psiturk Documentation

Release 2.0

McDonnell, J.V., Martin, J.B., Markant, D.B., Coenen, A., Rich, A.S.

Sep 16, 2018

Contents

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 Getting psiTurk installed on your computer page. Afterwards, head over to our quickstart guide , or for a more detailed tutorial demonstrating how to setup a simple experiment with **psiTurk**, visit Getting up and running with the basic Stroop task.

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 What is 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 How do I host a psiTurk experiment?

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 you personal computer.

1.1.5 Do I have to learn how to code?

Yes. To run your experiment in a web browser you need to have at least some basic web programming skills (especially using HTML, CSS, and JavaScript).

Fortunately there exist many resources and tutorials that can help get started. If you are completely new to web programming, you might want to check out Codecademy, for example, for interactive tutorials on building websites.

Once you mastered the basics, you can take advantage of the vast number of libraries and tools that can help you to build sharp and sophisticated experiments with the support of a large community of users. For specific questions, visit stackoverflow.com.

To get you started, **psiTurk** provides a fully functioning example experiment (Getting up and running with the basic Stroop task) that you can use as a template for your own study. **psiTurk** also includes a library of basic Javascript functions (psiTurk API) that you can insert into your code to handle page transitions, load images, and record data.

1.2 Getting psiTurk Installed on Your Computer

psiTurk can be installed on any modern computer which supports Python (<= 2.7). However, currently **psiTurk** is *not* supported on Windows (*see below*). It works well on most unix variants including Mac OS X, BSD, and Linux. Installation is *usually* not difficult.

When **psiTurk** is successfully installed, you will simply 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.

1.2.1 Installation requirements

Installation of **psiTurk** requires:

1. **A python installation (<= v2.7).** We recommend the Enthought python distribution on Mac OS X.
2. **The ``pip`` package manager.** Directions on installing this are given below.
3. **Access to a command line tool.** (e.g., Terminal.app on Mac OS X)
4. **A web browser.** A WebKit compatible browser such as FireFox, Safari, or Chrome is recommended.

An additional requirement for actually using **psiTurk** to run experiments is an Internet connected computer capable of receiving incoming requests.

1.2.2 Installation steps

To install the package there are two options currently. First, the current stable release of **psiTurk** is hosted on the python package index pypi. As a result, it can easily be installed as a standard python package using the python package manager tool `pip`. Alternatively, you can install directly from the development branch on github. The following instructions describe the general process. In addition, system specific notes are provided below.

Install stable version via pypi

The easiest way to install **psiTurk** is via `pip`. Linux users will likely prefer to install pip as described *below*. Otherwise, If you don't already have `pip`, you can install it by typing the following in a terminal:

```
cd /tmp  # Just to put us in a directory that will be cleaned up periodically
curl -O https://raw.githubusercontent.com/pypa/pip/develop/contrib/get-pip.py
python get-pip.py  # If you get a permissions error, try typing sudo python get-pip.py
```

If you want a single system to run different versions of **psiTurk** (or other python packages) on a per-experiment basis, follow the Virtual Environment instructions *below*.

Once `pip` is installed, type into a terminal:

```
pip install psiturk
```

If this doesn't work, try

```
sudo pip install psiturk
```

If the install was successful you will have a new command `psiturk` available on your command line. You can check the location of this command by typing

```
which psiturk
```

Install directly from github

You can also install the bleeding-edge development version directly from github using `pip`. To install the latest stable branch follow the instructions above to install `pip` and:

```
sudo pip install git+git://github.com/NYUCCL/psiTurk.git@master
```

If the install was successful you will have a new command `psiturk` available on your command line. You can check the location of this command by typing

```
which psiturk
```

Updating from a previous version

To avoid compatibility issues, if you upgrade from a previous version it can be useful to first uninstall then reinstall **psiTurk** using the following sequence of commands:

```
$ pip uninstall psiturk
$ git clone git@github.com:NYUCCL/psiTurk.git
$ cd psiTurk
$ sudo python setup.py install
```

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
```

And then start a new shell session. 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 (if mkvirtualenv is not recognized follow the instructions here) :

```
$ 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.
```

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

```
$ 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. Use the *deactivate* command to leave the virtualenv.

1.2.3 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 all four of the requirements listed above 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 7 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:

```
aptitude 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:

```
aptitude 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. We currently recommend that Windows users try a cloud-based install such as openshift.

Cloud-based install (experimental)

If your local computer does not support **psiTurk** is it still possible to use the package by using a free hosting solution such as openshift. Begin by creating an account at http://openshift.redhat.com/ and download the command line tools at https://www.openshift.com/developers/rhc-client-tools-install

Create a python-2.7 application and add a PostgreSQL cartridge to the app

```
rhc app create psiturk python-2.7 postgresql-8.4 --from-code git://github.com/
↪jbmartin/psiturk-on-openshift.git
```

or you can do this to watch the build

```
rhc app create -a psiturk -t python-2.7
rhc cartridge add -a psiturk20 postgresql-8.4
```

Add this upstream psiturk repo

```
cd psiturk
git remote add upstream -m master https://github.com/jbmartin/psiturk-on-openshift.git
git pull -s recursive -X theirs upstream master
```

Then push the repo upstream

```
git push
```

That's it, you can now checkout your application at

```
http://psiturk-$YOURNAMESPACE.rhcloud.com
```

To access the your openshift hosted database run

```
rhc port forward -a psiturk
```

Connect to the database using your favorite SQL app, the PostgreSQL Local specs, and your credentials.

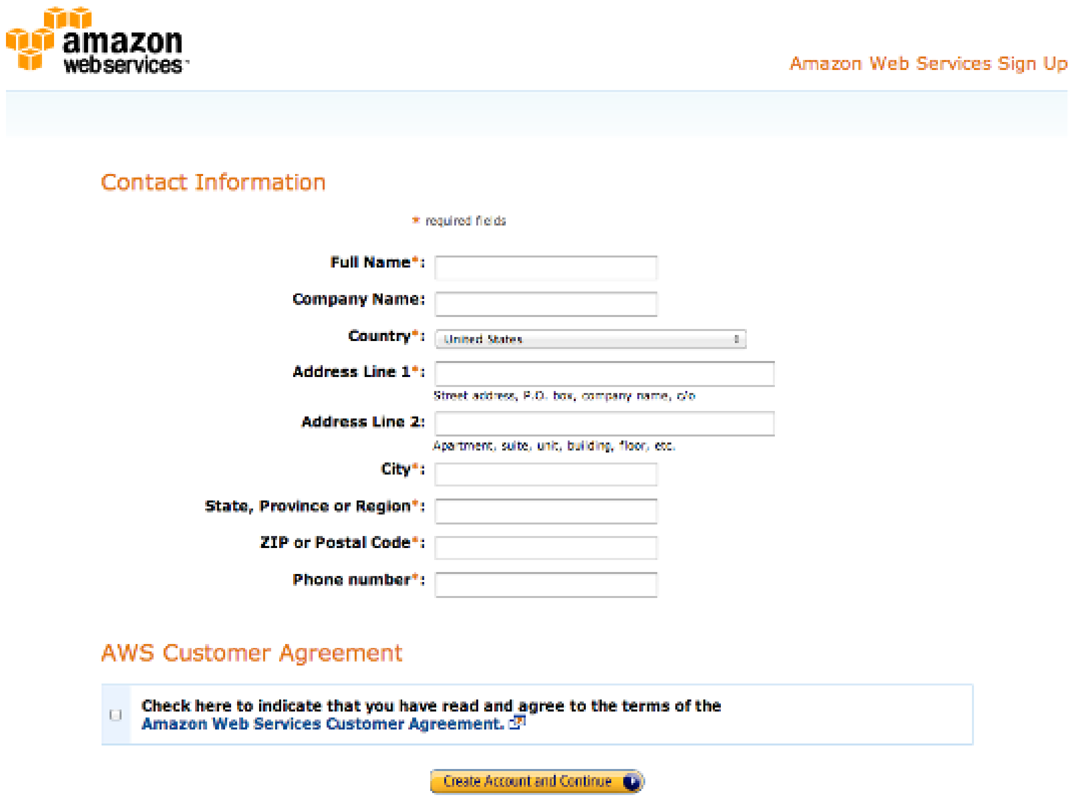
1.3 Getting setup with Amazon Mechanical Turk

psiTurk is a system for interfacing with Amazon Mechanical Turk. Thus, 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. Luckily they are a one-time process (possibly once for your entire lab if everyone shares a single AWS account).

1.3.1 Creating an AWS account

Start by going to the Amazon Web Services page here. If you made a Mechanical Turk account prior to this, sign in to your account and may skip to the next paragraph. Otherwise, click the Sign Up button at the top.

You should be redirected to a form asking for your contact information. Fill out the form and continue to the next section.



Amazon Web Services Sign Up

Contact Information

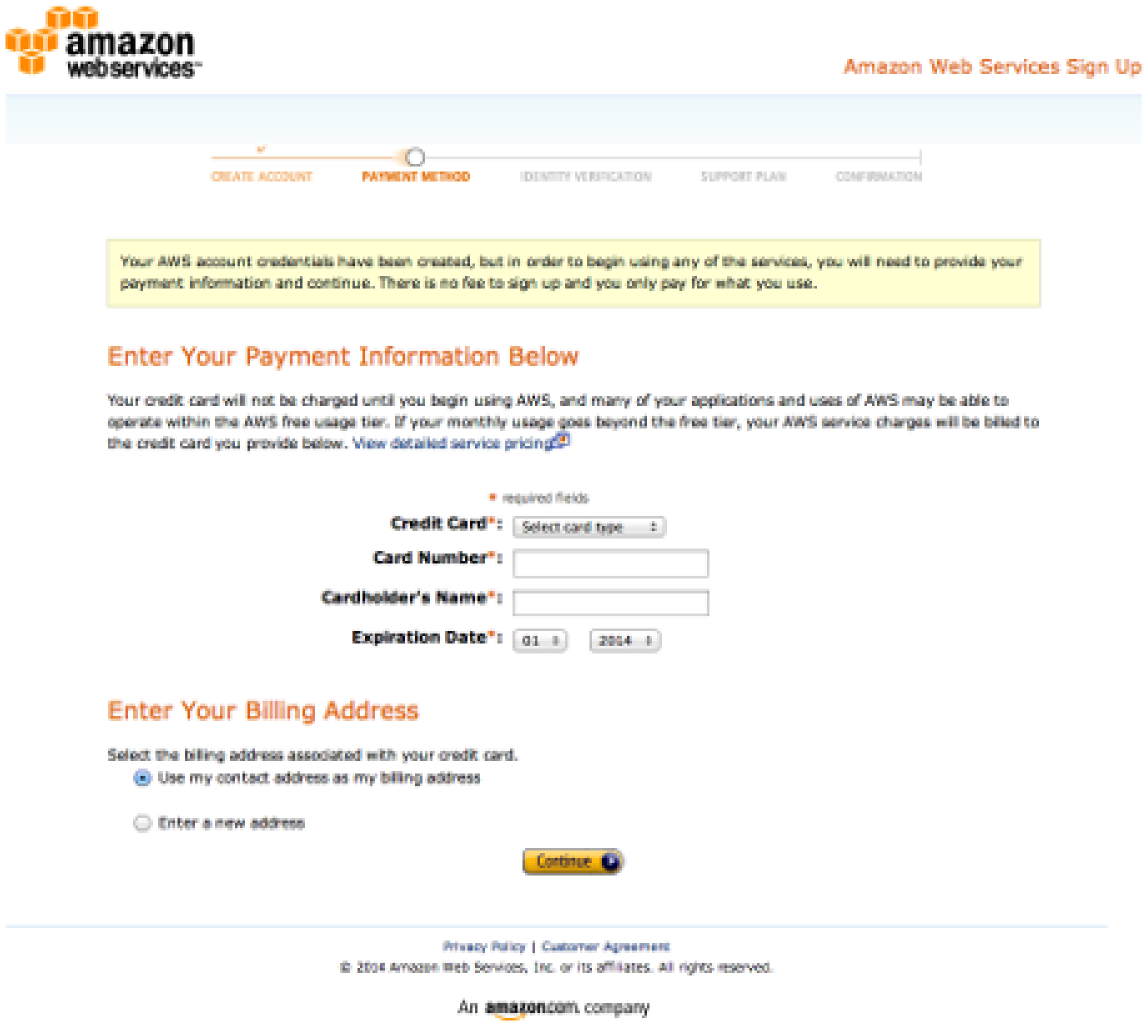
* required fields

Full Name*:	
Company Name:	
Country*:	United States
Address Line 1*:	Street address, P.O. box, company name, c/o
Address Line 2:	Apartment, suite, unit, building, floor, etc.
City*:	
State, Province or Region*:	
ZIP or Postal Code*:	
Phone number*:	

AWS Customer Agreement

Check here to indicate that you have read and agree to the terms of the Amazon Web Services Customer Agreement.

Next, you will need you credit card and your phone. The form should now ask for your credit card information.



Your AWS account credentials have been created, but in order to begin using any of the services, you will need to provide your payment information and continue. There is no fee to sign up and you only pay for what you use.

