Contents

1 User's Guide
   1.1 Forward .......................................................... 3
   1.2 Quickstart ......................................................... 4
   1.3 Installation ....................................................... 8
   1.4 Setting Up an Amazon Mechanical Turk Account .............. 10
   1.5 Configuration Overview ......................................... 12
   1.6 Settings .......................................................... 13
   1.7 Command-line Interface Overview ............................... 21
   1.8 psiTurk commands ............................................... 22
   1.9 Databases Overview ............................................. 34
   1.10 Tutorials ......................................................... 36
   1.11 Anatomy of a basic psiTurk project ............................ 47
   1.12 Recording Data .................................................. 56
   1.13 Retrieving Data .................................................. 57
   1.14 Customizing psiTurk ............................................ 59
   1.15 Dashboard Overview ............................................ 63
   1.16 Experiment Exchange ........................................... 65
   1.17 Frequently Asked Questions .................................... 75
   1.18 Getting help ...................................................... 77
   1.19 Disclaimer ....................................................... 78

2 Contributing to psiTurk
   2.1 Contributing to psiTurk .......................................... 79
   2.2 Project Roadmap .................................................. 82

3 API Reference
   3.1 psiturk.js API .................................................... 85

4 Citing ............................................................. 89

5 License ............................................................. 91
Welcome to psiTurk’s documentation. To learn more about the project, please visit https://psiturk.org.

To actually use psiTurk, you’ll first need to install it on your local computer or server. Instructions can be found on the Installation page. Afterwards, head over to our Quickstart.

For a more detailed tutorial demonstrating how to setup a simple experiment with psiTurk, visit Example project walkthrough. Also, check out the Experiment Exchange for examples to learn from, including demonstrations of integration with jspsych.

Contents

- User’s Guide
- Contributing to psiTurk
- API Reference
- Citing
- License
CHAPTER 1

User’s Guide

1.1 Forward

Read this if you want to find out more about Amazon Mechanical Turk (AMT) and how psiTurk can help you run web-based experiments on AMT painlessly and quickly. This section will also tell you what problems psiTurk does and does not solve to help you gauge whether it will be useful to you.

1.1.1 Understanding the psiTurk design philosophy: An analogy

Back before music was entirely digital people got their music on cassette tapes. To play the cassette you needed a player device (e.g., walkman or boombox). People would trade tapes, make copies of tapes, make mixtapes of their favorite songs. It was awesome.

psiTurk is like a player but instead of playing music, it plays (i.e., runs) experiments. You download and install the psiTurk application to your computer. This installs a command line tool `psiturk` which serves as a multi-function “player.” It can (figuratively speaking) run, pause, eject, and configure a given experiment.

To make it useful though psiTurk needs something to play. You can download from our experiment exchange library an archive which contains all the files specific to a given experiment. You basically “play” the downloaded experiment using the `psiturk` command. You can easily switch experiments by downloading another experiment archive and “playing” it. Even better, you can make your own experiments by remixing others (borrowing code from projects in the experiment exchange) or building your own from scratch.

The goal of psiturk was to build the “player” so you can spend more of your time on the important part of your research... the experiment (i.e., mix tape)!

Oh, and in case you missed it, “playing” someone else’s experiment posted to the experiment exchange basically means independently replicating it!

1.1.2 What is Mechanical Turk?

Amazon Mechanical Turk (AMT) is an online platform that lets you post a wide variety of tasks to a large pool of participants. Instead of spending weeks to run experiments in the lab, it lets you collect data of a large number of
people within a couple of hours.

Some key terminology for understanding the AMT model:

- HIT (Human Intelligence Task) - A unit of work (e.g. a psychology experiment)
- Requester - The person or entity that posts HITs (e.g. a researcher or lab)
- Worker - The person that completes HITs (i.e. a participant in your study)

Workers get paid a fixed amount for each HIT which is determined by the requester. Requesters can also make bonus payments to specific workers. Amazon collects a 10% fee for each payment.

### 1.1.3 Why psiTurk?

AMT provides some very basic templates that you can use to design HITs (particularly questionnaires), but these will most likely not serve your purposes as an experimenter. The psiTurk toolbox is designed to help you run fully-customized and dynamic web-experiments on AMT. Specifically, it allows you to:

1. Run a web server for your experiment
2. Test your experiment
3. Interact with AMT to recruit, post HITs, filter, and pay participants (AMT workers)
4. Manage databases and export data

psiTurk also includes a powerful interactive command interface that lets you manage most of your AMT activity.

### 1.1.4 Basic Requirement: A publicly-accessible server

Be aware that you will need to host your experiment on a server to which your participants have access.

### 1.1.5 Do I have to learn how to code?

Yes. psiTurk experiments are run in web browsers. To develop a web browser experiment, you need to have basic web programming skills with HTML, CSS, and JavaScript.

To get you started, psiTurk provides a fully functioning example experiment in the Example project walkthrough section that you can use as a template for your own study. psiTurk also includes a library of basic Javascript functions (see psiturk.js API) that you can insert into your code to handle page transitions, load images, and record data.

### 1.2 Quickstart

Interested in psiTurk? Try out this quick start guide to running a simple experiment online! It steps you from installing to paying participants and should work for most people using updated versions of Linux or Mac OS X.

This guide uses a simple example experiment provided in the default psiTurk installation, but can be used to run any psiTurk experiment.

---

**Contents**

- **Step 1: Install psiTurk**
- **Step 2: Create a default project structure**
• Step 3: Enter credentials
• Step 4: Launch psiTurk in the new project directory
• Step 5: Start the server
• Step 6: Debug/view your experiment
• Step 7: Create a sandboxed HIT/Ad
• Step 8: Check your data
• Step 9: Monitor progress
• Step 10: Approve workers
• Step 11: Switch to “live” mode

1.2.1 Step 1: Install psiTurk

psiTurk can be installed easily on any system that has the python package manager pip.

```bash
$ pip install psiturk
```

See also:
Installation

1.2.2 Step 2: Create a default project structure

psiTurk includes a simple example project which you can use to get started.

```bash
$ psiturk-setup-example
Creating new folder `psiturk-example` in the current working directory...
Creating default configuration file (config.txt)
```

1.2.3 Step 3: Enter credentials

In order to get access to the Amazon Mechanical Turk features of psiTurk, you need obtain and enter credentials for accessing Amazon Web Services. Both of these can be added to ~/.psiturkconfig. You can leave the aws_region at the default value.

```bash
$ cat ~/.psiturkconfig

[AWS Access]
AWS_ACCESS_KEY_ID = YourAccessKeyId
AWS_SECRET_ACCESS_KEY = YourSecretAccessKey
aws_region = us-east-1
```

See also:
Setting Up an Amazon Mechanical Turk Account

1.2. Quickstart
1.2.4 Step 4: Launch psiTurk in the new project directory

psiTurk should be run in the top level folder of your project. You should be greeted with a welcome screen and command prompt.

There is also an extensive help system in the command line. Type help to see a list of available commands. Type help <cmd> to get more information about a particular command (e.g., help server).

```bash
$ cd psiturk-example
$ psiturk
welcome...
psiTurk version 2.1.1
Type "help" for more information.
[psiTurk server:off mode:sdbx #HITs:0]$
```

See also:

*Command-line Interface Overview*

1.2.5 Step 5: Start the server

The psiTurk server is the web server which responds to external requests. To start or stop the server type server on, server off, or server restart.

```bash
$ [psiTurk server:off mode:sdbx #HITs:0]$ server on
Experiment server launching...
Now serving on http://localhost:22362
[psiTurk server:on mode:sdbx #HITs:0]$
```

1.2.6 Step 6: Debug/view your experiment

To debug or test the experiment, simply type debug. This will launch the default web browser on your system and point it at your experiment in a method which is helpful for testing.

Hint: If you are running on a remote server and want to disable launching the browser type debug -p (print only) which will print the debugging URL but not launch the browser.

Altering the experiment code is beyond the scope of the quick start, but see *this guide* for details on how to modify and extend the stroop example.

```bash
$ [psiTurk server:on mode:sdbx #HITs:0]$ debug
Launching browser pointed at your randomized debug link, feel free to request another.
http://localhost:22362/ad?assignmentId=debugX12JJ8&hitId=debugA7NP2T&workerId=debugS9K039
[psiTurk server:on mode:sdbx #HITs:0]$
```

1.2.7 Step 7: Create a sandboxed HIT/Ad

In order to make the experiment available to workers on Amazon Mechanical Turk you need to:

1. Run your psiturk server on a machine that is publicly-accessible.
2. Post a HIT on AMT, which will point MTurkers to your psiturk server address.

The example below uses the Amazon Mechanical Turk “sandbox,” which is a place for testing your task without actually offering it live to real paid workers.

Your HIT should be visible on http://workersandbox.mturk.com if you search for your requester account name or the HIT title word “Stroop” (set in config.txt).

**Warning: Important!** Test to make sure that your Ad URL can be accessed from a place external to the network from which you created the HIT. If it cannot be accessed, then MTurkers won’t be able to access your HIT!

### 1.2.8 Step 8: Check your data

By default psiturk saves your data to a SQLite database participants.db in your base project folder. You can check that everything is being recorded properly by opening that file in a SQLite tool like Base.

See also:

*Databases Overview*

### 1.2.9 Step 9: Monitor progress

One simple way to monitor how many people have actually accepted your HIT is with the `hit list --active` or `hit list --reviewable` commands.

This shows the HITid for each task, how many have completed, and how many are pending.

See also:

See these FAQs:

- How do I interpret the hit list counts of “Pending,” “Complete,” and “Remain”?
- Immediately after I post my HIT on the “live” mode of AMT, I cannot find it via an mturk dashboard search?

### 1.2.10 Step 10: Approve workers

psiturk provides many tools for approving workers, assigning bonuses, etc. Try `help hit` and `help worker`.

One simple approach is to approve all the workers associated with a particular HIT (once all the assignments are complete). To do this, use the `worker approve --hit <HITID>` command.

```
$ [psiturk server:on mode:sdbx #HITs:0]$ worker approve --hit ˓→28K4SME3ZZ2M2I004SETTTXTTAG44LT
Approving....
```

### 1.2.11 Step 11: Switch to “live” mode

In order to create public hits on the “live” AMT site, you need to switch modes in the command shell using the `mode` command. To get back to the sandbox, just type `mode` again.

To avoid mistakes, psiturk defaults to sandbox mode (this behavior can be changed in config.txt)

From here, everything is exactly the same as described for sandbox hits above.
1.3 Installation

psiTurk is supported for python >= v3.6 on any Unix-Like operating system (i.e., not Windows).

When psiTurk is successfully installed, you will have a new command line tool available called psiturk. The psiturk command provides a number of functions to you including launching the server and interacting with the Mechanical Turk and Amazon Web Services (AWS) systems.

Requirements:

- python (>= v3.6)
- pip (to install, see here.)

To install the latest released version of psiTurk:

```
pip install psiturk
```

To upgrade psiturk:

```
pip install -U psiturk
```

1.3.1 From Source

You can install the bleeding edge version of psiTurk from source just as you would install any other Python package:

```
pip install git+https://github.com/NYUCCL/psiturk.git
```

To update from source:

```
pip install -U git+https://github.com/NYUCCL/psiturk.git
```

1.3.2 Running inside a Virtual Environment

It can desirable to keep each of your experiments’ dependencies (python and python package versions) isolated from each other. For example, if you want to install the development version of psiTurk (as described above) in one experiment, but not all the others installed on your system, Virtual Environments provide a solution.

You can install via pip:

```
sudo pip install virtualenv virtualenvwrapper
```

This will install the virtualenv tool as well as the supplementary virtualenvwrapper tools that make working with virtualenvs easier. You create a virtual environment as follows:
$ mkvirtualenv my-experiment

Running virtualenv with interpreter /usr/bin/python2
New python executable in my-experiment/bin/python2
Also creating executable in my-experiment/bin/python
Installing setuptools, pip...done.

(if `mkvirtualenv` is not recognized, follow the instructions [here](#))

Then, at any point in the future, to activate the virtual environment, use the `workon` command:

```bash
$ workon my-experiment
(my-experiment) $ which python python pip easy_install
~/.virtualenvs/my-experiment/bin/python
~/.virtualenvs/my-experiment/bin/pip
~/.virtualenvs/my-experiment/bin/easy_install
```

As you can see, when the environment is active, running `python` or `pip` will run copies specific to your project. Any packages installed with `pip` or `easy_install` will be installed inside your `my-experiment` virtualenv rather than system-wide.

Install psiTurk as above into the _virtual_ environment—i.e., with the virtual environment activated:

```bash
(my-experiment) $ pip install psiturk
```

You can use the `deactivate` command to leave the virtualenv.

### 1.3.3 System-specific notes

**Mac OS X**

Apple users will need to install a C compiler via XCode; to do so, install XCode from the App store. Once you have downloaded it, install the command line tools from the preferences menu as instructed [here](#). For earlier versions of Mac OS X (e.g., Snow Leopard) you may need to install XCode using the installation disc that came with your computer. The command line tools are an option during the installation process for these systems.

**Linux**

psiTurk is relatively painless to install on most Linux systems since the installation requirements come installed by default in most distributions.

If you encounter install problems when installing using `pip` as above, a likely cause is that you are missing the package from your distribution that contains a needed header file. In this case, one way to troubleshoot the problem is to do a web search for the name of your distribution and the name of the missing header file (which often appears in the error text produced by a failed pip install). That search will likely turn up the name of the package for your distribution that supplies the needed header file.

As an example, before installing psiTurk on a minimal Debian server (such as the one provided by many server hosting companies) you will need to install some additional packages, as illustrated by the following example command:

```bash
apt install python-pip python-dev libncurses-dev
```

If you would like to use mysql as your backend database (which is optional, and can be done at any time), further packages are needed. On a Debian system, they are:

1.3. Installation
apt install python-pymysql python-sqlalchemy libmysqlclient-dev

If you have additional specific issues, or if you can report the steps needed to install psiTurk on a particular Linux distribution, please help us update the documentation!

Windows

psiTurk is currently not supported on Windows. This is due to a technical limitation in the ability to run server processes on Windows. However, there are a number of options to get around this (see below for details on each option):

- Windows Subsystem for Linux (WSL) on Windows 10. **Recommended.**
- Virtualization through VirtualBox or similar software.

Windows Subsystem for Linux (WSL)

Windows now has the option to run a Linux translation layer inside Windows (WSL 1) or even a full Linux kernel (WSL 2). Either will allow you to run psiturk within the Linux subsystem. See https://docs.microsoft.com/en-us/windows/wsl/install-win10 for instructions on how to activate WSL on your system.

After you activate WSL and install a Linux distribution of choice, install psiturk within a WSL-connected command prompt as above for Linux.

Virtualization

**Warning:** WSL may not be compatible with concurrent usage of other hypervisors.

You can install a program like VirtualBox on your pc. Programs like these are called hypervisors and emulate a computer within your computer. Your physical machine is called a host and the virtual machine is called a guest. This technique allows you to install a Linux guest regardless of what OS the host is running.

Virtualization requires some computing power from the host so this option is not recommended if your psiturk experiment requires a lot of computing power as well or if it’s is expected to have a lot of participants active at once. However, it is a good option to develop and test your psiturk experiments on Windows systems prior to Windows 10. If you are running Windows 10 or higher see below for the WSL option, which is much easier on your system than virtualization.

After you install the virtual machine software you need an installation image for a Linux based OS. You can choose any Linux distribution you like but Ubuntu is a good choice if you don’t know which one to pick. You can usually obtain an *.iso file for the Linux distribution you like. These are virtual cd-roms. You can load them into your virtual machine and begin installing the guest OS. Once that is complete you boot your virtual machine into Linux and follow the installation steps for Linux.

1.4 Setting Up an Amazon Mechanical Turk Account

psiTurk can interface with Amazon Mechanical Turk (although it doesn’t have to!). To do so, you need to create an account on Amazon’s website in order to use it. There are a number of steps involved here which have to do with signing up with Amazon and creating several accounts. Luckily they are a one-time process for a given AWS account.
1.4.1 Accounts Creation and Linking

Carefully follow AWS’s guide for setting up the necessary accounts for using Amazon Mechanical Turk. Before doing so, note the following:

- **Step 5** discusses setting up the Developer Sandbox. Carefully follow all steps in this section, including the steps in the note for linking your aws account *specifically to the sandbox.*
- **Step 6** in the guide is “Set up an AWS SDK”. You may skip this step – psiTurk uses the Python/Boto (Boto3) SDK under the hood.
- **Step 7** in the guide suggests the option of enabling AWS Billing for your account. However, at least one psiTurk user has reported difficulties doing so, needing to contact AWS customer support before being able to post hits.

1.4.2 AWS Credentials

psiTurk uses the Python/Boto (Boto3) SDK to communicate with the AWS API. In order to do so, boto must have access to the user’s AWS credentials, generated in section *Setting Up an Amazon Mechanical Turk Account.*

There are two approaches for setting the keys: (1) in a file called `.psiturkconfig` located in the user’s home directory, and (2) in any of the ways that Boto expects.

