# Speech perception on the web

**This description of the software is also available in the file readme.pdf**

The library and the examples in this package implement web-based versions of a number of different types of experiments that are commonly used by researchers who study speech perception and psycholinguistics.

The package includes several example experiment paradigms:

1. **Audio transcription** presents audio files and asks the listener to transcribe (type) the words that they hear. This method is used in speech intelligibility studies like the SPIN (speech intelligibility in noise) task, and can also be used for free response judgements about voice quality or talker characteristics.

2. **Picture transcription** presents pictures and asks for a typed response, as might be needed for various sorts of norming studies for projects that use images with speech.

3. **Audio categorization** presents audio files in a two alternative forced choice (2AFC) paradigm, as is used in classical identification and categorization studies. The labels that are presented on the screen can be specified in a stimulus list so different response labels can be presented for each sound file. Responses are given by pressing select keys on the keyboard (currently the ‘m’ and ‘z’ keys).

4. **Audio paired comparison** presents audio files in pairs for a comparative 2AFC judgment; for instance whether the audio files are the “same” or “different”. In this paradigm as well as in the categorization paradigm, a correct answer can be specified and the listener is warned when their answer is incorrect.

5. **Video Categorization** presents movie clips (mpeg4 files) in a two alternative forced choice (2AFC) paradigm. Together with the helper script ‘make_mp4s’ VCat.php can be used to implement the cross-modal priming paradigm by presenting a brief image timed to a sound file as a single downloaded movie, thus controlling the interstimulus interval between the audio and visual components of a cross-modal priming paradigm.

## Requirements

You must be able to host your experiment on a web server that can serve *.php files, and audio and video files. Helper scripts in this package are provided for creating or modifying stimuli and assembling stimulus list files and are written in perl and python, and use applications for manipulation of media files including *sox, lame, imagemagick, and ffmpeg*.

## Running Subjects

To collect data from subjects, all you have to do is give them a link to your study. For example, the last of the example links listed above (Video Categorization) is:

```
https://linguistics.berkeley.edu/~kjohnson/exp/VCat.php?list=1
```

This link points to a server (https://linguistics.berkeley.edu), and a page on that server (~kjohnson/exp/VCat.php), and passes a variable to the page (?list=1). You can give this link to students in a class, or send it to a mailing list of volunteers, or use the link with Amazon Mechanical Turk or Prolific. The experiment provides a unique identifier to the participant on completion of the task which can be used as proof of participation for purposes of providing payment. This is how the experiment is run on MTurk, if you are using that platform to recruit and pay participants. The link for your experiment changes in one small way for use on Prolific:

```
https://linguistics.berkeley.edu/~kjohnson/exp/VCat.php?list=1&prID={{%PROL
```

uses the additional variable (&prID={{%PROLIFIC_PID%}}) to fill our variable *prID* with the participant's prolific ID, which is provided by the prolific platform when linked from prolific.co. We annonymize this ID by just taking its last four characters. It isn't strictly necessary to collect the prolific ID, but having even a portion of it can provide a fail-safe to be sure that participants get paid. The prID also distinguishes participants who joined the study from Prolific from those who did not (who have prID=0). As this implies, the same study can be run simultaneously on various platforms.

**Audiocheck.php.** HeadphoneCheck from the Josh McDermott Lab at MIT is used to make sure that listeners have working audio, and that they are wearing headphones. To use this, point participants to audiocheck.php with variables 'list' and 'next'. This will send the listener to your experiment after they pass the headphone check. (Add the Prolific variable &prID={{%PROLIFIC_PID%}} to collect the Prolific participant ID.) In the example below the participant will do the audiocheck and then move on to the experiment in AX.php. Note: audiocheck.php needs to be in the same server directory as AX.php.

```
https://myserver.edu/~mydirectory/audiocheck.php?list=1&next=AX.php
```

An example experiment. An experiment is implemented as an interlinked set of pages and supporting data files. Information is passed from one page to the next through hidden forms and URL variables. To make your own experiment you can take one of the example paradigms in the package and edit the files to suit your needs.

