

Conducting experiments on Mechanical Turk via *mturkutils*

Florian Jaeger with help from Linda Liu, Zach Burchill, Wednesday Bushong, and Xin Xie

December 22, 2020

Contents

1 Overview	1
2 Develop and debug your experiment locally	1
3 Sandbox a HIT	1
4 Obtain and check sandbox results	3
5 Getting more out of your YAML configuration file	4
5.1 One YAML file for many HITs (and lists)	4
5.2 Avoiding the MTurk fee for 10 or more assignments	5
5.3 Preventing unwanted subjects from taking your experiment	6
6 Going live!	7
6.1 While the experiment is online	8
6.2 Getting your results	9
7 Soon after the experiment	9
7.1 Paying (approving) workers	9
7.2 Granting bonuses	9
7.3 Paying workers who couldn't complete the experiment due to technical difficulties	9
7.4 Blocking workers from future experiments	9

1 Overview

Make sure you have completed the tutorial on *Setting up Mechanical Turk and boto(3)*. Now that you are familiar with `boto(3)` and the `mturkutils`, it is time to get started on your own experiment. This tutorial walks you through how to sandbox your experiment on MTurk's sandbox, how to upload the (thoroughly tested) HIT to MTurk, and how to collect your data once all assignments for the HIT have been completed. This tutorial does *not* describe how to program your own experiment.

2 Develop and debug your experiment locally

After you have completed the HTML (and potentially Javascript or other) programming necessary for your experiment, the first recommended step is to debug the experiment locally on your computer. In many cases, you will not even need to install a local webserver on your computer. If your experiment does not record speech, video, or alike but only *presents* audio, video, or other stimuli then you likely will be able to simply open your HTML file on your computer in a browser. This allows you to go through your experiment and debug it. **Only after you have completed local debugging of your experiment, is it time to test whether your experiment also works on MTurk.**

3 Sandbox a HIT

The first step following completed developing and debugging is to *sandbox* your experiment. The MTurk sandbox provides an online testing environment that works like the real MTurk while avoiding that you have to pay for the HIT. You can host a HIT on the MTurk sandbox and then you or other team members can take that HIT in order to debug it. This will also be the first step in your debugging process at which you can test whether the data that you obtain from a completed HIT assignment is in the intended format.

You can walk through the steps described next using your own experiment, or using the short mock experiment contained in this tutorial. This mock experiment is hosted on the HLP Lab web server, and you won't have to modify it at all.

- Using your favorite text editor, check out the contents of `example-HIT.yaml` in the `hits/` folder. This is a YAML configuration file. **YAML configuration files specify the basic information for MTurk HITs** that *mturkutils* will use to upload your HIT(s). This includes the HIT description, the payment, the url that specifies where the experiment is hosted, and more (we'll cover configuration files in a more detail in Section 5.2).
- Carefully, check the YAML configuration file. This is the moment to check the YAML file you are using for the sandbox, as it likely the blueprint for the YAML file you'll be using once you move from sandbox to production.

Sandbox checklist—*before* uploading HIT(s) to sandbox

- Check whether the payment offered in the experiment matches the experiment offered in the HIT (as specified in the YAML configuration file).
- Check whether you want to batchify your HIT(s). Amazon Mechanical Turk now [charges an additional 20% fee for every HIT with more than 10 assignments](#). You can avoid this fee by splitting your HIT(s) into smaller batches of less than 10 assignments using *mturkutils*, as described in Section 5.2.
- Make sure that your YAML file(s) specify the right URLs and that those URLs point to HTML files that collect the data you intend to collect:
 - * Check whether your YAML file(s) create HITs for all of the intended conditions of your experiment, and *only* those conditions. Also keep in mind that data for all conditions should be elicited at the same time if you want to be able to state in your write-up that participants were randomly assigned to one of the conditions.
 - * Check whether the YAML file(s) for each HIT point to the correct URLs. If the URL that the YAML file points to does not exist all that you (or a future subject) might see is a blank page.
 - * Make sure that each URL or combination of URL parameters has the intended effect. This might require that you check for each of the HITs whether the stimuli, stimuli order, task, etc. correspond to the condition indicated in the URL / by the URL parameters.

