Online Shooter Simulation in Practice – Template, Tutorial, and Manual

Joost Timmerman

Jannis Kreienkamp

Maximilian Agostini

Nils Pontus Leander

Wolfgang Stroebe

Arie Kruglanski

If you are using the online version of the shooter task, please include the following citation:

Agostini, M., Kreienkamp, J., Kruglanski, A., Leander, N. P., Stroebe, W. (2018). *Handgun Ownership and the Racial Bias in Shooting Simulation Studies*. Manuscript submitted for publication

If you used this guideline please include the following in your references:

Timmerman, J., Kreienkamp, J., Agostini, M., T., Leander, N. P., Stroebe, W., & Kruglanski, A. W. (2018). Online Shooter Simulation (OSS). Groningen: Center for Psychological Gun Research. Retrieved from: https://www.thedataflowcompany.com/resources/oss/oss-manual.pdf

The online implementation of the shooter task is based on:

Correll, J., Park, B., Judd, C. M., & Wittenbrink, B. (2002). The police officer's dilemma: Using ethnicity to disambiguate potentially threatening individuals. *Journal of Personality and Social Psychology*, *83*(6), 1314–1329. http://doi.org/10.1037/0022-3514.83.6.1314

Abstract

This manual aims to assist researchers in adopting and implementing the gun shooter task within their (online) research programs. We offer a pre-built, ready to run survey-template (for Qualtrics). Additionally, we present a conceptual overview of the functional parts so that the task can be adjusted or adapted. Moreover, we also included a data processing and data analysis guide (step by step from data download to final per participant summary).

Keywords: Shooter Task, Online Experiment, Template, Analysis Tool

Online Shooter Simulation in Practice – Template, Tutorial, and Manual

The following document aims to guide researchers and practitioners to implement the Shooter Task (Correll, Park, Judd, & Wittenbrink, 2002) in an online study context (OUR REF). The goal is to provide readers with an easy-to-use and straight forward guide on how to implement the online shooter simulation (OSS). This handbook includes four parts:

1. Access to a pre-build, fully functional template (for Qualtrics)
2. Tutorial on implementations and adaptation of the template
3. More in-depth manual of the functional elements of the OSS
4. Processing and analyzing the OSS data

We offer two main ways to implement the shooter task in online capable questionnaires. The first option is a pre-build, fully functional Qualtrics questionnaire template that can be uploaded into your own account and then be filled with additional content. The second option is an in-depth discussion of the individual building blocks of this task, which can be used to change parts of the task (e.g., change and host your own target stimuli) or to adapt the task for other online survey options (including *Survey Monkey*, *Google Forms*, or *Typeform*).

Accessing the Template

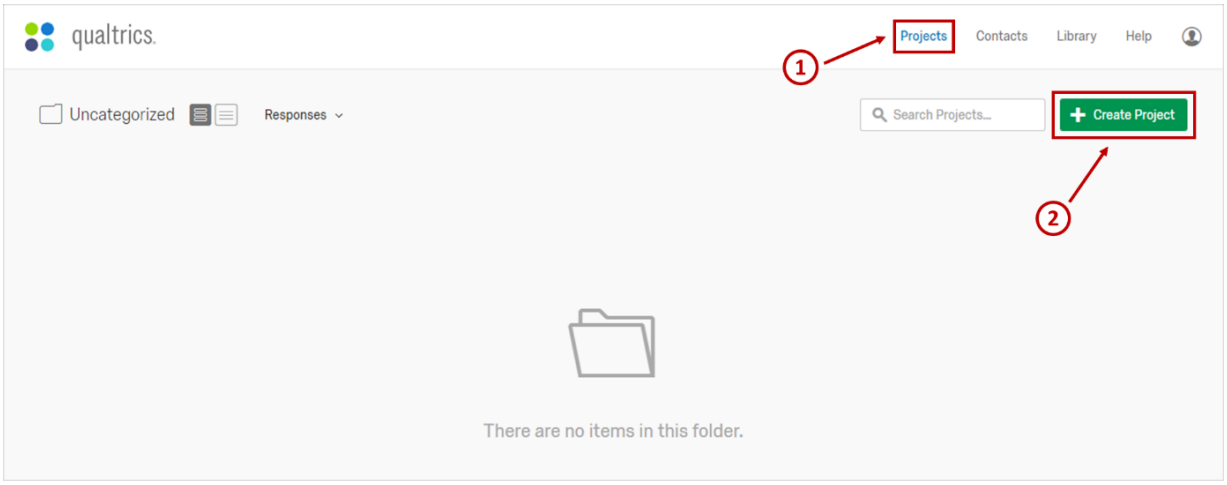
We offer a free, fully functional template of the online shooter task (OSS) for implementation in Qualtrics. The template relies on a "Qualtrics Survey Format" file (.qsf), which is a text file that stores all information of a survey structure (https://www.qualtrics.com/support/survey-platform/survey-module/survey-tools/import-and-export-surveys/). This setup allows for full data ownership by the user and open-source access to any aspect of the survey.