For example, the flow of the example Audio paired comparison experiment is:

```
AX.php  →  AX_trials.php → Questionnaire.php → finished.php
```

**AX.php** Introduction, instructions, consent form. User must provide a list number: AX.php?list=1

**AX_trials.php** - This is the main trial running page. The page assigns a random subject ID, presents the list of trials, optionally in random order, calls process.php after each trial, which writes data into *AX_Data.csv*.

**process.php** is called in the background by *AX_trials.php* to put data in *AX_Data.csv*. Data are written after every trial.

**Questionnaire.php** - A short reusable/modifiable questionnaire. The page passes subject ID, prID, and *.inc filename to *finished.php*.

**finished.php** - Writes questionnaire data to the destination file specified in the parameters file - e.g. *AX_Subjects.csv*, This pages also gives the subject their completion code number, or links to the Prolific completion page.

**AX_params.inc** - This set of parameters is included in several php files (*AX_trials.php, finished.php, and process.php*) and specifies type of trials, where to find the trial list, and where to store data.

**AX_Data.csv** is a writable csv file to store trial-by-trial data (trials from subjects working at the same time will be interleaved). You can use any name you want, just specify it in the *params.inc* file.

**AX_Subjects.csv** is a writable csv file for subject questionnaire data. Again, any name you want. Add a data header by hand.

**js/AX_list1.js** - This is a file that lists the stimuli and response options to be presented. The script *make_list.prl* can make this file from a csv file of trials.

**media** The package presents .wav, or .mp3 audio files, .mp4 movie files, or .jpg, .png image files to participants. In audio experiments both .wav and .mp3 versions of the audio must be available. The script *make_mp3s* uses 'sox' and 'lame' to prepare .wav and .mp3 files for the experiment (by default making them 16kHz, mono and amplitude normalized). The script *make_mp4s* uses ffmpeg to make movie stimuli, and the script *resize.py* uses imagemagik to prepare images for presentation.

## Multiple blocks of trials

If you want to have trials broken into separate blocks, then you can do something like:

```
AX.php → AX_trials1.php → AX_rest.php →
    AX_trials2.php → Questionnaire.php → finished.php
```

*AX_rest.php* in this experiment flow might have instructions for the second block of trials, and you could specify different params.inc files for the different blocks (for instance *block1_params.inc* and *block2_params.inc*).

## The parameter file

A parameter file is used to specify five key aspects of an experiment. The parameter file is consulted by the experiment *.php files when needed, using the php command 'include()'. The name of the parameter file is constructed in php code from a template *[x]_params.inc* and a the '[x]' portion of this name is specified in the PHP header of AX_trials.php

1. **$datafile** -- name of file where trial-by-trial data will be written <any name you wish>.

2. **$subjects** -- name of file where subject questionnaire data will be written <any name you wish.>

3. **$template** -- A template for the name of the .js file that specifies a list of trials. e.g. AX_list##.js where ## in the template is replaced by a list number (*AX_list1.js*).

4. **$resp_type** -- what type of response to expect. One of:

    - **2AFC**, two alternative forced choice
    - **2AFC_labels**, two alternative forced choice with dynamic labels
    - **rating**, Likert rating (a number from 1 to 7)
    - **text**, type into a text box

5. **$stim_type** -- what type of stimulus will be presented. One of:

- **audio1**, one audio file
- **audio2**, two audio files with an interstimulus interval (isi)
- **image**, an image file
- **video**, a movie file
- **text**, text in a box on the screen

If you want a second list of trials for a second block of trials you could define two file list templates (e.g. $template1 and $template2) and instruct the second block to use the second template to find a different list of trials.

## Response types

The javascript routines in js/audexp.js recognize four kinds of possible responses that you could ask from participants.

