TurkSurveyor Manual

Adam Kapelner

January 30, 2014

Contents

1 Introduction 3

2 Getting the Default Survey Running 3

2.1 Obtaining the source code . . . 3
2.2 Setting Your Passwords and Server Information . . . 3
2.3 Customizing how the HIT is Displayed on MTurk . . . 3
2.4 Deployment . . . 5
2.5 Creating new Surveys on MTurk . . . 6

3 Customizing the Survey Questions 7

3.1 The Question Hash . . . 7
3.1.1 :signature (string) [required] . . . 7
3.1.2 :question_text (string) [required] . . . 7
3.1.3 :question_type (symbol) [required] . . . 7
3.1.4 :other_response_params (hash) [optional] . . . 10
3.1.5 :response_choices (string array) [required or optional] . . . 10
3.1.6 :question_title (string) [optional] . . . 10
3.1.7 :question_description (string) [optional] . . . 10
3.1.8 :question_type (symbol) [optional] . . . 11
3.1.9 :submit_text (string) [optional] . . . 11
3.2 Creating General Questions . . . 11
3.3 Creating Check Questions . . . 11
3.4 Demographic Questions . . . 11
3.5 Need For Cognition Assessment . . . 11
3.6 Miscellaneous Questions . . . 12
3.7 The Question Order . . . 12

4 Customizing the Survey Presentation 12

4.1 Preventing Satisficing . . . 12
4.2 More Textual Customization . . . 13
4.2.1 Separate HIT Preview Page . . . 13
4.2.2 Question Counter . . . 13
4.3 Design Customization . . . 13

5 Experimentation Using TurkSurveyor **14**
5.1 Text Manipulation Studies 14
5.2 Presentation Method Study 15
5.3 Experimental Versions 15

6 The Administrator Console **15**

7 Development **16**
7.1 Platform Information 16
7.2 Add a database file 16
7.3 Code Contributions 16
7.4 Files you should not Commit 17

1 Introduction

TurkSurveyor was written by Adam Kapelner and Dana Chandler. The project began as a generalization of the code used for the study Kapelner and Chandler (2010).

TurkSurveyor is especially designed for the following tasks:

- Running custom surveys on MTurk
- Running experiments on MTurk that use surveys

2 Getting the Default Survey Running

This section will help you get the pre-packaged default survey running on MTurk. Customizing your survey questions is covered in detail in section 3 and the design is covered in section 4.

2.1 Obtaining the source code

Instructions on how to download or view the source code can be found at the source code's homepage:

https://github.com/kapelner/turksurveyor

You will need a git source control program to download the code:

http://git-scm.com/downloads

2.2 Setting Your Passwords and Server Information

There is one file that is required and is NOT part of the repository because it contains personal information. In order to use **TurkSurveyor** you need to create the following file:

`lib/personal_information.rb`

This is the file that contains all your personal passwords and login information. Create this file using the code template found in figure 1. Replace all constants with your own, personalized information. Read the comments for explanation of each constant. If you are not using the recommended Aptana platform, ignore the last four fields of this configuration file.

2.3 Customizing how the HIT is Displayed on MTurk

Parameters such as the wage, country restriction, HIT title, assignment length, description, keywords, etc can be set in the file:

```
module PersonalInformation
  #admin portal related
  #the name of your project
  ProjectName = "TurkSurveyor Test Project"
  #the password to access the administrator's portal
  AdminPassword = "password"
  #server information
  #The location of your development server (if any)
  DevServer = "127.0.0.1:3000"
  #The location of the production server
  ProdServer = "myserver.com"
  #MTurk information
  #The access key ID provided by Amazon (see aws.amazon.com)
  AwsAccessKeyID = "my AWS access key"
  #The secret key provided by Amazon (see aws.amazon.com)
  AwsSecretKey = "my AWS secret key"
  #contact information
  #where do crash reports get sent to?
  EmailForCrashReports = "me@me.com"
  #when contacting a worker, how do you sign the email?
  MTurkWorkerContactFormSignature = "Adam"
  #Aptana Server-related information
  #what is your aptana site's name?
  AptanaApplicationName = "my_website"
  #your aptana account username
  AptanaAccountUsername = "adam"
  #your aptana account password
  AptanaAccountPassword = "password"
  #your server database password (see setup control panel)
  AptanaDatabasePassword = "mysql_password"
end
```