Enter Your Payment Information Below

Your credit card will not be charged until you begin using AWS, and many of your applications and uses of AWS may be able to operate within the AWS free usage tier. If your monthly usage goes beyond the free tier, your AWS service charges will be billed to the credit card you provide below. View detailed service pricing

* required fields

Credit Card*:	Select card type
Card Number*:	
Cardholder's Name*:	
Expiration Date*:	01 2014

Enter Your Billing Address

Select the billing address associated with your credit card.

[✓ Use my contact address as my billing address]

Enter a new address

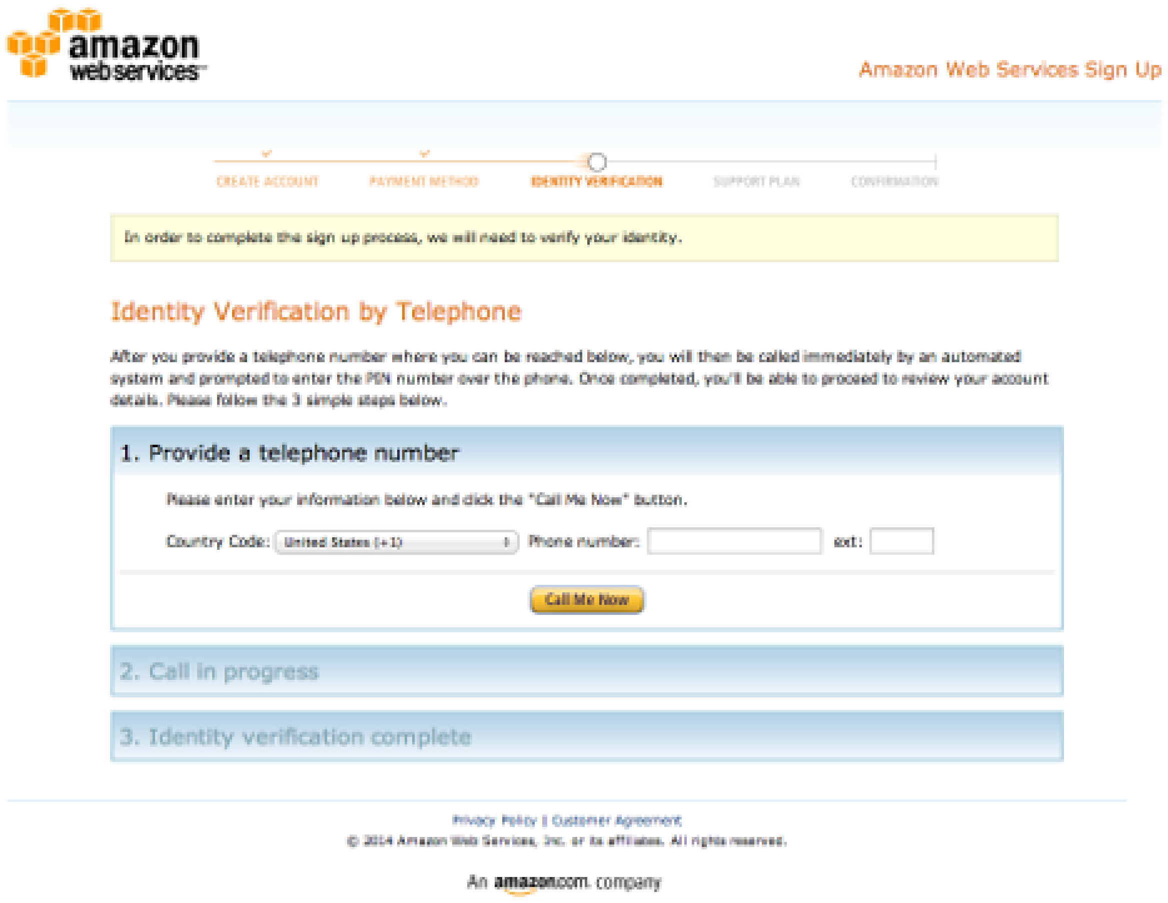
Privacy Policy | Customer Agreement

© 2014 Amazon Web Services, Inc. or its affiliates. All rights reserved.

An amazon.com company

If you do not see the forms to fill in your credit card information, go to the Payment Methods page either by clicking the link on the toolbar to the left or here. Enter in your credit card information. (Amazon will only charge you, if you use their cloud services. Signing up for an account should not incur any charges.)

On the next page, you will be asked to enter your phone number. Have your phone nearby. After you put in your phone number the webpage will display a 4-digit pin code and Amazon will call you. Enter the pin on your phone's keypad when prompted by the call.



Amazon Web Services Sign Up

In order to complete the sign up process, we will need to verify your identity.

Identity Verification by Telephone

After you provide a telephone number where you can be reached below, you will then be called immediately by an automated system and prompted to enter the PIN number over the phone. Once completed, you'll be able to proceed to review your account details. Please follow the 3 simple steps below.

1. Provide a telephone number

Please enter your information below and click the "Call Me Now" button.

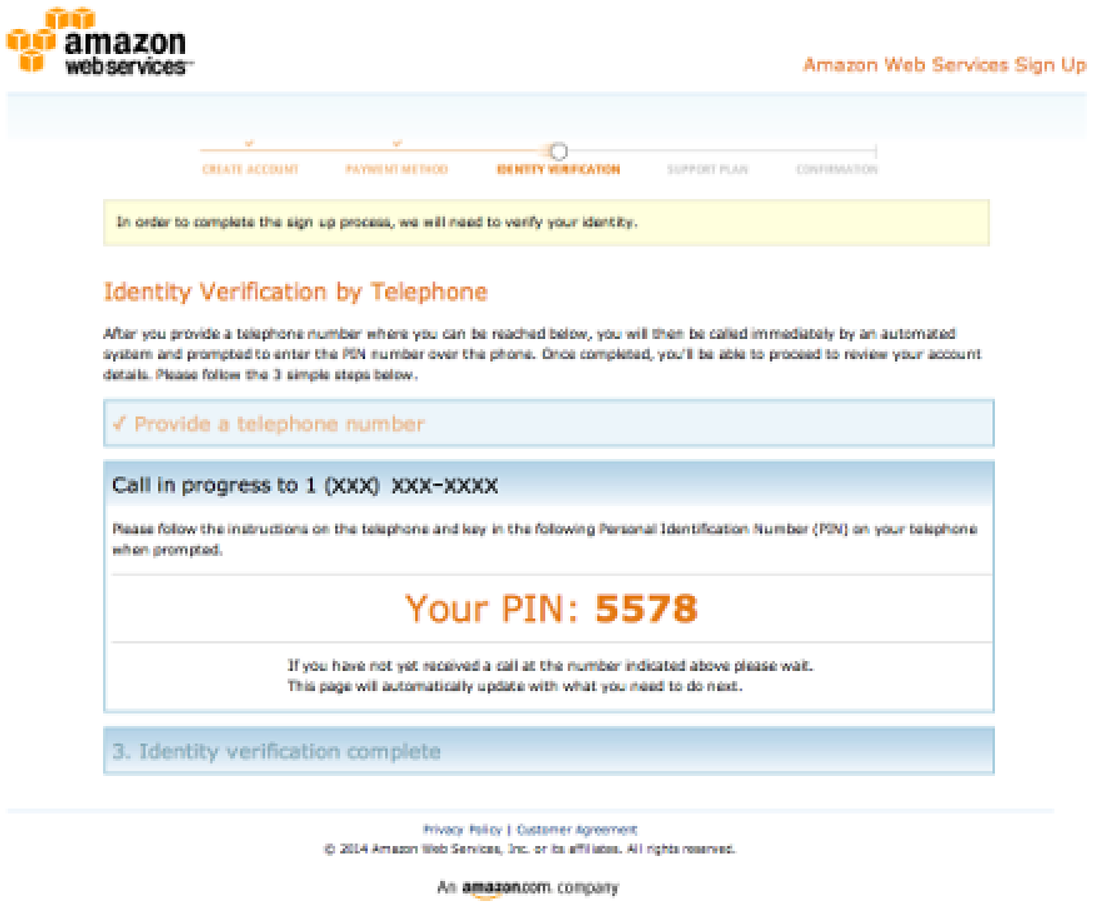
Country Code: United States (+1) Phone number: ext:

2. Call in progress

3. Identity verification complete

Privacy Policy | Customer Agreement
© 2014 Amazon Web Services, Inc. or its affiliates. All rights reserved.

An amazon.com. company



In order to complete the sign up process, we will need to verify your identity.

Identity Verification by Telephone

After you provide a telephone number where you can be reached below, you will then be called immediately by an automated system and prompted to enter the PIN number over the phone. Once completed, you'll be able to proceed to review your account details. Please follow the 3 simple steps below.

✓ Provide a telephone number

Call in progress to 1 (XXX) XXX-XXXX

Please follow the instructions on the telephone and key in the following Personal Identification Number (PIN) on your telephone when prompted.

Your PIN: 5578

If you have not yet received a call at the number indicated above please wait.
This page will automatically update with what you need to do next.

3. Identity verification complete

Privacy Policy | Customer Agreement
© 2014 Amazon Web Services, Inc. or its affiliates. All rights reserved.

An amazon.com company

Amazon will ask you to select a support plan. For the purposes of psiTurk, you only need the Basic(Free) plan. Click continue.

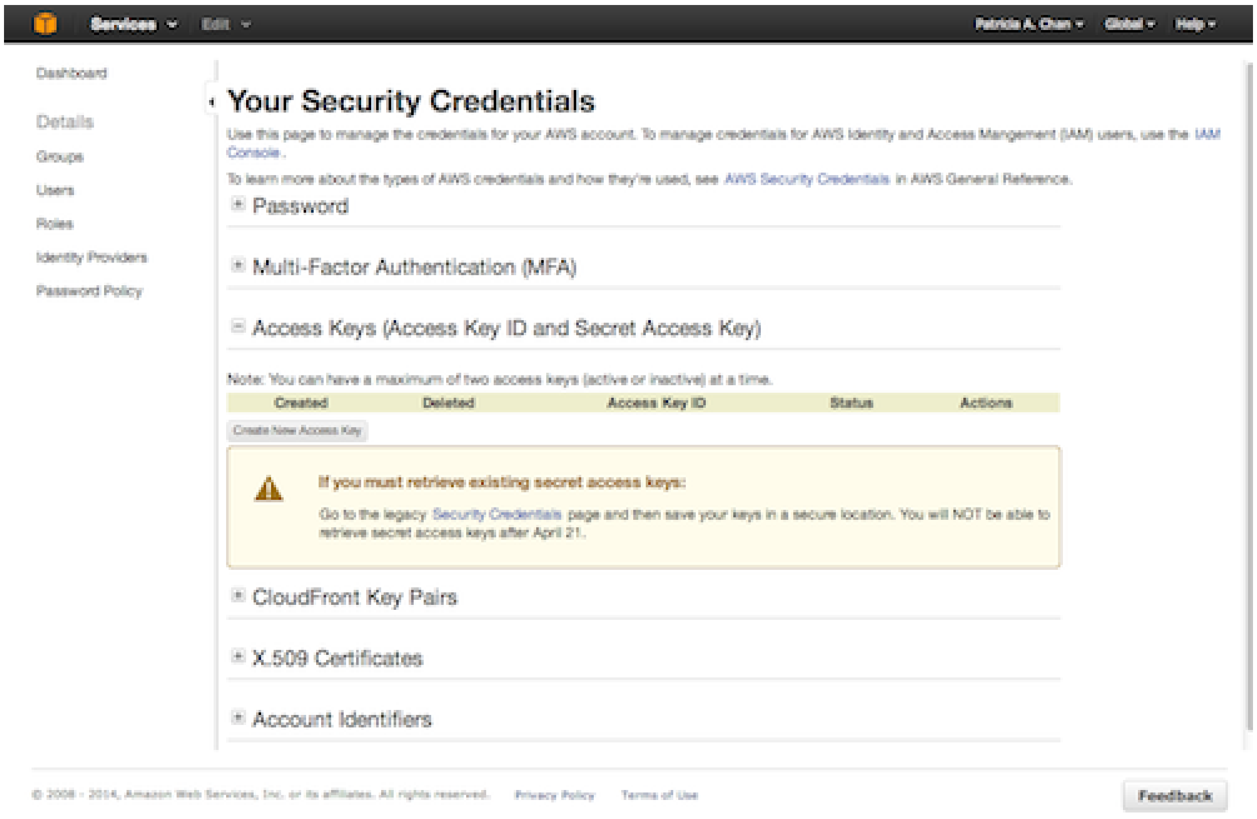
Your Amazon Web Service account should be set up now.

1.3.2 Obtaining AWS credentials

An AWS access key id and secret access key is required for posting new HITs to Mechanical Turk as well as monitoring existing HITs. If you created an AWS access key and did not save your secret access key, you will need to create a new access key. After April 21, 2014, AWS no longer allows users to retrieve their secret access key. Follow the steps below to create a new key.

You can create your keys after you open an Amazon Web Services account. Your keys can be generated in the AWS Management Console.

Click on the “Access Keys” tab. Your screen should look like this:



Your Security Credentials

Use this page to manage the credentials for your AWS account. To manage credentials for AWS Identity and Access Mangement (IAM) users, use the IAM Console.