**.psiturkconfig approach**

If set here, the keys should be lowercased, and under an ‘AWS Access’ section key, as follows:

```ini
[AWS Access]
aws_access_key_id = foo
aws_secret_access_key = bar
```

**Boto approach**

If AWS credentials are not found via the `.psiturkconfig approach`, then Boto will search for them via its typical methods. That is, psiTurk users can store AWS credentials in any way that Boto expects. Specifically, the credentials variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` can be set via one of the following methods, listed in order of Boto-preference:

1. Environment variables (can optionally be set in `.env`)
2. Shared credential file (`~/.aws/credentials`)
3. AWS config file (`~/.aws/config`)
4. Boto2 config file (`/etc/boto.cfg` and `~/.boto`)

**Note:** psiTurk sets the `AWS_DEFAULT_REGION` to ‘us-east-1’, and this cannot be overridden.

For example, if a user’s `AWS_ACCESS_KEY_ID` were ‘foo’, their `AWS_SECRET_ACCESS_KEY` were ‘bar’, they might set the following in their `~/.aws/credentials` file:

```ini
AWS_ACCESS_KEY_ID=foo
AWS_SECRET_ACCESS_KEY=bar
```

Note that Boto3 respects certain environment variables that alter which files are searched for credentials and configuration settings. See here for more information.

1.4. Setting Up an Amazon Mechanical Turk Account
1.5 Configuration Overview

While AWS credentials are handled separately, psiTurk reads configuration settings from the following three places, and in the following order, with former entries being preferred over later ones.

2. A config.txt file (i.e., user-specified defaults)
3. A .psiturkconfig file
4. psiTurk defaults

That is to say, (4) there exist psiTurk-set defaults for many settings, (3) but a user may specify preferred defaults in a config.txt file, which they may override in turn via a ~/.psiturkconfig file, with (1) environment variables superseding any previous setting.

Furthermore, psiTurk loads and uses the python-dotenv library, which means that, on platforms that respect it, you can store KEY=VALUE pairs in a file called .env in your project’s root directory, and they will be read in as environment variables by psiTurk.

1.5.1 Which go where? Consider security and privacy, as well as science replicability

While any setting can be set via environment variables, only some make sense to do so. For instance, configuration settings that are credentials-related should not be saved to source code repositories, from whence they may leak. Instead, they can be set via environment variables. Sensitive settings include the following:

- database_url, when it contains embedded credentials
- dashboard login username and password

In other instances, a different setting may be desired for “development” environments than in “production” ones. For example, local development may want to use one database, while a psiturk experiment deployed onto a cloud hosting service for “live” data collection may want to use a different database. Or, different server host, port, and threads settings may be desired. Other settings can enable or disable dashboard and task-runner features, which may only be desirable for certain deployments. Environment variables are ideal for situations like these, rather than committing them to a static configuration file.

However, most other settings are best stored where they can be shared with others, for science replication purposes. Such settings should go in config.txt.

Additionally, a user may set any available setting to a global preferred value in a file called ~/.psiturkconfig.

Finally, many but not all settings have psiTurk-set defaults. Look in an example config.txt file generated by the psiturk-setup-example, and on the settings page within these docs, for documentation on the different settings and their defaults.

**Note:** In general, changes made to configuration settings require restarting the server process as it may change the behavior. Generally it is best to edit these files while psiturk is not running, and then restart the server.

1.5.2 Local configuration file

A sample default local configuration file can be viewed on github [here](https://github.com). A copy of this file is placed in the example directory as part of running the psiturk-setup-example command. In order for changes to config.txt to be read by psiturk, this file must be placed in the root of the psiturk study directory.
See Settings for descriptions of the available configuration parameters.

### 1.5.3 The `.psiturkconfig` global config file

Any configuration option can actually be placed in either the global or local configuration file. For example, if you wanted to run different project from different AWS accounts, you could add an [AWS access] section to move the local config.txt files and have different values in different folders. And vice-versa – any setting could be placed in the `.psiturkconfig` file in lieu of in config.txt.

By default, `.psiturkconfig` is searched for in the user’s home directory (`~/.psiturkconfig`), but the search location can be customized by setting the `PSITURK_GLOBAL_CONFIG_LOCATION` environment variable. This env var cannot be set via the `.env` file – rather, it must be available in the shell environment.

**Important:** Keep in mind that settings in the local `config.txt` file always override settings in the global `.psiturkconfig` file. In turn, environment variables always override all other config settings.

### 1.6 Settings

This page is organized by each of the sections of a `config.txt` file.

Any of these settings can be set via an environment variable by setting the setting name (sans section name) prefixed by the key string `PSITURK_`. For instance, to set the number of server threads (`threads`, section `Server Parameters`) to 1 via env var, one would set: `PSITURK_THREADS=1`.

**See also:**

*Configuration Overview*

- HIT Configuration
- Database Parameters
- Server Parameters
- Task Parameters
- Shell Parameters

### 1.6.1 HIT Configuration

The [HIT Configuration] section contains details about your Human Intelligence Task.

**title**

Title of the task that will appear on the AMT worker site.

**Type**: string

Workers often use fields like this one to search for tasks. Thus making them descriptive and informative is helpful.
**description**

Descriptive text for your study’s listing on AMT.

**Type** string

Workers often use fields like this one to search for tasks. Thus making them descriptive and informative is helpful.

**keywords**

A list of keywords to be associated with your study on AMT.

**Type** comma-delimited string

Workers often use fields like this one to search for tasks. Thus making them descriptive and informative is helpful.

**lifetime**

How long your HIT remains visible to workers (in hours).

**Type** integer

After the lifetime of the HIT elapses, the HIT no longer appears in HIT searches, even if not all of the assignments for the HIT have been accepted.

This is in contrast to the HIT duration, which specifies how long workers have to complete your task, and which you provide at HIT creation time. See the documentation on hit create for more details.

**us_only**

Controls if you want this HIT only to be available to US Workers.

**Type** bool

This is not a failsafe restriction but works fairly well in practice.

**approve_requirement**

Minimum approval percentage for a worker to be able to accept your study.

**Type** integer

Sets a qualification for what type of workers you want to allow to perform your task. It is expressed as a percentage of past HITs from a worker which were approved. Thus 95 means 95% of past tasks were successfully approved. You may want to be careful with this as it tends to select more seasoned and expert workers. This is desirable to avoid bots and scammers, but also may exclude new sign-ups to the system.

**number_hits_approved**

How many hits a worker must have approved before they can take your study.

**Type** integer

Important to use in conjunction with approve_requirement, because mturk will default approve_requirement to 100% until a worker has at least 100 HITs approved. Override that behavior by setting this to at least be 100.
**require_master_workers**

If True, Will make it so that only workers with the “Master” qualification can take your study.

    Type bool

See Who Are Amazon Mechanical Turk Masters?

**Note:** Master workers cost an extra 5%.

**browserExcludeRule**

A set of rules you can apply to exclude particular web browsers from performing your task.

    Type comma-delimited string

When a user contact your psiturk server, the server checks to see if the User Agent reported by the browser matches any of the terms in this string. It if does the worker is shown a message indicating that their browser is incompatible with the task.

Matching works as follows. First the string is broken up by the commas into sub-string. Then a string matching rule is applied such that it counts as a match anytime a sub-string exactly matches in the UserAgent string. For example, a user agent string for Internet Explorer 10.0 on Mac OS X might looks like this:

```
Mozilla/5.0 (compatible; MSIE 10.0; Macintosh; Intel Mac OS X 10_7_3; Trident/6.0)
```

This browser could be excluded by including this full line (see this website for a partial list of UserAgent strings). Also, “MSIE” would match this string or “Mozilla/5.0” or “Mac OS X” or “Trident”. Thus you should be careful in applying these rules.

There are also a few special terms that apply to a cross section of browsers. `mobile` will attempt to deny any browser for a mobile device (including cell phone or tablet). This matching is not perfect but can be more general since it would exclude mobile version of Chrome and Safari for instance. `tablet` denies tablet based computers (but not phones). `touchcapable` would try to exclude computers or browser with gesture or touch capabilities (if this would be a problem for your experiment interface). `pc` denies standard computers (sort of the opposite to the `mobile` and `tablet` exclusions). Finally `bot` tries to exclude web spiders and non-browser agents like the Unix curl command.

**allow_repeats**

Specifies whether participants may complete the experiment more than once.

    Type bool

    Default false

If it is set to `false` (the default), then participants will be blocked from completing the experiment more than once. If it is set to `true`, then participants will be able to complete the experiment any number of times.

Note that this option does not affect the behavior when a participant starts the experiment but the quits or refreshes the page. In those cases, they will still be locked out, regardless of the setting of `allow_repeats`.

**whitelistQualificationIds**

A list of custom qualifications that participants must possess to perform your task.

    Type comma-delimited string
You may need to ensure that workers have some requisite skill or pass some previous screening factors, such as language proficiency or having already completed one of your tasks. AMT uses custom qualification types to perform this filtering. When you add a custom qualification to `whitelist_qualification_ids`, AMT will only show your ad to potential participants who already have that qualification set. Other MTurk workers will neither see your ad nor be able to accept the HIT.

See Managing worker cohorts with qualifications and Best practices for managing workers in follow-up surveys for additional details on custom qualifications.

### blacklist_qualification_ids

A list of custom qualifications that participants must not possess to perform your task.

**Type** comma-delimited **string**

When you add a custom qualification to `blacklist_qualification_ids`, MTurk workers with that qualification already set will neither see your ad nor be able to accept your HIT. This is the recommended way of excluding participants who have performed other HITs for you from participating in your new HIT.

### 1.6.2 Database Parameters

The [Database Parameter] section contains details about your database.

See also:

- **Configuring Databases** For details on how to set up different databases and get your data back out.
- **Recording Data** For details on how to put data into your database.

#### database_url

`database_url` contains the location and access credentials for your database (i.e., where you want the data from your experiment to be saved).

**Type** string - valid database url

To use a SQLite data base, simply type the name of the file:

```ini
database_url = sqlite:///participants.db
```

This example would write to a database file with the name “participants.db” in the top-level directory of your experiment.

To use an existing MySQL database:

```ini
database_url = mysql://USERNAME:PASSWORD@HOSTNAME:PORT/DATABASE
```

where `USERNAME` and `PASSWORD` are your access credentials for the database, `HOSTNAME` and is the DNS entry or IP address for the database, `PORT` is the port number (standard is 3306) and `DATABASE` is the name of the database on the server. It is wise to test that you can connect to this url with a MySQL client prior to launching.

#### table_name

Specifies the table of the database you would like to write to.

**Type** string
IMPORTANT: psiTurk prevents the same worker from performing as task by checking to see if the worker appears in the current database table already. Thus, for a single experiment (or sequence of related experiments) you want to keep the `table_name` value the same. If you start a new design where it no longer matters that someone has done a previous version of the task, you can change the `table_name` value and begin sorting the data into a new table.

### 1.6.3 Server Parameters

The [Server Parameter] section contains details about your local web server process that you launch from the command line.

**host**

Specifies the network address to which your server should bind (i.e., on which address it should listen).

*Type* `string`

There are two common values for this. If `host` is set to `localhost` (or synonymously `127.0.0.1`), then your experiment will only work for testing (i.e., even if you have an internet addressable computer, people outside of your local machine will not be able to connect). This is a security feature for developing and testing your application.

If set to `0.0.0.0`, then your psiturk server will be accessible to any traffic that can reach the computer on which your server is running. If your server has a public-internet interface, then participants anywhere in the world can access your study.

**port**

The port that your server will run on.

*Type* `integer`

If not running as `root`, must be greater than 1024. Max 65535. Typically a number greater than 5000 will work. If another process is already using a given port you will usually get an error message.

**logfile**

*Type* `string`

The location of the server log file. Error messages for the server process are not printed to the terminal or command line. To help in debugging they are stored in a log file of your choosing. This file will be located in the top-level folder of your project.

**loglevel**

*Type* `integer`

Sets how “verbose” the log messages are. See the python `logging` library.

**enable_dashboard**

Whether to enable the dashboard. If True, then the `login_username` and `login_pw` must also be set.

*Type* `bool`

*Default* False
do_scheduler

Whether to run the task scheduler, which is viewable and configurable from the dashboard.

  **Type** bool
  **Default** False

Tasks are loaded from the database. If True, then `threads` must be no greater than 1, because the task runner is not thread safe. Will only run while the psiturk server is running.

**login_username**

  **Type** string

If you want to have a custom-login section of your web application (e.g., see customizing psiturk) then you can set a login and password on certain web pages urls/routes. By default if you aren’t using them, this is ignored.

**login_pw**

  **Type** string

If you want to have a custom-login section of your web application (e.g., see customizing psiturk) then you can set a login and password on certain web pages urls/routes. By default if you aren’t using them, this is ignored.

**threads**

  **Type** the string ‘auto’ or integer

`threads` controls the number of process threads the the psiturk webserver will run. This enables multiple simultaneous connections from internet users. If you select `auto` it will set this based on the number of processor cores on your current computer.

**certfile**

Public ssl certificate for the psiturk server to use.

  **Type** path

`certfile` should be the /path/to/your/domain/SSL.crt

If both certfile and keyfile are set and the files readable, then the psiturk gunicorn server will run with ssl. You will need to execute the psiturk with privileges sufficient to read the keyfile (typically root). If you run `psiturk` with `sudo` and if you are using a virtual environment, make sure to execute the full path to the desired psiturk instance in your environment.

If you want to do this, you are responsible for obtaining your own cert and key. It is not necessary to run the psiturk server with `ssl` in order to use your own ad server. You can have a proxy server such as `nginx` in front of psiturk/gunicorn which handles ssl connections.

However, if you configure the psiturk server to run with SSL by setting the ‘certfile’ and ‘keyfile’ here, you must use a proxy server in front of psiturk to serve the content in your `/static` folder. An SSL-enabled psiturk/gunicorn server will not serve static content – it will only serve dynamic content.

See https://docs.gunicorn.org/en/stable/deploy.html for more information on setting up proxy servers with the psiturk (gunicorn) server.
**keyfile**

Private ssl certificate for the psiturk server to use.

    Type path

certfile should be the /path/to/your/domain/private-SSL.key. Although .crt files can contain .key files within them, psiturk currently requires that you point to separate .crt and .key files for this feature to work.

See the documentation for certfile for more information.

**server_timeout**

    Type integer
    Default 30

Number of seconds gunicorn will wait before killing an unresponsive worker. This timeout applies to any individual request.

If you expect that your experiment may take more than 30 seconds to respond to a request, you may want to increase this.

**Note:** See [https://docs.gunicorn.org/en/stable/settings.html#timeout](https://docs.gunicorn.org/en/stable/settings.html#timeout) for more information.

---

### 1.6.4 Task Parameters

The [Task Parameters] section contains details about your task.

**experiment_code_version**

    Type string

Often you might run a couple different versions of an experiment during a research project (e.g., Experiment 1 and 2 of a paper). Or, perhaps you make modifications to a single study after having already begun data collection.

experiment_code_version is a string which is written into the database along with your data helping you remember which version of the code each participant was given.

This variable is used by the server along with num_conds and num_counters to ensure an equal number of workers per condition for the current experiment_code_version. In other words, changing the experiment_code_version resets the number of workers per condition to [0 0].

**num_conds**

    Type integer

psiTurk includes a primitive system for counterbalancing participants to conditions. If you specify a number of condition greater than 1, then psiTurk will attempt to assign new participants to conditions to keep them all with equal N. It also takes into account the time delay between a person being assigned to a condition and completing a condition (or possibly withdrawing). Thus, you can be fairly assured that after running 100 subjects in two conditions each condition will have 50+- completed participants.
Note: If you want to reset the random assignment when changing `num_conds`, update the `experiment_code_version`.

`num_counters`

**Type** integer

`num_counters` is identical to `num_cond` but provides an additional counterbalancing factor beyond condition. If `num_counters` is greater than 1 then psiTurk behaves as if there are `num_cond*num_counters` conditions and assigns subjects randomly to the the expanded design. See [Issue #53](#) for more info.

`contact_email_on_error`

The email you would like to display to workers in case there is an error in the task.

**Type** string, valid email address

Workers will often try to contact you to explain what when want and request partial or full payment for their time. Providing a email address that you monitor regularly is important to being a good member of the AMT community.

`cutoff_time`

Maximum time in minutes that it should take for a participant to finish the task.

**Type** integer

Exclusively used in determining random assignment – basically, how long should a participant be given to complete the task after starting? How long should the task last? This is different than the `duration` specified when running `hit create`, because a participant may not start the task immediately after accepting it, while the hit `duration` starts ticking as soon as the hit is accepted (some workers queue their accepted hits before starting it).

### 1.6.5 Shell Parameters

The [Shell Parameters] section contains details about the psiturk shell.

`launch_in_sandbox_mode`

**Type** bool

If set to `true`, the psiturk shell will launch in sandbox mode. if set to `false`, the shell will launch in live mode. We recommend leaving this option to `true` to lessen the chance of accidentally posting a live HIT to mTurk.

`bonus_message`

**Type** string

If set, automatically uses this string as the message to participants when bonusing them for an assignment. If not set, you will be prompted to type in a message each time you bonus participants. (This message is required by AMT.)
1.7 Command-line Interface Overview

The psiTurk shell is a command line interface which allows users to communicate with their experiment server, and also with Amazon Mechanical Turk.

The psiturk command has several invocations.

1.7.1 Options

psiturk [options]

- `-v`, `--version`
  
  Print the currently installed version of psiTurk and exit.