Figure 1: Template for `lib/personal_information.rb`

lib/survey_m_turk.rb

As an example, for the experiment found in Kapelner and Chandler (2010), the parameters used are found in figure 2.

```
#hit defaults
EXPERIMENTAL_WAGE = 0.11
DefaultBonus = 0.01 #in USD
EXPERIMENTAL_COUNTRY = 'US'
DEFAULT_SURVEY_HIT_TITLE = "Take a short 30-question survey ---
    $#{EXPERIMENTAL_WAGE}USD"
DEFAULT_SURVEY_HIT_DESCRIPTION = %Q|Answer a few short questions
     for a survey.|
DEFAULT_SURVEY_HIT_KEYWORDS = "survey, questionnaire, poll,
    opinion, study, experiment"

#time related HIT defualts
DEFAULT_SURVEY_ASSIGNMENT_DURATION = 60 * 45 # 45min
DEFAULT_SURVEY_HIT_LIFETIME = 60 * 60 * 6 # 6hr
DEFAULT_SURVEY_ASSIGNMENT_AUTO_APPROVAL = 60 * 60 * 72 # 72hrs

#other things to set
DefaultAcceptanceMessage = 'Thanks for a job well done!'
DefaultBonusMessage = 'Thanks for a superb job!'
DefaultRejectionFeedback = "Sorry, you did not take our survey
    seriously."
```

Figure 2: Parameters that must be customized in lib/survey_m_turk.rb

2.4 Deployment

We recommend using Aptana to deploy the site that will control your survey and collect your data. This will allow you get up and running within minutes of downloading the code.

To do so, open a command prompt in the directory of the project and run the following commands:

1. `gem sources -a http://gems.aptana.com`
2. `gem install aptana_cloud`

Then, SSH into the server and install the following gems in the project directory using `gem install <gem name>`:

1. crypt

2. ruby-aws
3. rubystats

You can now deploy the project via:

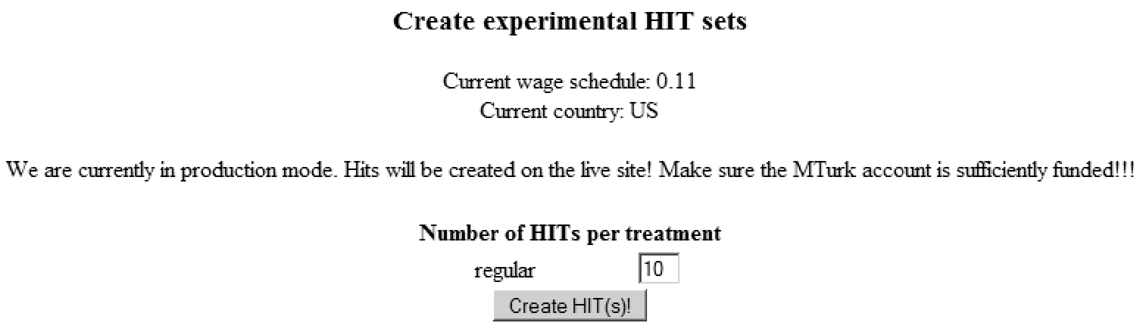
```
cap public deploy:migrations
```

Every time you make a change to your project, you will need to rerun this command.

2.5 Creating new Surveys on MTurk

Manually

You can create new survey HITs on MTurk by authenticating yourself in the admin console (see section 6), and following the instructions under the "create HITs" section (see figure 3).



Create experimental HIT sets

Current wage schedule: 0.11
Current country: US

We are currently in production mode. Hits will be created on the live site! Make sure the MTurk account is sufficiently funded!!!

Number of HITs per treatment

regular 10

Figure 3: The "create HITs" interface. For example, this would create 10 surveys on MTurk.

Automatically

You can set up **TurkSurveyor** to automatically create HITs for you by creating a cron job. A template is located in `config/crontab.txt` and is also shown in figure 4.[1] Make sure to replace the angle bracketed "project" with the name of your Aptana application.

```
0 2,8,14,20 * * * /usr/local/bin/ruby -C /home/aptana/homes/<
  project>/public/web/current script/runner "Survey.
  create_n_hit_sets(50)" -e production
```

Figure 4: Template for the cron tab file for running the cron job that automatically creates HITs. For example, this default cron job will create 50 HITs every 6hrs beginning at 10AM EST (note that the Aptana machines are on the UTC timezone).

