

Putting Psychology Studies Online

Using jsPsych, JavaScript and PHP

Isabella Sewell

Department of Psychology, University of Waterloo

Britt Lab Group

March 1, 2021

jsPsych, JavaScript and PHP	3
Naming Files for the BrittLab Server	3
Information and Consent Letter	4
Task	5
Skeleton	5
Page Layout	6
Script	7
SONA	7
Detect Browser	8
Collecting and Saving Demographic Information	8
Instructions and Text Screens	10
Trials	10
Fixation	10
Break Screen	11
Trials with Fixation and Implementing the Break Screen	11
Thank You and Feedback Questions	11
Finished Text	11
Feedback Question as Text	12
Feedback Question as Slider	12
Thank You	13
Timeline	14
Saving Data	14
Local Save	16
Feedback Letter	17
Working with Terminal – MAC	18
How to access the BrittLab server	18
Using scp to Transfer Files Between Server and Computer	19
Computer → Terminal	19
Terminal → Computer	19
Creating the Study on SONA	20

jsPsych, JavaScript and PHP

The document will include code that utilizes jsPsych, JavaScript and PHP (with some HTML).

[jsPsych](#) is a JavaScript library that was built for running psychological experiments online (in a web browser). The library has plugins that provide the general framework for building a task. jsPsych has very helpful documentation for each of their [plugins](#), the [steps for running a simple experiment](#) and useful [youtube tutorials](#) facilitated by the creators of jsPsych.

Beyond jsPsych, coding directly in JavaScript is useful for creating parts of your task that may be novel or more specific than any of the plugins available. The primary tool I used for understanding JavaScript was [w3schools.com](#). They had numerous tutorial chapters, each with specific examples.

Lastly, although you can code a task using jsPsych and JavaScript, in order to have your data save onto the BrittLab server, you must implement pieces of PHP. PHP is a server scripting language and is also used (like JavaScript) for creating interactive web pages. [W3schools.com](#) also has tutorials for learning PHP syntax. For your purposes, you will be primarily copying and pasting the pieces of PHP code created by Dr. Anderson and modified by Julia, while also calling upon `write_data.php` and `selectBrittLab.php`.

Naming Files for the BrittLab Server

When running studies online, you will need to include the information and consent letter, task and feedback letter. These three files will be put into your protocols folder on the BrittLab server. The following naming convention is useful to ensure your file names are concurrent with what is called upon in the sample code.

File	What to Name your file	Example
Information and Consent Letter	protocolname_consent.php	globallocal_consent.php
Task	protocolname_task.php	globallocal_task.php
Feedback Letter	protocolname_feedback.php	globallocal_feedback.php

Generally, when I was creating my task, I coded each of these as an .html file so that I could test it on my computer's web browser. However, some pieces of my code that I attached below are now specific to PHP, so you will run into errors if you try to run it as a .html file. Therefore I would suggest Julia's approach in that she utilizes a software ([XAMPP](#)) that mimics a server so that she can troubleshoot .php files directly while also having the data save onto her own computer (not the BrittLab server).

Information and Consent Letter

The template for the information and consent letter was adapted from Bill Eickmeier's old beehive link on "Psych Web Form Templates" (<https://beehive.uwaterloo.ca/dept/webformtemplate/>). The consent checkbox links this file to the task file.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
<title>Information and Consent Form</title>
</head>
<BODY bgcolor="#ffffff">
<form action=<?="https://artsresearch.uwaterloo.ca/~brittlab/protocols/" . $_POST['protocol'] . "/" . $_POST['protocol'] .
  "_task.php"?> method="POST">
  <input type="hidden" id="sonaid" name="sonaid" value=<?php echo $_POST['sonaid']?> >
  <input type="hidden" id="protocol" name="protocol" value=<?php echo $_POST['protocol']?> >
  <input type="hidden" id="pwsecret" name="pwsecret" value=<?php echo $_POST['pwsecret']?> >
  <input type="hidden" id="frompage" name="frompage" value="globallocal_consent.php"
```

Here I wrote out my information and consent form using HTML syntax.