The template can be downloaded at:

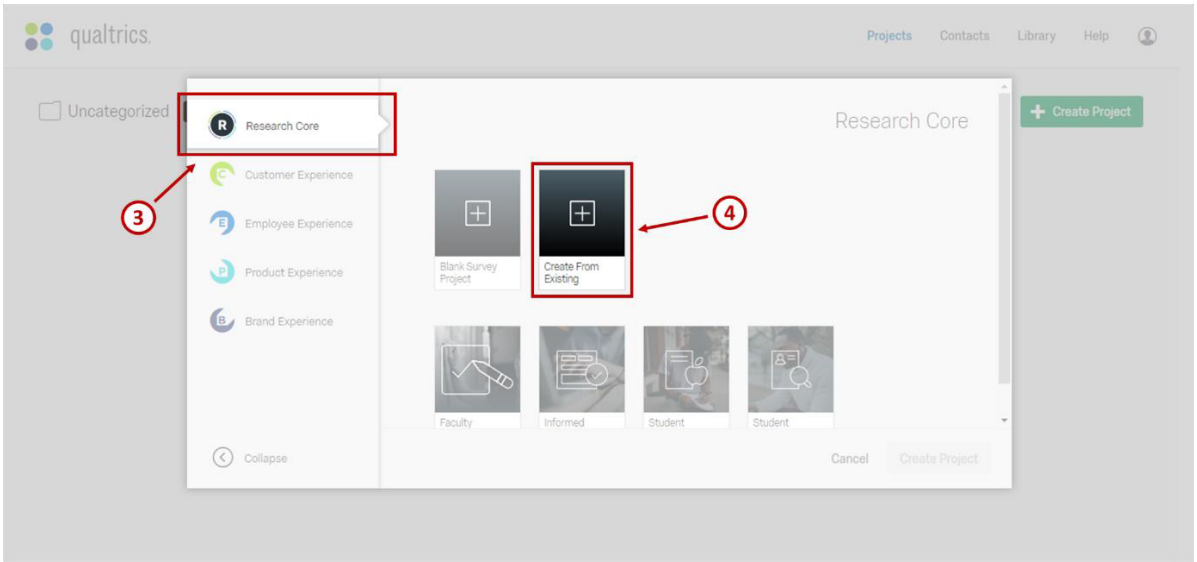
https://www.thedataflowcompany.com/resources/oss/OnlineShooterTask.qsf