Once you have written your cron job(s), you can install them by SSH'ing into the server, running "crontab -e", pasting the text, and saving.

[1]For more information on writing crontab files, see http://www.unixgeeks.org/security/newbie/unix/cron-1.html

3 Customizing the Survey Questions

This section will show you how to create questions, create experimental treatments, and customize the look and feel of your survey.

3.1 The Question Hash

Mastering how questions are stored in **TurkSurveyor** is essential to creating a custom survey. This section will go over the components of the question hash in detail.

Questions are stored as Ruby hash objects. They begin with a curly brace and end with a curly brace. Keys in the hash are ruby symbols which begin with a colon. Listed below are each customization key. For examples, see the file `lib/survey_true_and_check_questions.rb`.
Keys that are required are marked "required" and keys that are optional are marked as "optional".

3.1.1 :signature (string) [required]

Each question must have a *unique* identifier called the "question signature". If the question text is changed, we recommend you change the signature (since it now represents a different question). If the respondents have already filled in surveys with this question, do not change the signature, since you will not be able to access this past data from this question any longer.

3.1.2 :question_text (string) [required]

This is the question prompt itself (e.g. "What is your favorite color?").

3.1.3 :question_type (symbol) [required]

This key specifies the format of this question. The legal options are listed below, a short description is given for each, as well as an example hash:

:one_of_many A multiple choice question where the respondent is allowed to pick only one answer. This question's responses are rendered in HTML as radio buttons.

```
{
   :signature => 'dem_03_education',
   :question_type => :demographic,
   :question_text => %Q|What is your level of education?|,
   :response_type => :one_of_many,
   :response_choices => [
     'Some High School',
     'High School Graduate',
     'Some college, no degree',
     'Associates degree',
     'Bachelors degree',
     'Graduate degree, Masters',
     'Graduate degree, Doctorate'
   ]
}
```

:multiple_of_many A multiple choice question where the respondent can pick multiple responses or no responses at all. This question's responses are rendered in HTML as check boxes.

```
{
  :signature => 'dem_04_why_work_on_mturk',
  :question_type => :demographic,
  :question_text => %Q|Why do you complete tasks in Mechanical Turk?
    Please check any of the following that apply:|,
  :response_type => :multiple_of_many,
  :response_choices => [
    "Fruitful way to spend free time and get some cash (e.g., instead of
    watching TV)",
    %Q|For ''primary'' income purposes (e.g., gas, bills, groceries,
      credit cards)|,
    %Q|For ''secondary'' income purposes, pocket change (for hobbies,
      gadgets, going out)|,
    "To kill time",
    "I find the tasks to be fun",
    "I am currently unemployed, or have only a part time job"
  ]
}
```

:multiple_of_many_as_boxes A multiple choice question similar to the previous type with the difference being the responses are rendered as rounded boxes like. This question type was used to generate the IMC question found in both Kapelner and Chandler (2010) and Oppenheimer et al. (2009).

:likert_scale Question responses are scale from some minimum to some maximum. The responses are rendered in HTML on one horizontal line. Typical implementations are "do you agree or disagree with X?" with 7 levels, or "how happy do you feel on a scale of 1-10?".

You can specify the following optional parameters in :other_response_params (see section 3.1.4):

- :likert_min_score (integer) The minimum score on the Likert scale (defaulted to 1).
- :likert_max_score (integer) The maximum score on the Likert scale (defaulted to 7).

- :likert_descriptions (string array) Stores descriptions for each of the score levels (therefore, the array must have length $\ell = \max - \min + 1$). For instance, on a 1-5 agreement scale, the first entry could be "strongly disagree", the second could be "somewhat disagree", the third entry could be "no opinion", the fourth entry could be "somewhat agree", and the last entry could be "strongly agree". Note that there is no default.

```
Oppenheimer99MotivationCheck = {
  :signature => 'motivation_gauge_oppenheimer_2009',
  :question_type => :demographic,
  :question_title => "Motivation Question",
  :question_text => "How motivated were you to complete this survey?<br/>
    (1 - not motivated at all, 9 - very motivated)",
  :response_type => :likert_scale,
  :other_response_params => {:likert_max_score => 9}
}
```

:free_response_small Respondent answers inside a small text box. This is particular useful for simple questions like "What country were you born in?" or "What is your year of birth?", etc.

