Web experiments - Hands' on session NOW 2023

Romain Ligneul

Outline

Warning

If you did not do it already, perform the mandatory preparation steps: https://github.com/robustcircuit/now-workshop

1. Overview of the example code
2. Basic modifications to the code
3. Add a short survey at the end of the experiment
4. Modify the structure of the experiment
5. Serve the experiment online
6. Retrieve participant data

Overview of the experiment code

```
<!DOCTYPE html>
<html>
  <head>
    <script src="js/jspsych/dist/jspsych.js"></script>
    <script src="js/jspsych/dist/plugin-preload.js"></script>
    <script src="js/jspsych/dist/plugin-image-button-response.js"></script>
    <script src="js/jspsych/dist/plugin-image-keyboard-response.js"></script>
    <script src="js/jspsych/dist/plugin-html-button-response.js"></script>
    <script src="js/jspsych/dist/plugin-html-keyboard-response.js"></script>
    <link rel="stylesheet" href="css/jspsych.css" />
    <link href="img/favicon.ico" rel="icon" />
  </head>

  <body></body>

  <script>
    // retrieve information

    // name the current session
    var basename = "expNOWdata";

    // get timestamp of initial loading of the script
    var currDate = new Date();

    ///////////////////////////////////////////////////////
    // Define experiment parameters
    ///////////////////////////////////////////////////////
    // We store experiment parameters in a dedicated object
    var expdef = {};

    // define timing parameters in milliseconds (jsPsych default unit)
    expdef["choiceMaxDuration"] = 3000;
    expdef["feedbackDuration"] = 500;
    expdef["fixationDuration"] = 500;

    // define default repetition of images within blocks
    expdef["defaultRepetition"] = 1;

    // define the message displayed at the beginning of the experiment
    expdef["experimentIntroMsg"] =
      "<p>In this short experiment, you will have to learn to associate " +
      "specific images with specific response buttons, by trial and error.<br>" +
      "The experiment is divived in blocks that correspond to different categories of images
    expdef["experimentIntroButton"] = ["Click here to start the experiment"];

    // define the message displayed at the end of the experiment
    expdef["experimentEndMsg"] = "<p>Thanks for participating</p>";
    expdef["experimentEndButton"] = ["Click here to save the data"];
```

```
// define the message displayed at the beginning of each block
expdef["blockIntroMsg"] =
  "You are going to start a new block with images of ";
expdef["blockIntroButton"] = ["Click here to start the block"];

// define the message displayed below images
expdef["choiceMsg"] =
  "<p><b>Click</b> on the correct response button for this image</p>";

// define images and messages used for feedback
expdef["feedbackMsg"] = [
  "<p>Missed! <br>Respond faster next time</p>",
  "<p>Incorrect</p>",
  "<p>Correct</p>",
];

// define images paths for feedback
expdef["feedbackImg"] = [
  "img/missedFb.png",
  "img/thumbsDown.png",
  "img/thumbsUp.png",
];

// define the image names that will be used in the experiment
var imageNames = {};
imageNames["fruits"] = [
  "banana",
  "blueberry",
  "grapefruit",
  "kiwi",
  "pineapple",
  "raspberry",
];
imageNames["vegetables"] = [
  "beet",
  "brussels_sprouts",
  "carrot",
  "eggplant",
  "lettuce",
  "pumpkin",
];
```

```
//////////////////////////////////////////////////////
// Init jsPsych
//////////////////////////////////////////////////////

var jsPsych = initJsPsych({
  on_finish: function (done) {
    document.body.style.cursor = "pointer";
    var interaction_data = jsPsych.data.getInteractionData();
    interaction_data.trial_tag = "interaction";
    jsPsych.data.get().push(interaction_data);
    jsPsych.data.get().push({
      completedTask: "RLWM_expNOW",
      basename: basename,
      starting_time: currDate,
    });
    jsPsych.data.get().localSave("csv", basename + ".csv");
    display_data();
    jsPsych.data.get()
    $.ajax({
      type: "POST",
      url: "/save-file",
      data: jsPsych.data.get().json(),
      contentType: "application/json",
    });
  },
});

//////////////////////////////////////////////////////
// Generate trial list
//////////////////////////////////////////////////////

// block definition
var includedCategories = ["vegetables", "fruits"];
var setSizes = [1, 2];
var mapSR = [[0], [0, 1]];
var blockTrials = [expdef.defaultRepetition, expdef.defaultRepetition];

// prepare block shuffling
var blockIdx = [];
for (var b = 0; b <= includedCategories.length - 1; b++) {
  blockIdx.push(b);
}
```

```
blockIdx = this.jsPsych.randomization.shuffle(blockIdx);

// loop over blocks, images and repetitions
trialStructure = [];
for (var b = 0; b <= includedCategories.length - 1; b++) {
  // shuffle images (with respect to response mappings mapSR)
  blockImgs = this.jsPsych.randomization.shuffle(
    imageNames[includedCategories[blockIdx[b]]]
  );
  blockImgs = blockImgs.slice(0, setSizes[blockIdx[b]]);
  subStructure = [];
  // loop over images in the block
