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: 00.00000/joss.00000

Software

- Review
- Repository
- Archive

Submitted: 31 January 2019
Published: 01 March 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. Manually editing these files, to explore how different values affect a simulation, can be burdensome. xml2jupyter is a Python package that bridges this gap. 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. 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 et al. 2018), an open source, agent-based simulator for biology, and 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 allowed data types (currently) 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>
```

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 Journal of Open Source Software

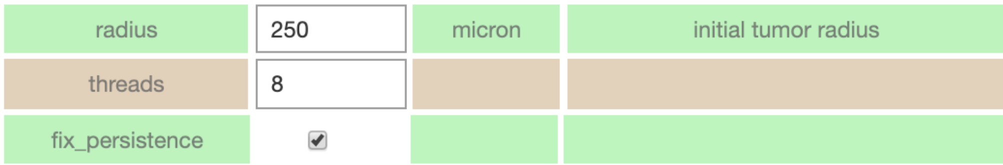

Figure 1: Simple example of XML parameters as Jupyter widgets.

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 a 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. With other Python modules that provide additional components (tabs) of the GUI, 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 below 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. We provide instructions for installing these additional dependencies in the repository README.

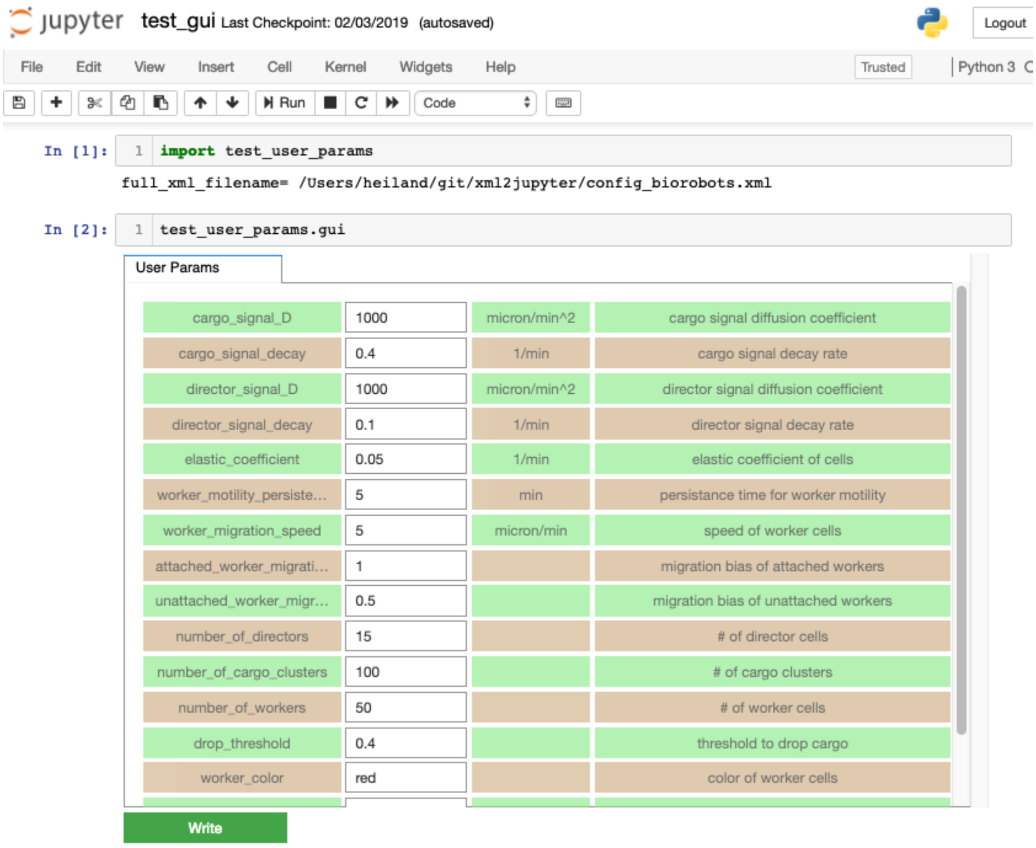

Figure 2: The biorobots parameters rendered as Jupyter widgets.

