Amazon Mechanical Turk

- Advantages
  - On demand workforce
  - Scalable workforce
  - Qualified workforce
  - Pay only if satisfied
Terminology

- Requestors
- HITs (Human Intelligence Tasks)
- Assignment
- Workers (‘Turkers’)
- Approval and Payment
- Qualification
Amazon Turk Pipeline

1. Create a HIT Template
2. Publish HITs
3. Manage Results
4. Manage Workers
HIT Template

- HTML page that presents HITs to workers
  - Non-variable: all workers see the same page
  - Variable: every HIT has the same format, but different content
HIT Template

- Define properties
- Design layout
- Preview

Sample HIT Templates

<table>
<thead>
<tr>
<th>HIT Template Name</th>
<th>HIT Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Open-ended Question</td>
<td>Answer a Simple Question [See an example]</td>
</tr>
<tr>
<td>Blank Template</td>
<td>Default Title [See an example]</td>
</tr>
<tr>
<td>Categorization</td>
<td>Pick the best category [See an example]</td>
</tr>
<tr>
<td>Categorization using Masters</td>
<td>Pick the best category [See an example]</td>
</tr>
<tr>
<td>Data Collection</td>
<td>Find the Website Address for Restaurants [See an example]</td>
</tr>
<tr>
<td>Data Correction</td>
<td>Provide the correct spelling of search terms [See an example]</td>
</tr>
<tr>
<td>Data Extraction</td>
<td>Get Product Name from Image [See an example]</td>
</tr>
<tr>
<td>Image Filtering</td>
<td>Flag offensive content images (WARNING: This HIT may contain offensive content. Worker discretion is advised.) [See an example]</td>
</tr>
<tr>
<td>Image Tagging</td>
<td>Tag an image [See an example]</td>
</tr>
</tbody>
</table>
HIT Template

- Properties
  - Template Name
  - Title
  - Description
  - Keywords
  - Time Allowed
  - Expiration Date
  - Qualifications
  - Reward
  - Number of assignments
  - Custom options
HIT Template

- Design
  - HTML
HIT Template

- Design
  - Template Variables
    - Variables are replaced by data from a HIT data file

```
<img width="200" height="200" alt="imagevariableName" style="margin-right: 10px;" src="$\{image_url\}" />
```
HIT Template

- Design
  - Data File
    - .CSV file (Comma Separated Value)

Row 1: Variable Names
Rows 2-5: Variable for each HIT
HIT Template

- Result
  - Also .CSV

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CompanyName</td>
<td>City</td>
<td>State</td>
<td>Country</td>
</tr>
<tr>
<td>2</td>
<td>Amazon</td>
<td>Seattle</td>
<td>WA</td>
<td>USA</td>
</tr>
<tr>
<td>3</td>
<td>Zappos</td>
<td>Las Vegas</td>
<td>NV</td>
<td>USA</td>
</tr>
<tr>
<td>4</td>
<td>Starbucks</td>
<td>Seattle</td>
<td>WA</td>
<td>USA</td>
</tr>
<tr>
<td>5</td>
<td>Coke</td>
<td>Atlanta</td>
<td>GA</td>
<td>USA</td>
</tr>
<tr>
<td>6</td>
<td>Facebook</td>
<td>Menlo Park</td>
<td>CA</td>
<td>USA</td>
</tr>
</tbody>
</table>

Table rows separated by line breaks.
Columns separated by commas.

First row is a header with labels for each column.

<table>
<thead>
<tr>
<th></th>
<th>CompanyName, City, State, Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Amazon, Seattle, WA, USA</td>
</tr>
<tr>
<td>3</td>
<td>Zappos, Las Vegas, NV, USA</td>
</tr>
<tr>
<td>4</td>
<td>Starbucks, Seattle, WA, USA</td>
</tr>
<tr>
<td>5</td>
<td>Coke, Atlanta, GA, USA</td>
</tr>
<tr>
<td>6</td>
<td>Facebook, Menlo Park, CA, USA USA</td>
</tr>
</tbody>
</table>
Accessing assignment details in JavaScript

```javascript
var assignmentId = turkGetParam('assignmentId', '');
if (assignmentId != '' && assignmentId != 'ASSIGNMENT_ID_NOT_AVAILABLE') {
    var workerId = turkGetParam('workerId', '');
}

function turkGetParam(name, defaultValue) {
    var regexS = "[\?&]"+name+"=([^&#]*");
    var regex = new RegExp(regexS);
    var tmpURL = window.location.href;
    var results = regex.exec(tmpURL);
    if (results == null) {
        return defaultValue;
    } else { return results[1]; }
}
```

Function automatically included by Amazon

Also commonly see a `gup` function used for the same purpose
Publishing HITs

- Select created template

Select HIT Template

All of the HITs in a batch will use the same HIT template. The HIT template describes the layout and properties of the HITs.