To learn more about the types of AWS credentials and how they're used, see AWS Security Credentials in AWS General Reference.

Password

Multi-Factor Authentication (MFA)

Access Keys (Access Key ID and Secret Access Key)

Note: You can have a maximum of two access keys (active or inactive) at a time.

Created	Deleted	Access Key ID	Status	Actions

If you must retrieve existing secret access keys:

Go to the legacy Security Credentials page and then save your keys in a secure location. You will NOT be able to retrieve secret access keys after April 21.

CloudFront Key Pairs

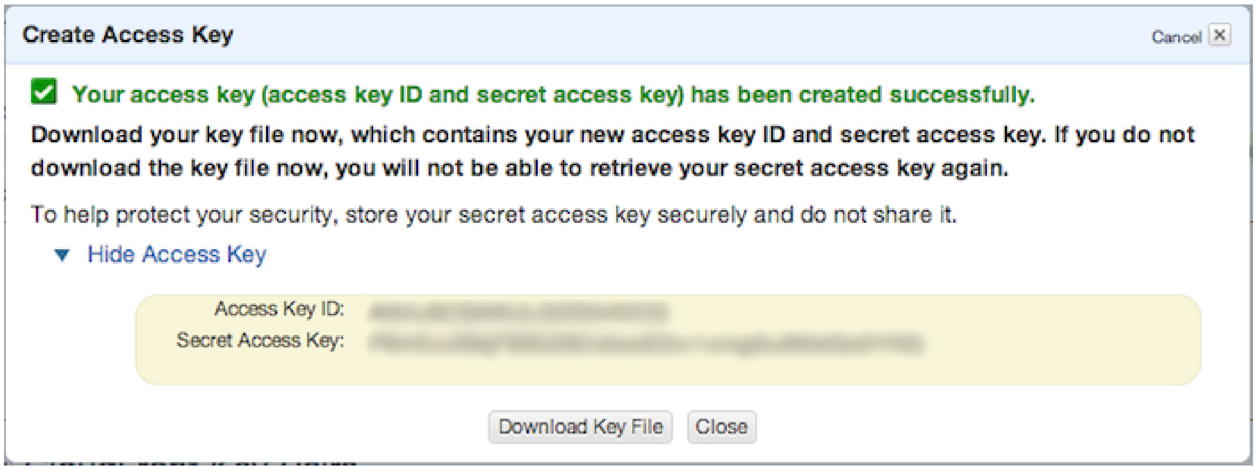
X.509 Certificates

Account Identifiers

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Press the "Create New Access Key" button to generate a set of access keys.

A popup window should appear on the screen to tell you that your access key has been created. Your access keys will appear in the popup box.



Create Access Key

Your access key (access key ID and secret access key) has been created successfully.

Download your key file now, which contains your new access key ID and secret access key. If you do not download the key file now, you will not be able to retrieve your secret access key again.

To help protect your security, store your secret access key securely and do not share it.

Access Key ID: [redacted]
Secret Access Key: [redacted]

If you do not see your access key, click the "Show Access Key" link in the popup box.

We recommend that you also download your access keys just in case. The "Download Key File" button will download the keys onto your computer in a CSV file.

The values of these keys need to be placed in your global `~/.psiturkconfig` file. The file is by default located in your home directory (see Configuration files for more info)

Note: If you are using IAM authentication, **psiTurk** requires that the *AmazonMechanicalTurkFullAccess* policy be added to the credentials it uses to connect to MTurk. See here for how to set up an IAM user.

1.3.3 Creating an AMT Requester account

To use your AWS keys to interface with Amazon Mechanical Turk, you need to create a requester account. Please see Amazon's instructions for this. In particular, it is necessary to at least once login to the requester site (http://requester.mturk.com) and also to at least once login to the sandbox requester site (https://requestersandbox.mturk.com), so that you can agree to the terms of service.

1.3.4 Linking funds

Under construction.

1.3.5 Additional instructions

Under construction.

1.4 Getting setup with psiturk.org

psiturk.org is a cloud-based system which provides users with information about their hits (who has accepted the hit, where they are located, etc. . .) and which provides a SSL-signed secure Ad server (ensuring that the majority of Workers can access your task). It is offered as a free service to anyone who uses **psiTurk**.

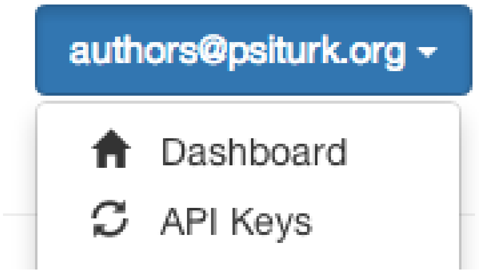
Note: To do anything beyond local testing a psiturk.org account is currently required.

1.4.1 Creating a psiturk.org account

The first step in using psiturk.org is to sign up with your email address. A free account can be created at https://psiturk.org/register.

1.4.2 Obtaining psiturk.org API credentials

To prevent your email and password from being passed repeatedly over the Internet when using psiturk.org, you access the psiturk.org API services using an API key (similar to how you interface with Amazon Mechanical Turk). To obtain your personal API keys login to psiturk.org (https://psiturk.org/login). On the main dashboard page, select the blue dropdown menu on the top right hand side of the page (shows your email address) and select "API Keys". Copy these keys into your `~/.psiturkconfig` file.



At any time you can regenerate these keys on the same page by pressing the “Regenerate API Keys” button.

At that point any old keys will no longer work, and you will need to update your `~/.psiturkconfig` file again. This way

1.5 psiturk.org Secure Ad Server

Participants recruited via Mechanical Turk first interact with your task via **ads**. Ads are simply the digital version of hanging a poster or flyer around your university building in order to recruit participants. Technically, **ads** are snippets of HTML code that describe what your task is about and what you’re offering for compensation. As a result, they are the front line for any subject recruitment online. It’s easy to overlook the importance of a good ad, and making that ad visible to as many participants as possible.

See also:

Getting setup with psiturk.org Use of the Secure Ad Server requires an account on psiturk.org.

1.5.1 Ads, Amazon Mechanical Turk, and the External HIT type

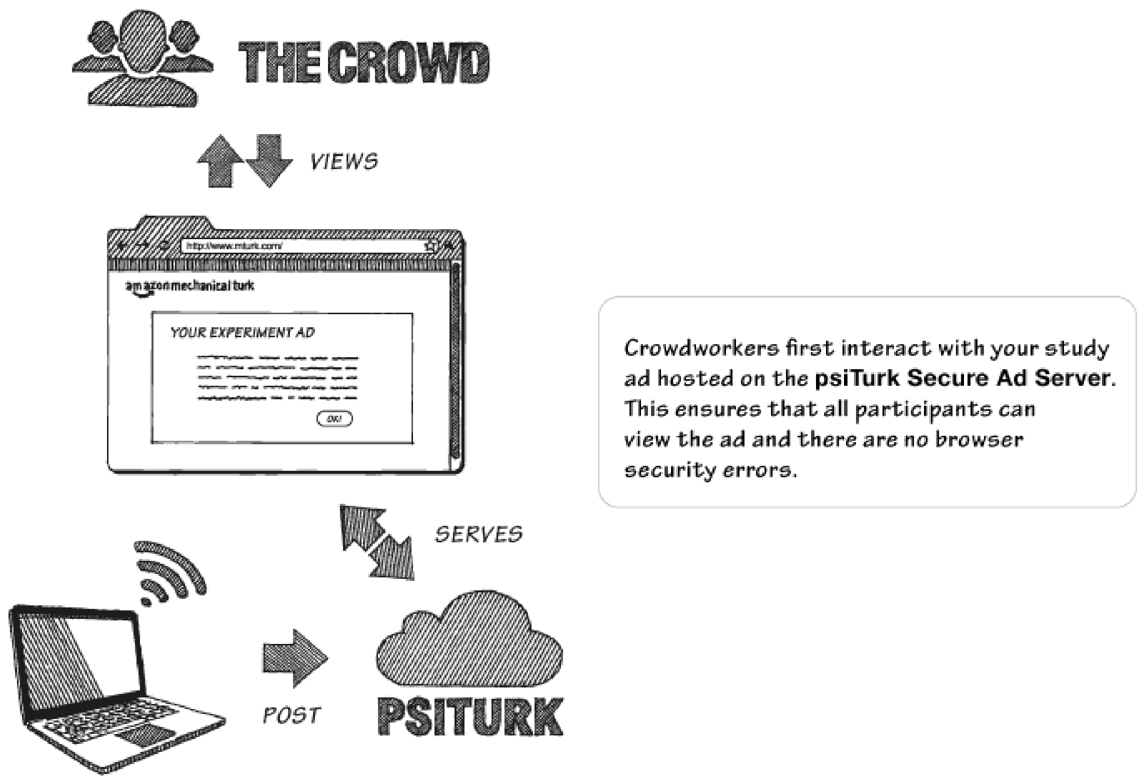
Any task (or HIT) which you deploy on your own server is listed using the “external HIT” type (a special name that Amazon uses for tasks which are hosted on external webservers). For these types of tasks, ads show up in users’ browsers as a HTML document. Due to recent changes in browser security, if your HTML is not encrypted and signed using an “official” SSL certificate (e.g., **https**://myschool.edu/myad.html works and the certificate signing authority is official) then the ad won’t display to potential participants at all!

There’s a good discussion of this issue here, here, and on Amazon’s own website.

This is **crazy**!

What’s worse is that many universities are not able to provide individuals with a signed SSL certificate. If that is the case, you can’t really use the external HIT mechanism without getting an account on some web hosting site.

However, the psiturk Secure Ad Server **solves this problem for all researchers**.


THE CROWD
VIEWS
http://www.mturk.com/
amazon mechanical turk
YOUR EXPERIMENT AD
OK!
Crowdworkers first interact with your study ad hosted on the psiTurk Secure Ad Server. This ensures that all participants can view the ad and there are no browser security errors.
SERVES
POST
PSITURK

Rather than getting your own signed certificate (a technically challenging process), when you use **psiTurk**, you can host your ad with us via https://ad.psiturk.org via a custom and unique URL made especially for you. We have already gone through the steps of getting an official, signed SSL certificate so you don't have to! **psiTurk** posts your custom ad text with us, and then participants access your task by first interacting with our secure server. We show them the ad, then forward them to you. No hassle, more potential participants!

A full "visual explanation" of the Secure Ad Server is provided here. Basically, you post the HTML of you "ad" to the psiturk.org cloud. Workers view the ad on the cloud server and decide if they want to accpet. If so they are forwarded to your local server or computer to complete the task.

1.5.2 Why use the psiturk.org Secure Ad Server?

As should be obvious, **psiTurk** already gets around a major technical hurdle for many scientists. However, the **psiTurk** Secure Ad Server not only serves up your SSL-signed Ad, but also provides you with some valuable data about people who view your HIT, people who accept it, and what other task they have completed on the **psiTurk** meta-platform. This can be very useful data. For example, when you use the **psiTurk** Ad server you can find out if your participants have done a version of your experiment before!

The public API for this data is coming soon, but just know that when you host your Amazon Mechanical Turk ads with us you are helping to build a valuable resource about which participants have done which types of experiments. This can be used to help filter your data or prevent certain participants from doing experiments for which they have already possibly been exposed to the important manipulation.

1.5.3 Sound great, how do I use it?

When you create a HIT from the command line in **psiTurk** your ad is posted to our servers. We begin forwarding people to your website instantly. You ad is never deleted (unless you want to delete it). Soon, you will be able to access statistics about who view, accepted, and returned your HIT and what other tasks they have completed on **psiTurk**. We also have plans to enable alternative ways of posting Ads to **psiTurk** including through a simple web interface. This would then allow researchers using survey-type (via Google Forms or Qualtrics) to take advantage of the features of the Secure Ad Server as well.

1.6 Sharing and replicating with the psiTurk.org Experiment Exchange

Under construction.

1.7 Using ssh tunnels

This is an experimental feature. Full documentation coming soon.

1.8 Quickstart

A simple quick start guide to running an existing **psiTurk** experiment is hosted here.

1.9 Configuration Files

There are two types of configuration files for **psiTurk**. Configuration files contain information needed to run an experiment as well as options which control how **psiTurk** behaves.

The first file is a "global" configuration file and resides in your home folder (*~/.psiturkconfig*). The second file is a "local" configuration file and resides in the folder of each experiment.

In general the "global" configuration file sets project-wide configuration options (i.e., those you want set the same for all the experiments or projects you are working on). The "local" configuration file contains the unique settings for individual experiments.

Note: In general, changes to either the local and global file 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 command shell.

1.9.1 Global configuration file

The global configuration file resides in your home folder in a "dot" file (*/.psiturkconfig*). This file is created automatically either the first time you run the *psiturk* command line tool or the first time you run *psiturk-setup-example*. The default file looks like this:

```
[AWS Access]
aws_access_key_id = YourAccessKeyId
aws_secret_access_key = YourSecretAccessKey
aws_region = us-east-1

[psiTurk Access]
```

(continues on next page)

(continued from previous page)

```
psiturk_access_key_id = YourAccessKeyId
psiturk_secret_access_id = YourSecretAccessKey
```

Other options can be added if you would like those to be global to all your projects. The default options include your access credentials/API keys for Amazon Web Services (and Mechanical Turk) as well as psiturk.org. You can learn how to obtain proper values for these settings by following those links.

