expfactory Documentation

Release 0.0.1

Poldracklab, Stanford University

Aug 06, 2020

Contents

The Experiment Factory is an effort to standardize, organize, and collaboratively develop experiments that can be run on multiple infrastructures. Currently supported is generating experiments in a battery intended to run with psiturk. on Amazon Mechanical Turk or in a standard web browser. These repos store, document, and deploy these experiments, and github is used to allow for open contribution from the larger community. expfactory-python is the controller for this infrastructure, and this documentation will guide usage of the tools:

Contents:

CHAPTER 1

Installation

Installation of the Experiment Factory Software requires some familiarity with the command line, as the software is based in python. "Pip" is a package management system for python, and you can use pip to install our software, expfactory-python, which is the controller of all pieces of this infrastructure. If you want to install the development version, you can install directly from Github with this command:

```
pip install git+git://github.com/expfactory/expfactory-python.git
```

Or to install the current release, use this command:

```
pip install expfactory
```

Please note that expfactory was developed with python 2.7 and has not been tested on other versions. Installation will place an executable, *expfactory* in your bin folder. You should be able to type *which expfactory* and see the location of your application. If you cannot, you likely installed the application locally, and the executable was placed in a folder not on your path. You can either try installation with sudo, or look at the output of the installation to find the path that it was installed to:

```
Installing expfactory script to /usr/local/bin
```

You can also use the module as a standard python module, meaning importing functions into scripts, for example:

```
from expfactory.battery import generate
```

1.1 Quickstart

1.1.1 Running Experiments

Deploy a battery of experiments, a game, or a survey on your local machine

```
expfactory --run --experiments stroop,nback
expfactory --run --survey bis11_survey
expfactory --run --game bridge_game
```

1.1.2 Developing Experiments

To run (or preview) a folder (experiment) you are working on, you can cd into that folder, and use the *–run* command:

```
cd my_experiment
expfactory --run
```

You can also specify the folder as an argument:

```
expfactory --run --folder /home/vanessa/Desktop/my_experiment
```

Validate the configuration file (config.json) for your experiment

```
cd my_experiment
expfactory --validate
```

Test your experiment with our experiment robot

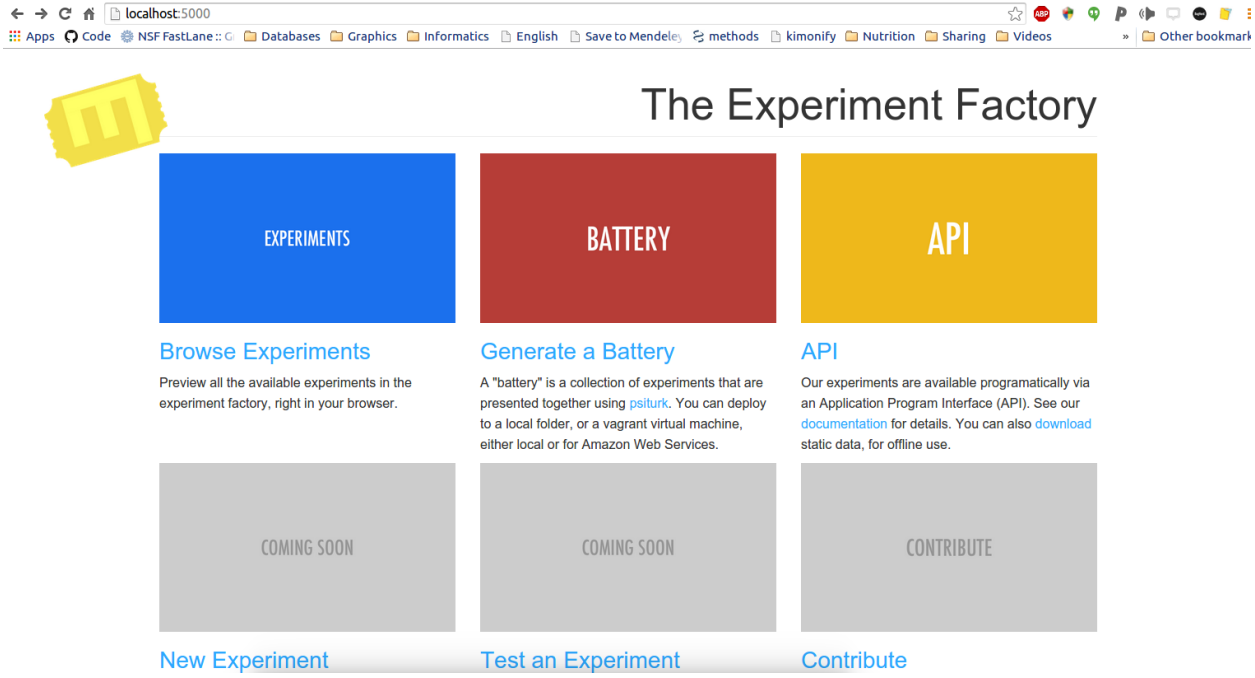
```
cd my_experiment
expfactory --test
```

Run the executable to open up a web interface to generate psiturk battery or virtual machine.

```
expfactory
```

You can also specify a port:

```
expfactory --port=8088
```



CHAPTER 2

Getting Started

You are probably a researcher that wants to collect behavioral data, and you have very specific requirements for your desired experiments, along with the environment that you will be collecting them in. The Experiment Factory is an open source, modular infrastructure that will make this easy for you. We store all of these components in a version control system called Github including 1) experiments, 2) surveys, 3) games, and different options for deployment (discussed below). Note that this is now a legacy version of the Experiment Factory. If you are looking for the newer (container-based and reproducible) experiment factory, please see expfactory/expfactory. Before we get started, let's define a few terms.

- **experiments**: an experiment is a web-based task, survey, or game. This means that it is presented in a browser, and coded in JavaScript, HTML, and CSS. If you look in any of the repos linked above, you will see an experiment is just a folder of files that will render in a browser and present visual and auditory stimuli, as well as collect behavioral metrics like response and reaction time, and accuracy.
- **battery**: a battery is a bunch of experiments presented in sequence. Typically, in research studies participants are presented with a battery of experiments.
- **deployment**: refers to where and how you want your participants to take the battery. You may choose to have them come into your lab, and sit at one of your computers. You may went to send your participants a unique, personal link to complete on their own computer. You may also want to serve the experiments from your own private server or database, or from a computer without an internet connection. For all of these situations, the experiment factory offers a solution.

How should you get started? We first recommend that you try out our paradigms. Your search may stop with this link - you can use this preview table to deploy the current version of an experiment, and data will be downloaded to your browser at the completion of each. If you like what you see, we encourage you to express interest in expfactory.org, a live deployment of expfactory docker that we are using at Stanford to deploy experiments locally and to Amazon Mechanical Turk. We are considering opening up this site to the public to offer Experiments as a Service (EaS) depending on (your!) expressed user interest. In the meantime, if you are comfortable with command line tools like git and pip, you can jump right into our development docs or deployment docs. We provide many avenues for deploying experiments, and all of this software is open source and free for you to use. We have a vision for reproducible, collaborative science, and want you to have control over your experiments and choice for deployment. Thus, we provide details about deployment options in the following sections.

2.1 I want to generate a custom battery

First you should follow instructions for installation. The Experiment Factory provides several options for deploying a battery, running one or more experiments locally, or generating a static battery to run on your own web server. A battery of experiments is a selection of experimental paradigms that are presented in sequence. We have made it easy to select one or more experiments from any of our repos, plug them into a expfactory battery, and deploy. What are your deployment options? You can use the paradigms as single units, as is, at expfactory.github.io, run a battery on demand using the command line tool, generate a custom folder to run later, or deploy a more substantial infrastructure, including vagrant virtual machines, a docker-based infrastructure.

2.1.1 Local Deployment of Experiments

A local deployment means running one or more experiments on a local machine, such as your computer or a lab machine. The simplest thing you can do is install the expfactory tool, and then run a battery of experiments:

```
expfactory --run --experiments stroop,nback
```

This command will use the latest experiments in the repo. If you want to ensure that your experiments do not change, or if you will be running the experiments without an internet connection, or if you want to modify experiments for your need, we recommend cloning the experiments repo, and specifying it as an argument to the command:

```
git clone https://github.com/expfactory/expfactory-experiments
expfactory --run --experiments stroop,nback --folder expfactory-experiments
```

You can do the same for a battery folder:

```
git clone https://github.com/expfactory/expfactory-experiments
expfactory --run --experiments stroop,nback --battery expfactory-battery
```

For each of the above, this will open up your browser to run the experiments, and data is stored to your computer via a file downloaded in the browser. You might want to specify a participant id to customize the naming of the output data, and to embed the id in the data itself:

```
git clone https://github.com/expfactory/expfactory-experiments
expfactory --run --experiments stroop,nback --subid DNS001
```

The above can also be done for any of our surveys or small selection of games:

```
expfactory --run --survey bis11_survey
expfactory --run --game bridge_game
```

2.1.2 Generation of Local Battery

If you want to generate a custom folder and run it later, you can use the expfactory command line "–generate" command. This situation is ideal for generating your own static HTML/CSS files to put on your own web server. We recommend our Windows users to do this in some Linux environment (if your university has a shared cluster or on a Virtual Machine) and then to move the battery folder back onto the Windows Machine. To generate a battery folder:

```
expfactory --generate --experiments stroop,nback
```

Or to generate a Psiturk battery folder:

```
expfactory --generate --experiments stroop,nback --psiturk
```

If you don't specify an output folder, a temporary directory will be created. For each of the above, you can specify the –output command, a full path to a folder (that does not exist) that you want the battery generated in.

Expfactory Portal

If you are a Linux or Unix-based user and want to dynamically generate a custom battery for Psiturk (called a "folder" deployment), or a Virtual Machine with Vagrant, you can do so with the command line tool:

```
expfactory
```

Which will open up a view in your browser. You can then click "battery." The web interface will take you through the following steps:

- A. collection of experiment details
- B. database connection parameters
- C. selection of local (folder) experiment, or deployment to AWS.
- D. selection of experiments

More details are provided about choosing a deployment, and configuring your battery.

Expfactory-docker

The Experiment Factory docker is a set of containers that can be run locally, or again on the cloud. The entire application comes packaged in a set of Docker images, meaning that installation and deployment of experiments happens in a web interface deployed by Docker Compose. We plan to offer experiment deployment as a service at expfactory.org and encourage you to sign up to express interest. You can also deploy our Docker infrastructure on your own server, however experience with docker and cloud computing is required.

2.2 I want to preview available experiments

We provide static versions of all experiments, along with meta-data, in our expfactory-experiments github pages. You can preview the currently available experiments in our online portal. You can generate this portal on the fly on your local machine as well:

```
from expfactory.views import generate_experiment_web
output_folder = os.path.abspath("/home/vanessa/Desktop/web")
generate_experiment_web(output_folder)
```

The output folder does not need to exist. This will generate the equivalent interface hosted on expfactory.github.io.

2.3 I want to contribute an experiment

The short story is that all of the experiments that can be selected are just folders on Github, https://github.com/expfactory/expfactory-experiments, and you can contribute by modifying an existing experiment or creating a new one by submitting a PR to this repository. Adding surveys are even easier, as a survey is just a tab delimited file in a folder in the expfactory-surveys repo. For complete details about experiment, survey, and game contributions, please see our development pages.

2.4 I want to learn about the expfactory-python functions

The generation of the batteries, along with experiment validation, and virtual machine deployment, are controlled by the expfactory-python functions. You can see complete function documentation under modindex, and we welcome any contributions to the code base via Github pull requests (PRs) or isses. We provide a few examples below of running tests and generating visualizations.

2.4.1 Run the experiment testing robot

```
expfactory --test
```

2.4.2 Validate an experiment folder

```
expfactory --validate
```

2.4.3 Preview a single experiment

```
expfactory --preview
```

2.4.4 Generate the entire expfactory.github.io interface

```
from expfactory.views import generate_experiment_web
web_folder = '/home/vanessa/Desktop/site'
generate_experiment_web(web_folder)
```

You can then run an experiment robot over experiments in this folder, either for all experiments:

```
experiment_robot_web(web_folder)
```

or a subset of experiments

```
experiment_robot_web(web_folder,experiment_tags=changed_experiments)
```

2.4.5 Checking static javascript with jshint

We recommend using the docker image to do this, across many experiment directories at once:

```
docker pull hyzual/jshint
cd expfactory-experiments
sudo docker run -it -v $(pwd):/lint hyzual/jshint --config /lint/.jshint_config .
```

2.4.6 Validate an entire set of experiment directories

```
from expfactory.tests import validate_experiment_directories, validate_experiment_tag
validate_experiment_directories('expfactory-experiments')
validate_experiment_tag('expfactory-experiments')
```

CHAPTER 3

Deployments

You should proceed with this steps after after installation of the Experiment Factory command line tool.

3.1 Local Battery

You can deploy a battery of experiments (meaning one or more experiments presented in a row) simply by using the expfactory command line tool. The basic function to generate a battery is the following:

```
expfactory --run --experiments local_global_shape,test_task
```

The command "–run" will tell the application that you want to run a local battery. The "experiments" variable is required, and should be a comma separated list of experiment unique ids (exp_id), meaning folder names in the experiment repo. You can also specify a subject id that will be embedded in the output data, and give a name for the data file (please don't use spaces):

```
expfactory --run --experiments local_global_shape,test_task --subid id_123
```

You can also specify a maximum running time (in minutes), in the case that you want to randomly select from experiments up to some time:

```
expfactory --run --experiments local_global_shape,test_task --time 30
```

The default is a very large number that (we hope) a battery would never go over. Finally, it could be the case that you want to use modified experiments, and in this case you can provide a folder of experiments as an argument:

```
expfactory --run --experiments local_global_shape,test_task --folder /path/to/your/
↪expfactory-experiments
```

or a battery folder as an argument:

```
expfactory --run --experiments local_global_shape,test_task --battery /path/to/your/
↪expfactory-battery
```

WARNING: Not specifying either the battery or experiments folder will always pull the latest from the repos. If you intend to run many subjects over time and want ensured consistency in the experiments, we reccommend that you clone both the battery and experiments repos, and specify them in the command:

```
git clone https://github.com/expfactory/expfactory-experiments
git clone https://github.com/expfactory/expfactory-battery
expfactory --run --experiments local_global_shape,test_task --battery expfactory-
↪battery  --folder expfactory-experiments
```

3.2 Generate Static

The above instructions will open up a web browser to run experiments on demand. If you want to generate a folder, either to be run locally or with psiturk, you can do the following:

```
expfactory --generate --experiments stroop,nback
```

Or to generate a Psiturk battery folder:

```
expfactory --generate --experiments stroop,nback --psiturk
```

If you don't specify an output folder, a temporary directory will be created. For each of the above, you can specify the –output command, a full path to a folder (that does not exist) that you want the battery generated in.