You can specify the following optional paramters in :other_response_params (see section 3.1.4):

- :input_max_characters (integer) The maximum number of characters the respondent is allowed in his response.
- :text_before_free_response (string) Text printed directly before the response box.
- :text_after_free_response (string) Text printed directly after the response box.
- :text_underneath_free_response (string) Text printed directly underneath the response box.

```
{
  :signature => 'dem_01_birth_year',
  :question_type => :demographic,
  :question_text => %Q|Which year were you born?|,
  :response_type => :free_response_small,
  :other_response_params => {
    :text_before_free_response => 19,
    :input_max_characters => 2
  },
}
```

:free_response_large Respondent answers inside a large response box. This is particularly useful for asking open-ended questions or for soliciting feedback.

You can specify the following optional paramters in :other_response_params (see section 3.1.4):

- :textarea_width (integer) The number of pixels wide the free response box is rendered
- :textarea_height (integer) The number of pixels high the free response box is rendered

```
FeedbackQuestion = {
  :signature => 'feedback_question',
  :question_type => :feedback,
  :question_title => "Please share your thoughts",
  :question_description => %Q|We are currently piloting this survey and would
    greatly appreciate any feedback you have.|,
  :question_text => %Q|What did you like most about this survey?<br/>What
     did you like least about this survey?<br/>Is there anything you would
     recommend to make it better?|,
  :response_type => :free_response_large,
  :other_response_params => {:needs_response => false}
}
```

3.1.4 :other_response_params (hash) [optional]

These are optional customizations (as a hash) that vary by the question type (see section 3.1.3). All question types accept the following customizations inside of the :other_response_params hash:

- :needs_response (boolean) If this is set to true, the page will prompt the user that they must enter an answer for this question. The default is true.

3.1.5 :response_choices (string array) [required or optional]

This is the string array that lists the answer choices for this question. This is required only for the :one_of_many, :multiple_of_many, and the :multiple_of_many_as_boxes question types.

3.1.6 :question_title (string) [optional]

Simply the title for this question.

3.1.7 :question_description (string) [optional]

The "fine print" for this question that is displayed between the title and the question prompt.

3.1.8 :question_type (symbol) [optional]

A signifier to distinguish between different types of questions. For regular questions, leave this option out. We recommend the following options:

- :imc_check This signifies a trick question with the intent to detect satisficing (see Oppenheimer et al. (2009)).
- :demographic This signifies this question is a demographic question, not part of the core of the survey.
- :need_for_cognition This signifies this question is part of the "need for cognition" assessment (see Cacioppo et al. (1984)).

3.1.9 :submit_text (string) [optional]

The text printed in the "submit" or "continue" button.

3.2 Creating General Questions

You can create one question hash (see section 3.1) for each general question and add this hash to the `TrueQuestions` array in the file `lib/survey_true_and_check_questions.rb`

3.3 Creating Check Questions

If you would like to give the same questions twice in order to immediately test question reliability, you can use the "check questions" feature.

Open the file `lib/survey_true_and_check_questions.rb` and add the index of the question you would like to duplicate in the `TrueQuestions` array. Remember, the first index is 0 and indices are separated by commas in Ruby arrays.

3.4 Demographic Questions

If you would like to give demographic questions, open the file `lib/survey_demographic_questions.rb`, add each question hash to the `DemographicQuestions` array. Remember to mark the `:question_type` key with `:demographic`.

If you change the age / gender questions, you should update the function `age_gender_to_s` found in `lib/survey_demographic_questions.rb` if you want your administrator console to display correctly.

3.5 Need For Cognition Assessment

If you would like to administer the "Need for Cognition" assessment found in Cacioppo et al. (1984), the 18-question assessment is found in `lib/survey_need_for_cognition_questions.rb`.

3.6 Miscellaneous Questions

You should add miscellaneous one-off questions to the file `lib/survey_miscellaneous_questions.rb` as individual question hashes (see section 3.1). We include as examples, a sample motivation question and a sample feedback question.