You can customize the location of this file to something other than the ~ folder by setting the *PSITURK_GLOBAL_CONFIG_LOCATION* in your shell environment.

1.9.2 Local configuration file

The local configuration file is specific to each project and resides in a file called *config.txt* in the top level of the project. Here is what *config.txt* looks like for the default **psiTurk** stroop project:

```
[HIT Configuration]
title = Stroop task
description = Judge the color of a series of words.
amt_keywords = Perception, Psychology
lifetime = 24
us_only = true
approve_requirement = 95
number_hits_approved = 0
require_master_workers = false
contact_email_on_error = youremail@gmail.com
ad_group = Default psiTurk Stroop Example
psiturk_keywords = stroop
organization_name = New Great University
browser_exclude_rule = MSIE, mobile, tablet
allow_repeats = false

[Database Parameters]
database_url = sqlite:///participants.db
table_name = turkdemo

[Server Parameters]
host = localhost
port = 22362
cutoff_time = 30
logfile = server.log
loglevel = 2
debug = true
login_username = examplename
login_pw = examplepassword
threads = auto
secret_key = 'this is my secret key which is hard to guess, i should change this'
#certfile = <path_to.crt>
#keyfile = <path_to.key>
#adserver_revproxy_host = www.location.of.your.revproxy.sans.protocol.com
#adserver_revproxy_port = 80 # defaults to 80
#server_timeout = 30

[Task Parameters]
experiment_code_version = 1.0
num_conds = 1
```

(continues on next page)

(continued from previous page)

```
num_counters = 1

[Shell Parameters]
launch_in_sandbox_mode = true
#bonus_message = "Thanks for participating!"
use_psiturk_ad_server = true
ad_location = false
```

This file is divided into a few sections which are described in detail. Each field is described by name and includes in brackets the type of data it expects.

Note: 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. Likewise, if you wanted to have the same *organization_name* in all your experiments, you could add a `[HIT Configuration]` section with an *organization_name* field to your *~/.psiturkconfig* file. Keep in mind that **settings in the local 'config.txt' file always override settings in the global '~/.psiturkconfig' file**.

HIT Configuration

The HIT Configuration section contains details about your Human Intelligence Task. An example looks like this:

```
[HIT Configuration]
title = Stroop task
description = Judge the color of a series of words.
amt_keywords = Perception, Psychology
lifetime = 24
us_only = true
approve_requirement = 95
number_hits_approved = 0
require_master_workers = false
contact_email_on_error = youremail@gmail.com
ad_group = My research project
psiturk_keywords = stroop
organization_name = New Great University
browser_exclude_rule = MSIE, mobile, tablet
allow_repeats = false
```

title [string]

The *title* is the title of the task that will appear on the AMT worker site. Workers often use these fields to search for tasks. Thus making them descriptive and informative is helpful.

description [string]

The *description* is the accompanying text that appears on the AMT site. Workers often use these fields to search for tasks. Thus making them descriptive and informative is helpful.

keywords [comma separated string]

keywords Workers often use these fields to search for tasks. Thus making them descriptive and informative is helpful.

lifetime [integer]

The *lifetime* is how long your HIT remains visible to workers (in hours). 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 [true | false]

us_only controls if you want this HIT only to be available to US Workers. This is not a failsafe restriction but works fairly well in practice.

approve_requirement [integer]

approve_requirement 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 [integer]

number_hits_approved is important to use in conjunction with *approved_requirement*, because mturk will default *approve_requirement* to 100% until a worker has at least 100 HITs approved. Override that behavior by setting *number_hits_approved* to something like 100.

require_master_workers [true | false]

require_master_workers will make it so that only workers with the “Master” qualification can take your study. See Who Are Amazon Mechanical Turk Masters?

Note: Master workers cost an extra 5%.

See also:

The following options help configure the psiturk.org Secure Ad Server.

Getting setup with psiturk.org How to get an account on psiturk.org.

psiturk.org Secure Ad Server An overview of the purpose and features of the Secure Ad Server.

contact_email_on_error [string - valid email address]

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

ad_group [string]

ad_group is a unique string that describes your experiment. All HITs and Ads with the same ad_group string will be grouped together in your psiturk.org dashboard. To create a new group in your dashboard simply create a new unique string. The best practice is to group all experiments from the same "project" with the same *ad_group* but assign different *ad_group* identifiers to different project (e.g., if two students in a lab were working on different things but shared a psiturk.org account then they might use different *ad_group* identifiers to keep things organized.)

psiturk_keywords [comma separated string]

psiturk_keywords [string, comma separated] are a list of key words that describe your task. The purpose of these keywords (distinct from the *keywords* described above) is to help other researchers know what your task involves. For example, you might include the keyword *deception* if your experiment involves deception. If it involves a common behavioral task like "trolly problems" you might include that as well. In the future we hope to allow researchers to query information about particular workers and task to find out if your participants are naive to particular types of manipulations. You should be careful not to include too general of terms here. For example, a researcher might want to exclude people who in the past had participated in a psychology study involving deception. They probably don't care to exclude people who did a "decision making task". Thus, being specific and using important keywords that are likely to be recognized by the research community is the best approach. (Ask yourself, if I wanted to exclude people who had done this study from a future study what keywords would I search for.)

organization_name [string]

organization_name [string] is just an identifier of your academic institution, business, or organization. It is used internally by psiturk.org.

browser_exclude_rule [comma separated string]

browser_exclude_rule is a set of rules you can apply to exclude particular web browsers from performing your task. When a users contact the Secure Ad 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* denys 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 [boolean]

allow_repeats specifies whether participants may complete the experiment more than once. 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*.

Database Parameters

The Database Parameter section contains details about your database. An example looks like this:

```
[Database Parameters]
database_url = sqlite:///participants.db
table_name = turkdemo
```

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 [url string]

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

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

```
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:

```
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 [string]

table_name specifies the table of the database you would like to write to. **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 not 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.

Server Parameters

The Server Parameter section contains details about your local web server process that you launch from the command line. An example looks like this:

```
[Server Parameters]
host = 0.0.0.0
port = 22362
cutoff_time = 30
logfile = server.log
loglevel = 2
debug = true
login_username = examplename
login_pw = examplepassword
threads = auto
#certfile = <path_to.crt>
#keyfile = <path_to.key>
#adserver_revproxy_host = www.location.of.your.revproxy.sans.protocol.com
#adserver_revproxy_port = 80
#server_timeout = 30
```

host [string]

host specifies the hostname of your server. There are really only two meaningful values of this. If host is set to 'localhost' or '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 *host* is set to *0.0.0.0* or the actual ip address or hostname of your current computer then your task will be available to the general internet.

port [integer]

This is the port that your server will run on. Typically a number greater than 5000 will work. If another process is already using a given port you will usually get an error message.

cutoff_time [integer]

Maximum time in minutes to finish the task. The connection will be closed after this time is up.

logfile [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 [integer]

Sets how "verbose" the log messages are. See the python logging library.

debug [true | false]

If debug is true, if there is an internal server error helpful debugging information will be printed into the webpage of people taking the experiment. **IMPORANT** this should be set to false for live experiments to prevent possible security holes.

login_username [string]

If you want to have 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 [string]

If you want to have 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 [auto | integer]

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

certfile [string]

Warning: SSL support for the psiturk server is an experimental feature.

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. See this gist for an example. **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 http://docs.gunicorn.org/en/stable/deploy.html for more information on setting up proxy servers with the psiturk (gunicorn) server.

See also:

use_psiturk_ad_server How to use your own ad_location. Does not require that the **psiTurk** server be SSL-enabled. (Although you will still need your own SSL certificate and key)

keyfile [string]

Warning: SSL support for the psiturk server is an experimental feature.

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

See the documentation for *certfile* for more information.

adserver_revproxy_host [string]

Normally when you create an ad on the psiturk ad server (*hit create…*), your external ip address is fetched and combined with the *port* that your psiturk gunicorn server is running on (the same port set in your config.txt). The psiTurk ad server directs all traffic directly to the psiturk gunicorn server.

If you want to put a reverse proxy in front of the psiturk gunicorn server (such as apache or nginx), set the hostname or ip address of the reverse proxy here. Set it even if it's the same as your external ip. Leave the protocol off (i.e., don't add *http://* to the front). (The psiturk ad server will add *http://* to the front of whatever you set here.)

If your reverse proxy port is different from 80, set it in *adserver_revproxy_port*.

Note: If you want to host your own ad, see the documentation for *use_psiturk_ad_server* and *ad_location*. The *adserver_revproxy_host* and *adserver_revproxy_port* settings are only used if you are using the psiTurk ad server.

See also:

- use_psiturk_ad_server
- ad_location

adserver_revproxy_port [integer]

Defaults to port 80 (the standard http port).

See the documentation for *adserver_revproxy_port* for more information.

Note: If you are hosting your experiment on *rhcloud.com*, this setting is ignored and 80 will always be used.

server_timeout [integer]

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.

Defaults to 30 seconds.

Note: See http://docs.gunicorn.org/en/stable/settings.html#timeout for more information.

Task Parameters

The Task Parameters section contains details about your task. An example looks like this:

```
[Task Parameters]
experiment_code_version = 1.0
num_conds = 1
num_counters = 1
```

experiment_code_version [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). *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 [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 [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.

Shell Parameters

The Shell Parameters section contains details about the psiturk shell.

```
[Shell Parameters]
launch_in_sandbox_mode = true
bonus_message = "Thanks for participating!"
use_psiturk_ad_server = true
ad_location = false
```

launch_in_sandbox_mode [true | false]

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.

See also:

Overview of the command-line interface The basic features of the **psiTurk** command line.

bonus_message [string]

If set to a string, 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.)

use_psiturk_ad_server [true | false]

Warning: Non-use of the psiturk ad server is an experimental feature.

If set to *true*, then the **psiTurk** secure ad server functionality will be enabled, and your ad will be hosted on psiturk.org when creating hits on AMT.

If you want to host your own ad, then set this to *false*. You are responsible for obtaining your own cert and key and for configuring your own proxy server in front of psiturk/gunicorn. It is not necessary to also include the cert and key in the [Server Parameters] section – you can have a proxy server such as nginx in front of psiturk/gunicorn which handles SSL connections. Although if you don't have your SSL certs in both places, then traffic between your proxy server and psiturk/gunicorn will not be encrypted. Perhaps that doesn't matter to you though if you configure your proxy server to pass traffic to your gunicorn/psiturk server via localhost.

If set to *false* then you must also specify your custom *ad_location* (see below).

See also:

See the [Server Parameters] certfile and keyfile configs for ssl-enabling the psiturk server (although this is not required to use your own ad location).

See also:

See this gist for an example nginx psiturk SSL configuration

ad_location [false | string]

Warning: Non-use of the psiturk ad server is an experimental feature.

ad_location is only used if *use_psiturk_ad_server* is *false*. Set to whatever you set up your proxy server to listen on. This will be sent directly to AMT when creating your HITs to tell AMT where to look for your ad.

Format is as follows:

```
https://<host>:<port>/ad
```

Some gotcha's:

- don't forget the */ad* at the end. And don't append a trailing backslash.
- you must use *https://* or AMT will explode.

- the *<port>* should be the port your *proxy server* (such as nginx) is running on, *not* the **psiturk** port. See the gist for a full example.

See also:

See the information for the *use_psiturk_ad_server* configuration above as well.

1.10 Command-line Interface

The **psiTurk shell** is a simple, interactive command line interface which allows users to communicate with Amazon Mechanical Turk, psiturk.org, and their own experiment servers.

1.10.1 Starting the psiTurk shell

Usage

The psiTurk shell can be launched from any psiTurk project folder (i.e., any folder with a `config.txt` file) by entering the command

```
psiturk
```

in the terminal.

Options

```
-v, --version
```

Print the currently installed version of psiTurk and exit.

```
-c, --cabinmode
```

Launch psiturk in cabin (offline) mode. This allows you to develop test experiments locally without an internet connection. Cabin mode offers only limited functionality, and lacks the `amt`, `db`, `hit`, `mode`, and `worker` commands.

```
-s, --script <filename>
```

Run a list of commands from a text file, then exit. Each line in the file is treated as a command.

1.10.2 The psiTurk shell prompt

The psiTurk shell prompt looks something like this:

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

and contains several pieces of useful information.

Server field

The `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

The `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

The `#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. The `#HITs` field is not displayed in cabin mode.

1.10.3 `amt_balance` command

Usage

```
amt_balance
```

The `amt_balance` command displays your current AMT balance, or your worker sandbox balance (always $10,000.00) if you are in sandbox mode.

Example

Checking your balance in sandbox mode:

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

1.10.4 `config` command + subcommands

Contents

Description

The `config` command is used with a variety of subcommands to control the current configuration context

`config print`

Prints the current configuration context (both local and global config options).

See also:

Configuration files More info about the global and local configuration files.