3.3 Interactive Battery Generation for Psiturk or Virtual Machine

Note that the interactive portal is a Flask application, and has not been tested on Windows systems.

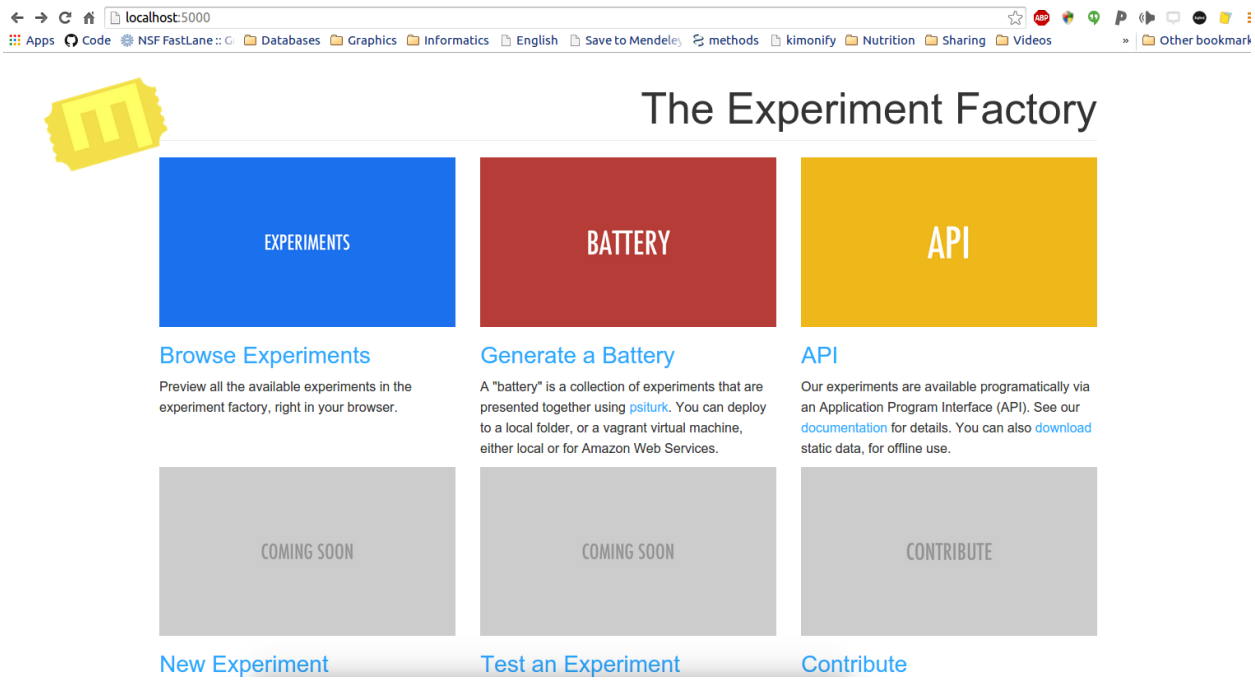
3.3.1 The Experiment Factory Application Portal

To run the executable to open up a web interface to design your psiturk experiment:

```
expfactory
```

```
^Cvanessa@vanessa-ThinkPad-T450s:~/Documents/Dropbox/Code/expfactory/expfactory-python$ expfactory
Found 37 valid experiments
Nobody ever comes in... nobody ever comes out...
 * Running on http://0.0.0.0:2020/ (Press CTRL+C to quit)
 * Restarting with stat
Created new window in existing browser session.
Found 37 valid experiments
Nobody ever comes in... nobody ever comes out...
```

This will open up your browser to the experiment factory portal. From here you can click on *battery* to start design of your psiturk experiment battery. You can also use this tool to serve a RESTful API of current experiments.



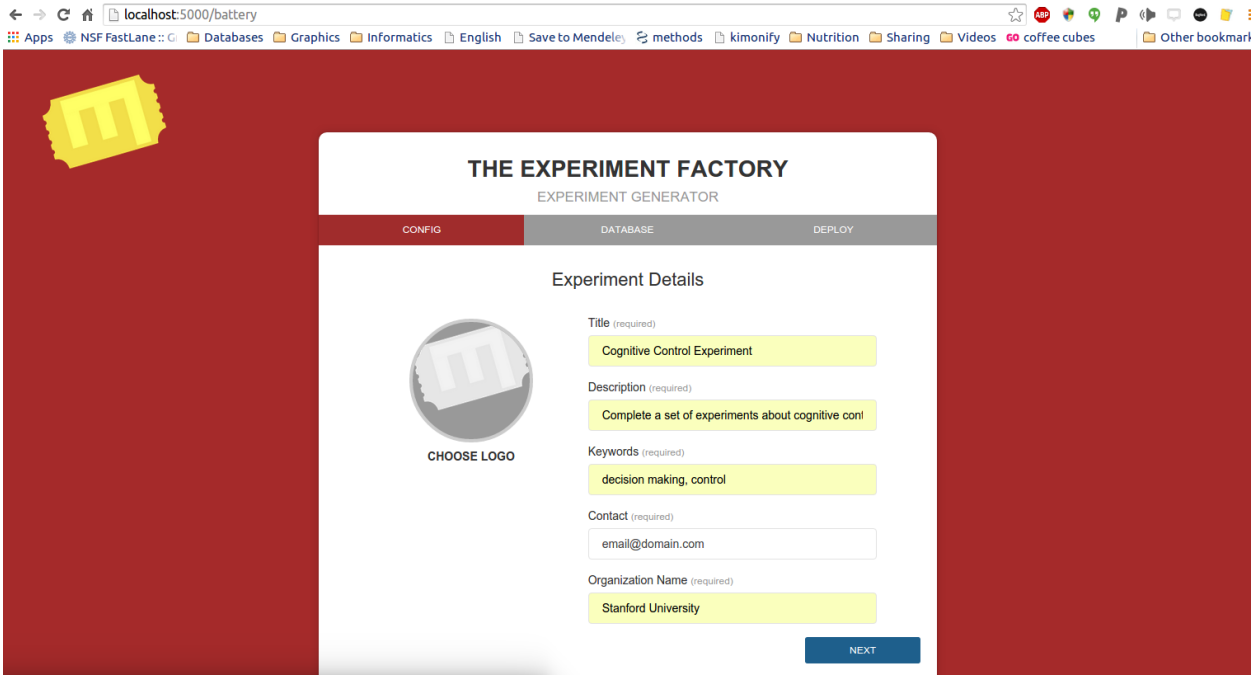
Configure a psiturk battery of experiments

The web interface will take you through the following steps:

- A. collection of experiment details
- B. database connection parameters
- C. selection of local (folder) experiment, or deployment to AWS.
- D. selection of experiments

3.3.2 A. Collection of experiment details

A psiturk battery is controlled via a config.txt file, in which you specify details of your experiment. Here we will collect those fields:

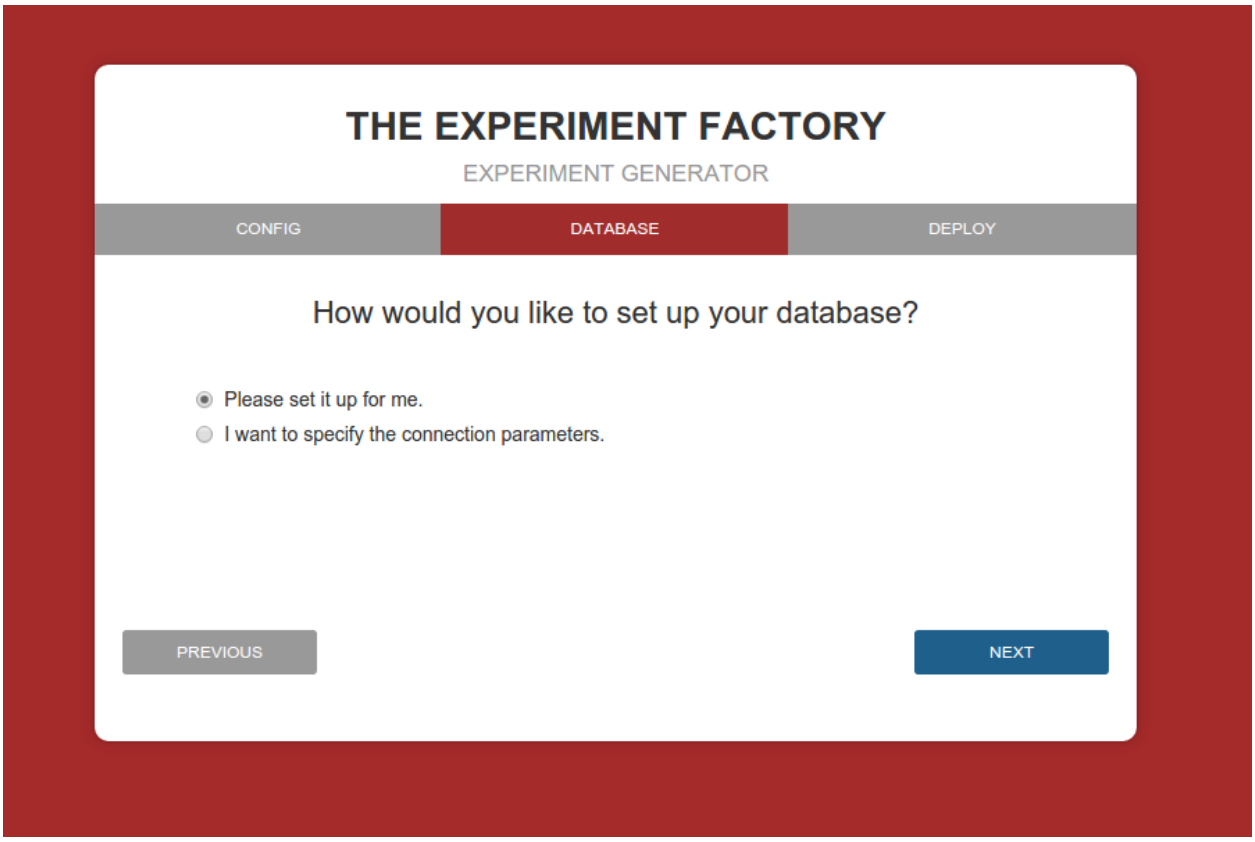


All fields are required. Specifically:

- Title: should be the title of your experiment.
- Description: should describe what your experiment is measuring, or its goals.
- Keywords: should be lower-case, comma separated
- Contact: should be a valid email for psiturk to contact *only on error*
- Organization Name: is typically your institution or university

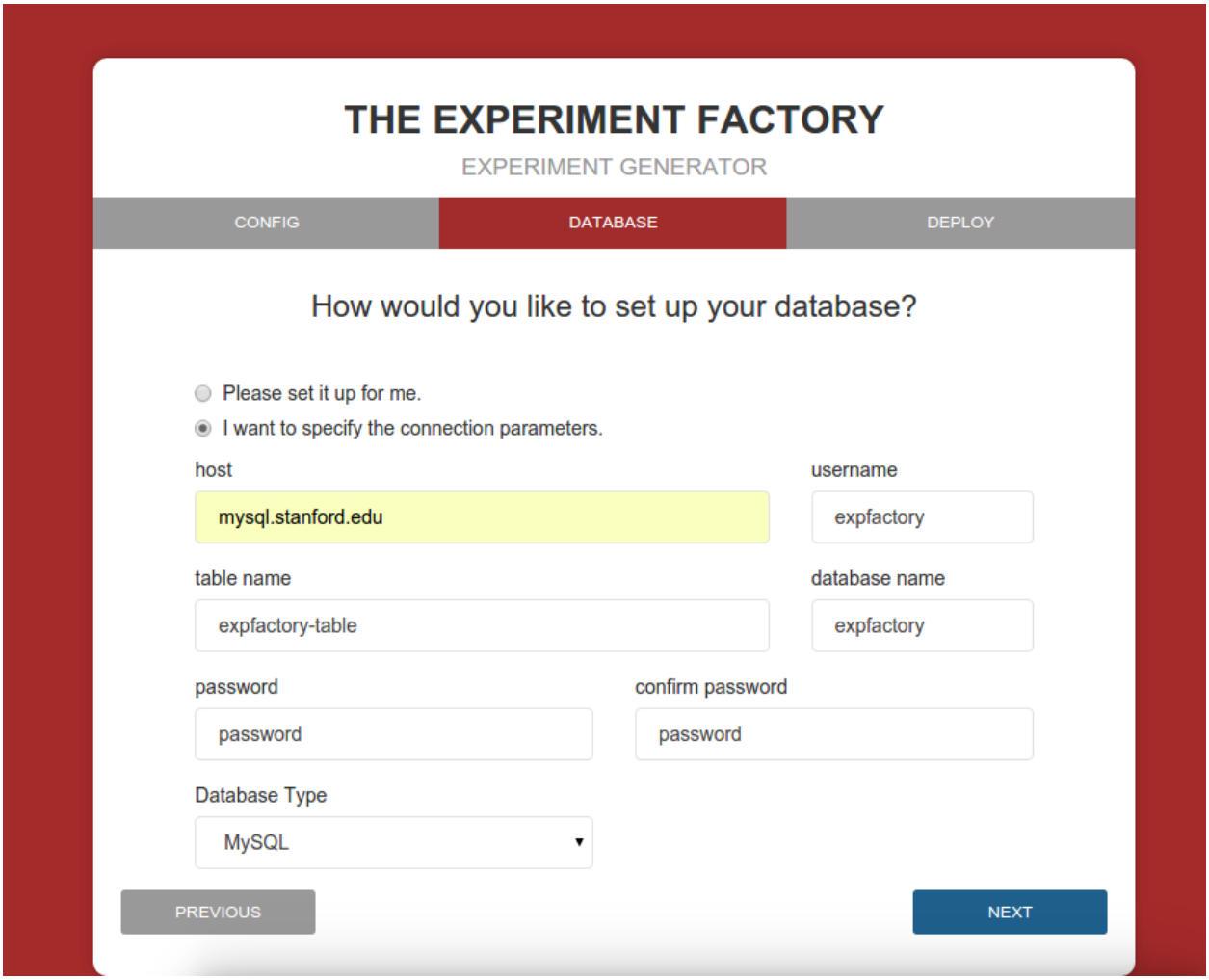
3.3.3 B. Database connection parameters

If you don't know anything about databases, you should let expfactory set up the database for you. In the case of a local folder, this will mean an sqlite3 file. In the case of a virtual machine or cloud (AWS) deployment, expfactory will configure a mysql database on the same server.