```
<label>I agree to take part in this study
<input type="checkbox" name="consent" value="Accept">
<br>
<label>I do not agree to take part in this study
<input type="checkbox" name="dont_consent" value="NotAccept">
<br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

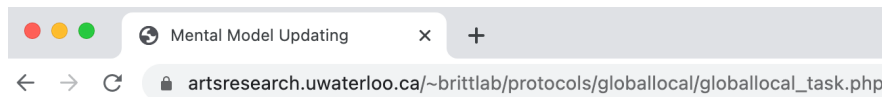
Task

Skeleton

Again, the general skeleton for the task was from Beehive's [Psych Web Form Templates](#). In the *head* section, ensure you have all of the appropriate jsPsych plugins that were used throughout the task. Also, ensure that the path to the plugin is the server's path, not the one for your personal computer. Julia has adapted a couple of plugins (notably, *jspsych-julia-html-form.js* which I used for all of my instruction screens).

An error I made when creating this section was having inconsistencies between jspsych and jsPsych. Some systems are case-sensitive when running the code. When referring to the path to the plugin use **jsPsych-6.2.0**. However, when referring to the plugin name itself use **jspsych**.

The *title* in the *head* refers to what participants will see in their tab.



```
<!DOCTYPE html>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
  <title>Mental Model Updating</title>
  <script src="/~brittlab/protocols/jsPsych-6.2.0/jspsych.js"></script>
  <script src="/~brittlab/protocols/jsPsych-6.2.0/plugins/jspsych-julia-html-keyboard- canvas.js"></script>
  <script src="/~brittlab/protocols/jsPsych-6.2.0/plugins/jspsych-html-keyboard-response.js"></script>
  <script src="/~brittlab/protocols/jsPsych-6.2.0/plugins/jspsych-julia-html-form.js"></script>
  <script src="/~brittlab/protocols/jsPsych-6.2.0/plugins/jspsych-survey-text.js"></script>
  <script src="/~brittlab/protocols/jsPsych-6.2.0/plugins/jspsych-survey-multi-choice.js"></script>
  <script src="/~brittlab/protocols/jsPsych-6.2.0/plugins/jspsych-html-slider-response.js"></script>
  <link rel="stylesheet" href="/~brittlab/protocols/jsPsych-6.2.0/css/jspsych.css">
```

```
<style>
</style>
</head>
```

This part in the *body* is in order to carry information over from SONA and the information and consent letter

```
<body>
  <input type="hidden" id="sonaid" value=<?=$_POST['sonaid']?> >
  <input type="hidden" id="protocol" value=<?=$_POST['protocol']?> >
  <input type="hidden" id="secret" value=<?=$_POST['pwsecret']?> >
  <input type="hidden" id="consent" value=<?=$_POST['consent']?> >
</body>
<script>
</script>
</html>
```

Page Layout

If you would like to make the background black with white writing, here is a couple pieces of code to use. The font (*body*) colour is modified in the *style* section, while background colour is modified in the *script* section. By default, the task would have a white background with black writing.

```
<style>
  body {
    color: white;
  }
</style>
```

```
<script>
  document.body.style.backgroundColor = "black";
</script>
```

The rest of the code for the task would be put in the *script* section of the .php file

Below is code that gathers data from the information and consent form.

The following if statement ensures that if a participant **did not** click “I agree to take part in this study” in the consent form, the study would be aborted and they would be redirected back to the SONA home page.

The following piece of code is in order to save the participants data from SONA and the information and consent letter. [jsPsych.data.addProperties](#) ensures that this piece of data appears on every trial (row in a .csv file) and as it's own separate column. This was important in order to ensure the participant's SONA ID is associated with each trial so that when you combine multiple datasets, you can still know which trial belongs to which participant. You can also do this for demographic questions (see next page).

[illegible]

Detect Browser

This is a short piece of code that I implemented to ensure that participants were only using Google Chrome as a browser (it was the only browser compatible for my study). The problem was that the code could run on Safari, but the data wouldn't save, so I would suggest checking other browsers for compatibility before running participants. I adopted the code from MDN Web Docs [Browser detection using the user agent](#).

```
var detectBrowser = navigator.userAgent;
if (detectBrowser.indexOf('Chrome') !== -1)
{
}
else
{
  alert("Study Aborted Due to Not Using Chrome")
  window.location.href = "https://uwaterloo.sona-systems.com/";
}
```

Collecting and Saving Demographic Information

For demographic information, again I would suggest implementing `jsPsych.data.addProperties` in order to save variables as a separate column for each trial (all rows will be filled). By default, the jsPsych plugins used for demographic or feedback questions will only save the variables as a string (column B).