Example

```
[psiTurk server:off mode:sdbx #HITs:0]$ config print
     [AWS Access]
     aws_region=us-east-1
     aws_access_key_id=XXXXXX
     aws_secret_access_key=XXXX
     ...
     [Shell Parameters]
     launch_in_sandbox_mode=true
[psiTurk server:on mode:sdbx #HITs:0]$
```

`config reload`

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

Example

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

`config help`

Display a help message concerning the config subcommand.

1.10.5 `db` command + subcommands

Contents

The `db` command is used with a number of subcommands to create and configure database instances. More information about database configuration can be found on the Configuring Databases page.

Note: The `aws_` subcommands are used to interact with the Amazon Web Services Relational Database Server (RDS) cloud service.

`db get_config`

Usage

```
db get_config
```

Display the current setting of the database (database_url).

Example

```
[psiTurk server:off mode:sdbx #HITs:1]$ db get_config
Current database setting (database_url):
    sqlite:///participants.db
```

`db use_local_file`

Usage

```
db use_local_file [<filename>]
```

Switch the current database to a local SQLite file with name <*filename*> (default is *participants.db*), or enter without filename and provide name when prompted.

Example

Setting database to a local SQLite file:

```
[psiTurk server:off mode:sdbx #HITs:1]$ db use_local_file
Enter the filename of the local SQLLite database you would like to use␣
↪[default=participants.db]: example.db
Updated database setting (database_url):
     sqlite:///example.db
[psiTurk server:off mode:sdbx #HITs:1]$
```

`db use_aws_instance`

Usage

```
db use_aws_instance [<instance_id>]
```

Switch the current database to a given instance <*instance_id*> on AWS RDS. Enter without an argument to display a list of instances from which to choose.

Example

Using an RDS database instance:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db use_aws_instance mydb
Switching your DB settings to use this instance.  Are you sure you want to do this? y
enter the master password for this instance: PasswordXXXXX
AWS RDS database instance mydb selected.
Here are the available database tables
        myexp
Enter the name of the database you want to use or a new name to create a new one:␣
↪myexp
Successfully set your current database (database_url) to
        mysql://UsernameXXXXX:PasswordXXXXX@mydb.cdukgn44bkrv.us-east-1.rds.amazonaws.
↪com:3306/myexp
```

`db aws_list_regions`

Usage

```
db aws_list_regions
```

Lists available AWS regions.

Example

```
psiTurk server:off mode:sdbx #HITs:1]$ db aws_list_regions
Avaliable AWS regions:
     us-east-1 (currently selected)
     us-gov-west-1
     eu-west-1
     us-west-1
     us-west-2
     sa-east-1
     ap-northeast-1
     ap-southeast-1
     ap-southeast-2
```

`db aws_get_region`

Usage

```
db aws_get_region
```

Displays the current AWS region you are communicating with.

Example

```
[psiTurk server:off mode:sdbx #HITs:1]$ db aws_get_region
us-east-1
```

db aws_set_region

Usage

```
db aws_set_region [<region_name>]
```

Sets the AWS region you are currently using to <*region-name*>. Enter without an argument to display a list of regions from which to choose.

Example

Setting region to *us-west-1*:

```
[psiTurk server:off mode:sdbx #HITs:1]$ db aws_set_region us-west-1
Region updated to  us-west-1
```

db aws_list_instances

Usage

```
db aws_list_instances
```

List instances and statuses in the current region/AWS account.

Example

1. Listing instances when there are none active in your region:

```
[psiTurk server:off mode:sdbx #HITs:1]$ db aws_list_instances
There are no DB instances associated with your AWS account in region  us-east-1
```

2. Listing instances when there is an active instance in your region:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_list_instances
Here are the current DB instances associated with your AWS account in region  us-
↪east-1
       --------------------
       Instance ID: mydb
       Status: available
```

db aws_create_instance

Usage

```
db aws_create_instance [<instance_id> <size> <username> <password>
<dbname>]
```

Create an RDS instance using MySQL on the AWS cloud, with the given instance id, size, username, password, and database name. `db aws_create_instance` can also be run interactively by running the command without parameters.

Example

Interactively creating a database instance:

```
[psiTurk server:off mode:sdbx #HITs:1]$ db aws_create_instance
*************************************************
Ok, here are the rules on creating instances:

instance id:
  Each instance needs an identifier.  This is the name
  of the virtual machine created for you on AWS.
  Rules are 1-63 alphanumeric characters, first must
  be a letter, must be unique to this AWS account.

size:
  The maximum size of you database in GB.  Enter an
  integer between 5-1024

master username:
  The username you will use to connect.  Rules are
  1-16 alphanumeric characters, first must be a letter,
  cannot be a reserved MySQL word/phrase

master password:
  Rules are 8-41 alphanumeric characters

database name:
  The name for the first database on this instance.  Rules are
  1-64 alphanumeric characters, cannot be a reserved MySQL word
*************************************************

enter an identifier for the instance (see rules above): mydb
size of db in GB (5-1024): 5
master username (see rules above): UsernameXXXXX
master password (see rules above): PasswordXXXXX
name for first database on this instance (see rules): myexp
*****************************
  Creating AWS RDS MySQL Instance
    id:  mydb
    size:  5  GB
    username:  UsernameXXXXX
    password:  PasswordXXXXX
    dbname:  myexp
    type: MySQL/db.t1.micro
    ________________________
Be sure to store this information in a safe place.
Please wait 5-10 minutes while your database is created in the cloud.
You can run 'db aws_list_instances' to verify it was created (status
will say 'available' when it is ready
[psiTurk server:off mode:sdbx #HITs:1]$
```

`db aws_delete_instance`

Usage

```
db aws_delete_instance [<instance_id>]
```

Delete the RDS instance with id *<instance_id>*. Enter without an argument to display a list of instances from which to choose.

Example

Deleting an AWS database instance:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_delete_instance
Here are the available instances you can delete:
            mydb ( available )
Enter the instance identity you would like to delete: mydb
Deleting an instance will erase all your data associated with the
database in that instance. Really quit? y or n: y
DBInstance:mydb
AWS RDS database instance mydb deleted.  Run `db aws_list_instances` for current␣
↪status.
```

1.10.6 `debug` command

Usage

```
debug [options]
```

`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.

Options

```
-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:

```
[psiTurk server:on mode:sdbx #HITs:0]$ debug -p
Here's your randomized debug link, feel free to request another:
http://localhost:22362/ad?assignmentId=debugDKSAAE&hitId=debug2YW8RI&
↪workerId=debugM1QUH4
[psiTurk server:on mode:sdbx #HITs:0]$
```

1.10.7 `download_datafiles` command

Usage

```
download_datafiles
```

The `download_datafiles` command 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

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

column 1	column 2	column 3	column 4	column 5
unique user ID	event type	interval	value	time

questiondata.csv

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

column 1	column 2	column 3
unique user ID	question name	response

trialdata.csv

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

column 1	column 2	column 3	column 4
unique user ID	trial #	time	trial data

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

1.10.8 `help` command

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:

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

psiTurk command help: ===================== amt_balance debug mode server tunnel 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 l 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:

```
[psiTurk 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`.

1.10.9 `hit` command + subcommands

Contents

Description

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

`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

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: $6.60
  Ad for this HIT now hosted at: https://ad.psiturk.org/view/Q3HWnfqzg3MP9VDbu3kFyn?
↪assignmentId=debugJCI80S&hitId=debug9AWC90
[psiTurk server:on mode:sdbx #HITs:1]$
```

`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
      max:13/pending:0/complete:0/remain:13
      Created:2014-03-07T21:36:33Z
      Expires:2014-03-08T21:48:33Z
```

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.

Example

1. Expiring two HITs at once:

```
[psiTurk server:on mode:sdbx #HITs:4]$ hit expire 2Y0T3HVWAVKIMG42A2S75Z9943NNFG␣
↪2RVZXR24SMEZFG314ME9X8P9CPPH0X
expiring sandbox HIT 2Y0T3HVWAVKIMG42A2S75Z9943NNFG
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 2776AUC26DG6NRIGNVRFN0COYO0B4R
expiring sandbox HIT 2VUWA6X3YOCCVET8PKOPWINIWJFPO0
[psiTurk server:on mode:sdbx #HITs:0]$
```

`hit dispose`

Usage

```
hit dispose (--all | <HITid>)
```

Dispose of one ore more HITs, or dispose of all HITs using the `--all` flag.

Note: To dispose of a HIT, it must not be active or have any unreviewed assignments

Example

```
[psiTurk server:off mode:sdbx #HITs:0]$ hit dispose 241KM05BMJTUXCDL0TG9UA7SJI3JEQ
deleting sandbox HIT 241KM05BMJTUXCDL0TG9UA7SJI3JEQ
[psiTurk server:off mode:sdbx #HITs:0]$
```

`hit list`

Usage

```
hit list [--active | --reviewable]
```

List all HITs, or list all active or reviewable HITs using the provided flags.

Examples

1. List all active HITs:

```
[psiTurk server:on mode:sdbx #HITs:1]$ hit list --active
Stroop task
   Status: Assignable
   HITid: 2ZFKO2L92HECCONGNYFCFF0C3R2FG1
   max:1/pending:0/complete:0/remain:1
   Created:2014-03-07T22:10:01Z
   Expires:2014-03-08T22:10:01Z

[psiTurk server:on mode:sdbx #HITs:1]$
```

2. List all HITs:

```
[psiTurk server:on mode:sdbx #HITs:1]$ hit list
Face Discrimination (5 - 10 minutes, up to $1.0 bonus!!)
   Status: Reviewable
   HITid: 2ZRNZW6HEZ6OUI7FRTZ6CGUMGIQPZ0
   max:1/pending:0/complete:0/remain:0
   Created:2014-03-03T23:53:08Z
   Expires:2014-03-04T23:53:08Z

Stroop task
```

(continues on next page)

(continued from previous page)

```
    Status: Assignable
    HITid: 2ZFKO2L92HECCONGNYFCFF0C3R2FG1
    max:1/pending:0/complete:0/remain:1
    Created:2014-03-07T22:10:01Z
    Expires:2014-03-08T22:10:01Z

[psiTurk server:on mode:sdbx #HITs:1]$
```

1.10.10 `psiturk_status` command

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]$
```

1.10.11 `quit` command

Usage

```
quit
```

The `quit` command 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

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.
$
```

1.10.12 `server` command + subcommands

Contents

Description

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

`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.

1.10.13 `status` command

Usage

```
status
```

The `status` command updates and displays the server status and number of HITs available on AMT or in the worker sandbox.

Note: This information is also displayed in the psiTurk shell prompt, but *#HITs* is not updated after every command (as every update requires contacting the AMT server). `status` provides a way to make sure the prompt is up-to-date.

Example

Using the `status` command in sandbox mode:

```
[psiTurk server:off mode:sdbx #HITs:1]$ status
Server: currently offline
AMT worker site - sandbox: 1 HITs available
```

1.10.14 `mode` command

Usage

```
mode
mode <which>
```

The `mode` command controls the current mode of the psiTurk shell. Type `mode live` or `mode sandbox` to switch to either mode, or simply `mode` to switch to the opposite mode. The current mode affects almost every psiturk shell command. For example, running `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 `worker list all` or `hit list all` will list assignments and HITs from either the live site or the sandbox, depending on your mode.

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 `<which>` specifier:

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

2. Switching mode with the server running:

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

Type `n` instead to abort the mode switch harmlessly.

1.10.15 `worker` command + subcommands

Contents

Description

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

`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.

Example

1. Approve a single assignment:

```
[psiTurk server:on mode:sdbx #HITs:0]$ worker approve␣
↪21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
approved 21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
```

2. Approve all assignments for a given hit:

```
[psiTurk server:on mode:sdbx #HITs:0]$ worker approve --hit␣
↪2QKHECWA6X3Y4QTYKCG5NXPTWYGMLF
approving workers for HIT 2QKHECWA6X3Y4QTYKCG5NXPTWYGMLF
approved 2MB011K274J7PY7FQ1ZN76UXH0ECED
approved 2UO4ZMAZHHRR1T7J8NEVUH1KJCAKBY
```

`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

Reject a single assignment:

```
[psiTurk 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

Unreject a single assignment:

```
[psiTurk 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 hte 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␣
↪21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
Type the reason for the bonus. Workers will see this message: Here's a bonus!
gave bonus of $2.00 to 21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
[psiTurk server:on mode:sdbx #HITs:0]$ worker bonus --amount 2.00␣
↪21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
Type the reason for the bonus. Workers will see this message: Here's another one!
gave bonus of $2.00 to 21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
```

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␣
↪2ECYT3DHJHP4RRU304P8USX9BCXU1O
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 2ECYT3DHJHP4RRU304P8USX9BCXU1O
gave a bonus of $2.00 to 2MB011K274J7PY7FQ1ZN76UXH0ECED
bonus already awarded to 21A8IUB2YU98ZV9C5BUL3FBJB5B8K7
```

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␣
↪2ECYT3DHJHP4RRU304P8USX9BCXU1O
Type the reason for the bonus. Workers will see this message: Thanks for moving␣
↪science forward!
bonusing workers for HIT 2ZQIUB2YU98JX6A4V3C0IBJ9W0HL9P
gave a bonus of $3.00 to 27UQ45UUKQOYW1ZFLNJ8RG012VYDVP
gave a bonus of $2.50 to 24IIHPCGJ2D2H2KFPX80MPPSKQM933
```