  for (var i = 0; i <= blockImgs.length - 1; i++) {
    // loop over repetition within blocks
    for (var r = 0; r <= blockTrials[blockIdx[b]] - 1; r++) {
      var trialspecs = {
        imgCategory: includedCategories[blockIdx[b]],
        imgId: blockImgs[i],
        repetitionCount: r,
        correctResponse: mapSR[blockIdx[b]][i],
        block: b,
        setSize: setSizes[blockIdx[b]],
      };
      subStructure.push(trialspecs);
    }
  }
  // randomize trials within a block
  subStructure = this.jsPsych.randomization.shuffle(subStructure);
  trialStructure.push(subStructure);
}
//console.log(trialStructure)

//////////////////////////////////////////////////////
// Generate trial objects
//////////////////////////////////////////////////////

var experimentIntro = {
  type: jsPsychHtmlButtonResponse,
  stimulus: function () {
    return expdef.experimentIntroMsg;
  },
  choices: expdef.experimentIntroButton,
  response_ends_trial: true,
};

var blockIntro = {
  type: jsPsychHtmlButtonResponse,
  stimulus: function () {
    return expdef.blockIntroMsg + includedCategories[blockIdx[blockNum]];
  },
  choices: expdef.blockIntroButton,
  response_ends_trial: true,
};

var fixationTrial = {
  type: jsPsychHtmlKeyboardResponse,
  stimulus: "<p style='font-size: 3rem;'>+</p>",
  trial_duration: expdef["fixationDuration"],
  response_ends_trial: false,
};

var choiceTrial = {
  type: jsPsychImageButtonResponse,
  stimulus: function () {
    var imgPath =
      "./img/" +
      trialStructure[blockNum][trialNum].imgCategory +
      "/" +
      trialStructure[blockNum][trialNum].imgId +
      ".jpg";
    return imgPath;
  },
  trial_duration: expdef["choiceMaxDuration"],
  choices: ["Left", "Middle", "Right"],
  prompt: expdef["choiceMsg"],
  stimulus_width: 400,
  maintain_aspect_ratio: true,
  response_ends_trial: true,
  on_start: function () {},
  on_finish: function (data) {
    if (data.response === null) {
      data.accuracy = -1;
    } else if (
      data.response == trialStructure[blockNum][trialNum].correctResponse
    ) {
      data.accuracy = 1;
    } else {
      data.accuracy = 0;
    }
  },
};

var feedbackTrial = {
  type: jsPsychImageKeyboardResponse,
  stimulus: function () {
    var lastCorrect = jsPsych.data.get().last(1).values()[0].accuracy;
    console.log(expdef["feedbackImg"]);
    return expdef["feedbackImg"][lastCorrect + 1];
  },
  prompt: function () {
    var lastCorrect = jsPsych.data.get().last(1).values()[0].accuracy;
    return expdef["feedbackMsg"][lastCorrect + 1];
  },
  stimulus_width: 150,
  maintain_aspect_ratio: true,
  trial_duration: expdef["feedbackDuration"],
  response_ends_trial: false,
};

var experimentEnd = {
  type: jsPsychHtmlButtonResponse,
  stimulus: function () {
    return expdef.experimentEndMsg;
  },
  choices: expdef.experimentEndButton,
  response_ends_trial: true,
};

// function to display data at the end of the experiment
var display_data = function () {
  // set an HTML div
  const display_element = this.jsPsych.getDisplayElement();
  display_element.innerHTML = '<pre id="jspsych-data-display"></pre>';
  var data = jsPsych.data.get();
  var correctN = data.filter({ accuracy: 1 }).select("accuracy").count();
  var overallN = data.select("accuracy").count();
```

```
    var correctRT = data.filter({ accuracy: 1 }).select("rt").values;
    var incorrectRT = data.filter({ accuracy: 0 }).select("rt").values;
    var htmlStr =
      "<p>You gave " +
      ((100 * correctN) / overallN).toFixed(1) +
      "% of correct responses. </p><br><br>";
    htmlStr +=
      "<p>On average, it took you " +
      (correctRT / 1000).toFixed(3) +
      "s to give a correct response and ";
    htmlStr +=
      (incorrectRT / 1000).toFixed(3) +
      "s to give an incorrect response.</p>";
    document.getElementById("jspsych-data-display").innerHTML = htmlStr;
  };

  ////////////////////////////////////////////////////////
  // Create nest timelines
  ////////////////////////////////////////////////////////

  // counters
  var trialGlobNum = 0;
  var blockNum = 0;
  var trialNum = 0;

  // trial timeline, loops until end of block
  var trial_timeline = {
    timeline: [fixationTrial, choiceTrial, feedbackTrial],
    on_timeline_start: function () {
      tBlock = 0;
    },
    loop_function: function () {
      console.log("block" + blockNum + " and trial" + trialNum);
      if (trialNum < trialStructure[blockNum].length - 1) {
        trialNum++;
        return true;
      } else {
        trialNum = 0;
        blockNum++;
        return false;
      }
    },
    };

    // block timeline, loops until the end of the experiment
    var block_timeline = {
      timeline: [blockIntro, trial_timeline],
      loop_function: function () {
        if (blockNum < trialStructure.length) {
          return true;
        } else {
          return false;
        }
      },
    };

    // main timeline, will only run once
    var main_timeline = [experimentIntro, block_timeline, experimentEnd];

    jsPsych.run(main_timeline);
  </script>
</html>
```