- `-s`, `--script <filename>`
  
  Run a list of commands from a text file, then exit. Each line in the file is treated as a command.

1.7.2 Command invocation

psiturk command [argument]...

Any single shell command can be run without launching the interactive shell, by invoking psiturk with the command as an argument. For example, to launch the psiturk server:

```
$ psiturk server on
```

1.7.3 Launch an interactive shell

psiturk

Alternatively, an interactive shell can be launched by running the command psiturk in any psiturk experiment server. A config.txt file will be loaded from the directory in which the shell is launched.

**Warning:** The interactive shell cannot be launched without valid AWS credentials having been set! This is because the prompt is intrinsically tied to AMT – its prompt displays the current mturk “mode” and the “number of hits”.

However, non-AWS psiturk commands can still be run via the psiturk <command> interface.

The psiTurk shell prompt looks something like this:

```
[psiTurk server:off mode:sdbx #HITs:0]$```

and contains several pieces of useful information:

- **Server field** – will generally be set to on or off and denotes whether the experiment server is running. If the server field says unknown, this likely means that a server process is running from an improperly closed previous psiTurk shell session. In this case, you may need to manually kill the processes in the terminal or restart your terminal session.

- **Mode field** – displays the current mode of the shell. In the full psiturk shell, the mode will be either sdbx (sandbox) or live. While in cabin mode, the mode will be listed as cabin. More about the psiturk shell mode can be found here.
• **#HITs field** – displays the number of HITs currently active, either in the worker sandbox when in sandbox mode or on the live AMT site when in live mode.

### 1.7.4 Create an Example Project

To create a sample project, run the following:

```bash
psiturk-setup-example
```

### 1.8 psiTurk commands

Each of these commands can be run either from an interactive shell, or as arguments to the psiturk command (e.g., `psiturk amt_balance` or `psiturk hit create 1 0.01 1` from a bash prompt).

**See also:**

*Command-line Interface Overview*

<table>
<thead>
<tr>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>amt_balance</code></td>
</tr>
<tr>
<td><code>config</code></td>
</tr>
<tr>
<td><code>debug</code></td>
</tr>
<tr>
<td><code>download_datafiles</code></td>
</tr>
<tr>
<td><code>help</code></td>
</tr>
<tr>
<td><code>hit</code></td>
</tr>
<tr>
<td><code>worker</code></td>
</tr>
<tr>
<td><code>quit</code></td>
</tr>
<tr>
<td><code>server</code></td>
</tr>
<tr>
<td><code>status</code></td>
</tr>
<tr>
<td><code>mode</code></td>
</tr>
</tbody>
</table>

#### 1.8.1 `amt_balance`

Displays your current AMT balance, or your worker sandbox balance (always $10,000.00) if you are in sandbox mode.

An example of checking your balance in sandbox mode:

```bash
[psiTurk server:off mode:sdbx #HITs:1]$ amt_balance
$10,000.00
```

#### 1.8.2 `config`

Used with a variety of subcommands to control the current configuration context.
Commands

- `config print`
- `config reload`
- `config help`

**config print**

Prints the current configuration context.

Example:

```
[psiTurk server:off mode:sandbox #HITs:0] config print
[Server parameters]
threads=auto
...
[Shell parameters]
launch_in_sandbox_mode=true
```

**config reload**

Reloads the current config context (both local and global files). This will cause the server to restart.

Example:

```
[psiTurk server:on mode:sandbox #HITs:0] config reload
Reloading configuration requires the server to restart. Really reload? y or n: y
Shutting down experiment server at pid 82701...
Please wait. This could take a few seconds.
Experiment server launching...
Now serving on http://localhost:22362
```

**config help**

Display a help message concerning the config subcommand.

**1.8.3 debug**

Makes it possible to locally test your experiment without contacting Mechanical Turk servers. Type `debug` to automatically launch your experiment in a browser window. The server must be running to debug your experiment. When debugging, the server feature that prevents participants from reloading the experiment is disabled, allowing you to make changes to the experiment on the fly and reload the debugging window to see the results.

- `debug -p, --print-only`
  
  Use the `-p` flag to print a URL to use for debugging the experiment, without attempting to automatically launch a browser. This is particularly useful if your experiment server is running remotely.

Example using the `-p` flag to request a debug link:
Here's your randomized debug link, feel free to request another:
http://localhost:22362/ad?assignmentId=debugDKSAAE&hitId=debugYW8RI&workerId=debugM1QUH4

1.8.4 download_datafiles

Accesses the current experiment database table (defined in config.txt) and creates a copy of the experiment data in a csv format. download_datafiles creates three files in your current folder:

- `eventdata.csv`
  
  Contains events such as window-resizing, and is formatted as follows:

<table>
<thead>
<tr>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
<th>column 4</th>
<th>column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique user ID</td>
<td>event type</td>
<td>interval</td>
<td>value</td>
<td>time</td>
</tr>
</tbody>
</table>

- `questiondata.csv`
  
  Contains data recorded with psiturk.recordUnstructuredData(), and is formatted as follows:

<table>
<thead>
<tr>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique user ID</td>
<td>question name</td>
<td>response</td>
</tr>
</tbody>
</table>

- `trialdata.csv`
  
  Contains data recorded with psiturk.recordTrialData(), and is formatted as follows:

<table>
<thead>
<tr>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
<th>column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique user ID</td>
<td>trial #</td>
<td>time</td>
<td>trial data</td>
</tr>
</tbody>
</table>

Note: More information about how to record different types of data in an experiment can be found here.

1.8.5 help

Usage:

```
help
help <command>
```

The help command displays a list of valid psiturk shell commands. Entering help followed by the name of a command brings up information about that command.

Examples:
1. List all commands:

```
[pstiurtk server:on mode:sdbx #HITs:0]$ help
psiTurk command help:
amt_balance debug mode server
config download_datafiles open setup_example version
db hit psiturk_status status worker

basic CMD command help:
EOF ed help li py run shortcuts
_load edit hi list q save show
_relative_load eof history load quit set
cmdenvironment exit 1 pause r shell
```

psiTurk commands are listed first, followed by commands inherited from the python *cmd2* module. More information about *cmd2* commands can be found here.

2. View the help menu for a command and its subcommands

```
[pstiurtk server:on mode:sdbx #HITs:0]$ help server
Usage:
server on
server off
server restart
server log
server help

'server' is used with the following subcommands:
on Start server. Will *not* work if server *is* already running.
off Stop server. May take several seconds.
restart Run 'server off', followed by 'server on'.
log Open live server log in a separate window.
help Display this screen.
```

**Note:** With commands with subcommands such as `server`, you can also view the help screen by entering `<command> help`. For example, `server help` has the same effect at `help server`.

### 1.8.6 hit

The `hit` command is used to create, view, delete, and modify Human Intelligence Tasks (“HITs”) on Amazon Mechanical Turk.

**Commands**

- `hit create`
- `hit extend`
- `hit expire`
hit create

Usage:

```
hit create [<numWorkers> <reward> <duration>]
```

Create a HIT with the specified number of assignments, reward amount, and duration. Will be posted either live to AMT or to the Worker Sandbox depending upon your current mode. `hit create` can also be run interactively by entering the command without parameters.

The duration specifies how long a worker can “hold on” to your HIT (in hours or hours.<fraction_of_hour>). This should be long enough for workers to actually complete your HIT, but sometimes workers will “accept” a HIT which is worth a lot of money but come back and do the work later in the day. You can specify a shorter duration if you want workers to complete your HIT immediately.

Example of creating a HIT in the sandbox with three assignments that pays $2.00 and has a 1.5 hour time limit:

```
[psiTurk server:on mode:sdbx #HITs:0]$ hit create 3 2.00 1.5
***************
Creating sandbox HIT
  HITid: 2XE40SPW1INMXUF9OJUNDB6BT8W2F4
  Max workers: 3
  Reward: $2.00
  Duration: 1.5 hours
  Fee: $0.60
________________________
Total: $6.60
Ad for this HIT now hosted at: https://ad.psiturk.org/view/Q3HNnfqzg3MP9VDbu3kFyn?
˓→assignmentId=debugJCI80S&hitId=debug9AWC90
```

hit extend

Usage:

```
hit extend <HITid> [--assignments <number>] [--expiration <time>]
```

Extend an existing HIT by increasing the amount of time before the HIT expires (and and is no longer available to workers) or by increasing the number of workers who can complete the HIT.

Example adding both time and assignments to a HIT:

```
[psiTurk server:on mode:sdbx #HITs:1]$ hit list --active
Stroop task
    Status: Assignable
    HITid: 2776AUC26DG6NRIGNVRFN0COYO0B4R
        max:3/pending:0/complete:0/remain:3
        Created:2014-03-07T21:36:33Z
        Expires:2014-03-08T21:36:33Z

[psiTurk server:on mode:sdbx #HITs:1]$ hit extend 2776AUC26DG6NRIGNVRFN0COYO0B4R --
˓→assignments 10 --expiration 12
HIT extended.

[psiTurk server:on mode:sdbx #HITs:1]$ hit list --active
Stroop task
    Status: Assignable
    HITid: 2776AUC26DG6NRIGNVRFN0COYO0B4R
```

(continues on next page)
Note that both the remaining number of assignments and the expiration time of the HIT have increased. One can also increase the number of assignments or the expiration independently.

**hit expire**

**Usage:**

```
hit expire (--all | <HITid> ...)
```

Expire one or more existing HITs, or expire all HITs using the `--all` flag.

**Examples:**

1. Expiring two HITS at once:

   ```
   [psiTurk server:on mode:sdbx #HITs:4]$ hit expire 2Y0T3HVXAVKIMG42A2S75Z9943NNFG
   expiring sandbox HIT 2Y0T3HVXAVKIMG42A2S75Z9943NNFG
   expiring sandbox HIT 2RVZXR24SMEZFG314ME9X8P9CPPH0X
   [psiTurk server:on mode:sdbx #HITs:2]$
   ```

2. Expiring all active HITS:

   ```
   [psiTurk server:on mode:sdbx #HITs:2]$ hit expire --all
   expiring sandbox HIT 2776AUC26DG6NRIGNVRFN0COY00B4R
   expiring sandbox HIT 2VUWA6X3YOCCVET8PKOPWINIWJFP00
   [psiTurk server:on mode:sdbx #HITs:0]$
   ```

**1.8.7 worker**

The `worker` command is used to list, approve and reject, and bonus worker assignments on Amazon mechanical Turk.

**Commands**

- `worker approve`
- `worker reject`
- `worker unreject`
- `worker bonus`
- `worker list`
- `psiturk_status`

**worker approve**

**Usage:**

```
```
worker approve (--hit <hit_id> | <assignment_id> ...)

Approve worker assignments for one or more assignment ID’s, or use the --hit flag to approve all workers for a specific HIT.

Examples:

1. Approve a single assignment:

```
[pstiTurk server:on mode:sdbx #HITs:0]$ worker approve
→21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
approved 21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
```

2. Approve all assignments for a given hit:

```
[pstiTurk server:on mode:sdbx #HITs:0]$ worker approve --hit
→2QKHECWA6X3Y4QTYKCG5NXPTWGYMLF
approving workers for HIT 2QKHECWA6X3Y4QTYKCG5NXPTWGYMLF
approved 2MB011K2747PY7FQJ1ZN76UXH0ECE
approved 2U42ZMAZHRX17J8NEVUH1KJCAKBY
```

**worker reject**

Usage:

```
worker reject (--hit <hit_id> | <assignment_id> ...)
```

Reject worker assignments for one or more assignment ID’s, or use the --hit flag to reject all workers for a specific HIT.

Example rejecting a single assignment:

```
[pstiTurk server:on mode:sdbx #HITs:0]$ worker reject 2Y9OVR14IXKOIZQL1E3WD6X30CD98U
rejected 2Y9OVR14IXKOIZQL1E3WD6X30CD98U
```

**worker unreject**

Usage:

```
worker unreject (--hit <hit_id> | <assignment_id> ...)
```

Unreject worker assignments for one or more assignment ID’s, or use the --hit flag to unreject all workers for a specific HIT.

**Note:** Unrejecting an assignment automatically approves that assignment.

Example of unrejecting a single assignment:

```
[pstiTurk server:on mode:sdbx #HITs:0]$ worker unreject 2Y9OVR14IXKOIZQL1E3WD6X30CD98U
unrejected 2Y9OVR14IXKOIZQL1E3WD6X30CD98U
```
**worker bonus**

Usage:

```
worker bonus (--amount <amount> | --auto) (--hit <hit_id> | <assignment_id> ...)
```

Grant bonuses to workers for one or more assignment ID’s, or use the `--hit` flag to bonus all workers for a specific HIT.

Enter the bonus `--amount <amount>` in an X.XX format, or use the `--auto` flag to bonus each worker according to the ‘bonus’ field of the database (requires a custom bonus route in the experiment’s `custom.py` file).

Upon running `worker bonus`, you will be asked to input a reason for the bonus. This message will be displayed to workers who receive the bonus.

**Note:** You must approve the worker assignment before you grant a bonus.

**Warning:** While it isn’t possible to approve an assignment more than once, it is possible to grant a bonus repeatedly. When running `worker bonus` with the `--hit` flag, only workers who have not yet received a bonus for the assignment will be bonused. However, when running `worker bonus` on individual assignments the worker will be bonused regardless of whether they have already received one.

Examples:

1. Bonusing an individual assignment. The bonus can be granted repeatedly, making this risky:

```
[psiTurk server: on mode:sdbx #HITs:0]$ worker bonus --amount 2.00
→21A8IUB2YU982V9C5BUL3FBJB5B8K7
Type the reason for the bonus. Workers will see this message: Here’s a bonus!
gave bonus of $2.00 to 21A8IUB2YU982V9C5BUL3FBJB5B8K7
[psiTurk server: on mode:sdbx #HITs:0]$ worker bonus --amount 2.00
→21A8IUB2YU982V9C5BUL3FBJB5B8K7
Type the reason for the bonus. Workers will see this message: Here’s another one!
gave bonus of $2.00 to 21A8IUB2YU982V9C5BUL3FBJB5B8K7
```

2. Say there are approved assignments for a HIT, one already bonused, one not yet bonused. Bonusing by HIT prevents repeated bonuses:

```
[psiTurk server: on mode:sdbx #HITs:0]$ worker bonus --amount 2.00 --hit 2ECYT3DHJHP4RRU304P8USX9BCXU10
Type the reason for the bonus. Workers will see this message: you haven’t been bonused yet. Here’s a bonus!
bonusing workers for HIT 2ECYT3DHJHP4RRU304P8USX9BCXU10
gave a bonus of $2.00 to 2MB011K274J7PY7FQ1ZN76UXH0ECED
bonus already awarded to 21A8IUB2YU982V9C5BUL3FBJB5B8K7
```

3. If a compute-bonus route exists in the experiment `custom.py`, we can also use the --auto flag to automatically give each worker the correct bonus:

```
[psiTurk server: on mode:sdbx #HITs:0]$ worker bonus --auto --hit 2ECYT3DHJHP4RRU304P8USX9BCXU10
Type the reason for the bonus. Workers will see this message: Thanks for moving science forward!
bonusing workers for HIT 2ZQIUB2YU98JX6A4V3C0IBJ9W0HL9P
```

(continues on next page)
gave a bonus of $3.00 to 27UQ45UKOYW1ZFLNJ8RG012VYDVP
gave a bonus of $2.50 to 24IHPCGJ2D2H2KFPX80MPPSKQM933

Note: Unlike the commands to approve, reject, or unreject workers, the worker bonus command requires the psiturk shell to be launched in the same project as the HIT for which workers are being bonused, since the information about which workers have been bonused is stored in the experiment database.

worker list

Usage:

worker list [--submitted | --approved | --rejected] [--hit <hit_id>]

List all worker assignments, or list worker assignments fitting a given condition using the provided flags. Use the --hit flag to list workers for a specific HIT.

Examples:

1. Listing all submitted workers:

```
[psiturk server:on mode:sdbx #HITs:0]$ worker list --submitted
[
  {
    "status": "Submitted",
    "assignmentId": "2VQHVI44OS2K18PW7EQSEAP5DPV5ZY",
    "workerId": "A2O6BB9HXEUXX1",
    "submit_time": "2014-03-04T16:14:32Z",
    "hitId": "2ZRNZW6HEZ6OUI7FRTZ6CGUMG1QPZ0",
    "accept_time": "2014-03-04T16:14:05Z"
  },
  {
    "status": "Submitted",
    "assignmentId": "2XB92NJKM05B2XAD1YN2JTP9TYXAFG",
    "workerId": "A2O6BB9HXEUXX1",
    "submit_time": "2014-03-03T23:35:17Z",
    "hitId": "2ZRNZW6HEZ6OUI7FRTZ6CGUMG1QPZ0",
    "accept_time": "2014-03-04T16:14:05Z"
  }
]
```

2. Listing approved workers for a specific HIT:

```
[psiturk server:on mode:sdbx #HITs:0]$ worker list --approved --hit 2ECYT3DHJHP4RRU304P8UX9BCXUI0
listing workers for HIT 2ECYT3DHJHP4RRU304P8UX9BCXUI0
[
  {
    "status": "Approved",
    "assignmentId": "21A8IUB2YU89ZV9C5BUL3FBJB5B8K7",
    "workerId": "A2O6BB9HXEUXX1",
    "submit_time": "2014-02-26T03:26:55Z",
    "hitId": "2ECYT3DHJHP4RRU304P8UX9BCXUI0",
    "accept_time": "2014-02-26T03:26:36Z"
  }
]
```
psiturk_status

Usage:

psiturk_status

Display startup screen with message from psiturk.org.

Example:

[psiTurk server:off mode:sdbx #HITs:1] $ psiturk_status

http://psiturk.org

an open platform for science on Amazon Mechanical Turk

--------------------------------------------------------------------
System status:
Hi all, You need to be running psiTurk version >= 1.0.5dev to use the
Ad Server feature!

Check https://github.com/NYUCCL/psiTurk or http://psiturk.org for
latest info.

psiTurk version 1.0.8dev
Type "help" for more information.

[psiTurk server:off mode:sdbx #HITs:1]$

1.8.8 quit

Usage:

quit

Quits the psiTurk shell. If you have a server running, psiTurk will confirm that you want to quit before exiting, since
quitting psiTurk turns off the server.

Example of quitting psiTurk with the server running:

[psiTurk server:on mode:sdbx #HITs:0] $ quit
 Quitting shell will shut down experiment server. Really quit? y or n: y
 Shutting down experiment server at pid 40182...
 Please wait. This could take a few seconds.

$
1.8.9 server

The `server` command is used with a variety of subcommands to control the experiment server.

- `server on`
- `server off`
- `server restart`
- `server log`

**server on**

Start the experiment server.

Example:

```
[psiTurk server:off mode:sdbx #HITs:0]$ server on
Experiment server launching...
Now serving on http://localhost:22362
[psiTurk server:on mode:sdbx #HITs:0]$
```

**server off**

Shut down the experiment server.

Example:

```
[psiTurk server:on mode:sdbx #HITs:0]$ server off
Shutting down experiment server at pid 32911...
Please wait. This could take a few seconds.
[psiTurk server:off mode:sdbx #HITs:0]$
```

**server restart**

Runs `server off`, followed by `server on`.

**server log**

Opens the server log in a separate window. Uses Console.app on Max OS X and xterm on other systems.

1.8.10 status

Usage:

```
status
```

The `status` command updates and displays the server status and number of HITs available on AMT or in the worker sandbox.
Example of using the \texttt{status} command in sandbox mode:

\begin{center}
\texttt{[psiTurk server:off mode:sdbx \#HITs:1]$ status}
\end{center}
\begin{center}
Server: currently offline
\end{center}
\begin{center}
AMT worker site - sandbox: 1 HITs available
\end{center}

### 1.8.11 mode

Usage:

\begin{verbatim}
mode
mode <which>
\end{verbatim}

The \texttt{mode} command controls the current mode of the psiTurk shell. Type \texttt{mode live} or \texttt{mode sandbox} to switch to either mode, or simply \texttt{mode} to switch to the opposite mode. The current mode affects almost every psiturk shell command. For example, running \texttt{hit create} while in sandbox mode will create a HIT in the sandbox, while running it in live mode will create a HIT on the live AMT site. Similarly, commands like \texttt{worker list all} or \texttt{hit list all} will list assignments and HITs from either the live site or the sandbox, depending on your mode.

\textbf{Note:} Switching the psiTurk shell mode while the server is running requires the server to restart, since at the end of the experiment participants need to be correctly redirected back to either the live AMT site or the sandbox. Therefore, you should not change modes while you are serving a live HIT to workers.

Examples:

1. Switching mode, with and without \texttt{<which>} specifier:

\begin{center}
\texttt{[psiTurk server:off mode:sdbx \#HITs:0]$ mode}
\end{center}
\begin{center}
Entered live mode
\end{center}
\begin{center}
\texttt{[psiTurk server:off mode:live \#HITs:0]$ mode sandbox}
\end{center}
\begin{center}
Entered sandbox mode
\end{center}

2. Switching mode with the server running:

\begin{center}
\texttt{[psiTurk server:on mode:sdbx \#HITs:0]$ mode}
\end{center}
\begin{center}
Switching modes requires the server to restart. Really switch modes? y or n: y
\end{center}
\begin{center}
Entered live mode
\end{center}
\begin{center}
Shutting down experiment server at pid 33447...
\end{center}
\begin{center}
Please wait. This could take a few seconds.
\end{center}
\begin{center}
Experiment server launching...
\end{center}
\begin{center}
Now serving on http://localhost:22362
\end{center}
\begin{center}
\texttt{[psiTurk server:on mode:live \#HITs:0]$}
\end{center}

Type \texttt{n} instead to abort the mode switch harmlessly.
1.9 Databases Overview

Databases provide a critical aspect of psiTurk, as they store data from experiments and help to prevent the same user from completing your experiment more than once. Databases provide an important function for web-based experiments – Because multiple people can complete your experiment at the same time, you need a system which can simultaneously write/read data. Databases are optimized for this type of environment and are thus very useful for experiments.

psiTurk can integrate with any database that is compatible with SQLAlchemy.

See also:

database_url – For details on how to configure databases in config.txt

1.9.1 Using SQLite

Perhaps the simplest quickstart solution is to use SQLite. This database solution writes to a local file on the same computer as is running the psiTurk server.

To use a SQLite database, simply set the database_url field in your local configuration file (config.txt):

database_url = sqlite:///FILENAME.db

where FILENAME is of your choosing. By default, psiTurk sets this like this:

database_url = sqlite:///participants.db

This will make a SQLite database file in the top-level folder of your project. If you change the database_url and restart psiTurk, a new database corresponding to the new filename will be created. If you set it to an existing file name, psiTurk will attempt to connect to this database.

It is best to do this while the server is not running (note in this example the “server” status says “off”). If you change this while the server is running you will need to type:

[psiTurk server:on mode:sdbx #HITs:0]$ server restart

While great for development and debugging, SQLite has a number of important downsides for deploying experiments. In particular, SQLite does not allow concurrent access to the database, so if the locks work properly, simultaneous access (say, from multiple users submitting their data at the same time) could destabilize your database. In the worst scenario, the database could become corrupted, resulting in data loss.

As a result, we recommend using a more robust database solution when actually running your experiment.

However, SQLite is a good solution particularly for initial testing. It is also possible to try to “throttle” the rate of signups on Mechanical Turk (by only posting one assignment slot at a time) so that database errors are less likely using SQLite.
Note: SQLite database are fine for local testing but more robust databases like MySQL are recommended especially if you plan to run many participants simultaneously. Again, any server compatible with SQLAlchemy can be used.

1.9.2 Using a postgresql database on Heroku

The Heroku free tier includes access to a postgresql database. See Running psiTurk on Heroku.

1.9.3 Using a SQL database server

A more robust solution is to set up a MySQL database. psiTurk’s reliance on SQLAlchemy for interfacing with database which means it is easy to switch between MySQL, PostgreSQL, or SQLite.

For example, to use an existing MySQL database:

```python
database_url = mysql://USERNAME:PASSWORD@HOSTNAME:PORT/DATABASE
```

where USERNAME and PASSWORD are your access credentials for the database, HOSTNAME is the DNS entry or IP address for the database, PORT is the port number (default is 3306) and DATABASE is the name of the database on the server.

Use 127.0.0.1 as the HOSTNAME for a database running locally to the psiTurk server rather than ‘localhost’. Mysql treats the HOSTNAME ‘localhost’ as a special case in Unix-based systems and will cause the psiTurk server to fail to boot.

It is wise to test that you can connect to this url with a MySQL client prior to launching, such as MySQL Workbench Sequel Pro.

Here’s an example of setting up a minimal MySQL database for use with psiTurk:

```bash
$ mysql -uroot -p
mysql> CREATE USER 'your_username'@'localhost' IDENTIFIED BY 'your_password';
Query OK, 0 rows affected (0.03 sec)
mysql> CREATE DATABASE your_database;
Query OK, 1 row affected (0.01 sec)
mysql> GRANT ALL PRIVILEGES ON your_database.* TO 'your_username'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

where your_username, your_password and your_database match the USERNAME, PASSWORD and DATABASE specified in config.txt’s database_url variable.

The table specified in config.txt:

```python
table_name = turkdemo
```

... will be created automatically when running the psiturk shell. MySQL is (fairly) easy to install and free. However, a variety of web hosting services offer managed MySQL databases. Some are even free.

1.9.4 Running a MySQL database on Amazon’s Web Services Cloud

While not terribly difficult, installing and managing a MySQL database can be an extra hassle. Interestingly, when you sign up with Amazon Mechanical Turk as a requester, you also are signing up for Amazon’s Web Services a very
powerful cloud-based computing platform that is used by many large web companies. One of the services Amazon provides is a fully hosted relational database server (RDS).

According to Amazon, “Amazon Relational Database Service (Amazon RDS) is a web service that makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing you up to focus on your applications and business.”

**Danger:** If you use Amazon’s RDS to host your MySQL database you may incur additional charges. At the present time a small RDS instance is free if you have recently signed up for Amazon Web Services. However, older account have to pay according to the current rates. This does **NOT** use the pre-paid mechanism that is used on Amazon Mechanical Turk. Thus launching a database server on the cloud and leaving it running run up monthly charges. You are responsible for launching and shutting down your own database instances if you use this approach. **PROCEED WITH CAUTION.**

## 1.10 Tutorials

### 1.10.1 Example project walkthrough

Perhaps the best way to learn about psiTurk is to go through the steps of configuring and running an experiment. This tutorial will take you through the steps required to run the example experiment – a Stroop task – that ships bundled with a psiTurk installation. This project can be a great starting place for developing your own experiment.

**Warning:** This guide assumes you already have the psiTurk command line tool installed on your computer. If you haven’t you should begin there and come back when it is installed. Instruction [here](#).

This guide also assumes you are using version 1.0.10dev or higher of the psiTurk command line tool. Type `psiturk --version` in your command shell/terminal program to verify your version number.

### Background

The Stroop effect is the finding that people show interference from reading while naming the font color of words. The task is used to suggest that reading has become a highly “automatic” cognitive skill. You can read more about the Stroop task [here](#). This guide won’t comment much on the psychology of it, rather focusing on the technical aspect of running such an experiment online that consists of a sequence of trials and which records response time and key presses.

### Initialize the demo code

The first step is to obtain the archive of code and resources specific to the Stroop demo. Additional experiments are shared on the psiTurk [experiment exchange](#). However, the Stroop demo comes bundled within the psiturk command line tool.

First use the `psiturk-setup-example` command to place fresh copies of the files into a new folder:

```bash
$ psiturk-setup-example
Creating new folder `psiturk-example` in the current working directory
Copying /Users/gureckis/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-packages/PsiTurk-1.0.10dev-py2.7.egg/psiturk/example to ./psiturk-example
Creating default configuration file (config.txt)
```
afterward you should have a new folder in the current directory named “psiturk-example” with the following listing of files:

```bash
$ cd psiturk-example
$ ls -la
total 16
drwxrwxr-x 6 gureckis staff 204 Mar 31 12:18 .
drw-xxxx 23 gureckis staff 782 Mar 31 12:18 ..
-rw-r--r-- 1 gureckis staff 796 Mar 31 11:55 config.txt
-rw-r--r-- 1 gureckis staff 3226 Mar 31 11:55 custom.py
drwxrwxr-x 9 gureckis staff 306 Mar 31 12:18 static
drwxrwxr-x 19 gureckis staff 646 Mar 31 12:18 templates
```

See also:
A full description of the individual files is provided [here](#). A few of the files described on the full documentation will not appear until the first time you start `psiturk` and launch the psiTurk server.

### Set Your AWS Credentials

To access Amazon Mechanical Turk and other Amazon Web Services features, you need to set your AWS Credentials and also your default AWS region (see [these instructions](#) for details).

### Configure the option for the demo experiment

Another of the files generated by `psiturk-setup-example` is the `config.txt` file, which contains a variety of experiment and server parameters. These values can be changed by altering the file in any text editor.

The default `config.txt` file is already mostly configured to help you test the Stoop demo. Three options you might want to adjust to begin with are:

1. **In the [Server Parameters] section ensure that the port listed is one** that is available on your computer (answer is usually yes unless you have particular firewall software running).

2. **In the [Server Parameters] section ensure that the host is** either `localhost` (if just testing/debugging locally) or set to `0.0.0.0` (if planning to test live on the AMT site).

See also:
A full description of the local configuration file and the meaning of the various option is available [here](#).

### Launch the psiTurk shell

All user commands to psiTurk, such as creating a HIT, launching the experiment server, or approving workers, are issued through the `psiturk command`. To open the shell, run `psiturk` in a valid experiment folder. You should see something like this (though probably colorized on your display):

```bash
$ psiturk
http://psiturk.org
```
The psiTurk shell prompt displays several useful pieces of information: whether the experiment server is on, whether you are in sandbox or live mode, and how many hits are online in your current mode (more on all of these below). While in the psiTurk shell, all commands entered will be executed by psiTurk. To exit the shell, type `quit`.

See also:

*Command-line Interface Overview*

**Start/stop the experiment server**

The psiTurk experiment server is a separate process that acts as a custom, local web server. To launch the server type `server on` in the command line interface:

```
[psiTurk server:off mode:sdbx #HITs:0]$ server on
Experiment server launching...
Now serving on http://localhost:
[psiTurk server:on mode:sdbx #HITs:0]$
```

Note that the command prompt has changed from showing `server:off` to `server:on` in this example (and also changed form red to green on colorized terminals). You can start or stop the server at any time using the `server on` and `server off` commands. Typically you want to have the server running when you are testing locally, testing on the AMT “sandbox”, or running your actual experiment. If the server stops when running your actual experiment, Internet users will no longer be able to participate in your experiment even if you still have HITs posted on AMT’s website. Thus, you should think of the experiment server as meaning your experiment is current “live.”

**Debug/test the experiment locally**

Frequently you would like to test your experiment in your browser locally without involving Amazon’s servers at all. To do so, ensure that the experiment server is running (the prompt should show `server:on`). Then enter the command `debug`. A new browser tab will open with the first screen of the experiment. The URL string for this will look something like this:

```
http://localhost:22362/ad?assignmentId=debug7FIXMF&hitId=debugI3XW1P& workerId=debugY3UNQY
```

The `http://localhost:22362/` part is set in the configuration options under Server Parameters in the fields “host” and “port”. The default value, `http://localhost:22362/` is a special term that refers to your own computer. As mentioned above, if you wanted to run this experiment publically you would want to change the host option to `0.0.0.0`.

The remaining part of the URL created random (i.e., fake) identifiers which stand-in for the values that Amazon provides identifying the user, hit, etc... Since by default psiTurk does not allow individuals to take the same exper-
ment more than once (it checks for you to see if the worker has already completed the task or read too far into the instructions) these random values are helpful during debugging.

**Important:** When running in debug mode (i.e., when the assignmentId, hitId, and workerId variables are prefixed with the word “debug”) everything proceeds as usual. However, the server will not block the same user from restarting the experiment after finishing the instructions (as is true normally). This helps debugging since you don’t have to keep inventing new fake workerId. However, good to keep in mind this difference.

The first page that you see in the experiment looks something like this:

![Thank you for accepting this HIT!](image)

This is the page the AMT worker would see when they first accept the hit. When you click the link, a full screen window will open up which will run the experiment. You can test it now if you like just to get a sense of things. If you want to stop midway through that is no problem. Just close that browser window. Running debug again will open a new browser window and let you repeat the process.

**Important:** In the typical development cycle you would make changes to the javascript, CSS, or HTML files in your project locally and use debug to see those changes and test them. This way the development environment is the same as the one in which you will eventually deploy your experiment on Mechanical Turk.

**Experiment Structure**

The basic stroop demo lays out a pretty standard experiment sequence. It is perhaps most helpful to step through this sequence yourself, but conceptually:
First the users view an “ad” for the study (that is what is displayed above). Then they view a consent form and are asked to verify that they read and understood the consent. Next they are given a sequence of instruction screens. The experiment logs how long they look at each instruction screen as well as if they shift back and forth using the next/previous buttons. Then the main experiment begins which dynamically re-draws the browser window using Javascript. The psiturk.js API records the data and synchronizes it with your server from time to time. After the experiment finishes the user is given a simple questionnaire about their experiences in the task. Finally control is returned to Amazon (or if debugging a stand-in message is displayed). While all this is going on the psiturk.js API records if the user is changing windows and prevent them from reloading the browser mid-way into the task to start over.

Launch in AMT sandbox

Now that you’ve tested the experiment locally, you may want to see how it would appear on mturk before running it live with paid workers. Amazon offers this ability through the worker sandbox – a simulated environment that allows developers to test their HITs.

To create a hit in the worker sandbox, first check that the server is on and that you are in sandbox mode; the psiTurk prompt should say on next to server and sdbx next to mode. If you are in live mode, enter the command mode to switch to sandbox mode. If you are in live mode it will post your task to the live, paid AMT website instead of the free demo site.