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": "2ZRNZW6HEZ6OUI7FRTZ6CGUMGIQPZ0",
        "accept_time": "2014-03-04T16:14:05Z"
    },
    {
        "status": "Submitted",
        "assignmentId": "2XB92NJKM05B2XAD1YN2JTP9TYXAFG",
        "workerId": "A2O6BB9HXEUXX1",
        "submit_time": "2014-03-03T23:35:17Z",
        "hitId": "2RWSCWY2AOO2W03X0OFGTSCMKZZ22I",
        "accept_time": "2014-03-03T23:34:19Z"
    }
]
```

2. Listing approved workers for a specific HIT:

```
[psiTurk server:on mode:sdbx #HITs:0]$ worker list --approved  --hit␣
↪2ECYT3DHJHP4RRU304P8USX9BCXU1O
listing workers for HIT 2ECYT3DHJHP4RRU304P8USX9BCXU1O
[
    {
        "status": "Approved",
        "assignmentId": "21A8IUB2YU98ZV9C5BUL3FBJB5B8K7",
        "workerId": "A2O6BB9HXEUXX1",
        "submit_time": "2014-02-26T03:26:55Z",
        "hitId": "2ECYT3DHJHP4RRU304P8USX9BCXU1O",
        "accept_time": "2014-02-26T03:26:36Z"
    }
]
```

1.11 Configuring Databases

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.

Databases can be configured via the command line or by editing the configuration files directly. See the db command documentation for a full list of database commands available in the **psiTurk** shell. You can also view your current database settings by typing:

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

in the command line shell.

See also:

Database parameters For details on how to configure databases in *config.txt*.

Local configuration file For details on the local configuration file *config.txt*.

1.11.1 Using SQLite

Perhaps the simplest solution is to use SQLite. This is a simple, easy to use database solution that is written to a local file on the same computer as is running the psiTurk shell/server. By default **psiTurk** will use a local SQLite database.

To use a SQLite data base, 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.

You can also change this setting using the command line:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db use_local_file FILENAME.db
```

and verify the changes using:

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

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 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. Luckily, **psiTurk** can help you set up such a database (usually for free).

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.

1.11.2 Using a self-hosted MySQL database (recommended)

A more robust solution is to set up a MySQL database. **psiTurk** relies on SQLAlchemy for interfacing with database which means it is easy to switch between MySQL, PostgreSQL, or SQLite. We recommend MySQL because we have tested it, but other relational database engines may works as well.

To use an existing MySQL database:

```
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 (standard 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. Sequel Pro is a nice GUI database client for MySQL for Mac OS X.

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

```
$ 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, *turkdemo* by default

```
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. Your university may be able to provide this as well. MySQL is a very ubiquitous piece of software.

1.11.3 Obtaining a low-cost (or free) MySQL database on Amazon's Web Services Cloud

While not terribly difficult, installing and mangaging 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.**

The **psiTurk** command line provides a way to create a small MySQL database on Amazon's cloud using the RDS service. The command for this are available under the *db* command. Type:

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

for a list of sub-commands. The commands that begin with *aws_* directly interface with the Amazon cloud.

Note: Of course, you must have valid AWS credentials to use this system. See Getting setup with Amazon Mechanical Turk and Global configuration file.

If you are using psiturk with an IAM user, and if you want to use AWS RDB services via psiturk, add the *AmazonRDSFullAccess* AWS policy or an equivalent custom policy to your IAM user. See AWS docs here.

1.11.4 AWS Regions

AWS divides their cloud into different "regions" based on the location of the data center. To see a list of available regions type:

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

This command will also show which region you are currently using. The region is also set in your *~/.psiturkconfig* Global configuration file. You can also get the current region by typing:

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

To change your region simply type:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_set_region [<region_name>]
```

where *region_name* is one of the options listed by *db aws_list_regions*.

Why is this important? If you start an instance in one region, then switch regions, it will not show up in your list anymore. The regions are sort of independent from one another. Thus it is important to remember **which region** your instance was started on (i.e., which data center).

Note: It is probably fine to just keep the region set to a single value perhaps geographically closer to your location. This functionality is just provided in case the default region isn't working for you.

1.11.5 Creating an RDS Instance

After you have decided on a region, it is fairly easy to create a database instance. Type:

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

to see all available instances associated with your account **in the current region**. If you haven't created any instances in this region yet you should get a message like:

```
There are no DB instances associated with your AWS account in region  us-east-1
```

To create a new instance use the *db aws_create_instance* command:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_create_instance [<instance_id> <size>
↪<username> <password> <dbname>]
```

The optional arguments allow you to create the database in one command. If you prefer you can use an interactive mode by just typing:

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

This will print the following message describing the various options you need to specify for your database instance:

```
*************************************************
Ok, here are the rules on creating instances:

instance id:
  Each instance needs an identifier.  This is the name
  of the virtual machine created for you on AWS.
  Rules are 1-63 alphanumeric characters, first must
  be a letter, must be unique to this AWS account.

size:
  The maximum size of you database in GB.  Enter an
  integer between 5-1024

master username:
  The username you will use to connect.  Rules are
  1-16 alphanumeric characters, first must be a letter,
  cannot be a reserved MySQL word/phrase

master password:
  Rules are 8-41 alphanumeric characters

database name:
  The name for the first database on this instance.  Rules are
  1-64 alphanumeric characters, cannot be a reserved MySQL word
*************************************************
```

Then you will be prompted to specify values for these fields. If you follow the rules correctly your command will execute successfully:

```
enter an identifier for the instance (see rules above): mydb
size of db in GB (5-1024): 5
master username (see rules above): UsernameXXXXX
master password (see rules above): PasswordXXXXX
name for first database on this instance (see rules): myexp
*****************************
  Creating AWS RDS MySQL Instance
    id:  mydb
    size:  5  GB
    username:  UsernameXXXXX
    password:  PasswordXXXXX
    dbname:  myexp
    type: MySQL/db.t1.micro
    ________________________
 Be sure to store this information in a safe place.
 Please wait 5-10 minutes while your database is created in the cloud.
 You can run 'db aws_list_instances' to verify it was created (status
 will say 'available' when it is ready
```

The instructions mention that it can take a few minutes for you database to "spin up". If you run *db aws_list_instances* after a few minutes you should now see your database in the cloud:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_list_instances
Here are the current DB instances associated with your AWS account in region  us-east-
↪1
        --------------------
        Instance ID: mydb
        Status: creating
```

Notice the status is "creating" (this means the database is not available yet). Just wait a bit longer. It really can take 10-15 minutes! Other possible status messages for an instance include *backing-up* (AWS automatically backs up your database in case of data loss. At this time **psiTurk** does not help you access those backups, you'll have to do that from the AWS web console.)

When your database is ready the message from *db aws_list_instances* should look like:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_list_instances
Here are the current DB instances associated with your AWS account in region  us-east-
↪1
        --------------------
        Instance ID: mydb
        Status: available
```

If you have multiple instances they will also appear in this list.

Danger: Multiple instances increase the possible charges you'll incur to Amazon since you are charged per-instance.

Once your instance is created and "available" if you type *db get_config* you'll notice that your experiment is still configured to use whatever setting you had previously:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db get_config
Current database setting (database_url):
        sqlite:///participants.db
```

To actually **use** your instance you need to tell **psiTurk** which instance:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db use_aws_instance mydb
Switching your DB settings to use this instance.  Are you sure you want to do this? y
enter the master password for this instance: PasswordXXXXX
AWS RDS database instance mydb selected.
Here are the available database tables
        myexp
Enter the name of the database you want to use or a new name to create a new one:␣
↪myexp
Successfully set your current database (database_url) to
        mysql://UsernameXXXXX:PasswordXXXXX@mydb.cdukgn44bkrv.us-east-1.rds.amazonaws.
↪com:3306/myexp
```

And now your experiment will save data to this MySQL database in the Amazon cloud! Notice that Amazon has assigned your computer a random looking hostname/ip (mydb.cdukgn44bkrv.us-east-1.rds.amazonaws.com). You can connect using any standard MySQL client (e.g., Sequel Pro) which is running on the same computer as you **psiTurk** process

Note: **psiTurk** automatically makes instances so that only the current computer's ip address can access the database for security reasons. To modify that you can use the Amazon Web Services control panel or simple delete and spin up a new database instance.

To switch back to a local SQLite file:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db use_local_file FILENAME.db
Updated database setting (database_url):
        sqlite:///FILENAME.db
```

It is **important** that you delete your instance when you are finished using it. Otherwise you will be charged (usually fractions of a penny per hour). Assuming I wanted to delete my new *mydb* instance here is an example session:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_list_instances
Here are the current DB instances associated with your AWS account in region  us-east-
↪1
        --------------------
        Instance ID: mydb
        Status: available
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_delete_instance
Here are the available instances you can delete:
          mydb ( available )
Enter the instance identity you would like to delete: mydb
Deleting an instance will erase all your data associated with the database in that␣
↪instance. Really quit? y or n: y
DBInstance:mydb
AWS RDS database instance mydb deleted.  Run `db aws_list_instances` for current␣
↪status.
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_list_instances
Here are the current DB instances associated with your AWS account in region  us-east-
↪1
        --------------------
        Instance ID: mydb
        Status: deleting
```

After waiting a bit verify that you instance actually has been deleted:

```
[psiTurk server:off mode:sdbx #HITs:0]$ db aws_list_instances
There are no DB instances associated with your AWS account in region  us-east-1
```

Overall we think this is pretty cool and nicely leverages the fact that you already got a Amazon Web Services account when you signed up to use Amazon Mechanical Turk! However, remember, this **can incur hosting charges**. We have set things up so that this process creates very small, very simple RDS instances (which are the cheapest kind). However, leaving an instance running – or multiple instances – for a really long time can incur service charges which will be billed to your account by Amazon at the end of the month (you may not realize the charges until later).

The point is that using a free MySQL database hosted by your university or another provider may be better, but this solution is available for researchers who can afford to pay the hosting fee and would like everything in one place.

1.11.6 Obtaining a free MySQL database via OpenShift

If you are hosting your experiment on OpenShift, if you add a *MySQL* cartridge to your gear, **psiTurk** will automatically save data to that db instead of to whatever is specified in your *database_url* config. OpenShift gears, including using MySQL cartridges, are free unless you change default configuration settings.

See also:

PsiTurk OpenShift documentation.

1.12 Step-by-step Tutorials

We have a number of helpful step-by-step tutorials that introduce key concepts in using **psiTurk**.

1.12.1 Getting up and running with the basic Stroop task

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 basic Stroop experiment that ships default with psiTurk. 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:

```
$ 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:

```
$ cd psiturk-example
$ ls -la
total 16
drwxrwxr-x   6 gureckis  staff   204 Mar 31 12:18 .
drwx------  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.

Configure your global psiTurk options

When you run `psiturk-setup-example` the first time, a global configuration file is created in your local directory named `~/.psiturkconfig`. In order to get access to all the psiTurk features you need to enter credentials for accessing Amazon Web Services and psiturk.org. Both of these can be added to *~/.psiturkconfig*.

To access Amazon Mechanical Turk and other Amazon Web Services features you needs to enter your AWS Credentials (see these instructions for details). You can leave the *aws_region* at the default value.

To access psiTurk online features such as the Ad Server you need to create an account on psiturk.org. Please visit http://psiturk.org/register to sign up or http://psiturk.org/login to obtain your crediations. On your psiTurk dashboard click "API Keys" and enter them into your file.

See also:

Please read more about the global configuration file, getting set up with Amazon Web Services, and getting setup with psiturk.org on their respective documentation pages.

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-line shell. To open the shell, run `psiturk` a valid experiment folder. You should see something like this (though probably colorized on your display):

```
$ psiturk

http://psiturk.org
 ______   ______     __     ______   __  __     ______     __  __
/\  == \ /\  ___\   /\ \   /\__  _\ /\ \/\ \   /\  == \   /\ \/ /
\ \  _-/ \ \___  \  \ \ \  \/_/\ \/ \ \ \_\ \  \ \  __<   \ \  _"-.
 \ \_\    \/\_____\  \ \_\    \ \_\  \ \_____\  \ \_\ \_\  \ \_\ \_\
  \/_/     \/_____/   \/_/     \/_/   \/_____/   \/_/ /_/   \/_/\/_/

             an open platform for science on Amazon Mechanical Turk
```

(continues on next page)

(continued from previous page)