Overview of the server code

```
// get packages
var fs = require("fs");
var path = require("path");
var express = require("express");
var cors = require("cors");
require("dotenv").config();

// --- INSTANTIATE THE APP
var app = express();

// manage cors policy
app.use(cors());

app.use(express.static(__dirname + "/public/"));
```

```
// set views
app.set("views", path.join(__dirname, "/public/"));

// set routes
app.get("/expNOW", function (request, response) {
  response.render("experiment.html");
});

// set view engigne
app.engine("html", require("ejs").renderFile);
app.set("view engine", "html");

// START THE SERVER
app.listen(3000, function () {
  console.log(
    "Server running. To see the experiment that it is serving, visit the following address:"
  );
  console.log("http://localhost:%d/expNOW", 3000);
});
```

Let's launch the server

Open Visual Code or the IDE of your choice and type

```
cd PATH_TO_THE_GIT_REPO
cd server
node app
```

Then, open http://localhost:3000/expNOW in your browser.

You should see the experiment running (if you've followed the installation steps).

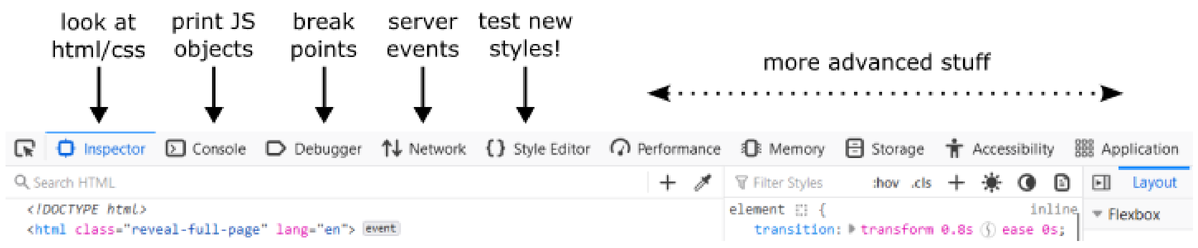
To stop and restart the server, just press *Ctrl+C* and type again `node app`.