When you are in sandbox mode if you type amt_balance you will see you have a never ending account with $10,000.00 of fake money to spend on sandbox HITs.

```
[psiTurk server:on mode:sdbx #HITs:0]$ amt_balance
$10,000.00
```

To create a hit, enter the command hit create, and then answer the prompts to set up the HIT. Your choices for the prompt answers are arbitrary for now, since the HIT won’t be completed by real workers. If the host variable in the config.txt file for this project is set to localhost (default) or 127.0.0.1 you will get an error reminding you that you server is no accessible to the general Internet. Please change this option before trying to post your task on AMT.

```
[psiTurk server:on mode:sdbx #HITs:0]$ hit create
number of participants? 5
reward per HIT? 1.00
duration of hit (in hours)? 1
**************************
Creating sandbox HIT
   HITid: 3SA4EMRVVJ2ALPN29GP6BDPNBS0P0
   Max workers: 5
   Reward: $1.00
   Duration: 1 hours
   Fee: $0.50

   Total: $5.50
Ad for this HIT now hosted at: https://ad.psiturk.org/view/oyG8sMCn9ySLTTrumsYgHe?
–assignmentId=debugFOFTCL&hitId=debugTSXLIB
```

This example create a hit with 5 “slots” for participants (or 5 assignments). The reward is $1.00 and the participant has 1 hour to complete the task after accepting the HIT before it will be returned. Finally the unique “ad” for this experiment/HIT is displayed at the bottom.
You can also run create_hit non-interactively by providing arguments when you run the command, for example create_hit 10 1.00 4.

You should now see the number “1” next to “#HITs:” in the psiturk prompt, denoting that you have one active HIT in the worker sandbox. If you type the command hit list active, you should see a description of your HIT including the HIT id:

```bash
[psiturk server:on mode:sdbx #HITs:1]$ hit list active
Stroop task
  Status: Assignable
  HITid: 3SA4EMRJZV2ALPNZGP6BDPNBS0P0
  max: 5/pending: 0/complete: 0/remain: 5
```

To test your HIT, go to the worker sandbox and search for your HIT by entering the name of your requester account in the search bar. You should see something like this:

Click “view a HIT in this group” to open a hit. You should see an ad for your HIT appear on the screen. Click “accept HIT”, then click the link in the HIT ad to open the experiment in a full-screen window. If you complete the HIT in this manner you it should go through all the steps of the AMT process. Afterwards you will have some data in your database.

**Accessing your data**

The simplest way to retrieve data is using the `download_datafiles` command. This creates three csv files containing the three kinds of data: trial data, question data, and event data.

If you are using the default SQLite database (see configuring databases) then another option is to use a GUI tool like Base to access the data in the `participants.db` file in your project folder.

If you set your database to use MySQL then you maybe able to connect and export the data using Sequel Pro.

**Automatically computing a bonus**

See *Example: Automatically computing performance bonus.*
Approve/Reject Workers

Todo: Document this

Assigning bonuses

Todo: Document this

Launch “live” experiment

To launch an experiment “live” you follow the same steps as launching in the sandbox but first set the “mode” of the command line to “live”:

```
[psiTurk server:on mode:sdbx #HITs:1]$ mode
Switching modes requires the server to restart. Really switch modes? y or n: y
Entered live mode
Shutting down experiment server at pid 55158...
Please wait. This could take a few seconds.
Experiment server launching...
Now serving on http://0.0.0.0:22362
[psiTurk server:on mode:live #HITs:0]$
```

Now if you run hit create it will post a hit on the live website. You must have enough money in your AMT account to pay for the HITs you are requesting, otherwise an error message will be displayed. The amt_balance command will let you check your current balance:

```
[psiTurk server:on mode:live #HITs:0]$ amt_balance
$178.70
```

Danger: Remember to switch back to “sandbox” mode when you are finished collecting data so that the command you type will not accidently create tasks that will charge you account money!

Conclusion

This concludes the conceptual overview of the Stroop example that ships with psiTurk.

1.10.2 Running psiTurk on Heroku

Heroku is a cloud service that lets you run applications in the cloud. You can run psiTurk on Heroku by preparing a git repository and then pushing it to Heroku which will deploy and autorun the code for you.

The benefits of Heroku include the following:

- It’s somewhat easier to manage than Amazon Web Services EC2 for the tech-wary (no need for security groups, no need to ssh in).
• You can set up a free PostgreSQL server (which is highly recommended to use over the default SQLite database that psiTurk uses). A database server is required on heroku as files, including participants.db, are ephemeral. Data would be lost every time the app spins down.

• You get free SSL for hosting your own ad.

• It’s scalable.

• You get a Heroku buffering server in front of your psiTurk gunicorn instance, which helps with performance a little bit.

One downside with Heroku is that it can get expensive if you need any kind of horsepower beyond 512MB memory and one node.

What follows is a step-by-step tutorial for setting up a psiTurk example experiment on Heroku (both the experiment itself and ad) with a PostgreSQL database for collecting data:

1. Go to the Heroku website and create a new account if you don’t already have one.

2. Make sure that psiTurk, git, and the Heroku Command Line Interface are installed on your computer.

3. Create a psiTurk example at a desired location (all commands listed in this tutorial are meant to be typed into your terminal application):

   ```
   psiturk-setup-example
   ```

   **Important:** If you’re starting from a preexisting psiturk app, you need to grab three files from `/psiturk/example`: requirements.txt, herokuapp.py, runtime.txt, and Procfile. Place them in your project root, next to your config.txt.

4. Navigate into your newly created psiTurk example folder:

   ```
   cd psiturk-example
   ```

   Or if you are starting from an already-existing psiturk project, navigate to your project root dir.

5. Initialize a Git repository in the root dir of your psiturk project the psiTurk (your current working directory):

   ```
   git init
   ```

6. Log in to Heroku (and put in your credentials when promted for them):

   ```
   heroku login
   ```

7. Create a new app on Heroku. Running this command will add a remote to your .git/config file, which will make it easier to run heroku commands from your project folder that are automatically associated with your newly-created Heroku app.:

   ```
   heroku create
   ```

8. Create a Postgres database on the newly created Heroku app:

   ```
   heroku addons:create heroku-postgresql
   ```

9. Get the URL of the Postgres database that you just created:

   ```
   heroku config:get DATABASE_URL
   ```

10. Get the URL of your app:
11. In your psiTurk example, open the config.txt file. Here, find and make the following settings for these rows, and then save the file.

**Note:** Heroku assigns a random port to every dyno. psiTurk handles this. The port setting in config.txt is ignored.

```plaintext
database_url = <Your Postgres database URL that you retrieved above>
host = 0.0.0.0
threads = 1
ad_location = https://<Your app URL that you retrieved above>/pub
use_psiturk_ad_server = false
```

12. Run the following commands, replacing <XYZ> with your access and secret keys for Amazon Web Services and psiTurk Secure Ad Server (you can also use this Python script to automatically run these commands, provided that you've filled out your credentials in your .psiturkconfig file. Running this script is the recommended approach!):

```plaintext
heroku config:set ON_CLOUD=true
heroku config:set psiturk_access_key_id=<XYZ>
heroku config:set psiturk_secret_access_id=<XYZ>
heroku config:set aws_access_key_id=<XYZ>
heroku config:set aws_secret_access_key=<XYZ>
```

You don’t need all of these settings if you run psiTurk standalone, independent of Amazon Mechanical Turk. However, you still need specify ON_CLOUD=true or psiTurk will not bind to the port specified by Heroku. In that case the app log on Heroku will say Error R10 (Boot timeout) -> Web process failed to bind to $PORT within 60 seconds of launch.

1. Stage all the files in your psiTurk example to your Git repository:

   ```plaintext
git add .
```

2. Commit all the staged files to your Git repository:

   ```plaintext
git commit -m "Initial commit"
```

3. Push the code to your Heroku git remote, which will trigger a build process on Heroku, which, in turn, runs the command specified in Procfile, which autolaunches your psiTurk server on the Heroku platform. Watch it run:

   ```plaintext
git push heroku master
```

4. Get the URL of the Postgres database that you just created:

   ```plaintext
heroku config:get DATABASE_URL
```

5. Get the URL of your app:

   ```plaintext
heroku domains
```

6. In your psiTurk example, open the config.txt file. Here, find and make the following settings for these rows, and then save the file:
Your local psiTurk instance needs these settings to communicate with the database on Heroku. You do not need to push these settings to Heroku to make them work. Specifically, be careful who has access to this file and do not push this information to any public git repositories:

Important: Anyone who has access to the database_url can connect to your database and has access to the data stored in it!

7. Run `psiturk` locally on your machine:

```
psiturk
```

8. To verify that your app is running, visit your `heroku` domain url in your browser. Obtain your `heroku` app url by running:

```
heroku domains
```

From that url, you can conveniently obtain a debugging url by clicking “Begin by viewing the ad.”

9. Run through your experiment hosted by heroku. You should now have some data in the database. To extract it into `csv` files, run locally:

```
psiturk download_datafiles
```

This should generate three datafiles for you in your local directory: `trialdata.csv`, `questiondata.csv`, and `eventdata.csv`.

Congratulations, you’ve now gathered data from an experiment running on Heroku!

From your local `psiturk` session, you can now create and modify HITs. When these are accessed by Amazon Mechanical Turk workers, the workers will be directed to the `psiturk` session running on your `Heroku` app. This means that it is never necessary to launch `psiturk` and run `server on` from _anywhere_ to run an experiment on Heroku. The server is automatically running, accessible via your Heroku domain url. (Of course, if you want to debug locally, you can still run a local server.)

Note: If you stay on the “Free” Heroku tier, your app will go to “sleep” after a period of inactivity. If your app has gone to sleep, it will take a few seconds before it responds if you visit its url. It should respond quickly once it “awakens”. Consider upgrading to a “Hobby” Heroku dyno to prevent your app from going to sleep.

Note: If you desire to run commands against your `postgresql` db, you can run `heroku pg:psql` to connect, from where you can issue postgres commands. You can also connect directly to your heroku postgres db by installing and running `postgresql` on your local machine, and passing the `DATABASE_URL` that you set in `config.txt` as a command-line option.
1.10.3 Using external survey tools with psiTurk

With the magic of iframes and javascript window messaging, you can integrate external survey tools into your psiTurk experiment. This is possible as long as the survey tool allows custom javascript to be triggered.

Window messaging allows cross-domain messaging via javascript, without having to configure security settings on the server. MDN says it best:

> “The window.postMessage method safely enables cross-origin communication. Normally, scripts on different pages are allowed to access each other if and only if the pages that executed them are at locations with the same protocol (usually both https), port number (443 being the default for https), and host (modulo document.domain being set by both pages to the same value). window.postMessage provides a controlled mechanism to circumvent this restriction in a way which is secure when properly used.”

Three special steps to hook up your survey to psiTurk:

1. Embed your survey as an iframe within one of your psiTurk pages or views.
2. Add a message event listener to your psiTurk window.
3. Post a message from the survey tool to the window.top when the survey is complete. window.top will be your psiTurk window. Do whatever you want via javascript once you receive the expected message.

To tie the psiTurk data and the external survey data together, embed a unique id into the iframe url you load, and then record that unique url into your survey data. Don’t forget to do this. If you forget, you won’t know to who to connect your survey data.

If you want to tie things both ways, post back your survey session id as part of the survey-complete post-back.

An example with Qualtrics

As of the time this documentation page was written, Qualtrics has an undocumented “feature”. Qualtrics automatically posts a window message to window.top when the Qualtrics “end of the survey event” is triggered. For Qualtrics surveys embedded as iframes in psiTurk experiments, we can take advantage of this behavior. The Qualtrics-posted message contains your survey_id and the participant’s Qualtrics-created unique session_id. You should already know the survey_id (because you just embedded a link containing this id), but the session_id is Qualtric’s unique id for whoever just finished your survey. You can record that with psiTurk as unstructured data (see Recording unstructured data) if you desire.

Don’t forget to explicitly log the psiTurk unique id as embedded data within Qualtrics. See here for more about embedding data into Qualtrics surveys.

The posted message when they finish a qualtrics survey is a string that looks like this:

```
QualtricsEOS|<survey_id>|<qualtrics_session_id>
```

So you can do something like this on your psiTurk page:

```javascript
// load your iframe with a url specific to your participant
$('#iframe').attr('src','<your qualtrics url>&UID=' + uniqueId);

// add the all-important message event listener
window.addEventListener('message', function(event){
    // normally there would be a security check here on event.origin (see the MDN link above), but meh.
    if (event.data) {
        if (typeof event.data === 'string') {
```

(continues on next page)
q_message_array = event.data.split('|');
if (q_message_array[0] == 'QualtricsEOS') {
    psiTurk.recordTrialData({'phase': 'postquestionnaire', 'status': 'back_from_qualtrics'});
    psiTurk.recordUnstructuredData('qualtrics_session_id', q_message_array[2]);
}

// display the 'continue' button, which takes them to the next page
$('#next').show();

This code can be put on a page that has a link with id #next default-hidden via css which advances the participant to the next experimental page. Note that this code checks that the event is QualtricsEOS before continuing on. That's because Qualtrics posts other events to window.top, too. This code is only interested in the EndOfSurvey event.

Also notice that this code doesn’t implement any security precautions. Normally it’s good practice to check to see where a message is coming from before you act on it. For instance, it might check to verify that the message is coming from a qualtrics.com domain. But in this code, the worst-case scenario is that a tech-savvy participant somehow triggers that they completed the survey before they actually did. In that case, their survey data would be blank, and after visual inspection their assignment could be rejected.

What about not-Qualtrics?

If your survey tool isn’t posting messages to window.top for you, just window.top.postMessage(<message>, <targetOrigin>) yourself. For instance, you might have javascript in your survey tool that does:

```javascript
window.top.postMessage("all_done|<survey_session_id>", ")
```

Then just listen for that event back on your psiTurk page, as in the Qualtrics example above.

1.11 Anatomy of a basic psiTurk project

Every psiTurk compatible project should include a few basic files. As an example, below is the file listing of the Stroop example which is included in a default psiTurk installation.

These files might all seem mysterious at first, but this section of the documentation explains their purpose. Of course, projects can include additional files as needed but these are basics that most projects will want to include.

1.11.1 config.txt

This is the basic configuration file for the project.

See also:
Local configuration files For details on the structure of these files.
1.11.2 custom.py

This file is optional. Most projects may not need this file at all. However, if you would like to extend the functionality of psiTurk in various ways, this file may be for you. In particular, this allows you to define custom “routes” or “urls” in your project. One example where this might be used is for creating routes that compute a participant’s bonus automatically.

See also:
Customizing psiTurk  For details on the structure of these files.

1.11.3 participants.db

By default, psiTurk will create a local SQLite database for storing data. You can also use a different database file or a MySQL database.

See also:
Configuring Databases  For a complete guide to databases with psiTurk.

1.11.4 server.log

The psiTurk web server process will not print to the Terminal. Instead, error messages and warning will be printed to the server log file. This will be created the first time you run the server.

See also:
Interacting with server log  The command for viewing the log file.
Logfile configuration options  Configuration options controlling the log file.

1.11.5 The static/ directory

The static folder holds files which are not dynamically altered by the psiTurk server (i.e., templated). This includes images, javascript libraries, CSS style sheets etc... You can add additional files and folders for static files if you need in your project.

It includes one top-level file (favicon.ico) which is the little icon that appears next to the URL in the browser window. You might want to customize this with the favicon.ico file used by your university or company.

In addition, there are typically four sub-directories:

The static/images/ directory

This folder should include all the image files (e.g., stimuli) used in your experiment. By default includes a university.png file which should be replaces with your university or company logo so that participants know the identity of your organization.

The static/css/ directory

This directory should hold all the CSS files you would like to use in your experiment (by default includes files shipped with Bootstrap and a style.css file which overrides some of those styles for particular parts of the instructions, ad, etc...).
The static/js/ directory

This folder should contain all your custom Javascript code for your project. In the Stroop example, this includes ‘task.js’ which includes the logic for the experiment and ‘util.js’ which includes some supporting/mathematical functions. You can add additional files as needed for your project.

The static/lib/ directory

This folder should contain all the external Javascript libraries that are needed by your project. It is a good idea to actually include copies of those libraries here instead of linking to a CDN or other URL. This was, far into the future, someone can re-run your experiment without have to hunt down an older version of the libraries you used. By default, the Stroop example includes libraries for Backbone, JQuery, d3.js, and underscore.js. These four are required for psiTurk to work properly but you can add other libraries for customization purposes.

The static/fonts/ directory

This directory should hold all the custom fonts for your project (by default includes fonts shipped with Bootstrap.)

1.11.6 The templates/ directory

The template folder holds the HTML templates for different parts of your experiment. You can add additional templates if needed for your project but this describes the basic set.

You can learn more about templates on the Jinja2 website.

The two most important files are ad.html and exp.html so be sure to review the documentation for those.

ad.html

This is a very important file. It contains the text of your HTML ad. This is the first thing participants taking your experiment will see. This file exists locally. When you are debugging in local mode, the local file will be used. When you create an ad on the Ad Server, a copy of this file is uploaded to the psiTurk cloud server.

See also:

psiturk.org Secure Ad Server You ad.html file is uploaded and stored on the Secure Ad Server when you create a hit.

Command line tool for creating HITs Info on how to create a HIT using the command line.

The structure of this file is very particular. There are two ways your ad will be viewed. First, when a potential participants is simply browsing the website, the will see one version of the ad. When the “Accept” the ad, the will see a second version that may include addition information (such as providing the link to launch your actual experiment).