If **$resp_type is 2AFC** then the routines will expect listeners to respond by pressing either the “z” or “m” key on each trial. It is expected and required that the _trials.php page which presents trials to have an html object which will be used to convey warnings such as “you must press ‘m’ or ‘z’” if the participant presses some other key, or “that response was too fast” if the response is faster than some experimenter-specified threshold, or “that response was incorrect” if the stimulus list includes an array that specifies which button is correct for each trial. In addition to having a span into which warnings can be presented, the _trials.php page should have some reminder about what the buttons mean.

The **$resp_type 2AFC_labels** is used to potentially present different labels on each trial. Unlike the 2AFC $resp*type in which the response labels are hard coded on the *_trials.php page, the 2AFC*labels type expects and requires that the *_trials.php page has html objects and , and that the trial list has arrays option1[] and option2[] that contain the labels to be presented on each trial. One nice feature of using the 2AFC_labels response type is that the response labels are written into the output datafile.

The **rating $resp_type** is used to give a Likert scale from 1 to 7.

In the **text $resp_type** the key listening code in audexp.js will assume that there is a text input element with the name and id “response” on the page, and that this form element will collect the text being typed by the participant, and the key listener will record a response time for the first key press that the participant types, and will monitor for the key to submit the answer and write data into the output data file.

## Stimulus types

The **audio1 $stim_type** is used when one audio file is presented, and **audio2** is used when two sound files will be played with a specified interstimulus interval. The **video $stim_type** assumes that the trials page as a canvas object, and the video will be played in this object. The **image $stim_type** also assumes that a canvas object is available on the trials page for the image.

## Stimulus list

The php file that plays trials to subjects (in the example this is *AX_trials.php*) will look for a list of stimuli in the 'js' directory of the webpage. A stimulus list may be as simple as the specification of an array called 'file1' that lists a media file, like an image, sound file, or video file. In the example experiment, 'AX_list1.js' defines five arrays - 'file1, file2' hold file names of the two audio files to be presented in a trial; 'option1, option2' hold labels that will be presented on the screen for the left and right response buttons, and 'correct' is an array that holds correct answers. When the 'correct' array is defined feedback is given to subjects.

```
AX\_list1.js
file1[0]='sounds/type_0'; file2[0]='sounds/type_0'; option1[0]='same'; opti
file1[1]='sounds/type_20'; file2[1]='sounds/type_0'; option1[1]='same'; opt
file1[2]='sounds/type_40'; file2[2]='sounds/type_0'; option1[2]='same'; opt
file1[3]='sounds/type_60'; file2[3]='sounds/type_0'; option1[3]='same'; opt
file1[4]='sounds/type_80'; file2[4]='sounds/type_0'; option1[4]='same'; opt
```

The '.js' file was created from a spreadsheet (AX_list1.csv) using the perl script *make_list.prl*. Each row in the spreadsheet specifies one trial. The array names are given as column headers in the first line of the .csv. These array names are not flexible at all - the javascript routines assume that there will be arrays called 'file1' and 'file2' for the two audio files presented in the AX trials (note that audio files are listed with a directory name where they can be found, and without a filename extension - .wav and .mp3 will be added based on the capability of the user's browser). The javascript routines for a response type "2AFC_labels" also assumes that there will be arrays named 'option1' and 'option2' to use as button labels, and if there is an array called 'correct' listeners in a 2AFC task will automatically be given feedback about the accuracy of their responses.

```
AX_list1.csv
file1,file2,option1,option2,correct
sounds/type_0,sounds/type_0,same,different,same
sounds/type_20,sounds/type_0,same,different,different
sounds/type_40,sounds/type_0,same,different,different
sounds/type_60,sounds/type_0,same,different,different
sounds/type_80,sounds/type_0,same,different,different
```