- To upload the HIT to MTurk's sandbox, run (in the `mturkutils/boto3/` directory):

```
python loadHIT.boto3.py -s -c ../hits/example-HIT.yaml
```

Don't forget the `sandbox` tag, otherwise the HIT may end up live on MTurk for other workers to do (and you may end up paying real \$\$\$)! If this was successful, you should see a message similar to the following:

```
[$10,000.00] is the initial balance
You can preview your new HIT at:
https://workersandbox.mturk.com/mturk/preview?groupId=some_long_ID
[$10,000.00] is the final balance
```

You should also notice that a new file (example-HIT.success.yaml) was automatically generated in hits/. Note that this success file will be overwritten everytime you run `loadHIT` for the same configuration file. While we are in sandbox mode, this does not harm but once the experiment is live this would mean that it will be difficult to manage. **So make sure *not* to delete or overwrite the example-HIT.success.yaml file: you will be using it to collect your results.**

By default *mturkutils* will use the AWS credentials specified under your default profile (typically in `./aws/credentials`). **If you would like to upload the HITs under an account that requires different AWS credentials**, you can specify the `-p profile_name` tag, where `profile_name` should refer to a profile in your AWS credentials file. For example:

```
python loadHIT.boto3.py -s -c ../hits/example-HIT.yaml -p lab
```

For this to work, there must be a AWS access key and secret access key below a section called `[profile_name]` in your `./aws/credentials` file. For example, for the above example to work your `./aws/credentials` file should contain the following lines:

```
[lab]
# the HLP lab account
aws_access_key_id = <access key>
aws_secret_access_key = <secret access key>
```

- Go to the URL that was output by `loadHIT` (https://workersandbox.mturk.com/mturk/preview?groupId=some_long_ID), and **complete the HIT**. You will need to log into your MTurk sandbox worker account. If you see a website telling you that you do not have permission to access the HIT, log into your workersandbox account, and then visit the URL again.

4 Obtain and check sandbox results

Congratulations! By this point, you have successfully created a HIT (as a requester) and completed a HIT (as a worker). Now you can obtain and carefully check your results.

- **Download your results from the sandbox.** For example, let's say we have completed a few assignments the example HIT uploaded in the previous section. You can then pull the results from Mechanical Turk's sandbox with *mturkutils*:

```
python getResults.boto3.py -s -f ../hits/example-HIT.success.yaml
-r ../hits/example-HIT.results_120120.tab
```

Note that the result file will be overwritten each time you run `getResults`—similar to what `loadHIT` does with the success file. Thus, **it's recommended to include the date in the filename of the results file**, as in the code example in this section. Finally, if you uploaded the HIT(s) to sandbox under a specific profile, make sure to include that same profile name when you get the results (again by adding `-p profile_name`).

- **Parse your results into an interpretable format.** The output of `getResults` is a tab-separated values (.tab) file, with one row per completed assignment, and one column for each variable that your experiment returned to Mechanical Turk. This is usually not the format we need for data visualization and analysis. For any non-trivial experiment, this .tab file will also be hardly human readable. Since

we should carefully check whether the result file contains all the information that we will need later, the next step is to parse this file into the format we need. This can be done in Python, R, or whatever scripting language you prefer. In R, for example, `read_tsv()` from the `readr` package (part of the `tidyverse` package) makes reading in tab-separated files easy, and the `dplyr` package (also part of the `tidyverse`) helps with wrangling the data into a suitable format. For example:

```
library(tidyverse)

d = read_tsv("PATH_TO_YOUR_RESULT_FILE/YOUR_RESULT_FILE.tab")
```

- Once you have parse your results into a suitable format, it is time to thoroughly check whether the data contains all the information you will need to visualize and analyze your data.

Sandbox checklist—after obtaining results from sandbox

- Check that the HIT results in data being recorded once the participant (you or your team members) press submit. For example, the experiment might hang.
- If the HIT does record data, make sure that the data is in the intended format and that all variables you will need for your analysis are actually being recorded.
- Finally, check that the *format* of the recorded data is the one you intend. For example:
 - * Are all variables included (e.g., subject and item IDs, trial order information, block information, all dependent variables, all conditions)?
 - * Are the values provided for the variables in the format you intended? Are there unexpected NAs?
 - * *Check your design and counter-balancing.* This usually involves both whether nuisance variables are balanced within conditions and across conditions, whether all combinations of items do actually occur in your data, and whether each combination only occurs as often as intended.

5 Getting more out of your YAML configuration file

In addition to the basic components of the YAML file described above, you can add additional information to the YAML file that `mturkutils` can interpret. This section describes three features that are often helpful.