These two types of adds are contained in the same file. Which one is displayed is set by the Jinja template (<http://jinja.pocoo.org/docs/>) The basic structure is:

```{% if assignmentid == "ASSIGNMENT_ID_NOT_AVAILABLE" %}
    HTML/CSS FOR AD BEFORE ACCEPTING
{% else %}
    HTML/CSS FOR AD AFTER ACCEPTING
{% endif %}```

(continues on next page)
Important: You cannot directly reference addition CSS or JS files in the ad since the ad server will host the ad using https://. As a result you need to include all CSS styles you want applied to your ad directly in the file. bootstrap.min.css is provided for free by the ad server.

For example, here is an example template that comes with the default stroop example.

```html
<!doctype html>
<!-- The ad.html has a very specific format.
Really there are two "ads" contained within this file.
The first ad displays to participants who are browsing
the Amazon Mechanical Turk site but have not yet accepted
your hit.
The second part of the ad display after the person selected
"Accept HIT" on the Amazon website. This will reload the
ad and will display a button which, when clicked, will pop
open a new browser window pointed at your local psiTurk
server (assuming it is running and accessible to the Internet).
See comments throughout for hints
-->
<html>
<head>
<title>Psychology Experiment</title>
<link rel=stylesheet href="/static/css/bootstrap.min.css" type="text/css">
<style>
/* these tyles need to be defined locally */
body {
    padding:0px;
    margin: 0px;
    background-color: white;
    color: black;
    font-weight: 300;
    font-size: 13pt;
}

/* ad.html - the ad that people view first */
#adlogo {
    float: right;
    width: 140px;
    padding: 2px;
    border: 1px solid #ccc;
}

#container-ad {
    position: absolute;
    top: 0px; /* Header Height */
</style>
</head>
<body>
</body>
</html>
```
1.11. Anatomy of a basic psiTurk project

---

Call for participants

The XXX Lab at XXXXX University is looking for online participants for a brief psychology experiment. The only requirements are that you are at least 18 years old and are a fluent English speaker. The task will that XXXXX minutes and will pay XXXXX.

This should display your experiment: who can participate, what the payment is, the time, etc...

If assignmentid is "ASSIGNMENT_ID_NOT_AVAILABLE" it means the participant has NOT accepted your hit.

{% if assignmentid == "ASSIGNMENT_ID_NOT_AVAILABLE" %}

{% endif %}

If you have already completed this task before the system will not allow you to complete it again.
allow you to run again. If this looks familiar please return the someone else can participate.

</div>
<p>Otherwise, please return the HIT so someone else can participate. click the "Accept HIT" button on the Amazon site the task.</p>

{% else %}

<!-- OTHERWISE

If assignmentid is NOT "ASSIGNMENT_ID_NOT_AVAILABLE"
participant has accepted your hit. thus show them instructions to begin the usually a button to launch a new browser pointed at your server.

important you do not change the code for the function below if you want you experiment

-->
<h1>Thank you for accepting this HIT!</h1>

By clicking the following URL link, you will be taken to the experiment, instructions and an informed consent agreement.

</p>
</script>

function openwindow() {
window.open('{{ server_location }}/consent?hitId={{ hitid }}&assignmentId={{ assignmentid }}&workerId={{ workerid }}','Popup','toolbar=no,location=no,status=no,menubar=no,scrollbars=yes,resizable=no,width='+1024+',height='+768+');
}

onunload = function() { location.reload(true) }
</script>

<div class="alert alert-warning">
<b>Warning</b>: Please disable pop-up blockers before continuing.
</div>

(continues on next page)
complete.html

This is a small HTML file that “completes” the HIT. When debugging locally this file does nothing other than display a message.

A different but similar version of this file is provided on the Secure Ad Server to register when tasks are completed.

consent.html

This is the informed consent form for your study. Place the text approved by your IRB here.

custom.html

A placeholder example of adding custom URLs/routes to your psiTurk application.

See also:
Customizing psiTurk For details on the structure of these files.

debriefing.html

This is the debriefing form for you study. It is optional, and up to you to display this HTML using your custom Javascript code.

default.html

A placeholder file that is shown when someone accesses the top-level route (i.e., http://myserver.edu:PORT/). It just redirects people to the ad.
error.html

A HTML file that handles various errors that can occur during your experiment. Most errors will result in this template being shown. You can customize what you want to show participants in the event of an error here.

A full description of error codes is available here.

exp.html

This is the main “experiment”. It is where the experiment “begins” for the subject.

**Important** this file MUST include the following code snippet

```html
<script src="static/lib/jquery-min.js" type="text/javascript"> </script>
<script src="static/lib/underscore-min.js" type="text/javascript"> </script>
<script src="static/lib/backbone-min.js" type="text/javascript"> </script>
<script src="static/lib/d3.v3.min.js" type="text/javascript"> </script>
<script type="text/javascript">
// Subject info, including condition and counterbalance codes.
var uniqueId = "{{ uniqueId }}";
var condition = "{{ condition }}";
var counterbalance = "{{ counterbalance }}";
var adServerLoc = "{{ adServerLoc }}"
</script>
<script src="static/js/psiturk.js" type="text/javascript"> </script>
```

In the header of the file. This sets up the necessary variables for communication with the psiTurk experiment server.

The last function that should be called in this file is `psiturk.completeHIT()` which will finalize the task.

Here is a default example experiment:

```html
<!doctype html>
<!--
    The exp.html is the main form that
    controls the experiment.
    see comments throughout for advice
-->
<html>
    <head>
        <title>Psychology Experiment</title>
        <meta charset="utf-8">
        <link rel="Favicon" href="static/favicon.ico" />
        <!-- libraries used in your experiment
            psiturk specifically depends on underscore.js, backbone.js_-->
        <script src="static/lib/jquery-min.js" type="text/javascript"> </script>
        <script src="static/lib/underscore-min.js" type="text/javascript"> </script>
        <script src="static/lib/backbone-min.js" type="text/javascript"> </script>
        <script src="static/lib/d3.v3.min.js" type="text/javascript"> </script>
    </head>
</html>
```

(continues on next page)
The instructions/ folder

This is is a folder of instruction screen you can configure for your experiment. You can add or remove files here. The psiturk.js API has functionality for a basic instructions system but you are welcome to write you own in Javascript.

list.html

A placeholder example of adding custom URLs/routes to your psiTurk application.

See also:

Customizing psiTurk For details on the structure of these files.
postquestionnaire.html

This is an example questionnaire you can give participants at the end of the task. The code for processing the form is contained in the psiturk.js API.

stage.html

This is a part of the default stroop example which is used to display the stimuli. It defines some default CSS elements which can be styled and used to show stimuli or instructions within a task.

1.12 Recording Data

1.12.1 Types of Data

To record data in your task, you make calls to the psiturk.js Javascript API. There are three kinds of data that psiTurk will help you produce:

1. Trial-by-trial log file
2. Unstructured (field, value) pairs
3. Browser events

Recording trial data

The first dataset that will be produced by your experiment will be a simple log file, which you add to a single line at a time. In order to add a line of data to the log, use psiturk.recordTrialData:

```
psiturk.recordTrialData(['this', 'is', 1, 'line'])
```

The list of values that you supply to recordTrialData will then be appended to the log. It is up to you how to structure those lists; you will have to parse them as part of your analysis. Each time you call psiturk.recordTrialData, it will also record the time it was called (with a UTC timezone).

Recording unstructured data

In addition to trial by trial data, there is often a need to record information about a participant in the form of (field, value) pairs, for which you can use psiturk.recordUnstructuredData:

```
psiturk.recordUnstructuredData('age', 24)
psiturk.recordUnstructuredData('response', 'yes')
```

Like the trial-by-trial data, it is up to you to decide whether or not to use this function. For some kinds of experiments (like simple surveys), this might be the only function you need.

Browser event data

The third dataset is generated automatically without any input from the experiment, and is used to track special kinds of events that occur as a worker is interacting with the page. Currently, this includes:

1. “resize” events: when the worker changes the size of their browser window (the first value recorded is the initial size of the window)
2. “focus” events: when the worker switches to and from a different browser window or application. If the worker leaves the experiment window, a “focus off” event is recorded; when they return a “focus on” event is recorded.

**Note:** Information about how to retrieve recorded data sets can be found here.

### 1.12.2 Saving the data

It’s important to remember that `psiturk.recordTrialData` and `psiturk.recordUnstructuredData` only modify the `psiturk` object on the client side. If you want to save the data that has been accumulated to the server, you must call `psiturk.saveData()`.

It’s up to you how often `psiturk.saveData()` syncs the task data to the server (e.g., after every block, or once at the end of the experiment). Using saveData frequently will limit the loss of data if the participant runs into an error, but keep in mind that it involves a new request to the server each time it is called.

### 1.13 Retrieving Data

This section covers methods for retrieving datasets, as well as the structure of saved data.

#### 1.13.1 Methods

There are several ways to retrieve experiment data from the database.

**Retrieving using `download_datafiles`**

The simplest way to retrieve data is using the `download_datafiles` command. This creates three csv files containing the three kinds of data: trial data, question data, and event data.

**Retrieving programmatically**

While the `download_datafiles` shell command is the simplest way to retrieve experiment data, a more powerful and flexible solution is to retrieve the data programmatically. Many languages offer libraries for interfacing with mysql and sqlite databases - below is an example using python and the sqlalchemy package to retrieve data from a mysql database. We add `+pymysql` to the `db_url` to let sqlalchneymake use of pymysql package. (You can leave the `database_url` in config.txt as `mysql://` though – psiturk adds `+pymysql` internally). By including code such as this at the beginning of your analysis script, you can be sure the the data you’re analyzing is always complete and up-to-date.

```python
from sqlalchemy import create_engine, MetaData, Table
import json
import pandas as pd

db_url = "mysql+pymysql://username:password@host.org/database_name"
table_name = 'my_experiment_table'
data_column_name = 'datastring'
# boilerplace sqlalchneym setup
engine = create_engine(db_url)
metadata = MetaData()
metadata.bind = engine
```

(continues on next page)
table = Table(table_name, metadata, autoload=True)
# make a query and loop through
s = table.select()
rows = s.execute()

data = []
# status codes of subjects who completed experiment
statuses = [3,4,5,7]
# if you have workers you wish to exclude, add them here
exclude = []
for row in rows:
    # only use subjects who completed experiment and aren't excluded
    if row['status'] in statuses and row['uniqueid'] not in exclude:
        data.append(row[data_column_name])

# Now we have all participant datastrings in a list.
# Let's make it a bit easier to work with:

# parse each participant's datastring as json object
# and take the 'data' sub-object
data = [json.loads(part)['data'] for part in data]

# insert uniqueid field into trialdata in case it wasn't added
# in experiment:
for part in data:
    for record in part:
        record['trialdata']['uniqueid'] = record['uniqueid']

# flatten nested list so we just have a list of the trialdata recorded
# each time psiturk.recordTrialData(trialdata) was called.
data = [record['trialdata'] for part in data for record in part]

# Put all subjects' trial data into a dataframe object from the
# 'pandas' python library: one option among many for analysis
data_frame = pd.DataFrame(data)

1.13.2 How the datastring is structured

The main data from an experiment participant is held in a string of text in the datastring field of the data table. Understanding how this string is structured is important to be able to parse the string into a useful format for your analyses.

The datastring is structured as a json object. In the description that follows, sub-objects are indicated by names wrapped in angle brackets (< >).
"currenttrial": trial_number_when_data_was_saved,
"useragent": useragent,
"data": <data>,
"questiondata": <questiondata>,
"eventdata": <eventdata>,
"mode": <mode>}

**data**

The data sub-object contains a list of the data recorded each time `psiturk.recordTrialData()` is called in the experiment:

```json
[
  {
    "uniqueid": uniqueid,
    "current_trial": current_trial_based_on_num_of_calls_to_recordTrialData,
    "dataTime": current_time_in_system_time,
    "trialdata": //<datalist>
  },
  //...
]
```

Here, `<datalist>` is whatever is passed to `psiturk.recordTrialData()` in the experiment. This could be in any format, such as a string or list, but we recommend saving data in a json format so that all data is stored in a clear, easy-to-parse “field-value” format. `<dataTime>` is recorded in UTC time.

**questiondata**

The questiondata sub-object contains all items recorded using `psiturk.recordUnstructuredData()`.

```json
{"field1": value1,
 "field2": value2,
 ...}
```

**eventdata**

The eventdata sub-object contains a list of events (such as window resizing) that occurred during the experiments:

```json
[{"eventtype": eventtype,
 "value": value,
 "timestamp": current_time_in_system_time,
 "interval": interval},
 ...]
```

# 1.14 Customizing psiTurk

Sometimes you might like to add additional urls or “routes” to your project. For instance you could make a password protected dashboard to visualize your data as it comes in, add additional functionality to your psiturk experiment, or
add more complex server-side computations (e.g., fitting a computational model to the subject in real time and using that to adapt the stimuli people view).

This can be achieved by using Flask Blueprints. psiTurk will look for a file called custom.py in the project directory, and import any blueprint from that module named custom_code. See below for examples.

### 1.14.1 Example: Automatically computing performance bonus

It is hard to use the main task to directly modify the database. However, you may use custom.py file with a function called compute_bonus to put the correct amount of bonus in the database. You could do this in Javascript perhaps but the problem is that participants can modify the javascript in their browser and increase their bonus. Instead it is better if bonuses are computed on the server side. The custom.py script may look like the following:

```python
from flask import Blueprint, request, render_template, jsonify, abort, current_app
# dealing with error
from psiturk.experiment_errors import ExperimentError
# Database setup
from psiturk.db import db_session, init_db
from psiturk.models import Participant
# dealing with json like reading from database
from json import dumps, loads
# explore the Blueprint
custom_code = Blueprint('custom_code', __name__, template_folder='templates', static_folder='static')
#----------------------------------------------
# example computing bonus
#----------------------------------------------
@custom_code.route('/compute_bonus', methods=['GET'])
def compute_bonus():
    # check that user provided the correct keys
    # errors will not be that gracefull here if being accessed by the Javascript client
    if not 'uniqueId' in request.args:
        # i don't like returning HTML to JSON requests... maybe should change this
        raise ExperimentError('improper_inputs')
    uniqueId = request.args['uniqueId']
    try:
        # lookup user in database
        user = Participant.query.
            filter(Participant.uniqueid == uniqueId).
            one()
        user_data = loads(user.datastring)  # load datastring from JSON
        bonus = 0

        for record in user_data['data']:
            # for line in data file
            trial = record['trialdata']
            if trial['phase'] == 'TEST':
                if trial['hit'] == True:
                    bonus += 0.02
        user.bonus = bonus
        db_session.add(user)
        db_session.commit()
    except:
        raise ExperimentError('improper_inputs')
```

(continues on next page)
resp = {"bonusComputed": "success"}
return jsonify(**resp)

except:
  abort(404)  # again, bad to display HTML, but...

Accordingly, in the main task file task.js, you would call this function with the computeBonus function. Add a piece of the code at the end of your experiment:

```javascript
psiTurk.computeBonus("compute_bonus", function () {
  psiTurk.completeHIT(); // when finished saving compute bonus, the quit
});
```

Now let’s walk through some key points of this process.

```python
from flask import Blueprint, request, render_template, jsonify, abort, current_app

custom_code = Blueprint('custom_code', __name__, template_folder='templates', static_folder='static')

@custom_code.route('/compute_bonus', methods=['GET'])

def compute_bonus():
  if not 'uniqueId' in request.args:
    # i don't like returning HTML to JSON requests... maybe should change this
    raise ExperimentError('improper_inputs')

  uniqueld = request.args['uniqueId']

Here we use request to receive the information sent from javascript. In our case it’s taken care by the computeBonus function. Looking into computeBonus to see where that “uniqueID” comes from:

```javascript
self.computeBonus = function(url, callback) {
  $.ajax(url, {
    type: "GET",
    data: {uniqueId: self.taskdata.id},
    success: callback
  })
};
```

As mentioned before, the url is the route name; the data is a dictionary with one key named “uniqueID”, which is being looked for in the python compute_bonus function.

Now let’s coming back to the compute_bonus function:
try:
    # lookup user in database
    user = Participant.query.\n        filter(Participant.uniqueid == uniqueId).\n        one()
    user_data = loads(user.datastring)  # load datastring from JSON

Now the database kicks in. We’ve created a user object which we will be able to read all data about this user that has been saved in the database, as well as write something.

```
    bonus = 0
    for record in user_data['data']:  # for line in data file
        trial = record['trialdata']
        if trial['phase'] == 'TEST':
            if trial['hit'] == True:
                bonus += 0.02

Now we calculate bonus by checking how many trials are correct.
```

```
user.bonus = bonus
    db_session.add(user)
    db_session.commit()
```

We assign value for the “bonus” column of this user and commit to the database. This will enable psiturk to give bonus.

```
    resp = {"bonusComputed": "success"}
    return jsonify(**resp)
```

Finally, we give this call-back message to the original query source, which is our psiTurk.computeBonus function. Trip is done, hurray!!

1.14.2 The basic logic of using custom.py

When is custom.py called?

It is loaded as a module when the psiturk server starts (called by psiturk/experiment.py) That is to say, you’d need to restart psiTurk whenever you’ve made some change of this script!

What is a route and why we need it?

A route is a URL served on the server. We need it because it is impossible for javascript to run python script (or any local files) directly. But you don’t have to call from javascript – equally, just access the address like http://localhost:5000/my_route in your browser!

(Note if my_route is expecting to receive arguments, like the participant ID, then the url becomes like http://localhost:5000/my_route?id=12345.)