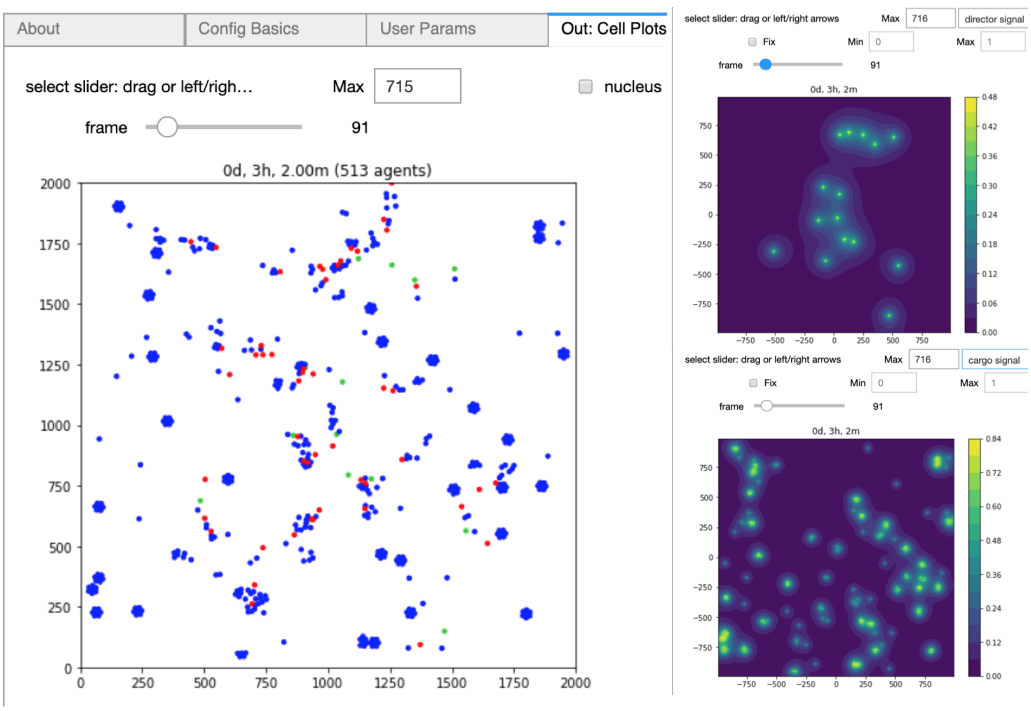

Figure 3: Plotting the biorobots (cells; left) and signals (substrates; right).

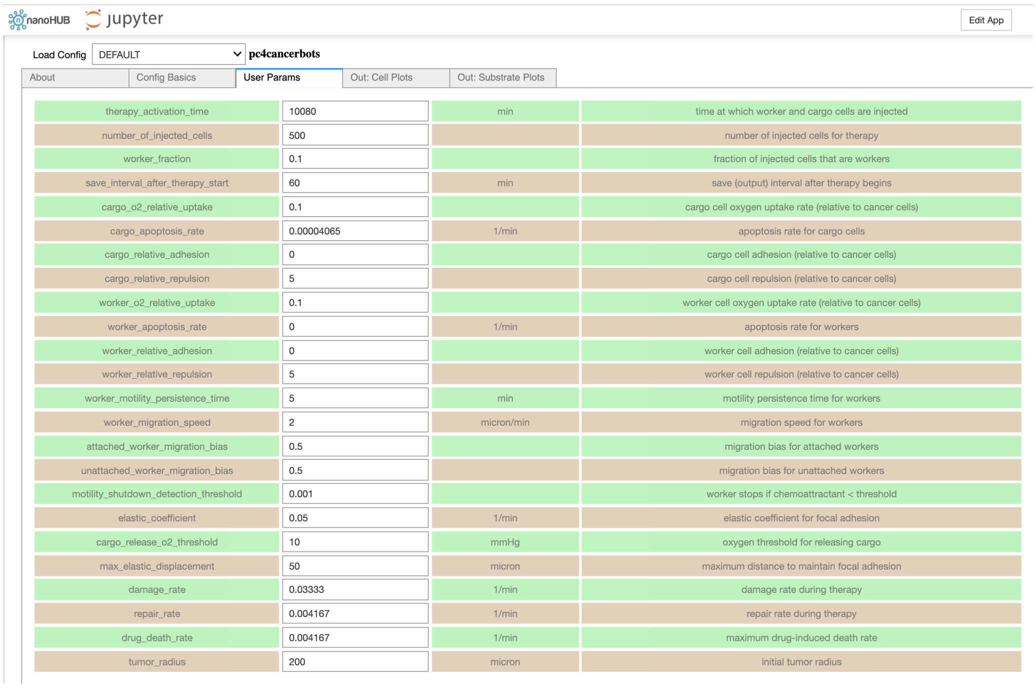

Figure 4: The cancer biorobots parameters in a nanoHUB Jupyter application.