So:

```
AX_list1.csv   →   make_list.prl   →   AX_list1.js
```

## AX_trials.php

The main .php file that runs the AX trials starts with a <?php … ?> block that includes the 'AX_params.inc' file to get the variables described above. The include file is needed by process.php and Questionnaire.php, as well as being needed here in AX_trials.php, so to avoid passing the name of the include file on the open web we pass the first part of the file name, and use the template '_params.inc' to complete the filename. The starting php block also defines $stimulus*list from a filename template variable $template, that is defined in the include file. The $stimulus*list variable is defined by replacing the '##' characters in the template with the contents of $list (which can be any characters, but for us is usually just a number). If the experiment is being run on Prolific, a shortened version of the subject's prolific ID is also passed. On other platforms the prID is 0.

```
<?php
    $list = $_GET["list"];  // determines which word list to use
    $prID = $_GET["prID"];

    $subj = uniqid($list);  // get a unique ID for this person
    $n = 'AX';  // this is part of the name of the include file

    $incfile = $n . '_params.inc';
    $success = include($incfile);
    if (!$success) {
             http_response_code(403);
             echo "Could not load include file.";
            exit();
    }
    $stimulus_list = str_replace("##",$list,$template);
?>
```

The portion of the php file refers to the library of javascript routines in js/audexp.js, and the list in the $stimulus_list.

```
<!DOCTYPE html>
<html>

<head>
<title>speech discrimination</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.
<script src="js/audexp.js"></script>
<script src=<?php echo $stimulus_list; ?>></script>
<style> td {text-align:center; width: 90px;} </style>
</head>
```

When the of the php file loads, some parameters are set by a load() function. “true” to randomize the order of the trials, ‘1500’ milliseconds of pause between trials, response times of less than ‘200’ milliseconds will generate a warning to the subject about a ‘too fast’ response, and response times greater than 4 seconds (4000 ms) will generate a ‘too slow’ warning. Finally, there will be a 500 millisecond pause between audio file1 and audio file2 in the AX pairs.

A ‘click here to start’ button is defined to play the first trial. (Most browsers need a user click on the page to enable audio on the page.) The javascript functions load() and play*first*trial() are defined in js/audexp.js.

```
<body onload="load(true,1500,200,4000,500);">

<center>
<h2>detect audio differences</h2>
<hr>

<p><button onclick="play_first_trial(stim_type='<?php echo $stim_type; ?>',
resp_type='<?php echo $resp_type; ?>', this)">Click here to start.</button>

<p>The differences can be very small, so listen carefully.<br />
Type either 'z' or 'm'</p>
```

Several aspects of the graphical interface are controlled dynamically by the js/audexp.js routines. For example the span is filled with the contents of the option1[] array for each trial. Similarly the ‘f1’ span is highlighted in yellow while the first file is playing, the ‘key’ span gives feedback to the subject showing what button they pressed, and the ‘warn’ span is used to present warnings to the subject. The names of these spans are not changeable - the audexp.js routines expect to find these particular spans in the .php file, based on the $stim*type and $resp*type given in the parameter file. For instance, if the $resp*type is '2AFC*labels' then there must be spans called 'response1' and 'response2', and the stimulus list file must define arrays 'option1' and 'option2' of labels to put into these spans.

```
<table>
<tr><td>z</td> <td></td> <td>m</td></tr>
<tr><td><span id="response1"></span></td><td></td><td><span id="response2">
</table>

<hr>
<span id="f1">first</span> &nbsp &nbsp
<span id="f2">second</span> <br \>
<span id="key">#</span></p>

<p><span id="warn"></span></p>
<p style="font-size:12px">now on: <span id="count">1</span>/    <span id="t
</center>
```

Finally, *AX_trials.php* defines two hidden forms. The first is filled and submitted after each response by the js/audexp.js routine *key_listener()* which monitors for key presses and processes them. Note that the first part (AX) of the name of the include file (*AX_params.inc*) is also passed to *process.php* in variable *$n* which reads the include file in order to get the name of the datafile into which the data will be written.