The server should be restarted each time you make a change to the server code.

Key approaches to develop and debug web applications

Use the developer tools of your browser!

To see the console, you must toggle the developer tab of your browser (most often using key **F12** or **Fn+F12**).



```
<!DOCTYPE html>
<html class="reveal-full-page" lang="en">
```

```
element {
  transition: transform 0.8s ease 0s;
```

Figure 1: Developer tabs of Firefox

Select `console` and type `expdef` + Enter to see our experiment definitions object.

You can also add `console.log(something)` statements anywhere in your scripts to print dynamically the value (and structure) of Javacript variables, objects, etc.

Inspect your HTML and CSS

All developer tabs have an "inspector" tool that allows you to check and understand the *static* structure of any webpage, as well as the name of the elements you can manipulate. For example, the first screen of the experiment contains the following code:

```
<body style="margin: 0px; height: 100%; width: 100%;" tabindex="0" class=" jspsych-display-el
  <div class="jspsych-content-wrapper">
    <div id="jspsych-content" class="jspsych-content">
      <div id="jspsych-html-button-response-stimulus">
        <p>In this short experiment, you will have to learn to associate specific images wit
      </div>
      <div id="jspsych-html-button-response-btngroup">
        <div class="jspsych-html-button-response-button" style="display: inline-block; margi
          <button class="jspsych-btn">Click here to start the experiment</button>
        </div>
      </div>
    </div>
  </div>
</body>
```

The `id` and `class` attributes can be used to access each of these objets (divs, buttons, paragraph) and change their appeareance or behavior.

Navigate documentations efficiently, use Stackoverflow or ChatGPT

Web apps involve a **huge** amount of concepts, tools, objects, librairies, etc. You don't need to understand everything in depth and many frameworks can do the hard work for you (a bit like jsPsych does).

- Don't start from scratch
- Run very frequently your code as you modify it
- Make sure you're reading the adequate documentation (version++)
- Ask questions: what example code means, what bugs mean, what are the most likely causes and solutions
- ChatGPT is pretty good at generating bits of JS code from natural language queries.

Start as short as possible

Experiments can fail at any stage, but they tend to fail much more often at the beginning or at the end. So, always keep them as short as possible when you develop them.

So, the first thing we will do is to shorten ours even more. Try the following changes in 'experiment.html

```
//expdef["feedbackDuration"] = 1000;
//expdef["fixationDuration"] = 1000;
expdef["feedbackDuration"] = 500;
expdef["fixationDuration"] = 250;
```

```
//var setSizes = [3, 6];
//var mapSR = [[0, 1, 2], [0, 1, 2, 0, 1, 2]];
var setSizes = [1, 2];
var mapSR = [[0], [0, 1]];
```

This should reduce quite drastically the duration without removing any key ingredients.

Modify further the example experiment

Create subject- and/or session-specific URL

URL parameters are passed in the URL using the symbol “?” and several parameters can be chained using the symbol “&”.

```
// just copy paste this somewhere at the top
  const queryString = window.location.search;
  const urlParams = new URLSearchParams(queryString);
  var SUBJECT = 'unknown'
  if (urlParams.has('SUBJECT')) {SUBJECT = urlParams.get('SUBJECT'); }
  var SUFFIX = 'unknown'
  if (urlParams.has('SUFFIX')) {SUFFIX = '_' + urlParams.get('SUFFIX');}
```

Note

This approach will allow to share specific URLs to our participants and therefore keep track of who did what.

Use the URL parameters to customize files and messages

Now, we should use a web address looking like: `localhost:3000/expNOW?SUBJECT=someone&SUFFIX=v1`

We also need to modify the following line if we want to use this information to identify the output data!

```
    // name the current session
    var basename = 'expNOW_' + SUBJECT + SUFFIX
```

Now, try to use the `SUBJECT` variable to give a personalized “Hello” to your participants!