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. It just happens to be the one of interest to us for PhysiCell. And while the additional Python modules that provide visualization are also tailored to PhysiCell output, they can serve as templates for other file formats and 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 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 an 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 can 1) be run and evaluated by the instructor, 2) 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.

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.

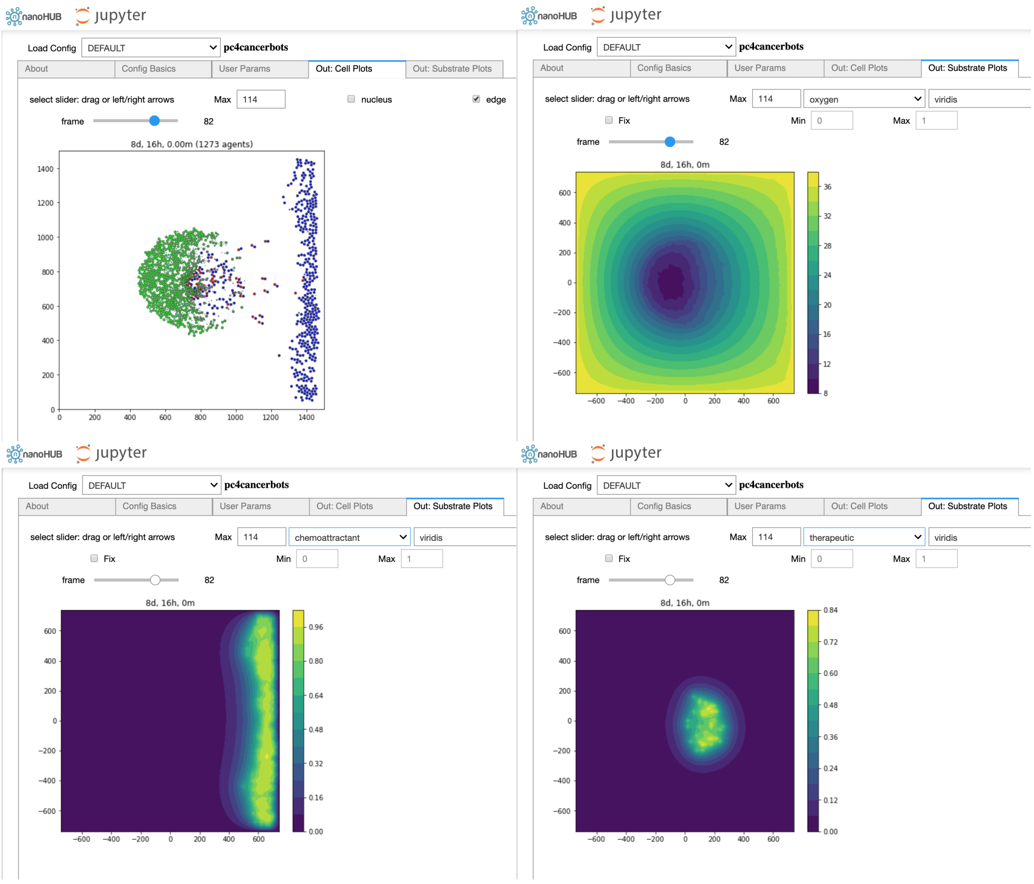

Figure 5: The cancer biorobots Jupyter notebook on nanoHUB.

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 depardment 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, Ahmadreza, Randy Heiland, Samuel H. Friedman, Shannon M. Mumenthaler, and Paul Macklin. 2018. "PhysiCell: An Open Source Physics-Based Cell Simulator for 3-d Multicellular Systems." *PLOS Computational Biology* 14 (2). Public Library of Science: 1–31. doi:10.1371/journal.pcbi.1005991.

Hunter, J. D. 2007. "Matplotlib: A 2d Graphics Environment." *Computing in Science & Engineering* 9 (3). IEEE COMPUTER SOC: 90–95. doi:10.1109/MCSE.2007.55.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, et al. 2016. "Jupyter Notebooks – a Publishing Format for Reproducible Computational Workflows." Edited by F. Loizides and B. Schmidt. IOS Press.

Madhavan, Krishna, Lynn Zentner, Victoria Farnsworth, Swaroop Shivarajapura, Michael Zentner, Nathan Denny, and Gerhard Klimeck. 2013. "NanoHUB.org: Cloud-Based Services for Nanoscale Modeling, Simulation, and Education." *Nanotechnology Reviews* 2 (1). doi:10.1515/ntrev-2012-0043.

Perkel, Jeffrey M. 2018. "Why Jupyter Is Data Scientists' Computational Notebook of Choice." *Nature* 563: 145–46. doi:10.1038/d41586-018-07196-1.