Call the route from javascript directly without the psiturk function?

In the example above, we used the built-in function of computeBonus to call the custom route. Of course you can customize your own call for your favorite route, especially specifying the data sent to it. The key helper is ajax which is a jquery API. Add a call in your task.js that looks like this:
$.ajax("my_route",{
    type: "GET",
    data: {id: myid, data:mydata},
    success: function (response) {
        console.log(response)
    }
});

Note the `type` argument should be consistent with what your route function wants (usually either “GET” or “POST”). The `data` argument is usually a dictionary.

### 1.14.3 Tips about debugging your custom route

Debugging custom.py is tricky since the error message won’t just appear in your browser console. You will most likely see an “5000 internal error” which just means there is bug when calling your route. You may, however, try the following:

- Find your error message at `server.log`, which is automatically generated in your current psiturk folder and will record the error messages. This is usually the most informative tool.
- Print messages within your python function, which will appear in the psiturk shell.
- If you are not sure the route is being called, return some error message that will show in your browser (go to your browser with `http://localhost:5000/my_route`)

### 1.15 Dashboard Overview

A dashboard is available at route `/dashboard`. The dashboard can be enabled by setting `enable_dashboard` to `True`, and by setting a `login_username` and `login_pw`.

The dashboard has many features, including a dynamic filtered table, batch actions, managing campaigns.

#### Contents

- Dynamic Filtered Table
- Batch Actions
- Campaigns
- Tasks
- Configuration

#### 1.15.1 Dynamic Filtered Table

View current status of participants, queried from the psiTurk database.

- Filter by:
  - mode
  - experimental condition
  - current code version
– whether the participant has a status of ‘complete’

• Group by:
  – condition

1.15.2 Batch Actions

Functionality to manually trigger the following actions:

• Workers:
  – Approve all HITs
  – Bonus all submissions via the “auto” method (based on the value set in the “bonus” column in the database).

• HITS:
  – Expire all
  – Approve all workers for all hits
  – Delete all

1.15.3 Campaigns

Campaigns are scheduled as tasks. Campaigns have the following features:

• Set a target “goal” for number of workers for a given task code version.

• Stagger the posting of HITs by a specified interval.

• Post HITs in batches of 9 assignments. This keeps the MTurk commission at 20%, instead of 40%.

• Monitors the number of available HITs, and continues posting rounds of HITs until the campaign goal has been met.

• Manually cancel a campaign.

The “Campaigns” tab also displays past campaigns.

1.15.4 Tasks

psiTurk “tasks” are stored in their own table in the database specified by the database_url. To enable a specific psi turk server to run tasks, set do_scheduler to true.

The “tasks” tab allows for scheduling an “Approve All workers” task which will be run at a set interval. Will “approve” all submissions currently marked as “Submitted” in the psiTurk database.

The tab will also display any currently running campaigns. To edit a campaign, visit the “Campaigns” tab.

**Warning:** The managing of tasks and the running of tasks is handled separately!

This is because psiTurk uses APScheduler for tasks, which does not currently handle interprocess synchronization (see this APScheduler FAQ).

This means that any dashboard can view, create, delete, and update tasks, while a single separate psiTurk server instance can be set up with only one thread for task-running.
Note!: if `do_scheduler` is set to True, and `threads` is greater than 1, psiTurk will refuse to start! This is a safeguard, because, again, APScheduler cannot handle interprocess task-running synchronization.

1.15.5 Configuration

- Set dashboard mode.
- View AMT balance.

1.16 Experiment Exchange

The principle of the experiment exchange is to allow researchers to share the code for psiTurk-compatible experiments, in the name of science reproducibility. Other researchers should be able to easily download psiTurk experiment code and re-run it using the same population and task code. In addition, researchers can use the experiment exchange to learn about the code used in other people’s experiments.

1.16.1 Contributing to the exchange

If you use psiTurk to design your experiment, please consider sharing the code! To do so, do the following:

- upload your code to a GitHub repository
  - Be sure not to commit any sensitive information, such as AWS credentials or a `database_url` containing credentials!
- propose an edit to this page, listing information about your experiment. This documentation page is hosted on GitHub.

Alternatively, if you have trouble with the above, open an issue on the project github repository page, and we’ll be glad to help you share your project. Science!

1.16.2 Browse experiments for study or replication

If you find one of these to be useful, consider “star”-ing it on GitHub to show your appreciation and to signal its usefulness to others.

Experiments

- Draw Together
- Go / no-go task using jsPsych
- Surveillance Task
- Mental rotation of 2D line drawings
- Whoops y’all
- Scalar Implicatures
- DFE sampling paradigm
- neurosemantic features
Draw Together

**Link**  https://github.com/NYUCCCL/drawtogether

**Keywords**  crowd source art, fun, demo

A simple experiment where workers are asked to draw an image using their computer mouse. A custom route (/gallery) provides a visual summary of all the creative images that people drew.
Go / no-go task using jsPsych

This is an example experiment to demonstrate how to use the jsPsych library with psiTurk. The experiment is a simple go/no-go task: subjects respond to blue circles as fast as possible, but ignore orange circles. Reaction times are measured and an average reaction time for correct responses is shown at the end. The experiment highlights a number of jsPsych features that can be used in a variety of experiments.

Surveillance Task

This task measures attention and rapid responding.
Mental rotation of 2D line drawings

This is a behavioral experiment based on classic mental rotation studies (Shepard & Metzler, 1971; Cooper, 1975). In this experiment, participants view pairs of images and must determine whether the images depict the same object (which has possibly been rotated), or mirror-image objects (which could also be rotated). For further details, see Hamrick & Griffiths (2014), available from http://www.jesshamrick.com/publications.

Whoops y’all

Whoops y’all is a psiTurk compatible experiment for paying people when an experiment goes badly for some reason. You enter the worker IDs of people who you owe money to and can reject all others. Payment is handled quickly and easily via psiTurk’s command line features. When you make a whoops, use “whoops y’all”!
Scalar Implicatures

Link  https://github.com/fdabl/scalar-implicatures
Keywords  pragmatics, incremental validity, decision-making

DFE sampling paradigm

Link  https://github.com/dmarkant/SamplingParadigmDFE
Keywords  decisions from experience, sampling paradigm, information search, decision making

The experiment implements a version of an incremental verification task in which binary truth-value judgments are collected for sentence-picture pairs, while the picture is only incrementally revealed upon subjects’ request. The task helps to obtain information about the relative preferences for different readings of potentially ambiguous sentences from categorical answers. Interesting from a programmer’s point of view is the use of [browserify](http://browserify.org/) which provides a module system for JavaScript. Using browserify transforms allowed us to write the experiment in the next version of JavaScript, providing us with nice new language features.
neurosemantic features

**Link**  https://github.com/stubridy/neurosemantic_norming

**Keywords** stimulus norming, semantic features

This code was used to get ratings of the association between “neurosemantic” features – features of concepts that map onto known functional specializations in the human brain (e.g., smell, color, motion) – and single word concepts. Contains examples of using task.js to request stimulus calculation and construction from the server side via ajax calls to custom.py. The examples included query a sql database to decide which words to show a participant, but this was particular to our data storage. That code could be swapped out for anything one might use to construct stimuli and return to js client in a jsonify’ed object containing the expected fields.

SearchExperiment

**Link**  https://github.com/kanakakis/search_experiment

**Keywords** trust

The aim of this experiment is to study the users’ trust dynamics regarding the accuracy and confidentiality of the ACME search engine

jsPsych CoffeeScript go/no-go example
This is an updated example of the integration between PsiTurk and jsPsych. It is a rewrite of the example code given in the jsPsych tutorial using Literate CoffeeScript. It uses some code used in jodeleeuw’s experiment example (also available on GitHub). It also demos post-trial data computation and collection from jsPsych to PsiTurk.

**Mushroom-foraging experiments with varying horizons**

[Images of mushrooms]

Four experiments in which the participant plays the role of a mushroom forager attempting to eat healthy mushrooms while avoiding poisonous ones. The purpose of the experiments is to test whether people employ a forward-looking exploration strategy, sampling mushroom species more when they expect to see them more times (Exp 1a and 1b) or with greater frequency (Exp 2a and 2b) in the future. The repository contains four experiments. To run an experiment, first `cd` into the appropriate folder after running the `psiturk-install` command.

**Biased hypothesis generation**

[Images of hypothesis generation]

Two experiments examining how biased hypothesis generation affects the ability to learn categorical rules through self-directed sampling. In both experiments, the representation of two feature dimensions is manipulated in order to affect the kinds of hypotheses people generate as they attempt to learn an unknown category boundary (i.e., either 1D
or 2D boundaries). Experiment 1 is a perceptual task while Experiment 2 applies the same design to a more abstract problem.

**Microchip intervention task**

![Microchip Task Image](image)

**Link** https://github.com/annacoenen/chipTask

**Keywords** causal learning; active learning; interventions

This is the causal intervention task “microchip task” used in Experiment 1 of Coenen, Rehder, & Gureckis (2015) http://gureckislab.org/papers/CoenenRehderGureckis2015.pdf. A few notes: * Caveat: This code is several years old and it is the first bit of JavaScript I’ve ever written. Please get in touch if I can help with any questions, at coenen.anna@gmail.com. * To change the appearance of the chips, edit static/images/mainboard.svg * To change the causal structures, you will need to change the information in static/lib/likelihoods.js, which holds outcome probabilities for every structure and every intervention, as well as the structural description of each graph.

**Stroop Example with jsPsych**

![Stroop Task Image](image)

**Link** https://github.com/alexanderrich/stroop-jspsych

**Keywords** stroop, simple, example, jspsych

A “port” of the psiTurk built in stroop experiment example to use jsPsych (version 6.0.0)
Change Detection

This is the change detection task that is often used to assess working memory capacity (K). Subjects are given a number of colored circles and are asked to remember as many as possible. After a short delay, the circles return to the screen. 50% of the time, one circle will have changed colors. The subject then responds with ‘S’ for same or ‘D’ for different. The following formula can then be used to get a measure of the subject’s capacity: 

\[ K = (\text{hit rate} + \text{correct rejection rate} - 1) \times \text{set size} \]

Probability Discrimination Game

The is the code for an experiment in which people see two groups of marbles and are asked to choose the group with the greatest chances of drawing a white marble at random. It a series of images sequentially for 750ms followed by an screen to reduce sensory memory. Right/Left response data as well as reaction time data are collected for each image.
Idea Generation / Divergent Thinking

**Link**  https://github.com/rickhass/Idea-Generation

**Keywords**  creativity, divergent thinking, semantic memory, Alternative Uses, Consequences

This experiment allows for administration of common divergent thinking tasks (e.g., Alternative Uses for Objects). It consists of six prompts (3 alternative uses, 3 consequences) that last 3 minutes each, and a practice prompt (naming colors). The key feature of the experiment is the collection of RTs in addition to typed responses. This allows for the examination of idea generation / divergent thinking with some of the tools used by semantic memory researchers. As is, the prompts are written text, but the code can be extended to include images as prompts. An R-Markdown file with the data from a recent experiment is available on OSF (https://osf.io/eux2k), which includes details on parsing the datastring, with a customized function for quickly doing so.

1yearproject

**Link**  https://github.com/yangyuxue1994/1yearProject

**Keywords**  picture description, cognition, language

This is a picture description task

Auditory Lexical Decision & Identification Tasks

**Link**  https://github.com/JSlote/cswro-exp-1

**Keywords**  auditory, spoken word, lexical decision, ldt, identification, sound, speech, speech perception
This is the code repository for Experiment One of the above-titled study. It includes implementations of auditory lexical decision and identification tasks. The experiment is designed to be run using psiTurk version 2.1.1 on the Amazon Mechanical Turk platform. Of general interest are the following features:

- Cursor auto-hiding during experiment proper,
- Audio preloading including a progress bar pop-up,
- Fullscreen requirement to mitigate distraction (participants are asked to enter fullscreen and the experiment is paused (all input blocked) if they exit prematurely),
- Basic asynchronous flow control for transitioning between stages of the experiment,
- and Audio reCaptcha integration (you will have to input your reCaptcha keys in custom.py and task.js for this feature to function).

You are welcome to use this code for personal or academic uses. If you use all or portions of this project in an academic paper, please cite as follows:

Slote, J., & Strand, J. (2015). Conducting spoken word recognition research online: Validation and a new timing method. Behavior Research Methods. doi: 10.3758/s13428-015-0599-7. For more information about this study or the Carleton Perception Lab, please visit https://apps.carleton.edu/curricular/psyc/jstrand/research/resources/

**1.17 Frequently Asked Questions**

**1.17.1 How do I host a psiTurk experiment for MTurkers?**

You need to host your server on any computer that is reachable from the public internet. Consider hosting your server on a cloud provider such as Heroku, AWS, DigitalOcean, etc.

See also:

*Running psiTurk on Heroku*

**1.17.2 I'm trying to run psiTurk at home using a cable modem or other connection. Will it work?**

In general this set up is possible via port fowarding, if you have access to and are comfortable configuring your home’s router. However, it is not recommended. Consider instead deploying your psiturk study on Heroku or another cloud provider.

**1.17.3 My university will not give me a static IP address. Can I still use psiTurk?**

You can still use psiTurk if you have access to a computer with a public IP address, or that can receive public traffic. See *How do I host a psiTurk experiment for MTurkers?*.

**1.17.4 I insist on running my experiment from my home, despite the insanity of doing the same.**

psiTurk experiments can be hosted on almost anything that has an internet connection and a public port, such as an office computer or laptop. You’ll need a static IP to prevent your experiment’s URL from changing. Users without one (e.g., most home users) can use a dynamic DNS service to forward a URL to their dynamic IP. Here’s a list of free DDNS providers. You also may need to forward a port from your home routers to your personal computer.
1.17.5 Can I use psiTurk for non-MTurk studies?

Yes! psiTurk launches a server that MTurk merely points to. For each accepted HIT, MTurk appends information about the workerId, assignmentId, and hitId that psiTurk uses to create a record for the participant in the psiturk database. psiTurk also reads a mode parameter from the url, which, for AMT studies, is either sandbox or live.

If you want to recruit participants via not-AMT, then you only must somehow generate URLs for your participants including the above keys.

Todo:
- also describe changing the complete.html template
- point to a google-group discussion of someone doing the above

1.17.6 There was an error in my experiment and I need to pay participants, but I can’t because they weren’t able to complete my HIT. How can I pay them?

You need Whoops y’all.

Whoops y’all is a psiTurk compatible experiment for paying people when an experiment goes badly for some reason. You enter the worker IDs of people who you owe money to and can reject all others. Payment is handled quickly and easily via psiTurk’s command line features. When you make a whoops, use “whoops y’all”!

See Whoops y’all on GitHub.

1.17.7 Why doesn’t psiTurk work on Windows?

Windows has very limiting security restrictions which prevent server processes from running. As a result we cannot support Windows. Instead we support all system based on an underlying Unix kernel which can run python. This include Mac OS X and Linux.

1.17.8 I need an experiment to do X, will psiTurk be able to do this?

Generally, any standard psychology experiment can be run using psiTurk. This means experiments with multiple trials, trials which change based on participant’s past responses, experiments with multiple phases or trial types, surveys, experiment recording reaction time, mouse tracking experiments, decision making, etc. The possibilities are actually not as much a function of psiTurk as of the capabilities of programming an experiment in Javascript. Any web application or applet that runs Javascript should play nicely with psiTurk with a little hacking. psiTurk mostly just provides the server and data logging capabilities, and it is up to you to define how your experiment actually looks and behaves.

There are examples in the experiment exchange which provide a more concrete understanding of the scope of things people have attempted with psiTurk.

One place where psiTurk currently hasn’t been used is group or multi-player experiments (although we’ve heard rumors of users who have reported success with this). In addition, we are not aware of people using psiTurk yet for multi-day or multi-session experiments. This is not a technical limitation per-se but may require some hacking. We’d be happy if someone tried to do these types of experiments and reported back about what we could add to the core psiTurk code to help with this.
1.17.9 I'm having trouble with my AWS/AMT credentials

In order to use your credentials you must create a requester account on Amazon Web Services. This usually involves providing a credit card number as well as a phone verification step. Finally, some users report having to log into http://requester.mturk.com at least once to agree to the software terms. Read the Setting Up an Amazon Mechanical Turk Account guide carefully.

1.17.10 Can you program my experiment for me?

Nope, sorry. Please check the Experiment Exchange for examples you might be able to draw insight from.

1.17.11 I'm having Javascript errors when designing my experiment. Can you help?

Sorry, but probably not. See the above about programming experiments. There are many ways of getting help with psiTurk specifically and many excellent tutorials online for developing web applications using Javascript.

1.17.12 Where is the /static/js/psiturk.js file? It doesn’t appear in any of the experiments I have downloaded!

psiturk.js doesn’t actually “exists” as a file in the static folder of any project. Instead, the psiturk server/command line tool automatically generates this file. The best way to view it is by “view source” in your browser while debugging your experiment. While somewhat unintuitive, this ensures that changes to psiturk.js are linked to new versions of the overall psiturk command line tool (since they are tightly interdependent). Alternatively, view the source of the file on GitHub.