If you are testing, we recommend that you let expfactory set it up for you. For an actual deployment, and especially a cloud deployment, it is essential that you specify custom MySQL or Postgresql database parameters. Letting expfactory set up a cloud deployment means that the connection parameters are available for all to see, and this is a risk to your data and to the privacy of your participants.

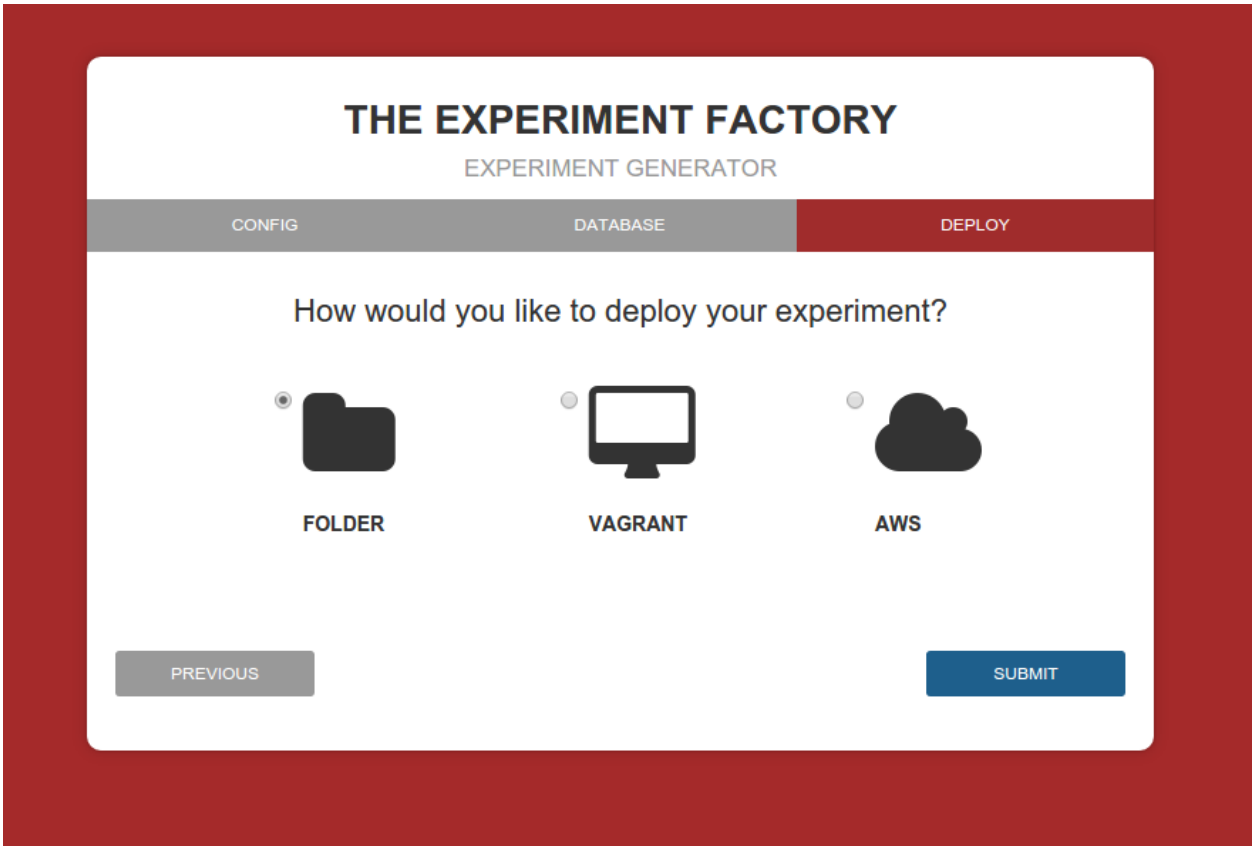
Finally, we do not allow for the option of an sqlite3 database beyond a local folder, because sqlite3 can only handle one read/write at a time, and thus is only appropriate for single-user, local testing. When setting up a database, example parameters are shown below:



Psiturkpy does not test these parameters for you.

3.3.4 C. Deployment

You have several deployment options, including a local folder, a virtual machine (vagrant), or a cloud deployment (Amazon Web Services, or AWS).



If you already have psiturk configured on your local machine and basically want to produce a folder to run, then you should choose the "Folder" option for deployment. If you do not specify database parameters, the default is to generate a sqlite3 file (a static database file in the same folder as your experiment).

If you want to use psiturk but don't want to install things on your local machine, a vagrant virtual machine would be appropriate. This means that you will ssh into the machine to "see" the folder with your experiment, and all required software and databases will be installed for you. The default IP address you will use to access the virtual machine in your browser is 192.128.0.20.

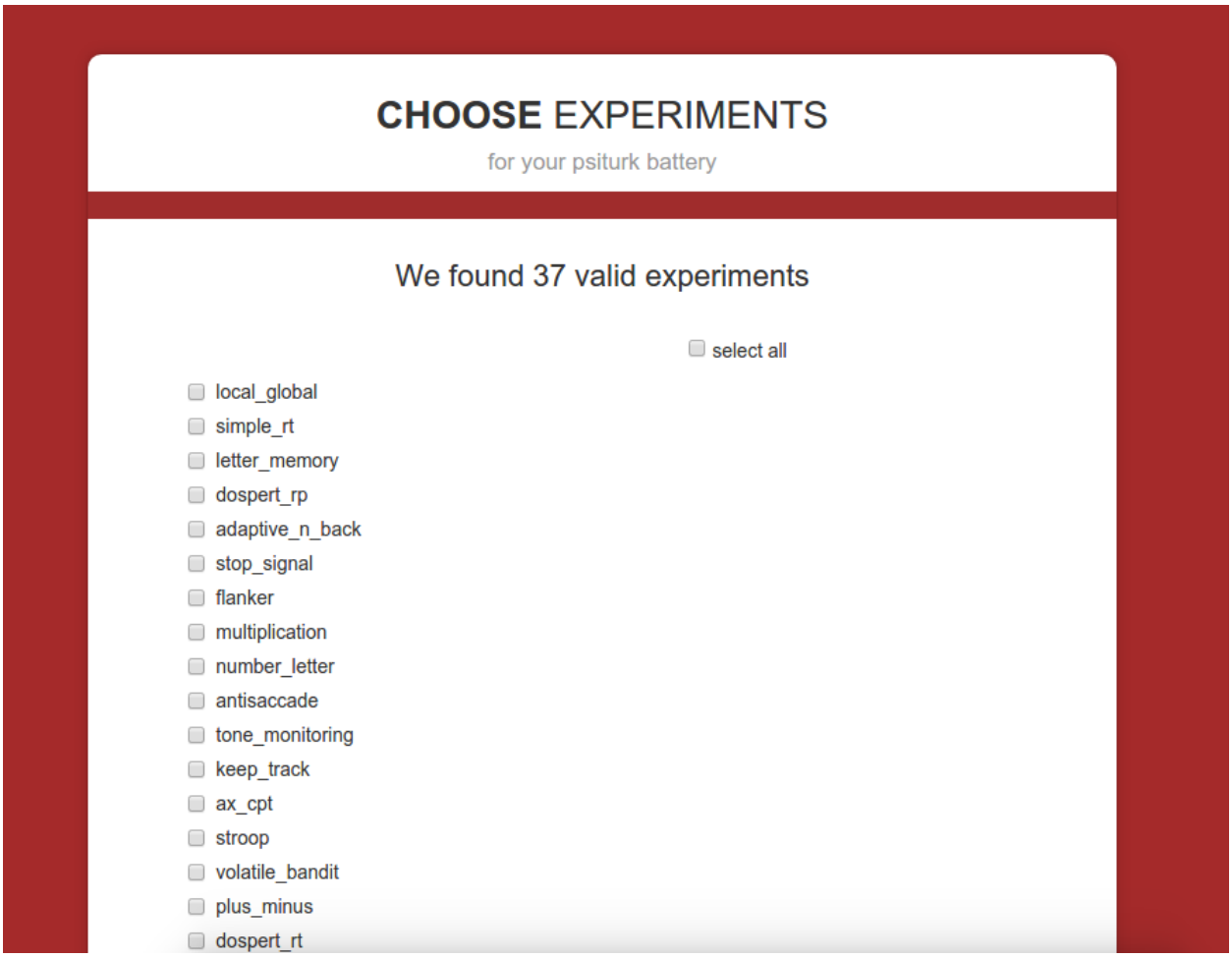
If you want a deployment that, for example, your lab could use to run many participants at once, you might want to set up psiturk on the cloud, for example, an Amazon Web Services EC2 Instance. If this is this case, we *strongly* recommend that you do not use the "default" database option, as it could compromise your data - you should enter the database parameters for a secure database that you create.

Before getting started, you should familiarize yourself with psiturk. Likely you will be interested in setting up an experiment on your local machine, and so you should follow the installation instructions and go through the entire demo to make sure things are working properly.

Choosing "folder" will generate a local experiment, and either of the virtual machine options will produce a Vagrantfile that can be run to deploy the Virtual Machine. Specific instructions for a local vagrant or vagrant-aws are provided. Note that you can use the VagrantfileLocal or VagrantfileAWS file (renamed to Vagrantfile) as is from the virtual machine repo to generate a battery with all available, valid experiments.

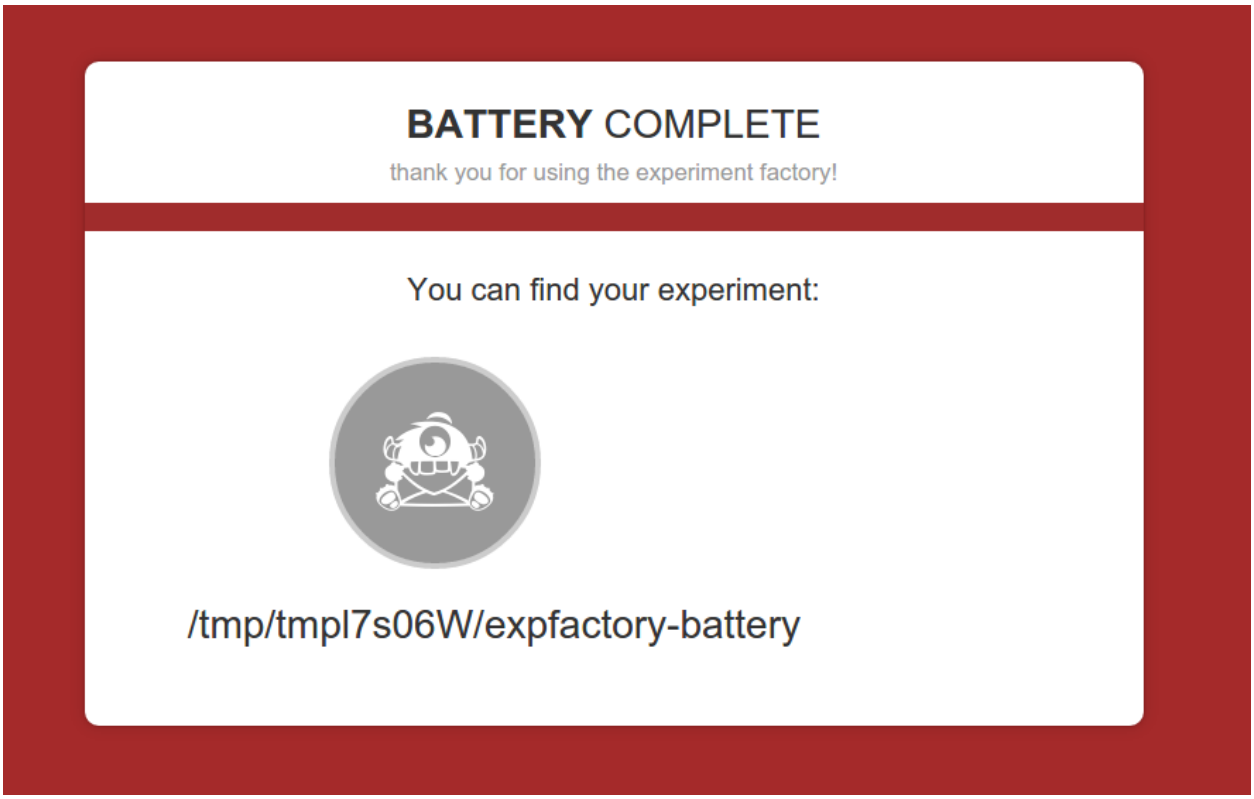
3.3.5 D. Experiment selection

When you click "Next" on part C above, behind the scenes the battery and experiment folders are downloading, and the most updated set of experiments are run through a validator. In this experiment selection screen, you are presented with experiment folders from http://www.github.com/expfactory/expfactory-experiments repo that pass validation:

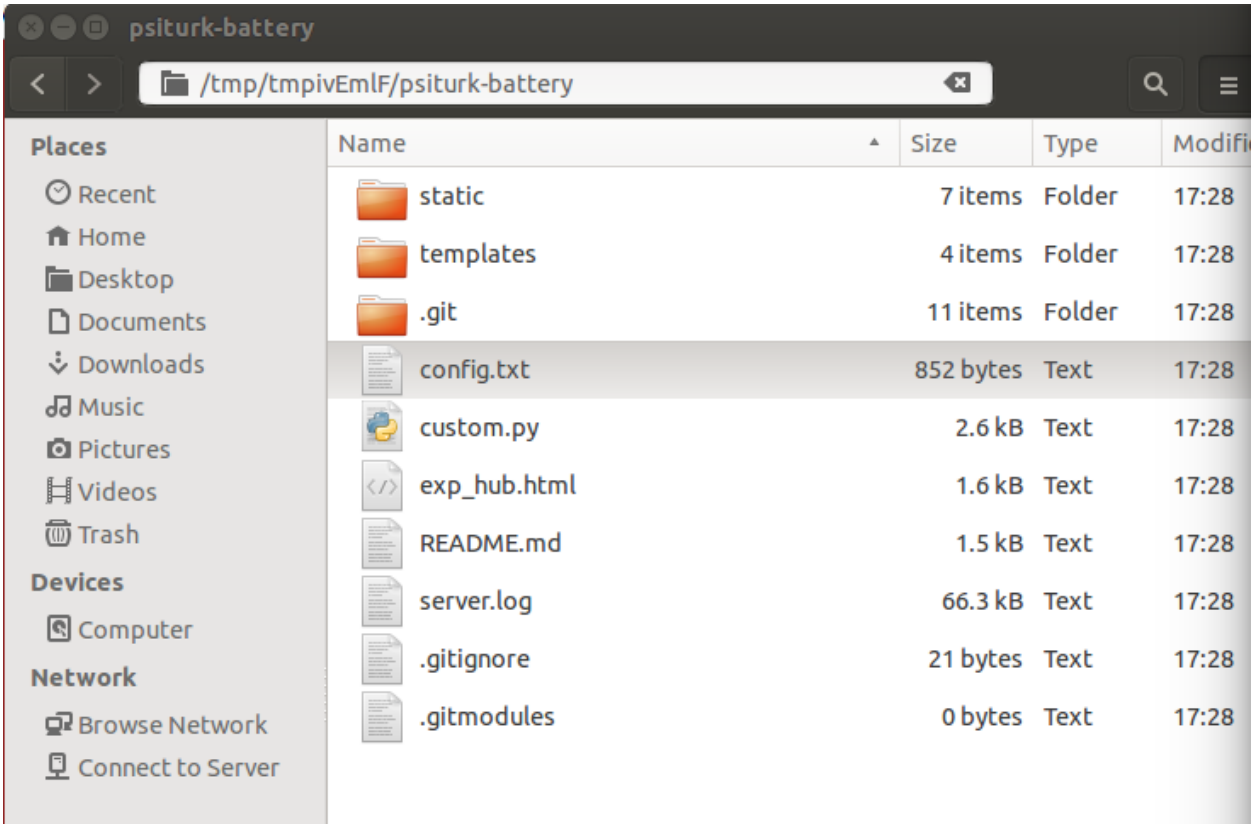


You can select as many or as few as you want, and they will be included in your custom battery. Note that in the future you will be able to select experiments based on task or concept from the cognitive atlas.

After experiment selection, your battery generation is complete, and the web interface will tell you where you can find the folder or Vagrantfile:

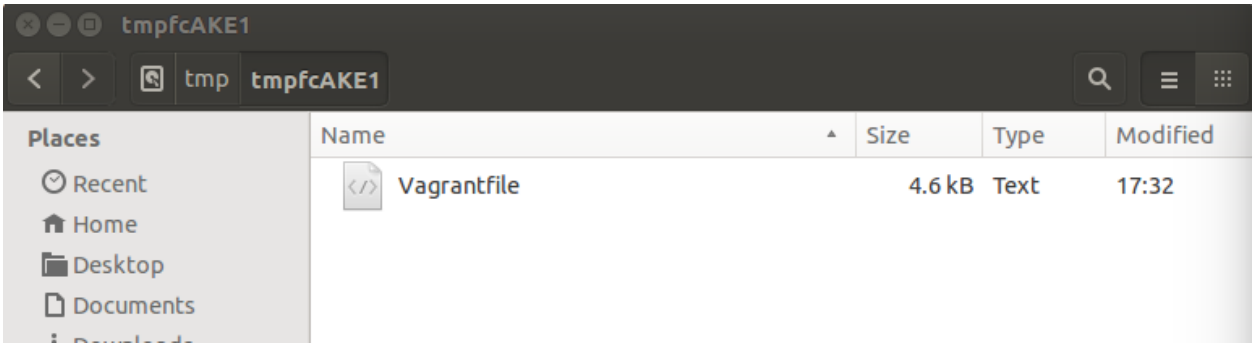


In the case of a "folder" generation, you will see a standard psiturk experiment structure:



We recommend you move this folder to where you would like to keep it, and then cd to the folder, and type *psiturk* to start the experiment. Note that this assumes that you have properly installed psiturk on your machine. If not, please return to these instructions.

In the case of a "vagrant" or "AWS" configuration, you will find a *Vagrantfile* in the output folder:



Vagrant Deployment

One of the options is to produce a "Vagrantfile" for use on your local machine. This is ideal if you want to use psiturk locally, but don't want to go through installation of the module or a database.

3.3.6 Setting up a local VM

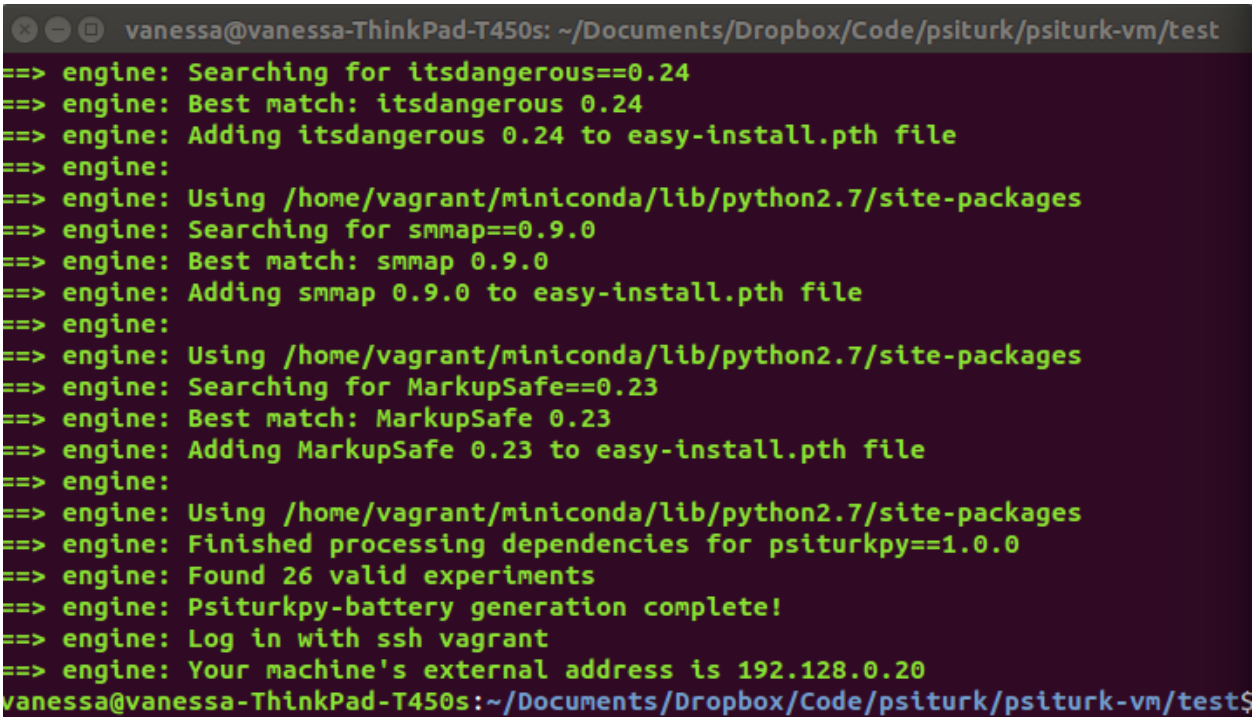
1. Install VirtualBox from https://www.virtualbox.org/wiki/Downloads
2. Install Vagrant from http://www.vagrantup.com/downloads. Vagrant is a provisioning system that sets up the virtual machine.
3. If you don't already have it, install git https://git-scm.com/downloads
4. You have two options to produce your Vagrantfile. If you want a custom battery, then you should run the executable expfactory, generate the file, and copy it to a folder outside of a the temporary directory. If you want to generate a vagrant machine with all valid tasks available, then you can clone the expfactory-vm repo and use the file "VagrantfileLocal" renamed to "Vagrantfile." Note that this is also the case for the AWS version (instructions below).

```
cd myvagrantdirectory
git clone https://github.com/expfactory/expfactory-vm.git`
cd expfactory-vm
```

We are now in a directory with the Vagrantfile. You can set up the virtual machine:

```
vagrant up
```

Note that this can take some time. Keep the lookout for red statements that may indicate an error. If you find an error please report it as an issue.



The above shows a successful build.

5. You can then log in with ssh

   vagrant ssh

7. Your experiment is located in $HOME/expfactory-battery. The build files are in expfactory-build. Note that when you turn the server on and debug, you will need the -p option to make sure the machine does not attempt to open the link with a browser in the terminal.

```
cd $HOME/expfactory-battery
psiturk
server on
debug -p
```

3.3.7 Deployment to AWS

You can equivalently produce a Vagrantfile with the expfactory module that can be deployed to AWS. Some expertise is assumed, namely that you are familiar with the EC2 section of the AWS console. You will need to log in and create a security group, download a key, and fill in all variables required in the SCRIPT section of the Vagrantfile.

1. Make sure you have the most up-to-date version of vagrant from https://www.vagrantup.com/downloads, and install vagrant-aws. If you do not, you will see this error (version 1.6.5)

```
vagrant-share can't be installed without vagrant login (RuntimeError)
```

And after updating:

```
    vagrant --version
    Vagrant 1.7.2
```

(continues on next page)

(continued from previous page)

```
2. Then you should install vagrant-aws, which will allow you to provision the Amazon␣
↪machine.
```

```
sudo vagrant plugin install vagrant-aws
Installing the 'vagrant-aws' plugin. This can take a few minutes...
Installed the plugin 'vagrant-aws (0.6.0)'!
```

You then need to add an aws compatible box. I found this box on the vagrant-aws plugin github repository:

vagrant box add aws https://github.com/mitchellh/vagrant-aws/raw/master/dummy.box

3. Fill in your Amazon credentials into the Vagrantfile, then bring up the machine

```
vagrant up --provider=aws
```

A few important notes:

- Make sure you mypem.pem file has the correct permissions:

```
  chmod 400 ~/.ssh/myfile.pem

* Many times you will not be able to connect to your machine because the security␣
↪group input/output is too stringent. Make sure to open ports to allow the default␣
↪psiturk port to come through (22362) as well as SSH.
```

3.4 expfactory-docker

The Experiment Factory docker is a set of containers that can be run locally, or again on the cloud. The entire application comes packaged in a Docker image, meaning that installation and deployment of experiments happens in a web interface deployed by the image. We plan to offer experiment deployment as a service at expfactory.org and encourage you to sign up to express interest. You can also deploy our Docker infrastructure on your own server, however experience with docker and cloud computing is required.

3.5 expfactory-server

Another deployment option is to deploy the batteries to a web server.

The batteries themselves are stored as static html files with a uid parameter in the query string to pass the subject id: e.g. https://mywebserver/itest/digit_span-en/?uid=123456789

PHP scripts, also hosted by the web server, are called on experiment completion to save the results (in JSON format) to a MySQL database.

The folder expfactory-python/server/mysql contains the required server resources to deploy. Ensure to define your database connection parameters in database_connect.php. In this folder, you will also find a create_expfactory_table.sql and a concerto.conf file as an example of how to make Concerto and Expfactory coexist and share the same database.

Edit the post_url variable in the templates/webserver-battery-template.html template to change the default URL to the PHP script (/itest/save_data.php) if needed. If not a local URL, cross-origin resource sharing should be enabled : refer to http://enable-cors.org/server_apache.html

The setup_battery_for_webserver.py script should be used to generate the battery in the target web directory.

```
python setup_battery_for_webserver.py --output /var/www/vhosts/expfactory-server/
↪digit_span-en --experiments digit_span
```

Though the run_battery.py script has no use in production, you may test your batteries easily with:

```
python run_battery.py --port 8080 --folder /var/www/vhosts/expfactory-server/digit_
↪span-en
```

CHAPTER 4

Application Program Interface (API)

4.1 Expfactory.org API

For experiments served with expfactory.org, we offer an interactive RESTful API, and a python module expanalysis specifically for retrieving and working with results. To use the API, you will need to provide a token generated when you log in to expfactory.org. First install the analysis package:

```
pip install expanalysis
```

Then when you have your token, here is how to use it from python:

```
from expanalysis.api import get_results

access_token = "abcdefghijklmnopqrstuvwxyz" # expfactory.org/token
results = get_results(access_token=access_token)
```

4.2 Experiments API

The experiment factory executable is useful for making new batteries and experiments, but it can also work to serve a RESTful API. This basically means you can type in URLs in your browser, and it will spit data back at you. The data is in the JavaScript Object Notation (JSON) format, which is super easy to read into your software of choice (python, R, matlab, etc.). If you want to download static versions of the experiment data, we also provide those in our Github repo. We recommend, however, that you use the API because it will always download and return the most up-to-date experiments.