A	B	J	K	L	M	N
rt	responses	subject	age	sex	key_press	TrialNumber
2381.07	{"Age":"21"}	348898	21	Female		
4218.985	{"Sex":"Female"}	348898	21	Female		
1268.06		348898	21	Female	40	0
868.06		348898	21	Female	38	1
251.28		348898	21	Female	38	2

The difference now for `jsPsych.data.addProperties` (compared to for protocol, consent and subject) is that it is now specific for each variable. Therefore, you must write `"on_finish: function(data){jsPsych.data.addProperties... etc.}"` (see next page).

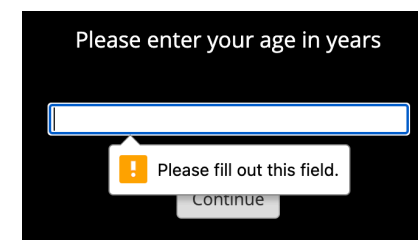
Survey Text

I used the [jspsych-survey-text plugin](#) for demographics like age where participants type out the answer. Jspsych-survey-text also allows for the question to be required, so that participants need to answer the question. If they try to skip it, they would be shown the alert “please fill out this field”. Lastly, both survey-text and survey-multi-choice (see below) allow for multiple questions on a single page.

```
var age = {  
  type: 'survey-text',  
  preamble: '<p> Please enter your age in years </p>',  
  questions: [  
    {prompt: " ", name: 'Age', required: true}  
  ],  
  on_finish: function(data) {  
    jsPsych.data.addProperties({  
      age: JSON.parse(data.responses).Age,  
    })  
  }  
};
```



A screenshot of a black rectangular interface. At the top, the text "Please enter your age in years" is displayed in white. Below the text is a white rectangular text input field with a blue border. At the bottom center is a white rectangular button with the word "Continue" in black text.



A screenshot of the same black rectangular interface as the previous one, but with an error message. The text "Please enter your age in years" is at the top. Below it is the white text input field. A white speech bubble with a red exclamation mark icon and the text "Please fill out this field." is pointing to the input field. At the bottom center is a white rectangular button with the word "Continue" in black text.

Multiple Choice

The [jspsych-survey-multi-choice plugin](#) also has a “required” parameter where if a participant tries to skip the question, they would be shown an alert that says “Please select one of these options”. Something useful for this plugin, is that it is easy to do multiple pages with different scales or even multiple questions on a page with different scales. This can be done by adding multiple prompts and specifying which option variable it should follow. The jsPsych page for this plugin shows a great example of this.

```
var page_sex_options = ["Male", "Female", "Prefer not to say"];  
  
var sex = {  
  type: 'survey-multi-choice',  
  questions: [  

```

```

    {prompt: "What is your sex?", name: 'Sex', options: page_sex_options, required:true},
  ],
  on_finish: function(data) {
    jsPsych.data.addProperties({
      sex: JSON.parse(data.responses).Sex,
    })
  }
};

```

Instructions and Text Screens

I used Julia's plugin *jspsych-julia-html-form.js* for almost all of my instruction pages. It shows text and a next button.

```

var start = {
  type: "julia-html-form",
  stimulus: "Press the next button to start the experiment!",
};

```

Trials

Fixation

For the fixation screen, I had a grey cross in the centre that was set to disappear automatically (i.e., `jsPsych.NO_KEYS`). I also set it so that the fixation screen appeared for varying durations (750 ms, 1000 ms, 1250 ms or 1500 ms). This was modelled from jsPsych's page on [Using Functions to Generate Parameters](#)

```

var fixation = {
  type: 'html-keyboard-response',
  stimulus: '<div style="font-size:60px; color:grey;">+</div>',
  choices: jsPsych.NO_KEYS,
  trial_duration: function(){
    return jsPsych.randomization.sampleWithoutReplacement([750,1000,1250,1500], 1)[0];
  }
};

```

Break Screen

I also implemented a break screen where participants simply had to press a key to continue.

```
var break_screen = {  
  type: 'html-keyboard-response',  
  stimulus: "This is a break. Press any key to continue"  
};
```

Trials with Fixation and Implementing the Break Screen

I didn't include the lines of code for my task, but if you want to see the steps I implemented please refer to *globallocal_task.php* in the protocols folder *globallocal*. The pieces of code combined my *allTrials* variable and the fixation variable. Julia helped set this up so that fixation screens followed each trial (i.e., *allTrials_withFixation*). We also implemented a break screen to occur when the current trial (i.e., *curTrialIndex*) was divisible by 250 (i.e., on the 250 trial), but not when the current trial number was 0.

```
allTrials_withFixation = []  
for (curTrialIndex = 0; curTrialIndex < allTrials.length; curTrialIndex++) {  
  allTrials_withFixation.push(fixation)  
  allTrials_withFixation.push(allTrials[curTrialIndex])  
  if (curTrialIndex % 250 == 0 && curTrialIndex != 0) {  
    allTrials_withFixation.push(break_screen)  
  }  
}
```

