“How To” Mesa Packages

The Mesa core functionality is just a subset of what we believe researchers creating Agent Based Models (ABMs) will use. We designed Mesa to be extensible, so that individuals from various domains can build, maintain, and share their own packages that work with Mesa in pursuit of “unifying algorithmic theories of the relation between adaptive behavior and system complexity (Volker Grimm et al 2005).”

DRY Principle

This decoupling of code to create building blocks is a best practice in software engineering. Specifically, it exercises the DRY principle (or don’t repeat yourself) (Hunt and Thomas 2010). The creators of Mesa designed Mesa in order for this principle to be exercised in the development of agent-based models (ABMs). For example, a group of health experts may create a library of human interactions on top of core Mesa. That library then is used by other health experts. So, those health experts don’t have to rewrite the same basic behaviors.

Benefits to Scientists

Besides a best practice of the software engineering community, there are other benefits for the scientific community.

1. **Reproducibility and Replicability.** Decoupled shared packages also allows for reproducibility and replicability. Having a package that is shared allows others to reproduce the model results. It also allows others to apply the model to similar phenomenon and replicate the results over a diversity of data. Both are essential part of the scientific method (Leek and Peng 2015).
2. **Accepted truths.** Once results are reproduced and replicated, a library could be considered an accepted truth, meaning that the community agrees the library does what the library intends to do and the library can be trusted to do this. Part of the idea behind ‘accepted truths’ is that subject matter experts are the ones that write and maintain the library.
3. **Building blocks.** Think of libraries like Legos. The researcher can borrow a piece from here or there to pull together the base of their model, so they can focus on the value add that they bring. For example, someone might pull from a human interactions library and a decision-making library and combine the two to look at how human cognitive function effects the physical spread of disease.

Mesa and Mesa Packages

Because of the possibilities of nuanced libraries, few things will actually make it into core Mesa. Mesa is intended to only include core functionality that everyone uses. However, it is not impossible that something written on the outside is brought into core at a later date if the value to everyone is proven through adoption.

An example that is analogous to Mesa and Mesa packages is Django and Django Packages. Django is a web framework that allows you to build a website in Python, but there are lots of things besides a basic website that you might want. For example, you might want authentication functionality. It would be inefficient for everyone to write their own authentication functionality, so one person writes it (or a group of people). They share it with the world and then many people can use it.

This process isn’t perfect. Just because you write something doesn’t mean people are going to use it. Sometimes two different packages will be created that do similar things, but one of them does it better or is easier to use. That is the one that will get more adoption. In the world of academia, often researchers hold on to their content until they are ready to publish it. In the world of open source software, this can backfire. The sooner you open source something the more likely it will be a success, because you will build consensus and engagement. Another thing that can happen is that while you are working on perfecting it, someone else is building in the open and establishes the audience you were looking for. So, don’t be afraid to start working directly out in the open and then release it to the world.

What is in this doc

There are two sections in this documentation. The first is the User Guide, which is aimed at users of packages. The section is a package development guide, which is aimed at those who want to develop packages. Without further ado, let’s get started!

User Guide

Note: MESA does not endorse or verify any of the code shared through MESA packages. This is left to the domain experts of the community that created the code.*

Step 1: Select a package

Currently, a central list of compatible packages is located on the Mesa Wiki Packages Page.

Step 2: Establish an environment

Create a virtual environment for the ABM you are building. The purpose of a virtual environment

is to isolate the packages for your project from other projects. This is helpful when you need to use two different versions of a package or if you are running one version in production but want to test out another version. You can do with either virtualenv or Anaconda.

- Why a virtual environment
- Virtualenv and Virtualenv Wrapper
- Creating a virtual environment with Anaconda

Step 3: Install the packages

Install the package(s) into your environment via pip/conda or GitHub. If the package is a mature package that is hosted in the Python package repository, then you can install it just like you did Mesa:

```
pip install package_name
```

However, sometimes it takes a little bit for projects to reach that level of maturity. In that case to use the library, you would install from GitHub (or other code repository) with something like the following:

```
pip install https://github.com/<path to project>
```

The commands above should also work with Anaconda, just replace the *pip* with *conda*.

Package Development: A “How-to Guide”

The purpose of this section is help you understand, setup, and distribute your Mesa package as quickly as possible. A Mesa package is just a Python package or repo. We just call it a Mesa package, because we are talking about a Python package in the context of Mesa. These instructions assume that you are a little familiar with development, but that you have little knowledge of the packaging process.

There are two ways to share a package:

1. Via GitHub or other service (e.g. GitLab, Bitbucket, etc.)
2. Via PyPI, the Python package manager

Sharing a package via PyPI make it easier to install for users but is more overhead for whomever is maintaining it. However, if you are truly intending for a wider/longer-term adoption, then PyPI should be your goal.

Most likely you created an ABM that has the code that you want to share in it, which is what the steps below describe.

Sharing your package

1. Layout a new file structure to move the code into and then make sure it is callable from Mesa, in a simple, easy to understand way. For example, `from example_package import foo`. See Creating the Scaffolding.
2. Pick a name.
3. Create a repo on GitHub.
    - Enter the name of the repo.
    - Select a license (not sure– click the blue ‘i’ next to the i for a great run down of licenses). We recommend something permissive Apache 2.0, BSD, or MIT so that others can freely adopt it. The more permissive the more likely it will gain followers and adoption. If you do not include a license, it is our belief that you will retain all rights, which means that people can’t use your project, but it should be noted that we are also not lawyers.
    - Create a readme.md file (this contains a description of the package) see an example: Bilateral Shapley
4. Clone the repo to your computer.
5. Copy your code directory into the repo that you cloned one your computer.
6. Add a requirements.txt file, which lets people know which external Python packages are needed to run the code in your repo. To create a file, run: `pip freeze > requirements.txt`. Note, if you are running Anaconda, you will need to install pip first: `conda install pip`.
7. `git add` all the files to the repo, which means the repo starts to track the files. Then `git commit` the files with a meaningful message. To learn more about this see: Saving changes. Finally, you will want to `git push` all your changes to GitHub, see: Git Push.
8. Let people know about your package on the MESA Wiki Page and share it on the email list. In the future, we will create more of a directory, but at this point we are not there yet.

From this point, someone can clone your repo and then add your repo to their Python path and use it in their project. However, if you want to take your package to the next level, you will want to add more structure to your package and share it on PyPI.

Next Level: PyPI

You want to do even more. The authoritative guide for python package development is through

the Python Packaging User Guide. This will take you through the entire process necessary for getting your package on the Python Package Index.

The Python Package Index is the main repository of software for Python Packages and following this guide will ensure your code and documentation meets the standards for distribution across the Python community.

References

Grimm, Volker, Eloy Revilla, Uta Berger, Florian Jeltsch, Wolf M. Mooij, Steven F. Railsback, Hans-Hermann Thulke, Jacob Weiner, Thorsten Wiegand, and Donald L. DeAngelis. 2005. “Pattern-Oriented Modeling of Agent Based Complex Systems: Lessons from Ecology.” American Association for the Advancement of Science 310 (5750): 987–91. doi:10.1126/science.1116681.

Hunt, Andrew, and David Thomas. 2010. The Pragmatic Programmer: From Journeyman to Master. Reading, Massachusetts: Addison-Wesley.

Leek, Jeffrey T., and Roger D. Peng. 2015. “Reproducible Research Can Still Be Wrong: Adopting a Prevention Approach.” Proceedings of the National Academy of Sciences 112 (6): 1645–46. doi:10.1073/pnas.1421412111.