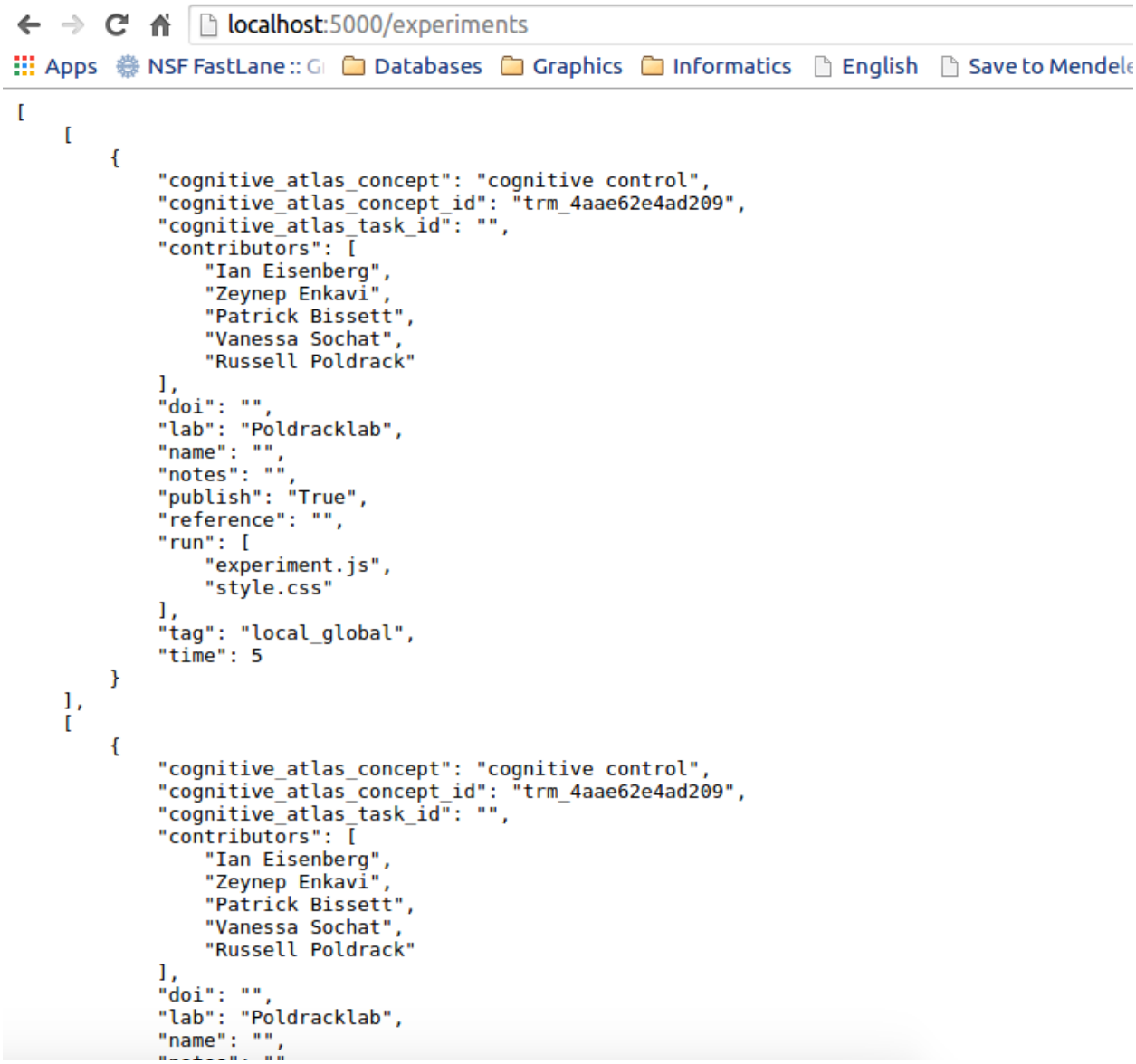
4.3 Running the API

After installation, start up the application.

```
expfactory
```

4.3.1 Retrieving all Experiments

```
http://localhost:8088/experiments
```

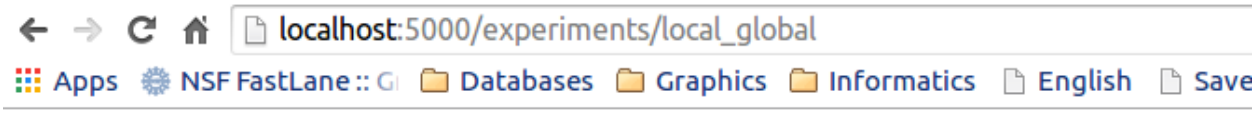



```
[
    [
        {
            "cognitive_atlas_concept": "cognitive control",
            "cognitive_atlas_concept_id": "trm_4aae62e4ad209",
            "cognitive_atlas_task_id": "",
            "contributors": [
                "Ian Eisenberg",
                "Zeynep Enkavi",
                "Patrick Bissett",
                "Vanessa Sochat",
                "Russell Poldrack"
            ],
            "doi": "",
            "lab": "Poldracklab",
            "name": "",
            "notes": "",
            "publish": "True",
            "reference": "",
            "run": [
                "experiment.js",
                "style.css"
            ],
            "tag": "local_global",
            "time": 5
        }
    ],
    [
        {
            "cognitive_atlas_concept": "cognitive control",
            "cognitive_atlas_concept_id": "trm_4aae62e4ad209",
            "cognitive_atlas_task_id": "",
            "contributors": [
                "Ian Eisenberg",
                "Zeynep Enkavi",
                "Patrick Bissett",
                "Vanessa Sochat",
                "Russell Poldrack"
            ],
            "doi": "",
            "lab": "Poldracklab",
            "name": "",
```


4.3.2 Retrieving a specific Experiment

The unique id of an experiment is its "exp_id," which also corresponds to the name of its folder and the "exp_id" variable in its config.json. You can select an experiment by this unique id:

```
http://localhost:8088/experiments/simple_rt
```




```
{
    "local_global": {
        "cognitive_atlas_concept": "cognitive control",
        "cognitive_atlas_concept_id": "trm_4aae62e4ad209",
        "cognitive_atlas_task_id": "",
        "contributors": [
            "Ian Eisenberg",
            "Zeynep Enkavi",
            "Patrick Bissett",
            "Vanessa Sochat",
            "Russell Poldrack"
        ],
        "doi": "",
        "lab": "Poldracklab",
        "name": "",
        "notes": "",
        "publish": "True",
        "reference": "",
        "run": [
            "experiment.js",
            "style.css"
        ],
        "tag": "local_global",
        "time": 5
    }
}
```


More API functions will come as requested. If there is a functionality you would desire, please tell us.

CHAPTER 5

Development

If you have not yet, please 'install http://expfactory.readthedocs.org/en/latest/installation.html the expfactory python application.

5.1 Contributing to code

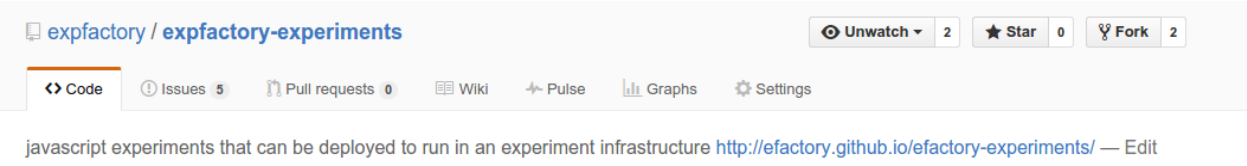
We recommend that you work with developing experiment code using Github. If you aren't familiar with Github, here is an introduction, and we provide more description below. This means that your general workflow will be as follows:

- A. fork our experiments repo
- B. clone the repository to your local machine
- C. checkout a new branch for your update
- D. when you are happy, commit changes locally, then to your fork
- E. submit a pull request (PR) to our master branch to review the changes
- F. keep in sync with our master

This will also be the general workflow for making any contributions to the battery, virtual machines, or python application.

5.1.1 A. Fork our repos

When you browse to our repo, you will see a "Fork" button in the upper right (and the same is true for expfactory-surveys or expfactory-games):



Click it, and select your Github username to create a copy of the repository under your account name.

A little about Github

You can think of Github as a home in the cloud to store your code, along with a history of all changes. This means that many people can collaboratively work on the same code and have a way to talk about changes, report issues, and merge it all together in the end. The way that Github works is based on having this "remote" version, along with a "local" version on your computer. The code base is generally called a "repository" (repo for short), and the way it is organized is based on branches. There is usually a default branch that is the "working" and "up to date" version of the code, and this is usually called "master." When you fork a repo, you should generally keep your version of "master" synced with the main (upstream) repository master. This is because we will use this branch as a template for all new branches. You can imagine if you are working on two features, you would want for each one to have a separate branch in your repo. If you use your master branch to work on the first feature, then you have no way of easily creating a fresh branch that is in sync with the upstream repo master that you forked. When someone talks about "getting code from" a repo, they usually will use the term "pull," and this is why when you make changes and ask another repo to accept (pull) your changes, it is called a "pull request." Sending code to the cloud repo will use the term "push." Rest assured that these terms and development practices will become second nature when you get into the groove of things. We will provide detailed instructions when possible, and please tell us if something is not clear.

5.1.2 B. Clone the repository to your local machine

After you fork the repo, you have created a version in the cloud associated with your account. But to work with it, you need to "clone" it to your local machine. You can do this by way of:

```
git clone https://github.com/[your-username]/expfactory-experiments
```

Note that if you use ssh (instructions for setup are here) you should clone like this:

```
git clone git@github.com:[your-username]/expfactory-experiments.git
```

where [your-username] is replaced with your Github account username.

A little about repos

A repository is "connected" to Github via a simple config text file. If you look at this file, you can understand quickly how simple it is. For example, let's say we just cloned the experiments repo:

```
cd expfactory-experiments
```

There is a hidden folder called ".git" with a config file in it. Generally you should not manually edit files in this folder, but the config is pretty easy to understand, and easy to edit to change the names of your repos or add other repos to push and pull to. Let's take a look at a standard config file:

```
vim .git/config
```

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = git@github.com:vsoch/expfactory-experiments.git
```

(continues on next page)

(continued from previous page)

```
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
```

We want to make a quick change to this file. The rationale for this change is that many people might be working on expfactory experiments, and our version of "master" can quickly fall behind the expfactory master. With this in mind, we want to be able to easily "pull" from the expfactory master, which we will call "upstream." Here is my config file for the expfactory-python. Since I want to work with the "upstream" (main expfactory Organization repo) along with my version (associated with username vsoch) you will notice that I have added a few lines to specify an "upstream" remote:

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "upstream"]
        url = git@github.com:expfactory/expfactory-experiments.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[remote "origin"]
        url = git@github.com:vsoch/expfactory-experiments.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
```

This means that, whenever I want to checkout my master to create a new branch from, or when either my PR or someone else's PR has been merged into the expfactory master, I should pull from upstream and push to my master:

```
git pull upstream master
```

It will open a text file that is asking for you to commit a merge. Save this file to do so. Then push to your master:

```
git push origin master
```

If you get errors about merge conflicts, this means that something was out of order, and you need to open up files that have conflicts to resolve the conflicts, and then commit. If you only use your master as a template to checkout new branches from, you should not run into this problem.

For the above example, you will also notice that the format of the URLs follows the SSH standard, because that is how I cloned it, and how it is setup on my computer. If you get errors about authentication, you likely have the wrong format specified in your config. A https url would look something like:

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = https://github.com/vsoch/expfactory-experiments.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
```

It's pretty common to clone with HTTPS (copy pasting the url for a repo), and then edit the config to change to SSH to push and pull. You can define as many remotes as you like in this file so you can easily pull from your colleagues repos if you want to test what they are working on.

5.1.3 C. checkout a new branch for your update

You've now cloned the repository to your local machine, and you would want to check out a new branch to make an update. First, see what branch you are on:

```
git branch
* master
```

Then checkout a new branch, it's a good standard to give it a tag to describe what you are doing (fix/update/enh) and then a specific name. For example:

```
git checkout -b update/add_my_experiment
```

You will then be switched to your new branch

```
git branch
master
* update/add_my_experiment
```

Once the branch is created, it should be standard practice for you to first see what branch you are on before starting work for the day. If you need to switch, you don't need the "-b" argument:

```
git checkout master
git checkout update/add_my_experiment
```

5.1.4 D. commit changes

It's good practice to commit changes locally when you are happy, adding a message to provide details about what you've changed. You may first want to compare your recent work to what is on Github, and you can do this with:

```
git status
```

It will show you changed, deleted, and added files. You can then "commit" these changes to your local repo, which basically means writing them into some local record in the .git folder to prepare for pushing to the remote repo:

```
git commit -a
```

is read as "git commit all" and it will open up a text editor to show you a summary of the changes. It is expected with each commit that you describe what you have done in a message. You can either uncomment lines in the file (remove the "#") or write a message right at the top. Saving the file will commit the changes locally.

Finally, you can push to your local branch. The format is like:

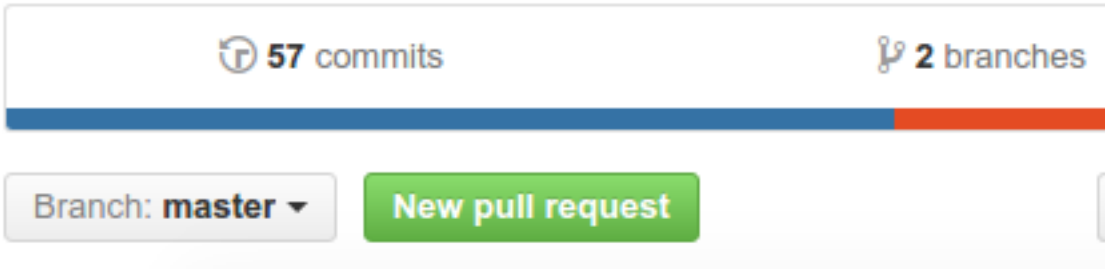
```
git push [remote] [branch]
```

where [remote] and [branch] correspond to the name of the remote and branch (in the .git/config) which means that for our example we might do:

```
git push origin update/add_my_experiment
```

5.1.5 E. submit a pull request (PR) to our master branch

Once you have made a change, and commit the change to your remote (cloud) repo, you would want to submit this change to us to consider merging into the expfactory master branch. It's easiest to do this with the online interface. You should go to your repo, and find the "Pull Request" button:



You will then want to select merging your base/branch into the expfactory/master branch. Please tell us in the comments what you have changed or added, and then submit the PR. We will be able to review your changes and give you feedback.

5.1.6 F. keep in sync with our master

As we detailed above, remember that many people might be submitting PRs to work on the same code base, which means that your master branch can fall behind. Whether you are updating your master after your PR (or someone else's) was merged, this should be the first thing you do before checkout of a new branch:

```
git checkout master
git pull upstream master
git push origin master
git checkout -b enh/add_function
```

Now that you are a development Github pro, you might want more details about the format of a new experiment.

5.2 Contributing a paradigm

Whether you want to contribute a survey, an experiment, or a game, the basic steps and structure of what you create will be the same. Each of surveys, experiments, and games has a repository under ' the expfactory github organization <http://www.github.com/expfactory>'_ and each repository is a collection of folders, with one paradigm for folder. You can store folders on your server or local machine, or just have our software grab the latest version directly from the repositories. The commonality between folders is that they all have a configuration file, the config.json, with a set of standard fields that are defined to allow the Experiment Factory Python Software to work with paradigms. A few notes about the config.json:

- You do not need to define fields in the config.json that are not required. Use expfactory –validate to see if you did it right.
- the folder name must correspond with the "exp_id" variable in the config.json
- experiment, survey, and game folder names should be all lowercase, no hyphens (-), or spaces.

5.2.1 config.json

A data structure that specifies the following:

- name: [required] the full name of the experiment, best is to use the name of the publication it is associated with.
- exp_id: [required] the experiment id (unique identifier) for the experiment, typically the folder name, all lowercase with no special characters.
- cognitive_atlas_task_id: [optional] the identifier for the experiment defined in the cognitive atlas
- template: [required] the javascript (or other) template that runs the experiment. We currently support jspsych, survey (for material-design lite surveys), or phaser (games). Please contact us to discuss adding your particular framework.
- contributors: a list of contributors to the task code base.
- reference: url(s) to referenced papers to develop the task. This field is to be removed, and the reference or DOI should be stored in the CognitiveAtlas.
- experiment_variables: should be a list of dictionaries to specify one or more variables to be available for use to measure performance, allocate bonus, or receive credit. The fields of this variable include "name" "type" "label" "range" and "description." The "label" field will determine if the Experiment Factory docker virtual machine will parse the variable as being available to use for a reward (eg, label == "reward") or for allocation of credit (eg, label == "credit").
- notes: any notes about the implementation, etc.
- publish: [required] either "True" or "False" to determine if the experiment should be revealed to the user of the expfactory-python application.
- deployment_variables: a dictionary of customizations for the javascript template. For example, jspsych has a jspsych_init object that you may want to customize.
- run: [required] entry javascript and css files for the experiment. Paths here should all be relative to the experiment folder, and will be used to embed the experiments in different contexts. For example, the following files in run in the folder *multisource*:

```
"run": [
        "experiment.js",
        "style.css",
        "static/js/jspsych/plugins/plugin.js"
       ],
```

will produce the following in a rendered template:

```
<link rel='stylesheet' type='text/css' href='static/experiments/multisource/style.css
↪'>

<script src='static/js/jspsych/plugins/plugin.js'></script>
<script src='static/experiments/multisource/experiment.js'></script>
```

Note that the full path is a file served in our expfactory-battery (a plugin common to many experiments), and the local paths are files in the multisource experiment folder.

An example of a config.json data structure is as follows:

```
[
    {
        "name": "Model-Based Influences on Humans' Choices and Striatal Prediction␣
↪Errors",
        "exp_id": "two_stage_decision",
        "cognitive_atlas_task_id":"trm_4aae62e4ad209",
        "template":"jspsych",
        "contributors": [
```

(continues on next page)

(continued from previous page)

```
                        "Ian Eisenberg",
                        "Zeynep Enkavi",
                        "Patrick Fisher",
                        "Vanessa Sochat",
                        "Russell Poldrack"
                       ],
        "run": [
                "experiment.js",
                "style.css",
                "plugin.js"
               ],
        "reference": "http://www.sciencedirect.com/science/article/pii/
↪S0896627311001255",
        "notes": "Condition = ordered stims in stage 1 and stage 2 (so [0, 1] or [1,␣
↪0] for stage 1 and [2, 3], [4, 5] etc. for stage 2 and FB for the FB condition (1␣
↪for reward, 0 for no reward)",
        "publish": "True"
   }
]
```

experiment_variables

Here are examples of different kinds of experiment variables. If you do not have any, you don't need to define this field, or you can define like this:

```
"experiment_variables":"",
```

A boolean variable that is to be used to calculate bonus for a task

```
"experiment_variables": [{
                          "name":"passed_check",
                          "type":"bonus",
                          "datatype":"boolean"
                          "description":"JavaScript boolean to indicate the␣
↪participant passed the check."
                         }]
```

A numeric variable (both int and float are stored as float) to be used to assess if credit should be given (or not)

```
"experiment_variables": [{
                          "name":"number_correct",
                          "type":"credit",
                          "datatype":"numeric",
                          "range":[0,100],
                          "description":"the total number of correct choices."
                         }]
```

A string variable that will be included as a model (we will be including other ways to explore results, not implemented for now) but not intended for allocation of credit / rejection, or bonus:

```
"experiment_variables": [{
                          "name":"quality_label",
                          "datatype":"string",
                          "range":["low","average","good","very good"],
```

(continues on next page)

(continued from previous page)

```
                                "description":"The quality of the responses."
                            }]
```

deployment_variables

A deployment variable is specific to the javascript template / architecture that the experiment is using. We only support jspsych, and so you can only specify "jspsych_init." You do not need to define this variable if you don't have any customizations, but here is how to define an empty variable:

```
"deployment_variables":""
```

Specify a field in jspsych_init

```
"deployment_variables":{"jspsych_init":
                                            {
                                              "max_load_time":40
                                            }
                        }
```

run

A list of javascript and css files that are essential for the experiment to run, specified in *load_experiments.js* (see details above). Typically, an experiment will have an *experiment.js* file and a *style.css*. For jspsych files that are included with the expfactory-battery/js folder, specify the complete path to the file relative to the static folder. For example:

```
static/js/jspsych/plugins/jspsych-call-function.js
```

Any files with full paths specified as the above will be checked for existance within the expfactory-battery folder. If found, the file will be linked successfully. If not found, the file will be looked for in the experiment folder. If the file does not exist in either place, an error will trigger upon generation of the battery.

5.3 Contributing to experiments

An experiment is just a folder with files that are expected to be a certain way. The "core" of an experiment is:

- config.json: a file with a bunch of information about an experiment, meta-data
- experiment.js: the main javascript file to run the experiment
- style.css: (optional) custom styling

A recommended strategy for developing a new experiment is to find an experiment similar to the one you want to make, copy the folder, and edit it. We recommend using JsPsych for tutorials, help, and examples, as the documentation is really great. We also provide an standard reaction time (commented) empty template for you to download, and in the future will provide a dynamic web interface for generating new templates. When you submit a PR to the expfactory-experiments repo, the experiment will be tested with continuous integration, and before doing this, we have provided a simple command line tool that can be used to validate the basics about the config.json and experiment folder.

```
cd my_new_experiment
expfactory --preview   # preview / run the experiment in the browser within the PWD
expfactory --validate  # will validate config.json and experiment structure
expfactory --test      # will run test robot we run on continuous integreation
```

5.3.1 Summary of Best Practices

- An experiment must minimally have an experiment.js and valid config.json file
- We use jspsych plugins for most experiments, and most are included with the battery repo, meaning you don't need to include them with your local folder, but rather specify their path in the "run" variable of your config.json (see below).
- you can include any images/sounds supplementary files in your experiment folder, it will be included
- these supplementary files specified in experiment.js should have paths relative to the battery experiment base directory, *static/experiments/[exp_id]/images/hello.png*
- supplementary files specified in style.css should be relative to the experiment folder.

Files hard coded into the experiment.js should have a path with the following format:

```
/static/experiments/[folder-name]/images/
```

where *folder-name* is replaced with the name of the experiment folder, and any subdirectories to that (e.g., "images" should exist in the experiment directory. For example, to link a sound file in an experiment folder, *tone-monitoring*:

```
practice_stims = [{sound: '/static/experiments/tone_monitoring/sounds/880Hz_-6dBFS_.
↪5s.mp3',
             data: {exp_id: 'tone_monitoring', trial_id: 'high', condition: 'practice'}
↪},
            {sound: '/static/experiments/tone_monitoring/sounds/440Hz_-6dBFS_.5s.mp3',
             data: {exp_id: 'tone_monitoring', trial_id: 'medium', condition: 'practice
↪'}},
            {sound: '/static/experiments/tone_monitoring/sounds/220Hz_-6dBFS_.5s.mp3',
            data: {exp_id: 'tone_monitoring', trial_id: 'low', condition: 'practice'}}
]
```

These files would be in the tone-monitoring experiment folder as:

```
tone_monitoring/
    sounds/
            880Hz_-6dBFS_.5s.mp3
            440Hz_-6dBFS_.5s.mp3
            220Hz_-6dBFS_.5s.mp3
```

as the entire thing will be included in the experiment's folder (under */static/experiments/[folder-name]*), with the same structure. You are free to include any necessary static files in your experiment, and can name subfolders in your experiment folder however you like. Most of these files do not need to be specified in the "run" variable of the config.json. These specified files should be those js and css that will be embedded in the dynamically produced experiment html so that the experiment functions. If you find that your experiment is not running, likely you can inspect the console and see that you are missing a file. When validation of an experiment occurs upon battery generation, all files specified in this "run" variable will be checked for existence, and an error output if not found.

5.3.2 experiment.js

This is the main javascript file to run the experiment. This file should include set up of the JsPsych experiment, and the requirement is that the resulting array of blocks be named {{exp_id}}_experiment (for example, if the folder is called "local_global" the array that finishes the file, onto which all blocks are pushed, is called "local_global_experiment." When defining your blocks, in the "data" variable you must include "exp_id" to also specify this unique identifier (e.g., local global). Not including the unique ID with the data will trigger an error during continuous integration testing, and will possibly mean that you could have data that cannot be linked back to a task.

Paths to images, sounds, and other files referenced in this file will be expected to follow the same pattern as above, e.g.: */static/experiments/[folder-name]/images/filename.png (required)*.

5.3.3 Supplementary files

You might want a folder of images, sounds, or other, to be included with your experiment. You are allowed to include, and specify these files as needed in the style.css or experiment.js. Important** if you specify an image/other in the style sheet, the path should be relative to the css file (the experiment folder). If you specify an image or file in your experiment.js file, the path should be relative to the battery base folder (e.g., /static/experiments/[folder-name]/images/.

5.3.4 style.css

Is the main style file for the experiment, which will be copied into the battery style folder and linked appropriately. Images that are defined in this file should have paths relative to the images folder. *(required)*.

5.4 Contributing to surveys

A survey is the easiest of our paradigms to contribute to, as an entire survey is just a tab delimited text file paired with a config.json in a folder in expfactory-surveys. In the config.json, "template" should be "survey". The easiest way to understand the tab delimited file is to look at an example.

You will notice the following columns in the header, which is the first row of the file:

- **question_type**: can be one of textfield, numeric (a numeric text field), radio, checkbox, or instruction. These are standard form elements, and will render in the Google Material Design Lite style.
- **question_text**: is the text content of the question, e.g., How do you feel when you wake up in the morning?
- **required**: is a boolean (0 or 1) to indicate if the participant is required to answer the question (1) or not (0) before moving on in the survey.
- **page_number**: determines the page that the question will be rendered on. If you look at an example survey you will notice that questions are separated by Next / Previous tabs, and the final page has a Finish button. It was important for us to give control over pagination to preserve how some "old school" questionnaires were presented to participants.
- **option_text**: For radio and checkboxes, you are asking the user to select from one or more options. These should be the text portion (what the user sees on the screen), and separated by commas (e.g, Yes,No,Sometimes. Note: these fields are not required for instructions or textbox types, and can be left as empty tabs.
- **option_values**: Also for radio and checkboxes, these are the data values that correspond to the text. For example, the option_text Yes,No may correspond to 1,0. Again, this field is typically blank for instructions or textbox types.

5.5 Contributing to games

Games are under development, and currently we are using "phaser" as a first template. As with surveys and experiments, you are required to have a config.json, and the "template" variable should be defined as "phaser". Standards are under development, and an updated list can be found at expfactory-games repo. Currently, the following is required:

- the main game file should be called Run.js
- the boolean variable "finished" included with every trial (0 to continue/1 to finish)

- the game should have an inputData function that updates a data object when appropriate.

While we further develop the standard, please see a finished game (code/live)as an example.

5.6 Testing

It makes most sense to work directly from a folder in the expfactory-experiments repo that you have cloned. This method will work if you have an internet connection, as the expfactory executable will quickly plug your experiment into a battery, and open up the page in your browser. To do this, you can cd into your experiment folder, and use the *–run* command. This command can be used to run or preview a single experiment locally:

```
cd simple_experiment
expfactory --run
```

You can also specify the folder as an argument:

```
expfactory --run --folder /home/vanessa/Desktop/simple_rt
```

You can also specify a port:

```
expfactory --run --port=8088
```

Your browser will open to show the experiment. You can press Control+C to shut down the server.

As stated previously, you should validate your config.json, folder structure and files, by running:

```
cd simple_experiment
expfactory --validate
```

This function also works to specify a folder

```
expfactory --validate --folder /home/vanessa/Desktop/simple_rt
```

You can use the experiment robot to test your experiment locally. The same functions are used that will be run for continuous integration, and it is a good idea to make sure that no errors are output before submitting a PR.