Note

Our first message to participants is defined by `expdef["experimentIntroMsg"]`

Display a full screen experiment

To do so, we need to use a plugin that jsPsych does not import by default, so we need to import it manually in our experiment, in the <head> section.

```
<script src="js/jspsych/dist/plugin-fullscreen.js"></script>
```

Find the actual file and have a look at the list of plugins provided by jsPsych. They might be useful for other experiments of yours!

Then, we just need to add the following jsPsych objects to our script, **after** the initialization of jsPscych.

```
  var enter_fullscreen = {
    type: jsPsychFullscreen,
    fullscreen_mode: true,
    message: function () {return "<p>The experiment will now go full screen.</p>"},
    button_label: function () {return "Click here to continue"},
  }
  var exit_fullscreen = {type: jsPsychFullscreen, fullscreen_mode: false}
```

Improve the visual appearance of our experiment

By default, jsPsych comes with very sober Cascading Style Sheets (CSS), but CSS can be easily updated to make things prettier. So, if we add the custom_jspsych.css stylesheet to **after** the default jspsych.css one, the experiment is already looking different.

```
  <link rel="stylesheet" href="css/jspsych.css">
  <link rel="stylesheet" href="css/custom_jspsych.css">
```

This is because `custom_jspsych.css` contains new styling information. Try to make additional changes to the visual appearance of the experiment by modifying or extending `custom_jspsych.css`.

```
/*to change all font colors of a given class */
.whateverclass {
  color: blue;
}
/*to change font weight of all several headings of a given class at once */
.whateverclass h2,h3,h4 {
  font-weight: 700;
}
/*to change the background of elements with a given ID */
#whateverID {
  background-color: rgb(2500, 0, 0);
}
```

Tip

Right click on any part of a web pages and click inspect to see its class or id. Do not forget semi-colons after each attribute!

Use jsPsych documentation to add a small survey at the end of the experiment

Open this page and adapt the demo code provided to add a survey.

- Use the *CDN-hosted Javascript file* or the Javascript file that is present in `js/jspsych/dist/survey-likert`
- Create just 2 or 3 questions that make sense in the context of this experiment

Add a third block using another image category

Study the code to find out how you can do this. Don't hesitate to make questions if you are stuck.

If you have access to Internet, you can be creative! Otherwise just use the third folder provided in `img`.

Just a hint to get you started, you might have to add code like this somewhere:

```
    imageNames['desserts'] = ["brownie","cake","dessert","donut","ice-cream_cone","milkshake'
```

Of course, there will be a few other changes to implement to make it work.

Display their data to your participants

```
// create a function to display data at the end of the experiment
var display_data = function () {
    const display_element = this.jsPsych.getDisplayElement(); // find main div of jsPsych
    display_element.innerHTML = '<pre id="jspsych-data-display"></pre>'; // set a new HTML di
    var data=jsPsych.data.get(); // get all jsPsych data
    // add the data to our new div
    document.getElementById("jspsych-data-display").textContent = JSON.stringify(data);
};
```

Then, you'll need to execute this function when jsPsych finishes to run:

```
  var jsPsych = initJsPsych({
    on_finish: function () {
      // ... add display_data to the end
      display_data();
    },
  });
```

Display their data to your participants (in a nice way)

Remove line `document.getElementById("jspsych-data-display").textContent = JSON.stringify(data);` and add instead:

```
  var correctN = data.filter({accuracy: 1}).select('accuracy').count();
  var overallN = data.select('accuracy').count();
  var correctRT=data.filter({accuracy: 1}).select('rt').mean();
  var incorrectRT=data.filter({accuracy: 0}).select('rt').mean();
  var htmlStr = '<p>You gave ' + (100*correctN/overallN).toFixed(1) + '% of correct response
  htmlStr += '<p>On average, it took you ' + (correctRT/1000).toFixed(3) + 's to give a corr
  htmlStr += (incorrectRT/1000).toFixed(3) + 's to give an incorrect response. '
  document.getElementById("jspsych-data-display").textContent = htmlStr;
```

Use Ngrok to serve your experiment(s) for piloting

- Download Ngrok for your OS (if you did not already)
- Sign up for a free account
- Check your Authtoken in the dashboard and copy-paste the line `ngrok config add-authtoken XXX` in your command line

- Open a new command line and type `ngrok http 3000`

Your experiment will be accessible from the address appearing just before `-> http://localhost:3000`

The data will still be downloaded on the computer of the pilot, rather than yours, so change the final message to ask the participant to see you their file by email.