3.7 The Question Order

The "SurveyOrder" array found in `lib/survey_customization.rb` allows you to control the question order of the survey. You can add whole arrays such as "TrueQuestions" or individual questions such as "FeedbackQuestion". The survey will appear in the order you set.

The example found in `lib/survey_customization.rb` are the true questions, however many that may be, followed by an IMC check (see Oppenheimer et al. (2009)), followed by duplication check questions (see 3.3), followed by a few demographic questions, and finally, one feedback question:

```
SurveyOrder = [
  TrueQuestions,
  Oppenheimer99IMC1,
  CheckQuestions,
  DemographicQuestions,
  FeedbackQuestion
]
```

Randomization of question order is a future feature that is not supported now.

4 Customizing the Survey Presentation

4.1 Preventing Satisficing

We encourage surveyors to use the *Kapcha* anti-satisficing technology whose effectiveness was demonstrated in Kapelner and Chandler (2010). Set the following boolean in the customization file (`lib/survey_customization.rb`):

```
UseKapcha = true
```

For a more upgraded *Kapcha*, you can force the fade-in to repeat when the user navigates away. This effect was not investigated in Kapelner and Chandler (2010), but we believe it to be effective based on trial runs. To use this set the following boolean in the same file:

```
UseKapchaDiscourageNavigation = true
```

In order to exempt certain questions types (see 3.1.3) from the `Kapcha` fade-in, use the following array in the customization file. For example, the following will allow demographic and feedback questions to display without the fade-in technology:

```
QuestionTypesExemptFromKapCha = [:demographic, :feedback]
```

The last anti-satisficing measure is including an exhortative message. This was shown in Kapelner and Chandler (2010) to decrease attrition. To place the message on the bottom of the question, set the following parameter in the customization file in to ":bottom", to place it on the top, use ":top", and to omit the message, set it to nil:

```
ExhortativeMessageLocation = :top
```

To specify the message, set the following string in the customization file:

```
ExhortativeMessage = "Please answer carefully. Your responses will
  be used for research."
```

4.2 More Textual Customization

In the customization file, there are a few more options for presenting surveys that are not related to prevention of satisficing.

4.2.1 Separate HIT Preview Page

If you would like a separate HIT preview page, set the following boolean:

```
UseSeparatePreviewHitPage = true
```

Otherwise, a worker previewing the HIT will be sent to the directions page with the continue button disabled. If the directions page differs by treatment, this may bias the selection of subjects when you run studies.

4.2.2 Question Counter

To display a counter in the top right of the survey (e.g. "Question: 3/20"), set the following boolean:

```
ShowQuestionCounter = true
```

4.3 Design Customization

Color schemes, font-sizes, layouts (within reason), can all be changed by editing the style file `public/stylesheets/turksurveyor.css`. The elements are self-explanatory. We recommend using firebug to iterate style changes.

5 Experimentation Using TurkSurveyor

5.1 Text Manipulation Studies

To test the effect of text changes such as those found in Thaler (1985); Oppenheimer et al. (2009); Kapelner and Chandler (2010), you can use **TurkSurveyor**'s text manipulation feature.

In the customization file, alter the following hash:

```
RandomizedTreatments = {
  :place => [[:run_down, :fancy], [0.5, 0.5]]
}
```

Each key in this hash is the name of the textual change. Each value in this hash is an array of two arrays. The first array are the names of the different text strings, the second array is the probabilities associated with these text strings. For example, when the HIT above is created, it will be randomly assigned to one of two groups for the ":place" text.

To create differences in the textual display of the question, we use the ":randomization_text_switches" key in the question hash (see section 3.1 for more information).

For example, inserting the following key-value into a question hash would yield two different treatments:

```
:randomization_text_switches => [
  [
    :place,
    {:run_down => 'run-down grocery store',
       :fancy => 'fancy resort'}
  ]
]
```

Make sure the keys here correspond to the keys specified in the `RandomizedTreatments` hash specified before.

This randomization specifies that the "run down" treatment will receive the text "run-down grocery store" and the second treatment will receive the text "fancy resort".