```
---------------------------------------------------------------------
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 1.0.10dev
Type "help" for more information.
[psiTurk server:off mode:sdbx #HITs:0]$
```

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:

More documention of the shell including documentation of each available command is available here.

Start/stop the experiment server

The **psiTurk** experiment server is a separate process that acts as a custom, local web server (similar to Apache). 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:

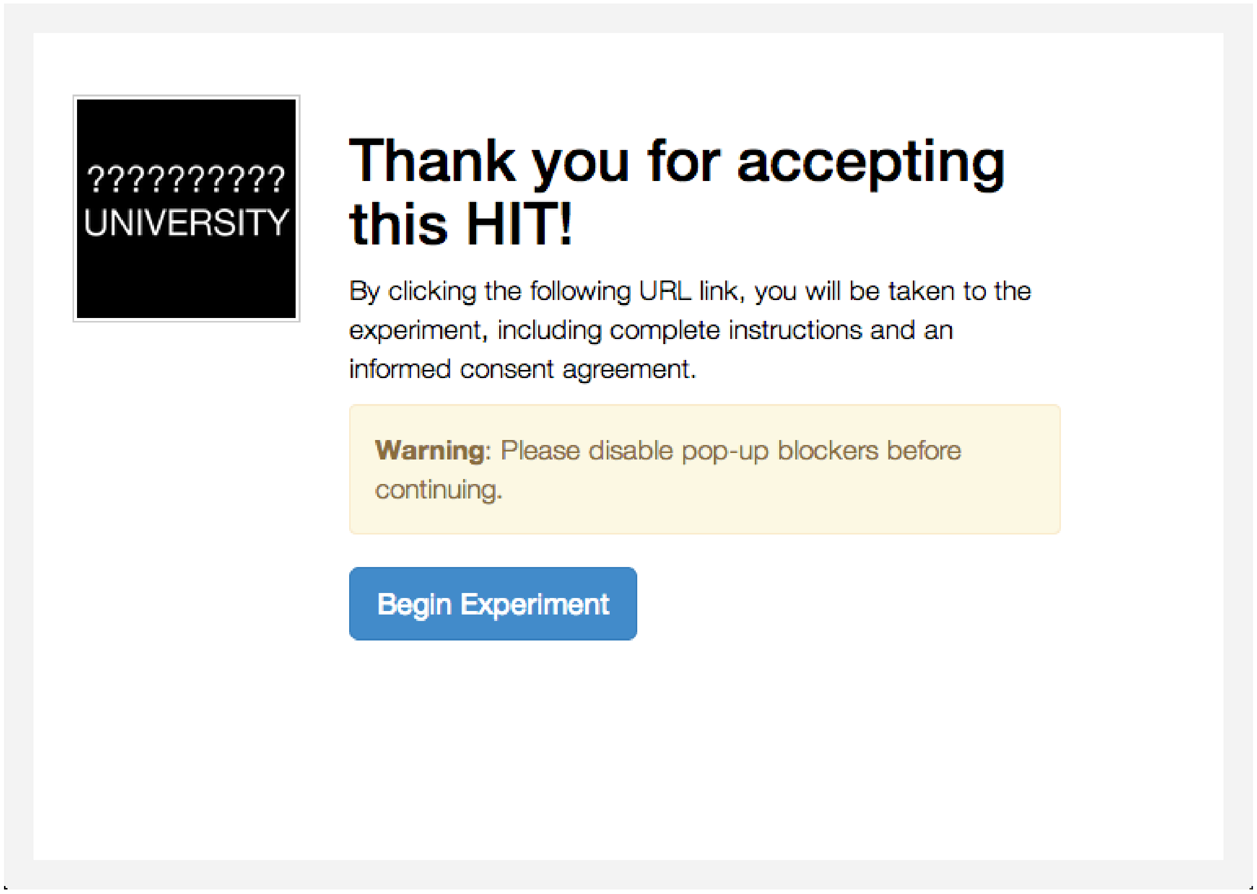
```
http://localhost:22362/ad?assignmentId=debug7FIXMF&hitId=debugI3XW1P&
↪workerId=debugY3UNQY
```

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 experiment 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!

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

Warning: Please disable pop-up blockers before continuing.

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 the 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 questionaire 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: 3SA4EMRVJV2ALPN29ZGP6BDPNBS0P0
    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
```

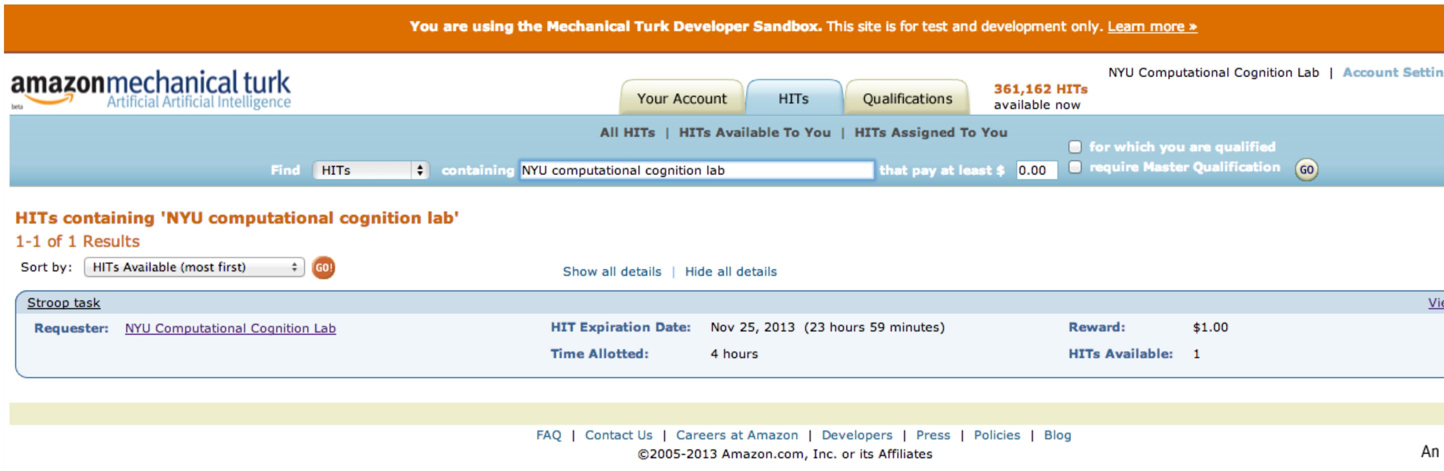
This example create a hit with 5 "slots" for participants (or 5 assignments). The reward is $1.00 and the participant has 1 hour to complete the task after accepting the HIT before it will be returned. Finally the unique "ad" for this experiment/HIT is displayed at the bottom. Notice that the ad is hosted on `https://ad.psiturk.org` which means it will always be visible to virtually all participants (see more info about the Secure Ad Server).

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:

```
[psiTurk server:on mode:sdbx #HITs:1]$ hit list active
Stroop task
        Status: Assignable
        HITid: 3SA4EMRVJV2ALPN29ZGP6BDPNBS0P0
        max:5/pending:0/complete:0/remain:5
        Created:2014-03-31T21:32:27Z
        Expires:2014-04-01T21:32:27Z
```

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:


You are using the Mechanical Turk Developer Sandbox. This site is for test and development only. Learn more »

amazonmechanical turk
Artificial Artificial Intelligence

Your Account | HITs | Qualifications

361,162 HITs available now

NYU Computational Cognition Lab | Account Settin

All HITs | HITs Available To You | HITs Assigned To You

Find HITs containing NYU computational cognition lab that pay at least $ 0.00

for which you are qualified

require Master Qualification

GO

HITs containing 'NYU computational cognition lab'

1-1 of 1 Results

Sort by: HITs Available (most first) GO!

Show all details | Hide all details

Stroop task

Vi

Requester: NYU Computational Cognition Lab

HIT Expiration Date: Nov 25, 2013 (23 hours 59 minutes)

Reward: $1.00

Time Allotted: 4 hours

HITs Available: 1

FAQ | Contact Us | Careers at Amazon | Developers | Press | Policies | Blog

©2005-2013 Amazon.com, Inc. or its Affiliates

An

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 SQLLite 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

Approve/Reject Workers

Assigning bonuses

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!

Further learning...

This concludes the conceptual overview of the Stroop example that ships with **psiTurk**. Continue reading the *decomposing the Stroop task <decompose_stroop.html>* section to learn more about the gritty details. This concludes the conceptual

1.12.2 Decomposing the Stroop task

1.13 Anatomy of a basic psiTurk project

Every **psiTurk** compatible project should include a few basic files. As an example here 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.

1.13.1 config.txt

This is the basic configuration file for the project.

See also:

Local configuration files For details on the structure of these files.

1.13.2 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.

1.13.3 participants.db

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

See also:

Configuring Databases For a complete guides to databases with **psiTurk**.

1.13.4 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.

1.13.5 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 lirbaries for customization purposes.

The *static/fonts/* directory

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

1.13.6 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 %}
```

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. boostrap.min.css is provided for free by the ad server.

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

```
<!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;
```

(continues on next page)

(continued from previous page)

```
                            padding: 2px;
                            border: 1px solid #ccc;
                        }

                        #container-ad {
                            position: absolute;
                            top: 0px; /* Header Height */
                            bottom: 0px; /* Footer Height */
                            left: 0px;
                            right: 0px;
                            padding: 100px;
                            padding-top: 5%;
                            border: 18px solid #f3f3f3;
                            background: white;
                        }
                </style>
        </head>
        <body>
                <div id="container-ad">

                        <div id="ad">
                                <div class="row">
                                        <div class="col-xs-2">
                                                <!-- REPLACE THE LOGO HERE WITH YOUR ␣
↪UNIVERSITY, LAB, or COMPANY -->
                                                <img id="adlogo" src="{{ server_
↪location }}/static/images/university.png" alt="Lab Logo" />
                                        </div>
                                        <div class="col-xs-10">

                                                        <!--
                                                                If assignmentid is
↪"ASSIGNMENT_ID_NOT_AVAILABLE"
                                                                it means the␣
↪participant has NOT accepted your hit.
                                                                This should display␣
↪the typical advertisement about
                                                                your experiment: who␣
↪can participate, what the
                                                                payment is, the time,␣
↪etc...

                                                        -->
                                                        {% if assignmentid ==
↪"ASSIGNMENT_ID_NOT_AVAILABLE" %}

                                                            <h1>Call for participants
↪</h1>
                                                            <p>
                                                                    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.
```

(continues on next page)

(continued from previous page)

```
                                                                          </p>
                                                                          <div class="alert alert-
↪danger">
                                                                                      <strong>This␣
↪task can only be completed once.</strong>
                                                                                      If you have␣
↪already completed this task before the system will not
                                                                                      allow you to␣
↪run again. If this looks familiar please return the
                                                                                      HIT so␣
↪someone else can participate.
                                                                          </div>
                                                                          <p>
                                                                                  Otherwise, please␣
↪click the "Accept HIT" button on the Amazon site
                                                                                  above to begin␣
↪the task.
                                                                              </p>


                                                                      {% else %}

                                                                              <!--
                                                                                      OTHERWISE
                                                                                      If␣
↪assignmentid is NOT "ASSIGNMENT_ID_NOT_AVAILABLE"
                                                                                      it means the␣
↪participant has accepted your hit.
                                                                                      You should␣
↪thus show them instructions to begin the
                                                                                      experiment ...
↪ usually a button to launch a new browser
                                                                                      window␣
↪pointed at your server.

                                                                                      It is␣
↪important you do not change the code for the
                                                                                      openwindow()␣
↪function below if you want you experiment
                                                                                      to work.
                                                                              -->
                                                                          <h1>Thank you for␣
↪accepting this HIT!</h1>
                                                                          <p>
                                                                              By clicking the␣
↪following URL link, you will be taken to the experiment,
                                                                              including complete␣
↪instructions and an informed consent agreement.
                                                                          </p>
                                                                          <script>
                                                                                      function␣
↪openwindow() {
                                                                                      popup =␣
↪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+'');
                                                                                      popup.
↪onunload = function() { location.reload(true) }
```

(continues on next page)

(continued from previous page)

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

                                                        <button type="button" class=
↪"btn btn-primary btn-lg" onClick="openwindow();">
                                                                  Begin Experiment
                                                                </button>


                                                        {% endif %}
                                                        <!--
                                                                endif
                                                        -->
                                        </div>
                        </div>
                </div>
        </body>
</html>
```

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

```
<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:

```
<!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␣
↪and jquery
        -->
                <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>
```

(continues on next page)

(continued from previous page)

```
                <script type="text/javascript">
                        // These fields provided by the psiTurk Server
                        var uniqueId = "{{ uniqueId }}";  // a unique string␣
↪identifying the worker/task
                        var condition = "{{ condition }}"; // the condition number
                        var counterbalance = "{{ counterbalance }}"; // a number␣
↪indexing counterbalancing conditions
                        var adServerLoc = "{{ adServerLoc }}"; // the location of␣
↪your ad (so you can send user back at end of experiment)
                </script>

                <!-- utils.js and psiturk.js provide the basic psiturk functionality -
↪->
                <script src="static/js/utils.js" type="text/javascript"> </script>
                <script src="static/js/psiturk.js" type="text/javascript"> </script>

                <!-- task.js is where you experiment code actually lives
                        for most purposes this is where you want to focus debugging,␣
↪development, etc...
                -->
                <script src="static/js/task.js" type="text/javascript"> </script>

        <link rel=stylesheet href="static/css/bootstrap.min.css" type="text/css">
        <link rel=stylesheet href="static/css/style.css" type="text/css">
    </head>
    <body>
            <noscript>
                        <h1>Warning: Javascript seems to be disabled</h1>
                        <p>This website requires that Javascript be enabled on your␣
↪browser.</p>
                        <p>Instructions for enabling Javascript in your browser can␣
↪be found
                        <a href="http://support.google.com/bin/answer.py?hl=en&
↪answer=23852">here</a><p>
                </noscript>
    </body>
</html>
```

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 `<div>` elements which can be styled and used to show stimuli or instructions within a task.

1.14 Recording 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

1.14.1 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.

1.14.2 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.

1.14.3 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.

1.14.4 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.

1.15 Retrieving Datasets

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

1.15.1 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.

1.15.2 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 sqlalchemy make 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.