1.17.13 How do I interpret the hit list counts of “Pending,” “Complete,” and “Remain”?

- MTurk defines “Completed” as submissions that you have either Approved or Rejected.
- MTurk defines “Pending” as submissions that have been “accepted” by a worker or that are being “viewed” by a worker. A worker has the “hit duration” to complete the hit. Many users use tools that automatically accept HITs for them and put them in a queue. Workers may not begin working on your hit until it is close to the duration expiry.
- Outstanding submissions that need to be either approved or rejected before the hit can be deleted.

1.17.14 Immediately after I post my HIT on the “live” mode of AMT, I cannot find it via an mturk dashboard search?

Many MTurkers use tools that automatically accept HITs for them and put them in a queue. If all of your HITs get gobbled up before the MTurk GUI refreshes, then your HIT will never appear via a search on the MTurk GUI.

1.18 Getting help

There are a number of ways to get help with psiTurk.

1. The https://psiturk.org has meta-information about the system.
2. Usage questions should go to the psiTurk Google Group here. Search for answers to common questions or post your own. Chances are if you run into a problem someone else will as well.

3. Potential bugs in the code, or feature requests, should be posted as a github issue. This is an open discussion of possible issues, bugs, feature requests, etc. Browse the open and closed issues first before posting. See the guide for contributors for more information about using the issues tracker.

4. Todd Gureckis taught a class covering online data collection and psiTurk at NYU Spring 2014. All lectures were videotaped and are available here.

5. Follow @psiturk on Twitter for helpful tips and breaking news.

6. If all else fails and you feel you simply cannot get help, you can consider emailing authors@psiturk.org, the benevolent dictators of the project and system architects. However, if you haven’t first pursued the above options, you may not get a quick response.

1.19 Disclaimer

psiTurk is free, open source software provided to scientists to aid in research. Because it helps you run paid experiments online using Amazon Mechanical Turk errors in the software, or in your use of the software, can lead to loss of money. This is the very nature of online research (errors may mean someone will actually do your task and you need to pay them as a result).

Our belief is that these types of errors can be best limited by having open, peer-reviewable software and sharing bug reports between labs and research groups. In other words, even if you wrote this software yourself it is possible that some bug could cost you money when getting started.

Danger: We take no responsibility for your use of the software. We make no claims that it is bug-free and any errors are not our responsibility. This is a community-run, community-supported system and not a company selling a product. We use the software in our lab and, when used correctly, has never caused us to lose money on Mechanical Turk due to mistakes. However, it is always possible to mis-use the software in a costly way.

In addition, while we strive to keep the psiturk.org Secure Ad Server running, crashes in that system could, in the short-term, affect your ability to collect data. Again, using the system you must understand what the risks are. The good news is that because the system is open source if there is a problem everyone can read the code themselves and make suggestions on how to fix things.

Some suggestions to avoid costly mistakes from happening:

1. Test your code a lot in the sandbox to make sure every stage is working and you understand what psiTurk is doing.

2. Run small batches at a time to verify everything is working

3. Keep your Amazon payments account balance reasonably low at any point in time. It is impossible to spend more money than is in your account at any point in time.

4. Exit the psiTurk server when you are not using it to collect data (i.e., do not leave psiTurk server running indefinitely). This ensures that no one will be able to actually perform your task and then claim they are owed payment. This also limits the ability of bots and other scammers to reverse engineer your task.

5. When testing “live”, explain in the text of your Ad that this is a test and you are looking for feedback. Workers get frustrated when you put bad or broken experiments online, but are often very helpful if you explain that you are hoping to get feedback on an unfinished project.
2.1 Contributing to psiTurk

Note: This guide is copied more or less from the contributors guidelines of the gunicorn project. Alternations were made for the nature of this particular project. An up to date copy of this guide always resides here.

Want to contributed to psiTurk? Awesome! Here are instructions to get you started. We want to improve these as we go, so please provide feedback.

2.1.1 Contribution guidelines

Pull requests are always welcome

We are always thrilled to receive pull requests, and do our best to process them as fast as possible. Not sure if that typo is worth a pull request? Do it! We will appreciate it.

If your pull request is not accepted on the first try, don’t be discouraged! If there’s a problem with the implementation, hopefully you received feedback on what to improve.

We’re trying very hard to keep psiTurk lean, focused, and useable. We don’t want it to do everything for everybody. This means that we might decide against incorporating a new feature. However, there might be a way to implement that feature on top of psiTurk.

Discuss your design on the mailing list

We recommend discussing your plans in our Google group before starting to code - especially for more ambitious contributions. This gives other contributors a chance to point you in the right direction, give feedback on your design, and maybe point out if someone else is working on the same thing.
Create issues…

Any significant improvement should be documented as a github issue before anybody starts working on it.

…but check for existing issues first!

Please take a moment to check that an issue doesn’t already exist documenting your bug report or improvement proposal. If it does, it never hurts to add a quick “+1” or “I have this problem too”. This will help prioritize the most common problems and requests.

Conventions

Fork the repo and make changes on your fork in a new feature branch:

- If it’s a bugfix branch, name it XXX-something where XXX is the number of the issue
- If it’s a feature branch, create an enhancement issue to announce your intentions, and name it XXX-something where XXX is the number of the issue.

Make sure you include relevant updates or additions to documentation when creating or modifying features.

Write clean code.

Pull requests descriptions should be as clear as possible and include a reference to all the issues that they address.

Code review comments may be added to your pull request. Discuss, then make the suggested modifications and push additional commits to your feature branch. Be sure to post a comment after pushing. The new commits will show up in the pull request automatically, but the reviewers will not be notified unless you comment.

Commits that fix or close an issue should include a reference like Closes #XXX or Fixes #XXX, which will automatically close the issue when merged.

Add your name to the THANKS file, but make sure the list is sorted and your name and email address match your git configuration.

Contributing to the docs

Our docs are currently hosted at readthedocs. Readthedocs uses Sphinx as the backend for their documentation so in order to update the docs you will first have to install Sphinx simply by typing:

```
easy_install -U Sphinx
```

on the command line.

There’s a Makefile in the docs directory, so you can generate the docs by running make on the command line, for example:

```
make html
```

will generate the html docs in _build/html. Running make with no arguments will show you the available subcommands.

All documentation files are in the docs folder and are formatted as reStructured Text. A good, detailed manual for the reStructured Text syntax can be found here.

Some essentials:
The index page is the main page that users see when they open the docs. It is also how readthedocs generates the sidebar that contains all the names of individual pages in the documentary so it is important that this is formatted correctly.

The main important feature is the toctree.

The toctree just looks like this:

```rst
.. toctree::
   :maxdepth: 1
   :titlesonly

   forward
   install
   quickstart
   recording
```

Sphinx will go through the pages listed in the toctree, search for subject headers and create both links for the index page and the sidebar in the correct format in the order that the pages are listed. For this reason, it is also very important that subjected headers be used correctly on the individual pages. For example, the forward page has a title that looks like this:

```
Forward
=======
```

and subtitles that look like this:

```
What is psiTurk?
~~~~~~~~~~~~~~
```

It actually doesn’t matter what character you use for the underline, it can be any of

```
= - ' " ^ _ * + # < >
```

but it must be consistent since all headers with the same character will be at the same level. For convenience, we are using ===== to mean title and ~~~~~ to mean sub header. Some other basic things in rST:

Links look like this:

```
"Getting psiTurk installed on your computer <install.html>"_
```

with the actual page in angle brackets. If the link is to another page within the docs, you only need to include the name of the page. Whenever you include a code example, put this line before:

```
.. code:: javascript
```

All pages on readthedocs.org (including this one) have a link to “Edit on Github.” This can be a great way to “steal” formatting ideas for your documentation edits.

### 2.1.2 Decision process

**How are decisions made?**

In general, all decisions affecting psiTurk, big and small, follow the same 3 steps:

- **Step 1:** Open a pull request. Anyone can do this.
- **Step 2:** Discuss the pull request. Anyone can do this.
- **Step 3:** Accept or refuse a pull request. The little dictators do this (see below “Who decides what?”)
Who decides what?

psiTurk, like gunicorn, follows the timeless, highly efficient and totally unfair system known as Benevolent dictator for life. In the case of psiTurk, there are multiple little dictators which are the core members of the gureckislab research group and alumni. The dictators can be emailed at authors@psiturk.org.

For new features from outside contributors, the hope is that friendly consensus can be reached in the discussion on a pull request. In cases where it isn’t the original project creators John McDonnell and/or Todd Gureckis will intervene to decide.

The little dictators are not required to create pull requests when proposing changes to the project.

Is it possible to become a little dictator if I’m not in the Gureckis lab?

Yes, we will accept new dictators from people esp. engaged and helpful in improving the project.

How is this process changed?

Just like everything else: by making a pull request :)

2.2 Project Roadmap

psiTurk is always looking to improve and to increase the number of contributors. We thought it would be helpful to lay out a basic roadmap of where we would like to see the project go in the future. This roadmap may inspire you to implement a new feature!

2.2.1 General priorities

Documentation

The documentation is greatly lagging behind progress on the psiTurk platform. We need help with people debugging documentation, improving it, and making additions! Notice how all documentation pages (including this one!) include a link to “Edit on GitHub”. Make a pull request and help us improve these docs!

Automated testing

The version 2.0 release introduced a number of new features which are fairly complex because they require communication over the Internet, RESTful APIs, etc. While there are automated unit tests for many of these features, it is important to have better tests of these features. Testing isn’t glamorous but writing tests improves your health, looks, and chances of getting in heaven.

Alternative database solutions

Currently psiTurk offers a variety of database solutions including local SQLite files, self-administered MySQL servers, and MySQL processes hosted on Amazon’s Web Services (RDS) platform. However, all of these are a little clunky and require users to know quite a bit about data management. The demands placed on these databases by a single experiment are not excessive, and thus there might be a more robust solution (e.g., NoSQL). One possibility is to host a robust cloud-based data API off psiturk.org.
psiturk.js

All projects currently should use psiturk.js to save data to the server and update the user status as they progress. It might be nice if these included additional features including easily displaying instructions, providing simple quizzes, etc... In theory many parts of the psiturk command shell could be moved into the psiturk.js library (e.g., one could even create hits and ads via javascript calls). This might eventually allow the power of the psiturk platform to be leveraged even on simple, standard web server platforms (i.e., not relying on Flask).

Ad Server

The Ad Server has the potential to gather valuable data about participants in studies, how naive they are, etc... Currently only a limited number of statistics are gathered, and much of this data is not publically accessible via an API or interface. Future versions of the psiturk.org dashboard could provide users with more interesting statistics about participants in their experiments, their geographic location, etc...

Unique IP issues

A major issue with psiTurk is that it requires a unique, Internet addressable IP address. This is a hurdle at some universities or companies. This is a bug and a feature at some level. The feature side is that for many users the ability to serve experiments off their local computer obviates the need for a dedicated server and simplifies some web security issues. For other users thought this is a frustrating hurdle to overcome in order to use psiturk. We are interesting in the community’s thoughts about this and suggestions about best practices include cloud based hosting systems like SalesForce’s Heroku and Amazon’s AWS.

2.2.2 Version 3.0

We envision that eventually psiturk could move entirely into the cloud (i.e., no need for user to install command line tool). This may be supported by changes and extensions to the psiturk.org API and the psiturk.js library. The emphasis in our initial development has been on advanced users/programmers comfortable in a unix environment, but future version could emphasize novice web programmers who are new to online experiments (e.g., undergrads).

If you have ideas about future directions for the project the Github issues tracker is a great place to share them.
3.1 psiturk.js API

Everything in the psiturk.js API is scoped under the psiturk namespace.

3.1.1 Creating the psiTurk object

To use the psiTurk library, a psiturk object must be created at the beginning of your experiment. It takes two key arguments uniqueId and adServerLoc. These two variables are first created in exp.html. They tell psiTurk which unique number/code corresponds to the current participant (allowing updating of data as the task progresses) and the location of the ad where users should be sent when the task is complete.

```javascript
// Create the psiturk object
var psiTurk = PsiTurk(uniqueId, adServerLoc);

// Add some data and save
psiturk.addUnstructuredData('age', 24)
psiturk.setData();
```

The following documents the javascript API.

3.1.2 psiturk.taskdata

taskdata is a Backbone model used to store all data generated by a participant and to sync it to the database.
taskdata has the following fields with these default values:

```
condition: 0
counterbalance: 0
assignmentId: 0
```

(continues on next page)
workerId: 0
hitId: 0,
useragent: "",
currenttrial: 0
data: ""
questiondata: {}
eventdata: []

These variables are either set during initialization or using the methods of the psiturk object. However, since taskdata is a Backbone model, you can always access their values directly using the Backbone `set` and `get` methods, which may be useful for debugging. For example:

```
psiturk.taskdata.set('condition', 2);
psiturk.taskdata.get('condition');
```

### 3.1.3 `psiturk.preloadPages(pagelist)`

For each path in pagelist, this will request the html and store in the psiturk object. A given page can then be loaded later using psiturk.getPage(pagename).

Example:

```
// Preload a set of HTML files
psiturk.preLoadPages(['instructions.html', 'block1.html', 'block2.html']);

// Set the content of the body tag to one of the pages
$('body').html(psiturk.getPage('block1.html'));
```

### 3.1.4 `psiturk.getPage(pagename)`

Retrieve a stored HTML object that has been preloaded using `psiturk.preLoadPages`.

### 3.1.5 `psiturk.showPage(pagename)`

Set the BODY content using an HTML object that has been preloaded using `psiturk.preLoadPages`.

Example:

```
psiturk.preloadPages(['instructions.html', 'block1.html', 'block2.html']);
psiturk.showPage('instructions.html');
```

### 3.1.6 `psiturk.preloadImages(imagelist)`

Cache each image in imagelist for use later.

### 3.1.7 `psiturk.recordTrialData(datalist)`

Add a single line of data (a list with any number of entries and any type) to the psiturk object. Using this will not save this data to the server, for that you must still call `psiturk.saveData()`.
Example:

```javascript
// data comprised of some list of variables of varying types
data = ['output', condition, trialnumber, response, rt];
psiturk.recordTrialData(data);
```

### 3.1.8 `psiturk.recordUnstructuredData(field, value)`

Add a (field, value) pair to the list of unstructured data in the task data object.

Example:

```javascript
psiturk.recordUnstructuredData('age', 24);
```

### 3.1.9 `psiturk.saveData([callbacks])`

Sync the current psiTurk task data to the database.

An optional argument `callbacks` can provide functions to run upon success or failure of the saving.

```javascript
psiturk.saveData({
  success: function() {
    // function to run if the data is saved
  },
  error: function() {
    // function to run if there was an error
  }
});
```

### 3.1.10 `psiturk.completeHIT()`

This finishes the task by passing control of the experiment back to the Secure Ad Server `<secure_ad_server.html>`.

When in debug mode this just cleans up the task. When running live on the sandbox or live site this passes control of the browser back to the Ad Server so that the subject can be marked as complete and the user’s browser will correctly finish the HIT on Amazon’s site.

### 3.1.11 `psiturk.doInstructions(pages, callback)`

psiTurk includes a basic method for showing a sequence of instructions. You are always free to write your own instructions code (and may need to). However, this provides a basic template for a pretty simple typical type of instructions composed of a sequence of multiple pages of text and graphics along with a “next” and (optionally) “previous” button.

The `doInstructions()` method takes two arguments. The first is a list of HTML pages that you would like to display. These should appear in the order you would like them to be displayed to participants. The instructions method uses the `showPage()` method to display the HTML of the page.

Prior to calling `doInstructions()` all the instruction pages you plan to display should be preloaded using the `preloadPages()` method.

Within each HTML page there should be a button or other HTML element with class equal to `continue` which the user can click to move to the next screen.

An Bootstrap example is:
In addition, if the HTML document includes an element with class `previous` it will, when clicked, go to the previous page. As a result you should not include a previous button on the first HTML page.

An example previous button using Bootstrap is:

```html
<button type="button" id="next" value="next" class="btn btn-primary btn-lg previous">
  <span class="glyphicon glyphicon-arrow-left"></span> Previous
</button>
```

The final argument to the instructions object is the method to be called when the “continue” button on the last page of the instructions is called.

Example

```javascript
psiturk = new PsiTurk(uniqueId, adServerLoc);
var pages = [
  "instructions/instruct-1.html",
  "instructions/instruct-2.html",
  "instructions/instruct-3.html"];
psiturk.preloadPages(pages); // preload the pages
var instructionPages = [
  "instructions/instruct-1.html",
  "instructions/instruct-2.html",
  "instructions/instruct-3.html"]; // however, you can have as many as you like
psiturk.doInstructions(instructionPages,
  function() { currentview = new StroopExperiment(); });
```

The last line in this example uses an anonymous function to launch the Stroop Experiment.

### 3.1.12 `psiturk.finishInstructions()`

`finishInstructions` is used to change the participant’s status code to 2 in the database, indicating that they have begun the actual task.

In addition, this removes the `beforeunload` handler such that if people attempt to close (or reload) the page, they will get an alert asking them to confirm that they want to leave the experiment.

You do not have to use `doInstructions()` in order to call `finishInstructions()`. In the example above you would want to call `psiturk.finishInstructions()` in the `StroopExperiment()` class.

Example

```javascript
psiturk = new PsiTurk(uniqueId, adServerLoc);
...
psiturk.finishInstructions();
```
See the github repo README for how to cite usage of psiTurk.
CHAPTER 5

License

See the LICENSEFILE on github.