However, where will this text be rendered? The final step is to somewhere in the question text, either in the title, text, or description, do the following:

```
:question_description => %Q|You are on the beach on a hot day.
  For the last hour you have been thinking about how much you
  would enjoy an ice cold can of soda. Your companion needs
  to make a phone call and offers to bring back a soda from
  the only nearby place where drinks are sold, which happens
  to be a #{FillInChars[0]}. Your companion asks how much you
  are willing to pay for the soda and will only buy it if it
  is below the price you state.|,
```

i.e. insert the Ruby string `FillInChars[0]` at the point you want **TurkSurveyor** to render the randomization. If you would like to insert a second text manipulation, you can use `FillInChars[1]`, etc.

5.2 Presentation Method Study

We recommend you declare the different treatments as symbols in the following array in the customization file:

```
Treatments = [:regular]
```

A bit of infrastructure is already built for you. Different treatments may need different text on the HIT introduction / directions page. This can be changed by editing the hash in the customization file:

```
TreatmentDirectionText = {
  :regular => "Please take the time to carefully read the
    instructions for each question and to give an accurate
    and honest answer."
}
```

Beyond this, best of luck designing the code for the experiment to test between presentation methods. For an example, you may want to download the code used for the Kapelner and Chandler (2010) study (http://danachandler.com/kapchastudy.html) and look at the code which tested for the "timing control".

5.3 Experimental Versions

During a study, you will find you are iterating as data comes in. To ensure each of your data points are consistent, we use experimental versioning. In the customization file, we change the experimental version by editing:

```
CURRENT_EXPERIMENTAL_VERSION_NO = 1
```

As for HIT listings in the administrator console, we may want to show HITs from more than one of versions. We can control this by editing the following array in the customization file:

```
ExperimentalVersionsToShow = [1, 3, 7]
```

6 The Administrator Console

Once the rails server is running the code, you can access the administrator console. Navigate to your server's URL and enter the administrator password you specified.[2]

Note that the "cleanup MTurk" function deletes all expired HIT listngs from the MTurk account. Note that the coded data dump is the same as the regular data dump as in it outputs the survey data in CSV, but the responses are coded and not raw text.

[2]i.e. in `lib/personal_information.rb`

7 Development

7.1 Platform Information

TurkSurveyor is written in Rails 2.1.0 and Ruby 1.8.6 although we hope this will be upgraded to newer technology soon. Rails is database-agnostic within reason. For information about downloading and setting up Ruby, Rails, and dependencies go to:

http://rubyonrails.org/download

Make sure you also install the gems described in section 2.4.

7.2 Add a database file

You need to create the database configuration file (`config/database.yml`) which is *not* included in the repository due to its personal nature.

You can copy the template in figure 5 and then change the text inside angle brackets.[3] After you create this file, make sure you create the databases you have specified.

```
development:
  adapter: mysql
  encoding: utf8
  database: <project>_dev
  username: root
  password: <password>
  host: localhost

test:
  adapter: mysql
  encoding: utf8
  database: <project>_test
  username: root
  password: <password>
  host: localhost
```

Figure 5: Template for `config/database.yml`

7.3 Code Contributions

We appreciate any contribution you can make to this project. We will maintain a list of want-to-have's at the source code's homepage, where contribution is especially encouraged.

[3]For more information, see http://www.tutorialspoint.com/ruby-on-rails/rails-database-setup.htm

7.4 Files you should not Commit

Do not commit any of the private files (`lib/personal_information.rb` and `config/database.yml`) as well as your log files, temp files, etc.

References

JT Cacioppo, RE Petty, and CF Kao. The efficient assessment of need for cognition. *Journal of Personality Assessment*, 48(3):306–307, 1984. URL `http://www.informaworld.com/index/785038000.pdf`.

Adam Kapelner and Dana Chandler. Preventing satisficing in online surveys: A "kapcha" to ensure higher quality data. *CrowdConf ACM Proceedings*, 2010.

Daniel M. Oppenheimer, Tom Meyvis, and Nicolas Davidenko. Instructional manipulation checks: Detecting satisficing to increase statistical power. *Journal of Experimental Social Psychology*, 45(4):867–872, July 2009. ISSN 00221031. doi: 10.1016/j.jesp.2009.03.009. URL `http://linkinghub.elsevier.com/retrieve/pii/S0022103109000766`.

Richard Thaler. Mental Accounting and Consumer Choice. *Marketing Science*, 4(3):199 – 214, 1985. URL `http://www.jstor.org/stable/183904`.