```
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 sqlalchemy setup
engine = create_engine(db_url)
metadata = MetaData()
metadata.bind = engine
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
```

(continues on next page)

(continued from previous page)

```
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)
```

1.15.3 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 (< >).

Top Level

The top level of the datastring contains summary information about the worker, as well as the datastring sub-objects:

```
{"condition": condition,
"counterbalance": counterbalance,
"assignmentId": assignmentId,
"workerId": workerId,
"hitId": hitId,
"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:

```
[{"uniqueid": uniqueid,
"current_trial": current_trial_based_on_#_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.

questiondata

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

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

eventdata

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

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

1.16 Customizing psiTurk

Describe *custom.py* file and more advanced techniques like automatically computing bonuses

1.17 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.

1.17.1 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:

```
// 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') {
            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.

1.17.2 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:

```
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.

1.18 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* are that:

- 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).
- You get free SSL if you want to host your own ad, which is good because the psiTurk Secure Ad Server goes down under heavy load.
- It's scaleable.
- You get a *Heroku* buffering server in front of your *psiTurk* gunicorn instance, which helps with performance a little bit (although it would be better to put nginx in front of gunicorn within the *psiTurk* instance).

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
   ```

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*

1. 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.

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

   ```
   git init
   ```

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

   ```
   heroku login
   ```

4. 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
   ```

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

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

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

   ```
   heroku config:get DATABASE_URL
   ```

7. Get the URL of your app:

   ```
   heroku domains
   ```

8. In your psiTurk example, open the *config.txt* file. Here, find and make the following settings for the these rows, and then save the file:

   ```
   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
   ```

9. 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 commmands, provided that you've filled out your credentials in your *.psiturkconfig* file. Running this script is the recommended approach!):

   ```
   heroku config:set ON_HEROKU=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>
   ```

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

```
git add .
```

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

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

12. 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:

```
git push heroku master
```

13. Run *psiTurk* locally on your machine:

```
psiturk
```

14. 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*."

15. Run through your experiment. You should now have some data in the database. To extract it into *csv* files, type:

```
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 HIT's. 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 that 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.

Also note that 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 runinng *postgresql* on your local machine, and passing the *DATABASE_URL* that you set in *config.txt* as a command-line option.

1.19 Running psiTurk on Amazon's Elastic Compute Cloud (EC2)

With *Amazon Web Services* (commonly abbreviated as *AWS*), you can host your experiment in the cloud, using *Amazon's Elastic Compute Cloud* (commonly abbreviated as *EC2*). What follows is a description of how to set up and modify *psiTurk* on *AWS* using a pre-built *EC2* image.

If you don't already have an *AWS* account, first follow the instructions in Getting setup with Amazon Mechanical Turk.

1.19.1 Setting up a psiTurk EC2 instance using a pre-built image

1. Sign in to your Amazon Web Services account and navigate to The AWS Console, then click on EC2 under the 'Compute' section, located under the 'All services' heading.
2. Make sure that the location in the top right corner is set to 'US East (N. Virginia)'. (If not, you will not find the pre-built image when searching for it.)
3. Either click the 'Launch Instance' button that appears on the EC2 dashboard, or click 'Instances' under the 'INSTANCES' section on the left menu, then click 'Launch Instance' there.
4. You should now be at 'Step 1: Choose an Amazon Machine Image (AMI)' in the EC2 Launch Instance Wizard. Click 'Community AMIs' on the left, then in the 'Search community AMIs' search box, search for 'ami-bcab37d4'. A single AMI should be listed: 'ubuntu-psiturk-2 - ami-bcab37d4'. Click 'Select' on this AMI.
5. Choose your instance type. The micro instance is free-tier eligible and should be sufficient unless you're expecting very high traffic, bandwidth or lots of heavy computation on your experiment server. Click 'Next: Configure Instance Details' at the bottom.
6. Click 'Next: Add Storage' and then 'Next: Add Tags'.
7. You should now be at 'Step 5: Tag Instance'. Click 'Add Tag' and name your EC2 instance in the 'Key' field so that you'll be able to tell it apart from other instances you might run in the future. Then click 'Next: Configure Security Group'.
8. A security group is a set of firewall rules that dictate who can access your server (based on IP) and through which ports. You can create multiple security groups and assign one or more of them to any of your EC2 instances. We'll use a single security group for our instance.

   Check the 'Create a new security group' radio button and fill in a name for your security group. There should already be a rule for SSH with its *Source* (which IPs can connect via SSH) set to *Anywhere* (any IP). You can change this to *My IP* for added security, but if your computer's IP address changes, which will likely happen if you change physical locations, you'll need to modify this rule before you can connect via SSH again.

   Click *Add Rule*, set the *Type* to Custom TCP Rule, and set the *Port Range* to '22362', the port that the psiTurk server runs on by default. If you set the *Source* to *My IP*, be sure to change it back to *Anywhere* before you try to run the experiment on Mechanical Turk, otherwise nobody will be able to access it. Click 'Review and Launch'.
9. If you chose *Anywhere* for either of the two Security Group Rules, you'll be shown a warning about this. Review your settings and click *Launch*.
10. You'll now be prompted to download a key pair to use for public key-based authentication when logging in via SSH. This is far more secure than password-based authentication. Select *Create a new key pair*, and name the key pair to whatever you want (preferably the same name as your instance). Click *Download Key Pair* and save the .pem file somewhere safe yet accessible as you'll need it every time you connect via *SSH*. Check the acknowledgment checkbox and click *Launch Instance* to complete the instance creation process.
11. On Linux or Mac, set the file permissions on the key so that only you can read it. You can do this by opening the terminal, navigate to the folder where you saved your key pair and then type

```
$ chmod 400 your-key.pem
```

1.19.2 Connecting to your EC2 instance using SSH

1. Navigate back to the EC2 console (*AWS Console <https://console.aws.amazon.com/console/>*. Then click on "Instances" under the "INSTANCES" section on the left menu and click in the checkbox for the instance that you want to connect to. In the info appearing at the bottom, look for the *IPv4 Public IP* entry.

2. Use the public key you downloaded during instance creation to connect to the machine at the public IP you just found. The default username for the pre-built image is *ubuntu*. On *Linux* or *MacOS*, open up a terminal session, navigate to the folder where you saved your key pair and type

   ```
   ssh -i your-key.pem.txt ubuntu@xx.xx.xx.xxx
   ```

   where *xx.xx.xx.xxx* should be replaced with the public IP you just found. Type *yes* when the system asks you whether to continue connecting. You should now be logged into the instance. If you get a *Permissions … are too open* error, follow the *chmod step* in the previous section to fix this.

1.20 Using psiTurk on OpenShift

Note: Consider trying the OpenShift PsiTurk cartridge. It involves less configuration, and you automatically get an nginx server in front of psiturk.

1.20.1 Get an OpenShift account

Setting up an OpenShift account is relatively easy. First, visit openshift.com and sign up for a new account. After verifying your account, choose the first step in the quick start guide, which starts taking you through the process of setting up an application. You want do to the following:

1. When choosing the type of application, select Python 2.7.
2. In the configuration step, choose a name as your public url that you would be happy for external visitors (i.e. your participants) to see. You do not need to link your application to a Git repository.
3. Before getting started, OpenShift will also ask you for a public SSH key. Here are some good instructions on how to find out whether you already have one, and to create one otherwise.

And that's it, you should now have created a python cartridge that you can use to run **psiTurk** experiments!

To access your application, visit the application overview page on OpenShift, click on your newly created python application, and under "Remote Access", copy the ssh command and paste it into your local terminal (Note: Windows users probably want to use an SSH client like PuTTY instead. Instructions for using PuTTY can be found here.). This should start a new OpenShift session. The command should look like this:

```
ssh SOMENUMBER@python-MYNAME.rhcloud.com
```

1.20.2 Using psiTurk on OpenShift

There are some idiosyncrasies involved in running psiTurk using OpenShift that you want to be aware of:

- You don't need to use 'sudo' to install psiTurk. Simply "pip install" from either pypi or github (see Installation Steps). "pip" is already included in your python application, so you don't need to install it.
- You do not have root access on OpenShift, so rather than placing the ".psiturkconfig" file in the root directory, psiTurk automatically stores it in "/app-root/data". So that's where you have to enter your AWS and psiTurk access keys. (Remember, the file gets created only after psiturk is run for the first time)
- Similarly, in order to make any changes or add files (e.g to start a new project), you need to cd into a directory where you have permission to do so. For example, you could use "/app-root/repo/".

- Note that on OpenShift, you can only edit text files (like ".psiturkconfig" or "config.txt") inside the terminal. If you don't know how to do that, one simple way is to use "nano", for example:

  ```
  nano config.txt # edit file, then exit with Ctr+X
  ```

- In your very first session, the port that you need to host your experiment (port "8080" - see below) will be blocked by another process, so you will have to kill it first. You can do so by first looking for the process, then grabbing its Process ID (the PID column) and killing it:

  ```
  $ lsof -i :8080
  $ kill INSERT_PID
  ```

1.20.3 Customize a new project

To run a psiTurk experiment (like the example Stroop task) you need to make two changes to the config.txt file, without which your experiment won't run on OpenShift:

1. The 'port' field needs to be set to '8080'
2. The 'host' field needs to hold your specific OpenShift ip address, which you can easily find like this:

```
echo $OPENSHIFT_PYTHON_IP
```

1.20.4 Example of first session

To put it all together, this is what your first OpenShift session could look like, in which you install psiturk and try out the Stroop task example.

```
$ pip install git+git://github.com/NYUCCL/psiTurk.git@dev
$ lsof -i :8080
$ kill INSERT_PID
$ cd app-root/repo/
$ psiturk-setup-example
$ cd psiturk-example
$ echo $OPENSHIFT_PYTHON_IP
$ nano config.txt # add IP and port number (8080) to config file and exit
$ psiturk # Start psiturk
```

Before you can go live, remember to change the global config file ("/app-root/data/.psiturkconfig").

1.21 Understanding Error Messages

When there is an error, various messages are shown to the work. This guide will document what the mean from an experimenter's point of view and what probably went wrong.

1.22 Frequently Asked Questions

1.22.1 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.

1.22.2 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.

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

psiTurk requires an generally internet-addressable computer. Some universities prevent this for security purposes. There are a couple of solutions if this situation applies to you. First you can run psiturk via an *ssh* session on any remote computer or server for which you can launch server processes. Examples would be a lab server that has a static ip address and allows users-lavel access to particular ports. Alternative there are a number of (free) services which will give you a unix command line "in the cloud" including Red Hat's OpenShift. Detailed instruction on how to do this are available here.

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

In general this set up is definitely possible. However, you may need to configure the wireless router that came with your internet service to forward particular incoming ports to your device (i.e., to you laptop instead of you phone or tablet). There are many excellent tutorials about this online.

Note: a new experimental feature called tunnels is in the works which may address this issue for many users.

1.22.5 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.

1.22.6 What do I need to know about running psiTurk on a remote server?

The **psiTurk** command line process and server generally works great over a *ssh* connection. Perhaps the only thing to be aware of are that you set the *host* field of your project's local configuration file to the ip address of the remote machine if you want to be able to easily access it. In addition, while the standard *debug* command automatically launches your web-browser, you usually don't want this behavior on the remove machine. Instead use *debug -p* to simply print the correct URL and copy/paste it into a browser on your local computer.

1.22.7 Can you program my experiment for me?

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

1.22.8 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. A good example is CodeAcademy's Javascript lessons.

1.22.9 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).

1.23 Getting help

There are a number of ways to get help with **psiTurk** if you are stuck.

1. The https://psiturk.org website has a wealth of information about the system.
2. There is a Google Group devoted to psiTurk located 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. Browse the issues list on github. This is an open discussion of possible issues, bugs, feature requests, etc… If your problem doesn't appear in the open or closed issues you might consider opening a bug report. 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 pursed the above options you may not get a quick response.

1.24 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 suggestion to avoid costly mistakes from happening are

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 fustrated 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.

CHAPTER 2

Contributing to **psiTurk**

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 will 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:

```
.. toctree::
  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 quizes, 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 fustrating 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 Red Hat's OpenShift 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.

CHAPTER 3

API Reference

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 <file_desc/exp_html.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.

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

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

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)

(continued from previous 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 `` <http://backbonejs.org/#Model-set>`__ and `` `get `` <http://backbonejs.org/#Model-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:

```
// 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:

```
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.

```
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:

```
<button type="button" id="next" value="next" class="btn btn-primary btn-lg continue">
    Next <span class="glyphicon glyphicon-arrow-right"></span>
</button>
```

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:

```
<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

```
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 = [ // any file here should be preloaded first
    "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

```
psiturk = new PsiTurk(uniqueId, adServerLoc);
...
psiturk.finishInstructions();
```

3.2 psiturk.org RESTful API