Send participant data to your server

Add this to the head section

```
<script src="js/jquery/jquery-3.7.1.js"></script>
```

Add this at the end of the on_finish function of initjsPsych

```
$.ajax({
  type: "POST",
  url: "/save-file",
  data: jsPsych.data.get().json(),
  contentType: "application/json",
}).done(function () {console.log('uploaded')})
```

Add the following lines to your server `app.js` (after creating a logdata subfolder), before the `app.listen()` call.

```
var bodyparser = require("body-parser");
app.use(bodyparser.json({ limit: "50mb" }));
app.post("/save-file", function (request, response) {
  var datestr = new Date();
  datestr = String(datestr.toISOString()).replace(/:|\s+|_/g, '')
  var filename = String(request.body[request.body.length - 1].basename + "_" + datestr + ".js
  fs.writeFile(path.join(__dirname, "logdata/" + filename), JSON.stringify(request.body), (e
});
```

That's it, you have a functional web experiment that you can use for piloting!

Bonus - Adapt an existing experiment for your server

We will use an experiment found on the web, but you can try with your own pick.

1. Open a terminal and `cd` inside now-workshop repository folder.
2. Type `git clone https://github.com/jspsych/experiment-demos`
3. Choose one of the 4 demo experiments and adapt it to serve it from `experiment2.html`

Overall steps to follows:

1. Copy-paste the content of a demo folder into your `public` subfolder.
2. Create a route towards the new .html file in our server script.
3. Adapt the `on_finish` method contained in our `jsPsych.init` section to send the data to your server at the end of the experiment! You will need to: (**i**) make decision about whether you want to use URL parameters or not (and implement them or define the `basename` variable in another way), (**ii**) push the basename attribute in jsPsych.data in the same way as it is done in `experiment.html`

Bonus - Add a second experiment to your server

If you want to serve different experiments at different addresses, you will need to modify a bit the server (`app.js`).

ExpressJS makes it very simple to serve a new page. You just need to duplicate and modify the following lines to serve a new HTML file (e.g. `visual_search.html`) at a new address (e.g. at `/visualsearch`).

```
app.get('/expNOW', function (request, response) {
  response.render('experiment.html');
});
```

Each `app.get()` defines a "route" linking a URL subaddress (such as localhost:3000/expNOW) to a specific HTML file present in our views folder (such as `experiment.html`).

Bonus - Preload all images

Use the documentation of the preload plugin to find a way to (manually) preload all images of the experiment.

Tip

You can easily construct a list of the paths of all the images displayed in the experiment in the constructor loop that runs over blocks and images.

Perspectives

Going beyond tutorials

Go beyond tutorials to implement professional experiments can be tough.

- Data protection requires a deeper understanding of server-side operations and encryption methods
- Large-scale data curation requires databases that require specific code
- Demanding experiments w.r.t timing accuracy require specific tools/code
- Unusual or complex experiments require to program custom JS plugins from scratch
- Scalability and usability of experiments across devices require *responsive* HTML or SVG
- Reusability of the same code base in the lab for fMRI, EEG, TMS, etc., as well as mobile or desktop apps, is feasible but it requires advanced JS skills.

Regulatory considerations

Web-based behavioral experiments are subjected to administrative and ethical rules.

1. Your experiment should obey EU GPRD legislation (informed consent, data protection, cookies, IP collection, geolocalization, etc.)
2. Identifying personal data should (in general) not travel through the your server (e.g. names, email & physical addresses, exact birthdate, etc.). Use random identifiers and deblinding tables.
3. Sensitive data should be encrypted during transfer and at rest (i.e., in your database)
4. Provided that criteria 1-3 are met, you should be able to run your web experiment only with an ethics committee approval (no CPP). CPP is in principle required only if identifying personal data is collected (including webcam recording).

Conclusion

Questions?

Please fill a small debriefing questionnaire at this address.

https://robustcircuit.eu/teaching/now-hands-on-2023/now-2023-debriefing-hands-on-session

Or flash this QRcode with your smartphone