Template Implementation

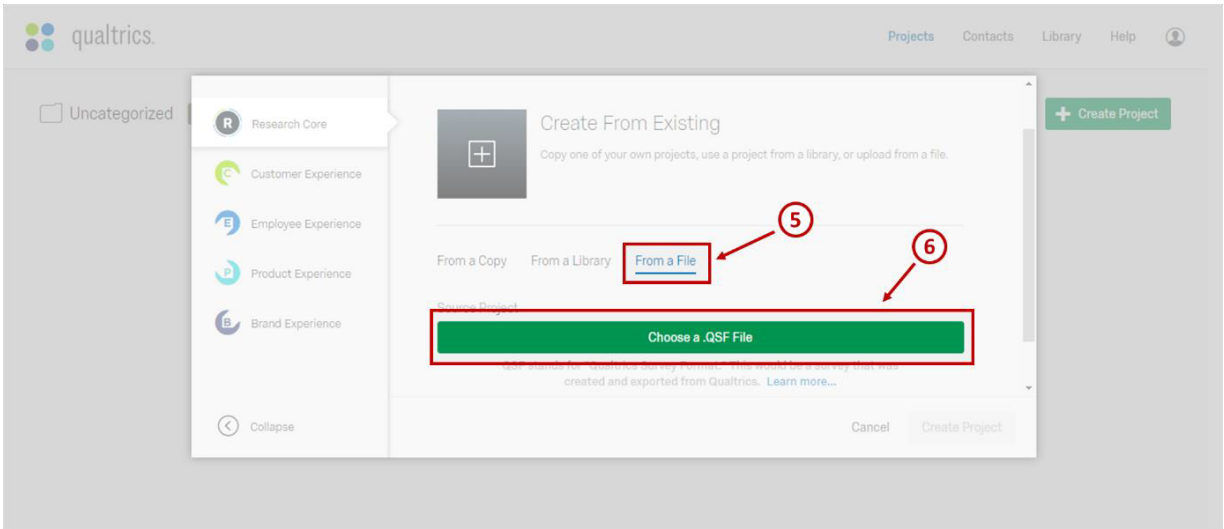
After you have downloaded the template you can immediately upload to the file into your own Qualtrics account. To upload the template log-in to your Qualtrics account and navigate to the **Projects** page ① (if you are inside another project you might have to click on **Projects** twice or even return to the start screen by clicking on the Qualtrics logo). Then click on + **Create Project** ② to open the creation dialog box.



In the new window, make sure you are in the **Research Core** tab ③ and select the **Create From Existing** option ④.

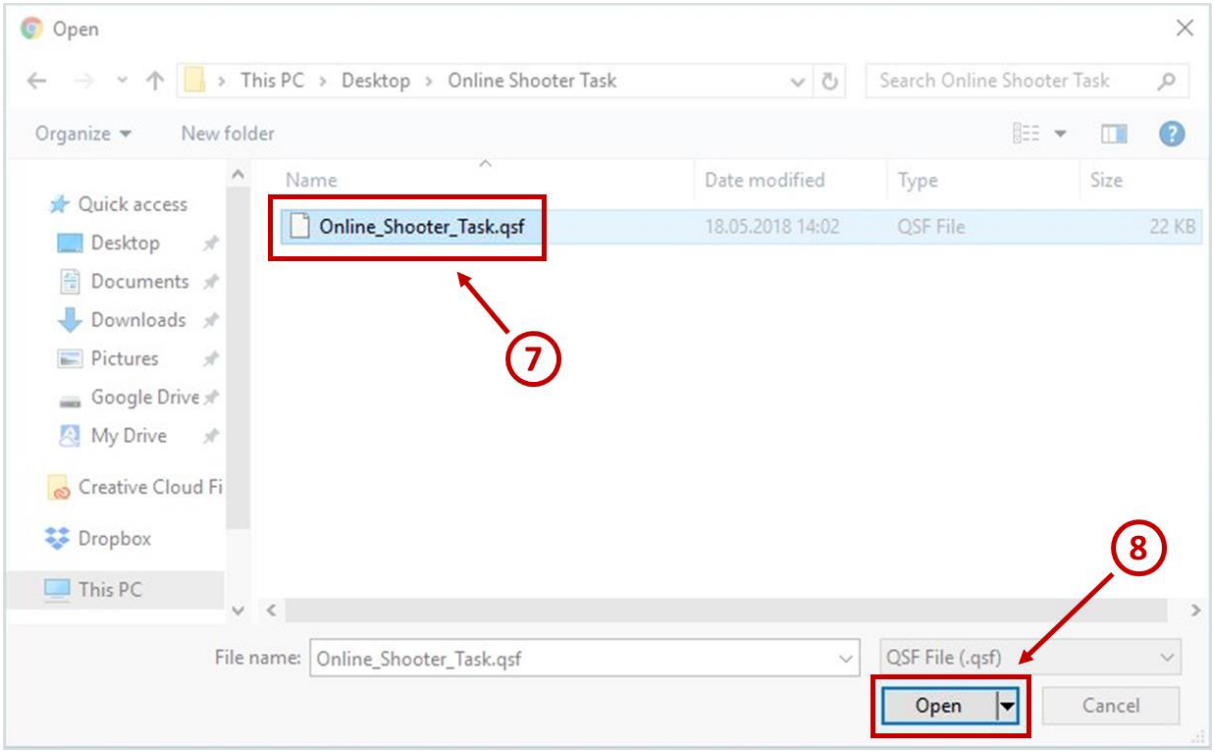


In the new window, select the **From File** tab ⑤ and click the **Choose a .QSF File** button ⑥.