<table>
<thead>
<tr>
<th>HIT Template Name</th>
<th>HIT Title</th>
<th>Creation Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Tagging</td>
<td>Identify landmarks <a href="#">See an example</a></td>
<td>October 01, 2010 8:26 AM PDT</td>
</tr>
</tbody>
</table>
Publishing HITs

- Upload Data File

**Upload Input Data**

To specify the data that will replace the variables in your HIT template, choose a Comma Separated Values (CSV) file that contains the input data. If you want to include images in your HIT, you will need to provide the URL. ([learn more](#))

**Image Tagging**

**Locate a New File**

Browse... [Upload]

[Download] a sample of the input file for this HIT template or learn more about [acceptable file formats](#)

**Or Choose an Existing File**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Input Lines</th>
<th>Creation Date</th>
</tr>
</thead>
</table>
Publishing HITs

- Preview and Publish
Qualification

- Make sure that a worker meets some criteria for the HIT
  - 95% Approval rating, etc.

- Requester User Interface (RUI) doesn’t support Qualification Tests for a worker to gain a qualification
  - Must use Mechanical Turk APIs or command line tools
Masters

- Workers who have consistently completed HITs of a certain type with a high degree of accuracy for a variety of requestors
  - Exclusive access to certain work
  - Access to private forum
- Performance based distinction
- Masters, Categorization Masters, Photo Moderation Masters – superior performance for thousands of HITs
Command Line Interface

- Abstract from the “muck” of using web services
- Create solutions without writing code
- Allows you to focus more on solving the business problem and less on managing technical details
- mturk.properties file for keys and URLs
- Input: *.input, *.properties, and *.question files
- Output: *.success, and *.results
- Tab delimited file
- Contains variable names and locations

<table>
<thead>
<tr>
<th>Image1</th>
<th>Image2</th>
<th>Image3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image1.jpg</td>
<td>Image2.jpg</td>
<td>Image3.jpg</td>
</tr>
</tbody>
</table>

**.input**
*.properties

- Title
- Description
- Keywords
- Reward
- Assignments
- Annotation
- Assignment duration
- Hit lifetime
- Auto approval delay
- Qualification
XML format

Define the HIT layout

Consists of:

- `<Overview>`: Instructions and information
- `<Question>`

Can be a QuestionForm, ExternalQuestion, or a HTMLQuestion
<Question>

- *QuestionIdentifier
- DisplayName
- IsRequired
- *QuestionContent
- *AnswerSpecification
  - FreeTextAnswer, SelectionAnswer, FileUploadAnswer
<Question>
  <QuestionIdentifier>my_question_id</QuestionIdentifier>
  <DisplayName>My Question</DisplayName>
  <IsRequired>true</IsRequired>
  <QuestionContent> [...] </QuestionContent>
  <AnswerSpecification> [...] </AnswerSpecification>
</Question>

<QuestionContent> (and <Overview>) can contain:
  • <Application>: JavaApplet or Flash element
  • <EmbeddedBinary>: image, audio, video
  • <FormattedContent> (later)
*.success and *.results

- ***.success**: tab delimited text file containing HIT IDs and HIT Type IDs
  - Auto-generated when HIT is loaded
  - Used to generate *.results

- **Submitted results in the last columns**
  - generate *.results with getResults command
  - tab-delimited file, last columns contain worker responses
Command Line Operations

- ApproveWork
- getBalance
- getResults
- loadHITs
- reviewResults
- grantBonus
- updateHITs
- etc
Loading a HIT

- loadHITs -input *.input -question *.question -properties *.properties -sandbox

- sandbox flag to create HIT in sandbox to preview
- preview flag also available
  - requires XML to be written in a certain way
Use **FormattedContent** inside a QuestionForm to use XHTML tags directly