Thank You and Feedback Questions

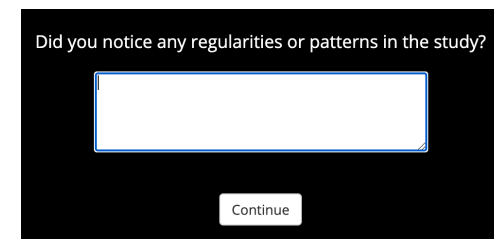
Finished Text

```
var finished = {  
  type: "julia-html-form",  
  stimulus: "<p>You've finished the experiment!</p><br>" +  
    "<p>Please press the next button to complete some feedback questions.</p>",  
};
```

Feedback Question as Text

Here, I used the same *survey-text* plugin as I did for the age demographic question, but the difference is I made the text box bigger (i.e., rows:5). This way participants can read their entire answer without it being cut-off as they type.

```
var feedback_text = {
  type: 'survey-text',
  questions: [
    {prompt: "Did you notice any regularities or patterns in the study?", name: 'Initial Observations', rows: 5, required: true},
  ],
  TrialType: "Survey",
};
```

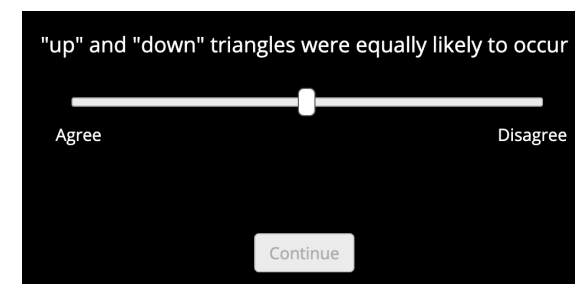


Feedback Question as Slider

The [jspsych-html-slider-response](#) plugin is a useful tool for replacing a Likert scale if you want participants to be more specific. Afterwards, I manually converted the numbers from 0-100 into Likert-like labels so that I could visualize it better. Similar to other plugins that had *required:true*, *html-slider-response* utilizes *require_movement:true*. This way, participants need to move the scale or at least click on the slider in order to go to the next question.

```
var feedback_instructions = {
  type: "julia-html-form",
  stimulus: "<p>For the following questions, <br><br>please move the slider to indicate how much you agree with each statement</p>",
};
```

```
var feedback_slider = {
  type: 'html-slider-response',
  stimulus: '<p>"up" and "down" triangles were equally likely to occur</p>',
  labels: ['Agree', 'Disagree'],
  require_movement: true,
  slider_width: 400,
```



```

TrialType: "Survey",
on_finish: function(data) {
  if(data.response <= 20){
    var scale = "agree";
  };
  if(data.response >20 && data.response < 40){
    var scale = "somewhat agree";
  };
  if(data.response >=40 && data.response < 60){
    var scale = "neutral";
  };
  if(data.response >=60 && data.response <= 80){
    var scale = "somewhat disagree";
  };
  if(data.response > 80){
    var scale = "disagree";
  };
  data.scale = scale
},
};

```

Thank You

```

var thank_you = {
  type: "julia-html-form",
  stimulus: "<p>Thank you very much for participating!</p><br>" +
    "<p> Press the next button to read the feedback letter</p>",
};

```

Timeline

This is an example of what my timeline would look like if I included all of the variables aforementioned.

```
timeline = [].concat(  
  age,  
  sex,  
  start,  
  allTrials_withFixation,  
  finished,  
  feedback_text,  
  feedback_instructions,  
  feedback_slider,  
  up_down_equal,  
  thank_you,  
)
```