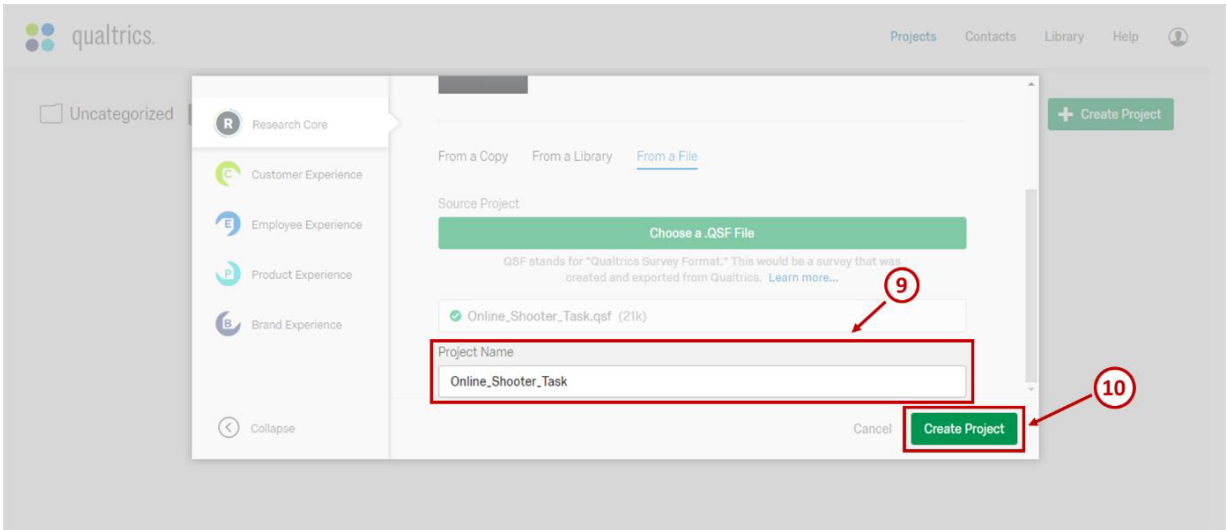


In the pop-up window search and select the OSS template you downloaded earlier ⑦.

Then click on the **Open** button ⑧ to upload the OSS template.



Finally, choose your **Project Name** ⑨ and click on **Create Project** ⑩.



If you have further questions about the setup of a survey via a QSF file in Qualtrics you can find a full documentation of format and process on the Qualtrics support website (https://www.qualtrics.com/support/survey-platform/my-projects/creating-a-project/#CreatingFromAFile). Additionally, the QSF file can be opened with any text editor. The file is usually a single delimited text string, but we have indented and formatted the file so that users can inspect or adapt the template even before uploading it.

Functional Elements of the OSS (Manual)

The OSS was originally programmed for Qualtrics Research Core because the online tool is flexible, offers a free version, and is widely used (especially, a lot of educational licenses; Hine, 2016). Technically, the implementation is, however, equally applicable to any online survey tool that supports: (1) modifications of the header (to load external JavaScript resources), (2) custom Cascading Style Sheets (CSS), (3) question HTML source formatting, (4) access to the response storage via JavaScript (e.g., embedded data or custom variables), and ideally (4) JavaScript compatibility on question elements. As such the online shooter task can be changed in parts (e.g., change stimuli set), be adapted for other platforms (e.g., Survey

Monkey) or set up on your own servers. To aid any of these needs we have compiled a description of the 4 functional elements of the OSS.

1. Hosting stimuli and task JavaScript file
2. Adapting and loading JavaScript files, style sheets, and assets
3. Setting up Task (HTML and JavaScript)
4. Saving the data

In a nutshell, the task works as such: (1) The images, sounds, JavaScripts, and utility assets are stored on your own servers or hosted by content delivery networks (CDN). (2) The JavaScript files (including the OSS task) are specified in the header's <script> tag to make them accessible on all pages (using the *scr* attribute). (3) The task page is set up using HTML elements, a cascading style sheet (CSS) and a JavaScript snippet to initiate the task. And (4) the online tool's data storage is set up to receive the participants' response patterns and reaction times.