- No JavaScript
- No XML comments
- No element IDs
- No class and style attributes
- No `<div>` and `<span>` elements
- Etc.
Specified in XML CDATA block inside a FormattedContent element

```xml
<QuestionContent>
  <FormattedContent><![CDATA[
    <font size="4" color="darkblue">Select the image below that best represents: Houses of Parliament, London, England</font>
  ]]></FormattedContent>
</QuestionContent>
```
Qualification Requirements

- qualification.1: qualification type ID
- qualification.comparator.1: type of comparison (greaterthan, etc.)
- qualification.value.1: integer value to be compared to
- qualification.locale.1: locale value
- qualification.private.1: public or private HIT

Increment the *.1 to specify additional qualifications
Worker must have 25% approval rate and HIT can be previewed by those that don’t meet the qualification
External HIT

Use an ExternalQuestion

```xml
<ExternalQuestion
    xmlns="http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2006-07-14/ExternalQuestion.xsd">
    <FrameHeight>400</FrameHeight>
</ExternalQuestion>
```

${helper.urlencode($urls)} to encode urls from *.input to show in externalpage.htm
External HIT

- In the external .htm:

```html
<form id="mturk_form" method="POST"
action="http://www.mturk.com/mturk/externalSubmit">
(...question...)
</form>
```

And then submit the assignment to Mturk

```javascript
if (gup('assignmentId') == "ASSIGNMENT_ID_NOT_AVAILABLE") {
    ...
} else {
    var form = document.getElementById('mturk_form');
    if (document.referrer && (document.referrer.indexOf('workersandbox') != -1)) {
        form.action = "http://workersandbox.mturk.com/mturk/externalSubmit";
    }
}
```
Create five questions, where the first 3 are required

```plaintext
#set($minimumNumberOfTags = 3)
#foreach($tagNum in [1..5])
<Question>
  <QuestionIdentifier>tag${tagNum}</QuestionIdentifier>
  #if($tagNum <= $minimumNumberOfTags)
    <IsRequired>true</IsRequired>
  #else
    <IsRequired>false</IsRequired>
  #end
</Question>
```
Qualification Test

- Given a request for a qualification from a worker, you can:
  - Manually approve qualification request
  - Provide answer key and Mturk will evaluate request
  - Auto-grant qualification

- Qualifications can also be assigned to a worker without a request
Qualification Test

- *.question, *.properties, *.answer
- Define the test questions in *.question and answers in *.answer

```
createQualificationType -properties qualification.properties
  -question qualification.question
  -answer qualification.answer
  -sandbox
```
<QuestionForm xmlns="http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2005-10-01/QuestionForm.xsd">
  <Overview>
    <Title>Trivia Test Qualification</Title>
  </Overview>
  <Question>
    <QuestionIdentifier>question1</QuestionIdentifier>
    <QuestionContent>
      <Text>What is the capital of Washington state?</Text>
    </QuestionContent>
    <AnswerSpecification>
      ...
    </AnswerSpecification>
  </Question>
</QuestionForm>
Qualification Test (Answer Key)

Auto-assign qualification and score with answer key
Qualification Test Properties

- name
- description
- keywords
- retrydelayinseconds
- testdurationinseconds
- autogranded
Matlab Turk Tool

aws_access_key = ;
aws_secret_key = ;
sandbox = true;

Initialize with keys and sandbox option

turk = InitializeTurk(aws_access_key, aws_secret_key, sandbox);

result = RequestTurk(turk, 'GetAccountBalance',
{'ResponseGroup.0','Minimal','ResponseGroup.1','Request'});

Parameters  Operations
<GetAccountBalanceResult>
  <Request>
    <IsValid>True</IsValid>
  </Request>
  <AvailableBalance>
    <Amount>10000.000</Amount>
    <CurrencyCode>USD</CurrencyCode>
    <FormattedPrice>$10,000.00</FormattedPrice>
  </AvailableBalance>
</GetAccountBalanceResult>
Paid By Bonus

- Approve individually or by batch
- Reject individually or by batch
- Give bonuses to good workers
- Can download batch into a .CSV, mark accept/reject, then upload updated .CSV to the Mechanical Turk
TurkCleaner

- Have the user select a subset of images that satisfy certain rules.

Is this a office cubicles? Yes

Copy .html into template, parse .CSV into Matlab readable format
DrawMe

- Line drawing on an image.
- Copy .html into Mturk template
- .CSV file can be parsed into Matlab cell arrays for processing
Demographics

**Nationality**
- 57% U.S.
- 32% India
- 5% Other
- 1% Romania
- 1% Pakistan
- 1% U.K.
- 0.50% Phillipines
- 0.50% Others

**Gender**
- 45% Female
- 55% Male
Demographics

Age

- 60+
- 51-60
- 41-50
- 31-40
- 25-30
- 18-24

Education

- Advanced
- Bachelors
- Associates
- Some College
- High School
Best Practice

- **Motivation**
  - Incentives: entertainment, altruism, financial reward

- **Task Design**
  - Easy to understand visuals, design interface such that accurate task completion requires as much effort as adversarial task completion, financial gain for amount of work tradeoff for worker
  - Creation task vs. Decision task

- **High Quality Results**
  - Heuristics such as gold standard and majority vote

- **Cost Effectiveness**
Creation Task vs Decision Task

- Creation:
  - Write a description of an image

- Decision:
  - Given two descriptions for the same image, decide which description is best
Iterative and Parallel

- **Iterative**: sequence of tasks, where each task’s result feeds into the next task (better average response)

- **Parallel**: workers are not shown previous work (better best response)
Task Design

(A) Easy for Humans
Chair? Airplane? ...

(B) Hard for Humans
Finch? Bunting?...

(C) Easy for Humans
Yellow Belly? Blue Belly? ...
Gold Standard

- Present workers with control questions where the answer is known to judge the ability of the worker.
- Requires keeping track of workers over time or presenting multiple questions per task.
Majority Vote

- Check the responses from multiple turkers against each other.
- Averaging multiple labels, etc.
Cost Effectiveness

- [Welinder, et. al.] Estimation of annotator reliabilities
  - Use the reliability of the annotator to determine how many additional labels are needed to correctly label the image.
Augmenting Computer Vision

- Using humans to improve performance
Augmenting Computer Vision

- Deterministic Users: assumed perfect users
- Turkers: subjective answers degrade performance (brown vs buff)

Q: Is the belly red? yes (Def)
Q: Is the breast black? yes (Def.)
Q: Is the primary color red? yes (Def.)
Augmenting Computer Vision

- Human answer corrects computer vision’s initial prediction
TurKit

- Toolkit for prototyping and exploring algorithmic human computation
generates ideas for things to see from 5 different workers and getting workers to sort the list
Crash-and-rerun programming

- Script is executed until it crashes
- Every line that is successfully run is stored in a database
- If script needs to be rerun, cost of rerunning human computation task is avoided by looking up the previous result (use keyword once)
- waitForHIT function that crashes unless results are ready
TurKit: Quicksort

quicksort(A)
  if A.length > 0
    pivot ← A.remove(once A.randomIndex())
    left ← new array
    right ← new array
    for x in A
      if compare(x, pivot) A
        left.add(x)
      else
        right.add(x)
    quicksort(left)
    quicksort(right)
    A.set(left + pivot + right)

A
compare(a, b) A
  hitId ← once createHIT(...a...b...)
  result ← once getHITResult(hitId)
  return (result says a < b) A

Use once if function is:
• deterministic
  • once Math.random() would result in the same value every run
• high cost
• has side-effects
  • ex: approving results from a HIT multiple times causes errors
TurKit: Parallelism

- If HIT A doesn’t finish, crash that fork and the next fork creates HIT C
- Subsequent runs will check each HIT to see if it’s done
- `join()` to ensure previous forks were successful
  - if previous forks unsuccessful, join crashes current path

```javascript
fork(function () {
    a = createHITAndWait() // HIT A
    b = createHITAndWait(...a...) // HIT B
})
fork(function () {
    c = createHITAndWait()  // HIT C
})
```
TurKit IDE
Turker Forum and Browser Plugin

- Turkopticon: (Union 2.0) shows reviews of requestors on Amazon MTurk
- TurkerNation
- Helpful Blogs for Requestors:
  - [Tips for Requestors]
  - [The Mechanical Turk Blog]