Saving Data

I used a different saveData function for my code, but Julia has now created a new function to resolve an error I was running into. Before, data for my experiment would not save if the overall trial number exceeded 300 trials. Julia described it as the data was still saving when the participant proceeded to the feedback letter, so the data couldn't save properly. I resolved the problem by [filtering jsPsych.data.get\(\)](#) to include only trials (not fixation screens) in the .csv file. The following piece of code is Julia's version which should resolve this issue (this can be found in the protocols folder *sampleprotocol* as *sampleprotocol_task.php*).

```
function saveData (protocol, name, data) {  
  return new Promise((resolve,reject) => {  
    var xhr = new XMLHttpRequest();  
    xhr.onreadystatechange = function() {  
      if (this.readyState == 4 && this.status == 200) {  
        resolve('file successfully saved')  
      }  
    }  
  });  
};
```

```

xhr.open('POST', "../write_data.php");
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.send(JSON.stringify({protocol: protocol, filename: name, filedata: data}));
})
}

```

Lastly, these lines of code call the saveData function. Again, Julia has fixed the error that I ran into when saving data, so this looks a little different from my original code. Here, I've called the saveData function twice so that data saves either when the participant finishes (i.e., on_finish) or when they close the tab or window (i.e., on_close).

```

jsPsych.init({
  timeline: timeline,
  on_finish: function() {
    saveData(jsprotocol, jssonaId + "_" + jsprotocol, jsPsych.data.get().csv())
    .then((x) => {
      console.log(x)
      let windowloc = jsprotocol + "_feedback.php?sonaId=" + jssonaId
      window.location.href = windowloc
    })
  },
  on_close: function() {
    saveData(jsprotocol, jssonaId + "_" + jsprotocol, jsPsych.data.get().csv())
    .then((x) => {
      console.log(x)
      let windowloc = jsprotocol + "_feedback.php?sonaId=" + jssonaId
      window.location.href = windowloc
    })
  }
});

```

Local Save

If you are looking to troubleshoot your task as an .html file, I would suggest utilizing `jsPsych.data.get().localSave('csv', idvariable.concat('_protocolname.csv'))`. This will save the data to your personal computer. If you're running it as an .html file, you won't have access to `jsPsych.value` (from SONA). So you can create a variable `subject_id` at the beginning of the task. Again, **this is just for troubleshooting your task, not for running actual participants.**

I mentioned this at the beginning of the document, but instead of testing it as a .html file, Julia uses a software ([XAMPP](#)) that mimics a server so that she can troubleshoot .php files directly while also having the data save onto her own computer (not the BrittLab server). This way, Julia can utilize the `saveData` function (mentioned earlier), rather than `localSave` (shown below).

```
var subject_id = jsPsych.randomization.randomID(10);

jsPsych.init(
{
  timeline: timeline ,
    on_finish: function() {
      jsPsych.data.get().localSave('csv', subject_id.concat('_globallocal.csv'))
    },
    on_close: function() {
      jsPsych.data.get().localSave('csv', subject_id.concat('_globallocal.csv'))
    }
  },
);
```


Feedback Letter

This is modelled similarly to the information and consent letter.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
<title>Feedback Form</title>
</head>
<BODY bgcolor="#ffffff">
  <form action="https://uwaterloo.sona-
systems.com/webstudy_credit.aspx?experiment_id=5060&credit_token=e4847f42ee0e4bbab390b38c76ce41bd&survey_code=" +
<?=$_GET['sonaid']?>; method="POST" >
  <input type="hidden" id="feedback" name="feedback" value=<?php echo $_POST['feedback']?> >
```

Here, I wrote out my feedback text using HTML syntax.

```
<a href='https://uwaterloo.sona-
systems.com/webstudy_credit.aspx?experiment_id=5060&credit_token=e4847f42ee0e4bbab390b38c76ce41bd&survey_code=<?=$_GET['sonaid']?>'>Return to Sona for Credit</a>

</form>
</body>
</html>
```

Working with Terminal – MAC

How to access the BrittLab server

1. Open Terminal via the applications' utilities folder or doing a Spotlight Search via command space and typing in Terminal.
2. Connect to the University of Waterloo VPN via Cisco AnyConnect Secure Mobile Client. If you've never connected before, connect to **cn-vpn-uwaterloo.ca**
 - Username is your WATIAM username
 - Password is your WATIAM password
 - Second password is via Duo Mobile (at least for me), where you open the app and under University of Waterloo is shows a 6-digit password.
3. Once you've connected to the VPN, in your terminal type in the following:
`ssh brittlab@artsresearch.uwaterloo.ca`
`ls`
 - If you want to see the contents of where you're located in the server
`cd public_html/protocols`
`ls`
 - Here is where all of the protocols live
 - *selectBrittLab.php* and *write_data.php* will be used in all of the protocols (ensures participants are redirected to your specific study and that the data saves onto the server, respectively)`cd *protocolname*`
 - Each protocols folder should have:
 - Data (folder)
 - Img (if necessary)
 - protocolname_task.php
 - protocolname_consent.php
 - protocolname_feedback.php

Using scp to Transfer Files Between Server and Computer

Ensure that this is occurring in a new terminal window separate where you did `ssh brittlab@artsresearch.uwaterloo.ca`

Computer → Terminal

E.g., uploading your study onto the server.

Refer to the following naming convention.