5.1 One YAML file for many HITs (and lists)

Conveniently `mturkutils` allows you to use a single YAML file to specify many different HITs. Specifically, the URL specified in the YAML file can contain a URL parameter whose value is described by a variable surrounded by curly parentheses. Here is an example of just the URL specification from a YAML file with the URL parameter `list` and the parameter value being specified by the variable `condition`:

```
question:
  url: https://www.hlp.rochester.edu/mturk/YOUR_DIR/exp.html?list={condition}
```

In the same YAML file, we then need to specify values that the variable `condition` can take. This is done in the `input` section of the YAML file following the URL specification. Say for an experiment with a 2-by-2 design crossing condition A and B with short and long exposure:

```
question:
  url: https://www.hlp.rochester.edu/mturk/YOUR_DIR/exp.html?list={condition}
  height: 750
  input:
    - condition: A_short
    - condition: B_short
    - condition: A_long
    - condition: B_long
```

This will create four HITs, one each for the following URLs:

- https://www.hlp.rochester.edu/mturk/YOUR_DIR/exp.html?list=A_short
- https://www.hlp.rochester.edu/mturk/YOUR_DIR/exp.html?list=B_short
- https://www.hlp.rochester.edu/mturk/YOUR_DIR/exp.html?list=A_long
- https://www.hlp.rochester.edu/mturk/YOUR_DIR/exp.html?list=B_long

Of course, this will only have the desired result if the HTML and Javascript code at these URLs knows how to handle the different values for the URL parameter *list*. Note that the number of assignments that you have specified at the top of the YAML file determines how many assignments will be collected from MTurk *for each of the URLs*. **The total number of assignments that this approach results in is thus the # of assignments specified in the YAML file *times* the # of rows specified in the input section of the YAML file** (four in this example).

Unfortunately, it **not *not* currently possible to specify multiple separate URL parameters through this method**. If your Javascript is using multiple URL parameters, you need to either create multiple separate YAML files for each complete URL (including the combination of URL parameters) or you need to modify your Javascript program to parse the single URL parameter that can be specified using the method described here into the multiple separate parameters expected by the remainder of your Javascript.

5.2 Avoiding the MTurk fee for 10 or more assignments

The same approach described in the previous section can also be used to avoid the 20% fee that Amazon charges on HITs with 10 or more assignments. Simply a) set the number of assignments specified in the YAML file to 9 or less, and b) use the input specification described in the previous section to make as many separate 'lists' (all of which are copies of the same list) as needed to reach the targeted number of participants.

For example, let's say you would like to collect data from 24 participants in each of two conditions A and B. We can then get 8 assignments three times for each of the two conditions:

```

---
title: Speech Perception Experiment. MUST BE NATIVE ENGLISH SPEAKER AND WEAR HEADPHONES.
description: Listen to sentences. Takes approximately 30 minutes.
keywords: psychology,experiment,speech,words
reward: 3.00
assignments: 8
#####
## HIT Timing Properties
#####

# this Assignment Duration value is 60 * 90 = 1.5 hour
assignmentduration: 7200

# this HIT Lifetime value is 60*60*24*5 = 5 days
hitlifetime: 172800

# this Auto Approval period is 60*60*24*15 = 15 days
autoapprovaldelay: 1296000

qualifications:
  builtin:
    # this is a built-in qualification -- user must have > 95\% approval rate
    - qualification: PercentAssignmentsApprovedRequirement
      comparator: GreaterThan
      value: 95
      private: true
    # this is a built-in qualification -- user must be in the United States
    - qualification: LocaleRequirement
      comparator: EqualTo
      locale: US
      private: false

question:
  url: https://www.hlp.rochester.edu/mturk/YOUR_DIR/exp.html?list={condition}
  height: 750
  input:
    # 24 subjects per between-subject condition; 48 subjects total
    - condition: A_1
    - condition: A_2
    - condition: A_3
    - condition: B_1
    - condition: B_2
    - condition: B_3

```

Like described in the previous section, this approach assumes that your the HTML and Javascript that the HIT's url points to will recognize the parameters values for list (A_1, A_2, ...) and select the intended list based on those values.

5.3 Preventing unwanted subjects from taking your experiment

Qualifications can be used to keep people from taking your experiment. For example, we standardly include the following qualifications in the YAML file for our HITs, limiting our experiments to workers who are in the US (as far as MTurk can tell based on their IPs) and who have 95% approval ratings from previous HITs they have taken:

```

qualifications:
  builtin:
    # this is a built-in qualification -- user must have > 95% approval rate
    - qualification: PercentAssignmentsApprovedRequirement
      comparator: GreaterThan
      value: 95
      private: true
    # this is a built-in qualification -- user must be in the United States
    - qualification: LocaleRequirement
      comparator: EqualTo
      locale: US
      private: true

```

Additionally, you can use qualifications to prevent anyone who has previously taken an experiment in the same series from taking another experiment of that type. This **Note that qualifications made on your production MTurk account are not visible during sandboxing (and vice versa). This will cause error messages if you try to include private qualifications in the YAML file for sandboxing.** The upshot of this is that you will typically have two separate YAML configuration files—one for sandboxing and one for production. Only the latter will contain the additional qualifications described next.