1. Hosting stimuli and task JavaScript file

The task stimuli (png files with backgrounds and populated backgrounds; i.e. with armed and unarmed targets) and the task JavaScript file have to be made accessible in the online survey tool. We opted with an option to host the task stimuli in a single archive file (ZIP file), which together with the task JavaScript we made accessible using an absolute URL (e.g., "https://gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/shooter-task-1000ms.js"). As described in Agostini, Kreienkamp, Leander, Kruglanski, and Stroebe (2018), we used a validated, reduced stimulus set of only 48 stimuli (8 practice trials, and 40 task trials; available at: https://gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/stimuli/img_48.zip). The full stimulus sets (80 and 100 stimuli) are available at Joshua Correll's personal web page:

http://psych.colorado.edu/~jclab/FPST.html. Finally, the JavaScript for the shooter task itself can be accessed at: https://gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/shooter-task-1000ms.js.

2. Adapting and loading JavaScript files, style sheets, and assets

2.1. The Shooter Task JavaScript is the key stone of the OSS. It specifies the task parameters (e.g., number of empty backgrounds between trials, maximum response time), loads and displays the stimuli, and sets up the response matrices to be saved in your online survey provider. Even novel JavaScript users will be able to adjust the task parameters:

```
BACKGROUNDS_MIN             = 1,     // minimum no. random backgrounds
BACKGROUNDS_MAX             = 3,     // maximum no. random backgrounds (then target background and then target)
PRACTICETRIALS              = true,  // practice trials Yes / No
BACKGROUND_DISPLAYTIME_MIN  = 500,   // minimum time before next background
BACKGROUND_DISPLAYTIME_MAX  = 1000,  // maximum time before next background
TARGET_DISPLAYTIME_TIMEOUT  = 1000,  // maximum time to respond (before timeout)
FIXATION_DISPLAYTIME        = 1000,  // time fixation cross is displayed
FEEDBACK_DISPLAYTIME        = 1500,  // time feedback message is displayed
BREAK_INTERVAL              = 20,    // every X trials, a rest period is introduced
BREAK_DURATION              = 10000, // duration of break
// Set up point system for response options (see Signal Detection Paradigm)
FEEDBACK_POINTS_HIT         = 10,    // hit
FEEDBACK_POINTS_CR          = 5,     // correct rejection
FEEDBACK_POINTS_FA          = -20,   // false alarm
FEEDBACK_POINTS_MISS        = -40,   // miss
FEEDBACK_POINTS_TOO_SLOW    = -25,   // too slow (time out)
// Set keyboard keys for response options
KEYCODE_SHOOT               = 74,    // J
KEYCODE_DONT_SHOOT          = 70,    // F
KEYCODE_SPACEBAR            = 32,    // Spacebar,
// specify destination for response matrix
QUALTRICS_EMBEDDED_FIELDNAME = 'taskData', // the survey's embedded datafield name for storing the response data.
// Specify location of stimuli file
images_zip_url              =  'https://gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/stimuli/img_48.zip',
```

The adjustments can easily be done by opening the JavaScript file with any common text editor (e.g., Windows Notepad, Apple TextEdit, Notepad++, or VS Code). More technical adaptations such as fitting the response arrays to your preferences or exploring other data storage options and specifying other survey engine options are also possible and are easily accessible to proficient JavaScript developers, would, however, go beyond the scope of this user manual.

2.2. Loading files into your survey environment relies upon the HTML header <script> tag. The task loads six essential JavaScript files.

1. The *jQuery* library (version 2.2.4, works equally with version 3.x.x) to ensure cross-browser compatibility and to simplify the methods used in other code (more information see: https://jquery.com/). Note that some survey engines, such as the Qualtrics engine, preload this library for you (for more information see here) and note that the library can also be accessed through content delivery networks to increase speed and ensure server accessibility (e.g.: https://code.jquery.com/jquery-3.3.1.min.js).