```
scp *space* file path/file name *space* brittlab@129.97.222.16:/home/brittlab/public_html/protocols/*your  
protocols folder*/file name*
```

Here is an example of a scp of my task that is copied from my desktop and pasting onto my globallocal protocols folder.

```
scp /Users/Bella/desktop/globallocal_task.php  
brittlab@129.97.222.16:/home/brittlab/public_html/protocols/globallocal/globallocal_task.php
```

Terminal → Computer

E.g., downloading a participant's dataset onto your computer

The difference here is now you are copying from the server and pasting onto your computer. So, the naming convention is reversed.

```
scp brittlab@129.97.222.16:/home/brittlab/public_html/protocols/globallocal/data/348898_globallocal.csv  
/Users/bella/desktop/348898_globallocal.csv
```

Creating the Study on SONA

When creating a new study or changing study information on SONA, copy and paste the following under study URL:

https://artsresearch.uwaterloo.ca/~brittlab/protocols/selectBrittLab.php?id=%SURVEY_CODE%

Ensure that you add **?id=%SURVEY_CODE%**. %SURVEY_CODE% will be replaced by the participant's SONA ID.

Also, when clicking on “*Detailed Help*” (under *My Studies* → *Study Information*), it explains that adding **?id=%SURVEY_CODE%** also ensures that participants get granted the credit automatically after they complete the study.

Website

[View Study Website](#)

[Sample Link with Embedded ID Code](#)

Completion URLs:

<https://uwaterloo.sona-systems.com/we>

(client-side)

<https://uwaterloo.sona-systems.com/ser>

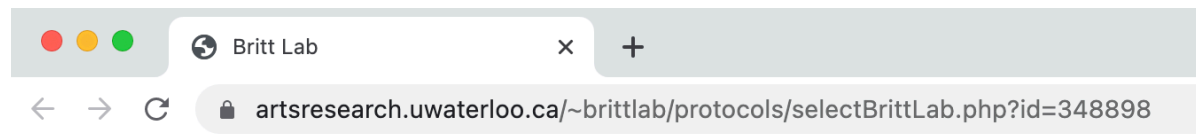
(server-side)

[Instructions](#)

You can also configure it so that participants receive credit in the system immediately after finishing the survey.

[Detailed Help](#)

In the example URLs above, replace the XXXX with the identifier passed with SURVEY_CODE to your external webstudy. If your external webstudy supports this functionality, then this will allow participants to be marked as having completed the study in the system as soon as they complete it at the external URL.



When I created the study, I also make sure to write this under *Preparation*.

IMPORTANT: Please complete the study in Google Chrome and use password globallocal

When participants follow the study URL, they will be redirected to the BrittLab artsresearch home page. Their SONA ID will be automatically filled in, but they will need a password (which is the name of your protocols folder on the BrittLab server). This piece of HTML syntax makes it more salient to participants that they need a password to access the study. It's also a useful place to mention if they need to use a specific browser

	Please note that a study sign-up is a firm commitment to the researcher running the study so select time slots carefully and if you cannot attend the lab study session at the specified time [do the online survey before the specified deadline], please cancel the Sign-Up or contact the researcher in advance. Failure to appear for too many studies without providing adequate prior notice to the researcher may result in denial of access to further studies. Details are provided on the frequently asked questions (FAQs) page on your SONA account.
Preparation	IMPORTANT: Please complete the study in Google Chrome and use password globallocal
Eligibility Requirements	Eligible Courses: Psych 101; 207, 211, 218, 226R, 230, 231, 238, 253, 253R; 257, 261 081, 082; 291, 292, 318, 334R 041, 042; 339, 041; 361, 363, 372, 390 044, 391, 444R, 453, 455, 461, 492; Geron 218; Hlth 218; Ls 230
Website	View Study Website



Hello and welcome to the [BrittLab](#).

Thank you for your time and for selecting one of our studies. To direct you to the correct study please enter the appropriate password below and click the *submit* button.

Sona ID:

Password: