). PhysiCell: An open source physics-based cell simulator for 3-d multicellular systems. *PLOS Computational Biology*, *14*(2), 1–31. doi:10.1371/journal.pcbi.1005991

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., et al. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. (F. Loizides & B. Schmidt, Eds.). IOS Press. doi:10.3233/978-1-61499-649-1-87

Madhavan, K., Zentner, L., Farnsworth, V., Shivarajapura, S., Zentner, M., Denny, N., & Klimeck, G. (2013). NanoHUB.org: Cloud-based services for nanoscale modeling, simulation, and education. *Nanotechnology Reviews*, *2*(1). doi:10.1515/ntrev-2012-0043

Oliphant, T. E. (2007). Python for scientific computing. *Computing In Science & Engineering*, *9*(3), 10–20. doi:10.1109/MCSE.2007.58

Perkel, J. M. (2018). Why jupyter is data scientists' computational notebook of choice. *Nature*, *563*, 145–146. doi:10.1038/d41586-018-07196-1