2-3. The *PreloadJS* (version 0.6.2) and *SoundJS* (version 0.6.2) libraries created by GSkinner can add sound effects to the OSS. Detailed documentation can be found at https://createjs.com/ or at https://github.com/CreateJS and CDN links can be found at http://code.createjs.com/.

4-5. The *JSZip* (version 3.1.3) and *JSZip utilities* (version 0.0.2) libraries by Stuart Knightley (https://github.com/Stuk) to load the stimuli images into the task as a single ZIP file.

6. The *Online Shooter Task* itself.

If you were to host these libraries and the task file yourself the header tag would point to the external JavaScript files (for information on the Qualtrics header access see link). For example:

```
<head> <!-- Not needed for Qualtrics header -->
   <script src="//gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/assets/libs/jquery-2.2.4.min.js"></script>
   <script src="//gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/assets/libs/preloadjs-0.6.2.min.js"></script>
   <script src="//gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/assets/libs/soundjs-0.6.2.min.js"></script>
   <script  src="//gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/assets/libs/jszip.js"></script>
   <script src="//gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/assets/libs/jszip-utils.js"></script>
   <script src="//gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/shooter-task-1000ms.js"></script>
</head> <!-- Not needed for Qualtrics header -->
```

2.3. The Cascading Style Sheet has to be referenced in the <head> tag as well. To set up the task screen for the participant and ensure consistent styling of text, graphics, and fixation cross the following markup can be used (accessible at https://gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/assets/css/style.css):

```css
/* use border box sizing */
*,
*:before,
*:after {
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}

.instructions p,
.instructions-practice p,
.feedback p,
.instructions-finished p {
    margin: 30px 0;
}

/* styling */
.shootertask-header{
    text-align: center;
    margin: 0;
    padding-top: 50px;
    font-family: Georgia, Times, "Times New Roman", serif;
    font-weight: normal;
    font-size: 3em;
}

.shootertask {
    position: relative;
    width: 1000px;
    height: 600px;
    margin: 0 auto;
    padding: 30px;
    text-align: center;
    background: #fff; /* background: #eee; */
    font-family: 'Arial', sans-serif;
}

/*
.shootertask img {
    max-width: 800px;
}
*/

.fixationchar,
.feedback,
.restbreak,
.instructions,
.instructions-practice,
.instructions-finished {
    display: none;     /* initially hidden */
}

.fixationchar {
    font-size: 40px;
}

.feedback {
    font-size: 32px;
}

.trial-img,
.fixationchar,
.instructions,
.instructions-practice,
.loading,
.feedback,
.restbreak,
.instructions-finished {
    position: absolute;
    top: 50%;
    left: 50%;
    width: 100%;
    transform: translate(-50%, -50%);
}
```

```
.trial-img {
    height: 100%;
}

.loading-img {
    margin-right: 10px;
    padding-bottom: 5px;
    vertical-align: middle;
}

.loading-error {
    color: #CC0000;
}
```

And the file can similarly be referenced in the header using an absolute URL (for CSS access options in Qualtrics see: https://www.qualtrics.com/support/survey-platform/survey-module/look-feel/advanced-look-feel-settings/):

```
<head>
   <link rel="stylesheet" href="//gmw-qualtrics.webhosting.rug.nl/npleander/shootertask/assets/css/style.css">
</head>
```