1. [Make an arbitrary qualification on the MTurk website.](#)
2. Assign that qualification to all subjects who have taken the previous experiments you are worried about. This can be done with `assignQual.py`. For details, check out:

```
python assignQual.py -h
```

3. In the YAML file for your new experiment add that workers should *not* have that qualification. I.e., your YAML file should contains the following lines under the qualifications section:

```

custom:
  # Has not done one of my other experiments
  - qualification: ID-OF-QUALIFICATION-YOU-MADE
    comparator: DoesNotExist
    private: true

```

4. Since these qualifications are not visible from the sanbox, you will typically add them to only the YAML file once you move to production. **If you intend to block subjects that have taken previous experiments, remember to include these additional qualifications in the YAML configuration file for production.**

6 Going live!

Before you upload your HIT to MTurk, make sure to go through a final check:

Final checklist

- Have you gone through both the both of the Sandbox checklists?
- Does your YAML configuration file for the live production version match the YAML production file for sandboxing (which you have carefully debugged), *except* for:
 - The number of assignments per HIT (which might have been set to 1 for sandboxing but now should yield the desired number of assignments).
 - **Any additional private qualifications**, such as required to block participants who have taken similar previous experiments from your lab.
- Ethics:
 - Does this experiment fall under a protocol approved by the human subject review board?
 - Is the protocol still valid, i.e., not expired?
 - Have you been added as an experimenter to this protocol?
 - Do you have valid CITI?
 - Does your experiment provide a contact (email) address?
- In the Javascript (.js) file for the experiment:
 - Is the consent form up to date?
 - Are you using the right RSRB protocol number?
 - Is the payment (*Reward*) specified in the YAML file correct?
 - Does the payment (*Reward*) specified in the YAML file match the payment specified in the instructions of your HTML file?
 - Has any code been removed that was meant for testing? (best not to ever add such code)

Once you have completed this checklist, you are ready to upload your HIT to MTurk. Simply repeat the steps outlined above for sandboxing a HIT but without the sandbox tag (-s or -sandbox).

6.1 While the experiment is online

Monitoring while the experiment is in progress

- Check progress of your experiment. If nobody is completing assignments for the HIT, it is likely that something went wrong.
- Regularly check the email account provided as contact for the experiment for emails from MTurk workers. Workers might run into technical difficulties. If that happens, consider the following questions. The sooner you act, the better your chances to create a bad reputation for the lab, to cause frustration among workers, or to waste money and time:
 - Is the problem so severe and likely to apply to others that you should abort the experiment?
 - What can be learned from this problem for future experiments? Keep notes in your to-do file that avoid the same problem in future experiments.
 - Be responsive to workers concerns and complaints.
 - Offer to pay people that have tried to complete a HIT but then ran into technical difficulties (e.g., the computer hung). See Section 7.3

6.2 Getting your results

Follow the steps described above for getting results from the sandbox except without the `-s` flag.

7 Soon after the experiment

7.1 Paying (approving) workers

The YAML configuration file for your experiment specifies a period after which workers who have completed an assignment for the HIT(s) will be automatically approved. It's this part in the YAML file:

```
# this Auto Approval period is 60*60*24*15 = 15 days
autoapprovaldelay: 1296000
```

Since IRB rules dictate that any participant has the right to abort an experiment at any point, we approve every worker who has taken the experiment. Even if you follow this guideline, it is recommended though that you keep track of whether some workers do seem to be gaming the system. You can add these workers to a qualification to avoid running them again in the future. Be careful though to ask yourself whether it wasn't lack of clarity of the instructions, rather than nefarious intent on the participant's side that caused the pattern of data. In my experience, the former is the far more common cause for data that appears to suggest that participants are not really doing the task you intended.

7.2 Granting bonuses

In addition to payments, MTurk provides the option to grant a bonus to a worker. This option can come in handily, for example, when there are multiple visits to an experiment (e.g., a longitudinal study). In that case the reward would be the basic payment that the worker received for completing any part of the experiment, and the bonus would be used as an incentive to complete *all* visits of the experiment. *Mturkutils* comes with a separate script to grant bonuses to workers.

7.3 Paying workers who couldn't complete the experiment due to technical difficulties

7.4 Blocking workers from future experiments

You can use *mturkutils*'s `blockWorkers` program to create a new private qualification on your MTurk requester account that identifies all participants who have taken your experiment. For that you will need a result file with worker IDs. All of the IDs in that file will be assigned the new qualification. For more details, see:

```
python blockWorkers.boto3.py -h
```