When all of the trials have been completed the *Questionnaire.php* is presented by submitting the 'continue_form'. The subject ID, list number, and param file identifier are passed as form parameters to the questionnaire. A javascript function automatically submits this "continue_form" after the last trial has been presented.

```
<form method="POST" id="dataform" action="process.php?n=<?php echo $n; ?>">
  <input type="hidden" name="subject" value=<?php echo $subj;?> />
 <input type="hidden" name="prID" value=<?php echo $prID;?> />
    <input type="hidden" name="list" value=<?php echo $list;?> />
  <input type="hidden" name="trial" />
  <input type="hidden" name="file1" />
  <input type="hidden" name="file2" />
  <input type="hidden" name="filedur" />
  <input type="hidden" name="loadtime" />
  <input type="hidden" name="mystatus" />
  <input type="hidden" name="response"/>
  <input type="hidden" name="rt" />
</form>

<form method="POST" id="continue_form" action="Questionnaire.php">
 <input type="hidden" name="subject" value=<?php echo $subj; ?> />
 <input type="hidden" name="prID" value=<?php echo $prID; ?> />
 <input type="hidden" name="list" value=<?php echo $list; ?> />
 <input type="hidden" name="n" value=<?php echo $n; ?> />
</form>

</body>
</html>
```

## PTrans_trials.php

The picture transcription task (*PTrans.php*) also has a ‘trials’ php file and the contrast between ‘PTrans_trials.php’ and ‘AX_trials.php’ may be instructive. The include file (‘PTrans_params.inc’) defines $stim*type as ‘image’, and $resp*type as ‘text’, so the trials php file must have some different elements.

The starting <?php … ?> block is the same as in the AX experiment except that we specify a different parameter file. With $n = ‘PTrans’ the include file will be ‘PTrans_params.inc’.

```
<?php
   $list = $_GET["list"];  // determines which word list to use -- a number
  $prID = $_GET["prID"];

   $subj = uniqid($list);  // get a unique ID for this person

   $n = 'PTrans';  // this is part of the name of the include file
   $incfile =  $n . '_params.inc';
   $success = include($incfile);
   if (!$success) {
          http_response_code(403);
          echo "Could not load include file.";
          exit();
   }
   $stimulus_list = str_replace("##",$list,$template);
?>
```

The html block is identical to the AX experiment. In the body block we give different instructions, and critically include a <canvas id="canvas"> object which the java script routines will fill with pictures from a list of image files in the 'file1' array which we defined in *PTrans_list1.js*.

```
<body onload="load(true,1500,200,4000);">

<center>
<h2>Enter a word or phrase for this picture</h2>
<hr>

<p><button onclick="play_first_trial(
    stim_type='<?php echo $stim_type; ?>',
    resp_type='<?php echo $resp_type; ?>', this)">
    Click here to start.</button></p>

<canvas id="canvas" width="576" height="384" /></canvas>
```

The data form that is passed to 'process.php' is also slightly different. It does not include information about the duration of a sound file, or on how long it took to load a sound file. Also, crucially, the 'response' input of the form is defined as a 'text' input and is given the id value of 'response'. This makes it so a response box appears for the subject to type answers into, and the response object can be identified and cleared between trials by the javascript keylistener() function when the subject presses the 'enter' key.

```
<form method="POST" id="dataform" action="process.php?n=<?php echo $n; ?>">
  <input type="hidden" name="subject" value=<?php echo $subj;?> />
  <input type="hidden" name="prID" value=<?php echo $prID;?> />
  <input type="hidden" name="list" value=<?php echo $list;?> />
  <input type="hidden" name="trial" />
  <input type="hidden" name="file1" />
  <input type="hidden" name="mystatus" />
  <input type="text" id="response" value="" name="response" />
  <input type="hidden" name="rt" />
</form>
```