```
cd simple_experiment
expfactory --test
```

or to specify a folder

```
expfactory --test --folder /home/vanessa/Desktop/simple_rt
```

Finally, if you add a new experiment, you will need to add a line to test it in the circle.yml. You can test the circle.yml before pushing as follows:

```
from expfactory.tests import validate_circle_yml
validate_circle_yml('expfactory-experiments')
```

Continuous integration is a term from software development that basically means we test as we change and modify our code - in this case “code” is referring to the different experiments. This means that when you submit a pull request (PR) to the expfactory-experiments repo, along with local testing that you might do, we also test the code on a continuous integration server called CircleCI. You will see a link to your PR from Github as soon as the PR is submit. Your contribution will be assessed based on the following:

- validation tests passing, meaning that your config.json and experiment folder are formatted and named properly

- jshint: we test the static code with jshint. If there are issues for your experiment, tests will not pass.
- robot tests: we have developed a robot that can be run locally or with continuous integration. The robot will click through the task, make random selections, and trigger an error if a 404 is found.
- experiment running: If you click on "Artifacts" tab at the top when the testing finishes, you will see a hierarchy of the machine, and you can go to ubuntu/expfactory-experiments/web/index.html to open up a browser and see the dynamically generated updated experiment set. If you click on "browse our experimental paradigms" from the box in the top right corner, and you should be able to find your experiment in the table and preview it. Note that the first tree view of the experiments points to expfactory.github.io. You should open your developer console, as you would on your local machine, and run through the experiment both looking for errors (that might have been detected by the robot) and seeing that things run as expected.

Currently, we only test changes between the last successful build and your contribution, meaning experiment folders with changed files. Since the entire battery would take upwards of 10 hours, we recommend you submit PRs limited to a few experiments, otherwise the running will take a very long time.

If you want to work offline, or simply want to develop an experiment using psiturk, then you will want to take the following approach:

- use the expfactory tool to generate a battery
- move the battery folder from your tmp directory to where you want it
- copy experiment folders / create new folders into the battery "/static/experiments" folder
- test and debug using psiturk (instructions not provided here)
- move the folder back to your expfactory-experiments repo, commit changes, and push as normal.

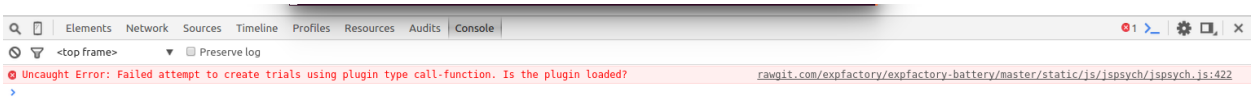
5.6.1 Getting Help

Please submit an issue to our code base, or email someone in the lab, and we will do our best to help. Please remember that we are a small group of (mostly) graduate students, and will respond as promptly as we can.

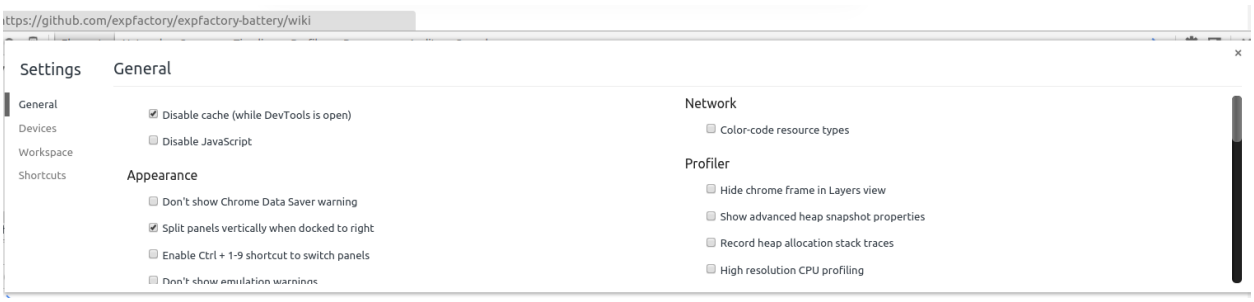
5.6.2 Debugging

If you see a white screen, it usually means there is a JavaScript error. While we can't provide details here for how to debug everything, generally if you right click –> Inspect Element and click the "console" tab you will see red text that indicates some JavaScript error. Questions to ask:

- Did you specify all required jspsych scripts in your "run" variable? A script that is not specified will not be included in the page, and you will likely see a message like this:



- Is the jspsych script included with the battery? If you specify a path in the run variable in config.json and it's not found, it's likely not included in the battery repo, and you can look to check. If you are missing a script and would like one added, please submit a PR to the repo and it will be done, and this will fix your error.
- Is your browser caching an old script? You might make a change, but then the error persists, and this is because the browser caches style and js files by default. You can disable this in the settings, so it won't cache when the development console is open:



or you can use Incognito mode (Control + Shift + N).

- Do you have a syntax error or a missing variable?
- What is the scope of your variable? If you reference a variable inside a function, it will not be available outside of that function. This is called scope.

Debugging JavaScript is hard and annoying. We chose to use jspsych rather arbitrary, and please let us know if you want support for other libraries (meaning we provide them in the battery repo). May the force be with you.

5.7 Contributing to this documentation

The documentation lives in the "doc" folder. Specifically, this is sphinx documentation that gets built automatically. The "build" folder contains the output that we serve on github pages, and this works by linking the index.html in the repo base to doc/build/html. The "source" folder contains files that you will want to edit.

You will need to install sphinx to work with these files

```
sudo pip install sphinx
```

And familiarize yourself with the restructured text (rst) documentation syntax.

The basic infrastructure is set up, so you should be able to edit the places that you need, and if you need help with something more advanced (and you can't figure it out) contact someone else in the Poldracklab or just submit *an issue <https://github.com/expfactory/expfactory-python/issues>_*.

5.7.1 Basic workflow

1. Edit a file
2. Enter the documentation base folder

```
cd doc
make html
```

Note that you should READ the error messages in the console when you build - it will tell you if you forgot to include files, or have any other syntax errors. Then:

```
cd build/html
python -m SimpleHTTPServer 9999
```

This will start a local web server using python at localhost:9999.

3. Open your browser to localhost:9999 to preview

CHAPTER 6

Tools Available

6.1 expfactory-python

A python module for using these tools, including functions to compile experiments. into a psiturk battery. and deploy in a virtual machine.

6.2 expfactory-battery

An instance of a psiturk experiment that contains a specific set of experiments, chosen by the user.

6.3 expfactory-experiments

The experiments deployed into the battery.

6.3.1 Modules

expfactory package

Submodules

expfactory.battery module

battery.py: part of expfactory package Functions to generate batteries

`expfactory.battery.`**`add_custom_variables`**(*custom_variables*, *template*)

add_custom_variables takes a list of tuples and a template, where each tuple is a (“[TAG]”,”substitution”) and the template is an open file with the tag. :param custom_variables: a list of tuples (see description above) :param template: an open file to replace “tag” with “substitute”

expfactory.battery.**generate**(*battery_dest=None*, *battery_repo=None*, *experiment_repo=None*, *experiments=None*, *config=None*, *make_config=True*, *warning=True*, *time=30*)

will create a battery from a template and list of experiments :param battery_dest: is the output folder for your battery. This folder MUST NOT EXIST. If not specified, a temp folder is created :param battery_repo: location of psiturk-battery repo to use as a template. If not specified, will be downloaded to a temporary directory :param experiment_repo: location of a expfactory-experiments repo to check for valid experiments. If not specified, will be downloaded to a temporary directory :param experiments: a list of experiments, meaning the "exp_id" variable in the config.json, to include. This variable also conincides with the experiment folder name. :param config: A dictionary with keys that coincide with parameters in the config.txt file for a expfactory experiment. If not provided, a dummy config will be generated. :param make_config: A boolean (default True) to control generation of the config. If there is a config generated before calling this function, this should be set to False. :param warning: Show config.json warnings when validating experiments. Default is True :param time: maximum amount of time for battery to endure (default 30 minutes) to select experiments

expfactory.battery.**generate_base**(*battery_dest*, *tasks=None*, *experiment_repo=None*, *survey_repo=None*, *game_repo=None*, *add_experiments=True*, *add_surveys=True*, *add_games=True*, *battery_repo=None*, *warning=True*)

generate_base returns a folder with downloaded experiments, surveys, and battery, either specified by the user or a temporary directory, to be used by generate_local and generate (for psiturk) :param battery_dest: [required] is the output folder for your battery. This folder MUST NOT EXIST. :param battery_repo: location of psiturk-battery repo to use as a template. If not specified, will be downloaded to a temporary directory :param experiment_repo: location of a expfactory-experiments repo to check for valid experiments. If not specified, will be downloaded to a temporary directory :param survey_repo: location of a expfactory-surveys repo to check for valid surveys. If not specified, will be downloaded to a temporary directory :param tasks: a list of experiments and surveys, meaning the "exp_id" variable in the config.json, to include. This variable also conincides with the tasks folder name. :param warning: show warnings when validating experiments (default True)

expfactory.battery.**generate_config**(*battery_dest*, *fields*)

takes a dictionary, and for matching fields, substitues and prints to "config.txt" in a specified battery directory :param battery_dest: should be the copied, skeleton battery folder in generation :param fields: should be a dictionary with fields that match those in the config non matching fields will be ignored.

expfactory.battery.**generate_local**(*battery_dest=None*, *subject_id=None*, *battery_repo=None*, *experiment_repo=None*, *experiments=None*, *warning=True*, *time=30*)

generate_local deploys a local battery will create a battery from a template and list of experiments :param battery_dest: is the output folder for your battery. This folder MUST NOT EXIST. If not specified, a temp directory will be used :param battery_repo: location of psiturk-battery repo to use as a template. If not specified, will be downloaded to a temporary directory :param experiment_repo: location of a expfactory-experiments repo to check for valid experiments. If not specified, will be downloaded to a temporary directory :param experiments: a list of experiments, meaning the "exp_id" variable in the config.json, to include. This variable also conincides with the experiment folder name. :param subject_id: The subject id to embed in the experiment, and the name of the results file. If none is provided, a unique ID will be generated. :param time: Maximum amount of time for battery to endure, to select experiments

expfactory.battery.**get_concat_js**(*valid_experiments*)

Return javascript concat section for valid experiments, based on psiturk.json :param valid_experiments: full paths to valid experiments to include

..note:

```
        case "simple-rt":
                experiments = experiments.concat(simple-rt_experiment)
                break;
        case "choice-rt":
```

(continues on next page)

(continued from previous page)

```
                experiments = experiments.concat(choice-rt_experiment)
                break;

Format for experiment variables is [exp_id]_experiment
```

expfactory.battery.**get_config**()
load in a dummy config file from expfactory

expfactory.battery.**get_experiment_run**(*valid_experiments*, *deployment='local'*)
returns a dictionary of experiment run code (right now just jspsych init objects) :param valid_experiments: full path to valid experiments folders, OR a loaded config.json (dict)

expfactory.battery.**get_load_js**(*valid_experiments*, *url_prefix=''*)
Return javascript to load list of valid experiments, based on psiturk.json :param valid_experiments: a list of full paths to valid experiments to include

..note::

Format is:

{

case "simple_rt": loadjscssfile("static/css/experiments/simple_rt.css","css") loadjscssfile("static/js/experiments/simple_rt.js","js") break;

case "choice_rt": loadjscssfile("static/css/experiments/choice_rt.css","css") loadjscssfile("static/js/experiments/choice_rt.js","js") break;

... }

expfactory.battery.**get_load_static**(*valid_experiments*, *url_prefix=''*, *unique=True*)
return the scripts and styles as <link> and <script> to embed in a page directly :param unique: return only unique scripts [default=True]

expfactory.battery.**get_timing_js**(*valid_experiments*)
Produce string (json / dictionary) of experiment timings :param valid_experiments: a list of full paths to valid experiments to include

..note:

```
Produces the following format for each experiment

{name:"simple_rt", time: 3.5}, {name:"choice_rt", time: 4}, ...
```

expfactory.battery.**move_experiments**(*valid_experiments*, *battery_dest*, *repo_type='experiments'*)
Moves valid experiments into the experiments folder in battery repo :param valid_experiments: a list of full paths to valid experiments :param battery_dest: full path to battery destination folder :param repo_type: the kind of task to move (default is experiments)

expfactory.battery.**template_experiments**(*battery_dest*, *battery_repo*, *valid_experiments*, *template_load=None*, *template_exp=None*, *template_exp_output=None*, *custom_variables=None*)
template_experiments: For each valid experiment, copies the entire folder into the battery destination directory, and generates templates with appropriate paths to run them :param battery_dest: full path to destination folder of battery :param battery_repo: full path to psiturk-battery repo template :param valid_experiments: a list of full paths to experiment folders to include :param template_load: the load_experiments.js template file. If not specified, the file from the battery repo is used. :param template_exp: the exp.html template file that runs load_experiment.js. If not specified, the psiturk file from the battery repo is used. :param template_exp_output:

The output file for template_exp. if not specified, the default psiturk templates/exp.html is used :param custom_variables: A dictionary of custom variables to add to templates. Keys should either be "exp" or "load", and values should be tuples with the first index the thing to sub (eg, [SUB_THIS_SUB]) and the second the substitition to make.

expfactory.experiment module

experiment.py: part of expfactory package Functions to work with javascript experiments

expfactory.experiment.**check_acceptable_variables**(*experiment_name*, *field_dict*, *template*, *field_dict_key*)

check_acceptable_variables takes a field (eg, meta[0][field]) that has a dictionary, and some template key (eg, jspsych) and makes sure the keys of the dictionary are within the allowable for the template type (the key). :param experiment_name: the name of the experiment :param field_dict: the field value from the config.json, a dictionary :param field_dict_key: a key to look up in the field_dict, which should contain a dictionary of {"key":"value"} variables :param template: the key name, for looking up acceptable values using get_acceptable_values

expfactory.experiment.**check_boolean**(*experiment_name*, *value*, *variable_name*)

check_boolean checks if a value is boolean :param experiment_name: the name of the experiment :param value: the value to check :param variable_name: the name of the variable (the key being indexed in the dictionary)

expfactory.experiment.**dowarning**(*reason*)

expfactory.experiment.**find_changed**(*new_repo*, *comparison_repo*, *return_experiments=True*, *repo_type='experiments'*)

find_changed returns a list of changed files or experiments between two repos :param new_repo: the updated repo - any new files, or changed files, will be returned :param comparison_repo: the old repo to compare against. A file changed or missing in this repo in the new_repo indicates it should be tested :param return_experiments: return experiment folders. Default is True. If False, will return complete file list

expfactory.experiment.**get_acceptable_values**(*package_name*)

expfactory.experiment.**get_experiments**(*experiment_repo*, *load=False*, *warning=True*, *repo_type='experiments'*)

return loaded json for all valid experiments from an experiment folder :param experiment_repo: full path to the experiments repo :param load: if True, returns a list of loaded config.json objects. If False (default) returns the paths to the experiments :param repo_type: tells the user what kind of task is being parsed, default is "experiments," but can also be "surveys" when called by get_surveys

expfactory.experiment.**get_valid_templates**()

expfactory.experiment.**get_validation_fields**()

Returns a list of tuples (each a field)

..note:

```
specifies fields required for a valid json
(field,value,type)
field: the field name
value: indicates minimum required entires
       0: not required, no warning
       1: required, not valid
       2: not required, warning
type: indicates the variable type
```

expfactory.experiment.**load_experiment**(*experiment_folder*)

load_experiment: reads in the config.json for an :param experiment folder: full path to experiment folder

expfactory.experiment.**load_experiments**(*experiment_folders*)
a wrapper for load_experiment to read multiple experiments :param experiment_folders: a list of experiment folders to load, full paths

expfactory.experiment.**make_lookup**(*experiment_list*, *key_field*)
returns dict object to quickly look up query experiment on exp_id :param experiment_list: a list of query (dict objects) :param key_field: the key in the dictionary to base the lookup key (str) :returns lookup: dict (json) with key as "key_field" from query_list

expfactory.experiment.**notvalid**(*reason*)

expfactory.experiment.**validate**(*experiment_folder=None*, *warning=True*)

Parameters

- **experiment_folder** – full path to experiment folder with config.json
- **warning** – issue a warning for empty fields with level 2 (warning)

..note:

```
takes an experiment folder, and looks for validation based on:

- config.json
- files existing specified in config.json

All fields should be defined, but for now we just care about run scripts
```

expfactory.interface module

expfactory.utils module

utils.py: part of expfactory package

expfactory.utils.**clean_fields**(*mydict*)
Ensure utf-8

expfactory.utils.**copy_directory**(*src*, *dest*)
Copy an entire directory recursively

expfactory.utils.**find_directories**(*root*, *fullpath=True*)
Return directories at one level specified by user (not recursive)

expfactory.utils.**find_subdirectories**(*basepath*)
Return directories (and sub) starting from a base

expfactory.utils.**get_installdir**()

expfactory.utils.**get_template**(*template_file*)
get_template: read in and return a template file

expfactory.utils.**get_url**(*url*)
return a url as string

expfactory.utils.**is_type**(*var, types=[<type 'int'>, <type 'float'>, <type 'list'>]*)
Check type

expfactory.utils.**remove_unicode_dict**(*input_dict*)
remove unicode keys and values from dict, encoding in utf8

expfactory.utils.**save_pretty_json**(*outfile*, *myjson*)

expfactory.utils.**save_template**(*output_file*, *html_snippet*)

expfactory.utils.**sub_template**(*template*, *template_tag*, *substitution*)
make a substitution for a template_tag in a template

expfactory.scripts module

script.py: part of expfactory api Runtime executable

expfactory.scripts.**main**()

expfactory.views module

views.py

part of the experiment factory package functions for developing experiments and batteries, viewing and testing things

expfactory.views.**embed_experiment**(*folder*, *url_prefix=''*)
return an html snippet for embedding into an application. This assumes the same directory structure, that all jspsych files can be found in static/js/jspych, and experiments under static/experiments/[folder] :param folder: full path to experiment folder with config.json

expfactory.views.**generate_experiment_web**(*output_dir*, *experiment_folder=None*, *survey_folder=None*, *games_folder=None*, *make_table=True*, *make_index=True*, *make_experiments=True*, *make_data=True*, *make_surveys=True*, *make_games=True*)
get_experiment_table Generate a table with links to preview all experiments :param experiment_folder: folder with experiments inside :param survey_folder: folder with surveys inside :param games_folder: folder with games inside :param output_dir: output folder for experiment and table html, and battery files :param make_table: generate table.html :param make_index: generate d3 visualization index :param make_experiments: generate experiment preview files (linked from table and index) :param make_data: generate json/tsv data to download :param make_surveys: generate static files for surveys repos :param make_games: generate static files for games repos

expfactory.views.**get_cognitiveatlas_hierarchy**(*experiment_tags=None*, *get_html=False*)
return :param experiment_tags: a list of experiment exp_id tags to include. If None provided, all valid experiments will be used. :param get_html: if True, returns rendered HTML template with hierarchy. False returns json data structure.

expfactory.views.**get_experiment_html**(*experiment*, *experiment_folder*, *url_prefix=''*, *deployment='local'*)
return the html template to test a single experiment :param experiment: the loaded config.json for an experiment (json) :param experiment_folder: the experiment folder, needed for reading in a survey :param url_prefix: prefix to put before paths, in case of custom deployment :param deployment: deployment environment, one of docker, docker-preview, or local [default]

expfactory.views.**preview_experiment**(*folder=None*, *battery_folder=None*, *port=None*)
preview an experiment locally with the –preview tag (for development) :param folder: full path to experiment folder to preview. If none specified, PWD is used :param battery_folder: full path to battery folder to use as a template. If none specified, the expfactory-battery repo will be used. :param port: the port number, default will be randomly generated between 8000 and 9999 :param robot: if True, a web server is started as a separate process for a robot to run

expfactory.views.**run_battery**(*destination=None*, *experiments=None*, *experiment_folder=None*, *subject_id=None*, *battery_folder=None*, *port=None*, *time=30*)
run_battery runs or previews an entire battery locally with the –run tag. If no experiments are provided, all

in the folder will be used. :param destination: destination folder for battery. If none provided, tmp directory is used :param experiments: list of experiment tags to add to battery :param experiment_folder: the folder of experiments to deploy the battery from. :param subject_id: subject id to embed into battery. If none, will be randomly generated :param battery_folder: full path to battery folder to use as a template. If none specified, the expfactory-battery repo will be used. :param port: the port number, default will be randomly generated between 8000 and 9999 :param time: total number of minutes for experiments to add to battery

expfactory.views.**run_single**(*exp_id*, *repo_type*, *destination=None*, *source_repo=None*, *battery_repo=None*, *port=None*, *subject_id=None*)

run_survey runs or previews an entire battery locally with the –run tag. If no experiments are provided, all in the folder will be used. :param destination: destination folder for battery. If none provided, tmp directory is used :param exp_id: exp_id for survey, experiment, or game to run (unique ID) :param repo_type: must be within experiments,surveys, or games :param source_repo: a source repo for the experiment, survey, or game :param subject_id: subject id to embed into result. If none, will be randomly generated :param battery_folder: full path to battery folder to use as a template. If none specified, the expfactory-battery repo will be used. :param port: the port number, default will be randomly generated between 8000 and 9999

expfactory.views.**tmp_experiment**(*folder=None*, *battery_folder=None*)

generate temporary directory with experiment :param folder: full path to experiment folder to preview (experiment, survey, or game). If none specified, PWD is used :param battery_folder: full path to battery folder to use as a template. If none specified, the expfactory-battery repo will be used.

expfactory.vm module

vm.py: part of expfactory package Functions to generate virtual machines to run expfactory batteries

expfactory.vm.**add_custom_logo**(*battery_repo*, *logo*)

Add a custom logo to the vm battery :param battery_repo: the full path to the battery repo base, assumed to have an "img" folder :param logo: the full path to the logo to copy. Should ideally be png.

expfactory.vm.**custom_battery_download**(*tmpdir=None, repos=['experiments', 'battery', 'surveys', 'games']*)

Download battery and experiment repos to a temporary folder to build a custom battery, return the path to the tmp folders :param tmpdir: The directory to download to. If none, a temporary directory will be made :param repos: The repositories to download, valid choices are "experiments" "battery" and "vm"

expfactory.vm.**download_repo**(*repo_type*, *destination*)

Download a expfactory infrastructure repo "repo_type" to a "destination" :param repo_type: can be one of "experiments" "battery" "vm" :param destination: the full path to the destination for the repo

expfactory.vm.**generate_database_url**(*dbtype=None*, *username=None*, *password=None*, *host=None*, *table=None*, *template=None*)

Generate a database url from input parameters, or get a template :param dbtype: the type of database, must be one of "mysql" or "postgresql" :param username: username to connect to the database :param password: password to connect to the database :param host: database host :para table: table in the database :param template: can be one of "mysql" "sqlite3" or "postgresql" If specified, all other parameters are ignored, and a default database URL is produced to work in a Vagrant VM produced by expfactory

expfactory.vm.**get_jspsych_init**(*experiment*, *deployment='local'*, *finished_message=None*)

return entire jspsych init structure :param experiment: the loaded config.json for the experiment :param deployment: specify to deploy local (default), or docker-mturk,docker-local (expfactory-docker). :param finished_message: custom message to show at the end of the experiment with Redo and Next Experiment Buttons

expfactory.vm.**get_stylejs**(*experiment*, *url_prefix=''*)

return string for js and css scripts to insert into a page based on battery path structure

expfactory.vm.**prepare_vm**(*battery_dest*, *fields=None*, *vm_repo=None*, *vm_type='vagrant'*)

Prepare virtual machine to run local with vagrant, or with vagrant-aws :param battery_dest: the battery destina-

tion folder :param fields: if specified, will be added to the config.txt :param vm_repo: the expfactory-vm repo to use for templates. If not provided, one will be downloaded to a temporary directory for use. :param vm_type: one of "vagrant" or "aws" Default is "vagrant" meaning a localvirtual machine with vagrant.

expfactory.vm.**specify_experiments**(*battery_dest*, *experiments*)

Specify experiments for a Vagrantfile in an output folder :param battery_dest: destination folder for battery :param experiments: a list of experiment tags to include

Module contents

CHAPTER 7

About

The Experiment Factory is developed by the Poldracklab at Stanford University, and all code is publicly available on Github. The current landscape of behavioral paradigms is messy at best, and so the Experiment Factory is an effort to standardize, organize, and collaboratively develop experiments that can be run on multiple infrastructures. Currently supported is generating experiments in a battery intended to run with psiturk on a local computer, a vagrant virtual machine, or Amazon Web Services. We intend to add support for docker and other web applications soon, and welcome your contributions to experiments, surveys, and games, along with the infrastructure for psiturk battery generation, virtual machines, and the software itself.

We are called the Experiment Factory because we want the software to support flexibility in designing the experiments, infrastructure, and deployment for your experiments. Psychology and neuroscience researchers should have the same modern tools available to them as a commercial effort, and long gone should be the days of passing around paradigms from computer to computer on USB sticks. If we work together we can refine experiments to make them better, and making behavioral measurments with the same thing is one step toward a vision of reproducible science.

7.1 License

The Experiment Factory code is licensed under the MIT open source license, which is a highly permissive license that places few limits upon reuse. This ensures that the code will be usable by the greatest number of researchers, in both academia and industry.

7.2 Citation

Please cite the Experiment Factory paper if you use it in your work:

Sochat VV, Eisenberg IW, Enkavi AZ, Li J, Bissett PG and Poldrack RA (2016). The Experiment Factory: standardizing behavioral experiments. Front. Psychol. 7:610. doi: 10.3389/fpsyg.2016.00610

7.3 Integrations and Tools

7.3.1 Psiturk

These experiments are intended to run via the psiturk infrastructure, and the experiment factory battery is the main experiment code that, when fit with experiments, can run on Amazon Mechanical Turk.

7.3.2 Phaser

We are collaborating with the Stanford Cognitive and Systems Neuroscience Lab to make simple, fun HTML event driven games that are integrated into the Experiment Factory frameowrk. This work is in its infancy, and it's going to be fun to see where we take it!

7.3.3 Material Design Lite

We use Google Material Design Lite as our template for forms, as it offers a beautiful user experience, and have implemented a basic validation for it and pagination using JQuery Wizard.

7.3.4 JsPsych

We use the JsPsych javascript framework to design our javascript experiments. While the current infrastructure is extendable to other frameworks, we have chosen JsPsych for our initial core of experiments.

7.4 Cognitive Atlas

Each experiment (task) is defined in the Cognitive Atlas, as are the cognitive concepts that the experiment measures. This means that, as more experiments are generated, a user can generate a battery intended to measure one or more cognitive concepts of interest. You can browse different tasks and cognitive concepts on the Cognitive Atlas, and specify their unique IDs in the experiment config.json files.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

M

N

P

R

S

T

V