3. Setting up the Task (HTML and JavaScript)

Another crucial aspect of the OSS is to set up the task interface for the participant. With the script and styling information in place only the HTML elements and the task initiation script have to be set up. The HTML elements replace a question text without answer options in your online survey provider. Here any aspect of the instruction texts during the task can be altered and adjusted. The HTML markup is the following:

```
<div class="shootertask">
    <div class="instructions-practice">
         <p>You will have less than a second to make each decision.
             <br> You will receive points based on your performance.
             <br> This first round of the game is for practice.</p>

        <p>Remember: Press the "J" key to shoot and the "F" key to not shoot.</p>

        <p>If you are ready for the practice phase, use your right trigger finger (pointer finger) to
        press the "J" key.</p>

        <p>
            <small>(if this doesn't work, please focus on this window first by clicking it)</small>
        </p>
    </div>

    <div class="instructions">
        <p>You have finished the practice phase. The test phase of the Active Shooter Task is longer.
            <br> You will receive feedback as you go. Please stay focused so we can accurately gauge
            your performance.</p>

        <p>Remember: Press the "J" key to shoot and the "F" key to not shoot.</p>

        <p>When you are ready, press the SPACEBAR to start the test phase.</p>

        <p>
            <small>(if this doesn't work, please focus on this window first by clicking it)</small>
        </p>
    </div>
```

```
    <div class="loading" id="loading">
        <img alt="Loading..." class="loading-img" src="//gmw-
        qualtrics.webhosting.rug.nl/npleander/shootertask/assets/img/loading.gif">
        <span class="loading-percentage">Loading 0%</span>

        <div class="loading-error"> </div>
    </div>

    <div class="trial-img">
        <span class="fixationchar">+</span>
    </div>

    <div class="restbreak">
        <p>Rest for seconds before continuing...</p>
         

        <p>Remember:
            <br> "J" = shoot
            <br> "F" = don't shoot.</p>
    </div>

    <div class="feedback">
        <p>
            <span class="feedback-trialpoints">0</span> points</p>

        <p> </p>

        <p>Total:
            <span class="feedback-totalpoints">0</span> points</p>
    </div>

    <div class="instructions-finished">
        <p>This is the end of the task. Please click the '&gt;&gt;' button to continue.</p>
    </div>
</div>
```

Additionally, the OSS script has to be called and populated with information about the survey engine. In Qualtrics, JavaScipt functionality can be added to individual questions using an included JavaScript editor (for documentation see https://www.qualtrics.com/support/survey-platform/survey-module/question-options/add-javascript/). Here you add the following code to the question itself**:**

```
Qualtrics.SurveyEngine.addOnload(function()
{
    var qualtricsSurveyEngine = Qualtrics.SurveyEngine,
        qualtricsQuestionData = this;

        initShooterTask(qualtricsSurveyEngine, qualtricsQuestionData);
});
```

If such a feature is not available to you, similar functionality and initiation of the OSS might be reached through the HTML <script> tag within the the question element.

5. Saving the data

The final aspect is setting up the system to retrieve the OSS response data. The participant's data will be saved in a matrix array that gets converted into a JSON formatted

array per participant. This array includes the characteristics of the target stimuli (i.e., a practice trial Boolean, target ethnicity, an armed Boolean, picture id), the response information (i.e., shoot decision, reaction time in milliseconds), as well as an ID variable of the participant. In Qualtrics this JSON array is saved in an embedded data field that represents a custom variable with information per participant (can be specified in the Shooter Task JavaScript file, see 2.1.). Should you not have access to the survey engines data storage an alternative would be to ensure that identifying information is loaded into the response array and to then save the OSS data separately on your own servers and merge it with the survey responses at a later time point (note, however, that this would be part of a greater effort of adapting the OSS to a different survey provider and would include major technical changes to the OSS JavaScript file – recommended only to experienced users).

Dealing with OSS Data

In this concluding section we will illustrate the data preparation steps using the data output format from Qualtrics and the freely available analysis program R Studio (the full R Markdown can be downloaded at https://www.thedataflowcompany.com/resources/oss/Dealing-with-OSS-Data.RMD). We assume a basic understanding of R and the R Studio environment. **Importing the Data from Qualtrics**

In a first step, we will load the Qualtrics data into the R environment. We use the `data.table` package's `fread` function to use all CPU cores while loading the comma-separated values file (csv file).

```
if (!require("pacman")) install.packages("pacman")
## Loading required package: pacman
pacman::p_load(data.table)

dat.Qualtrics = fread('OSS Data - CSV Qualtrics.csv',header=T,sep=',')
```

[The first bit of code (pacman package) checks whether the package was previously installed. If necessary, installs it and then loads the package into the R library for you to use]

Preparing the JSON file.

In a second step we look at the OSS shooter data in more detail. We extract it from the larger Qualtrics `data frame`, clean up Qualtrics export oddities and transform the JSON file into a R `data frame` to be processed further.

```
pacman::p_load(jsonlite,plyr,psych,knitr)

json = dat.Qualtrics$taskData # extract (subset) OSS data only

json = json[-(1:2)]  # remove first two rows (headers from Qualtrics)
json = gsub("\"\"", "\"", json) # replace douple """" with single ""
json = json[json != ""] # remove all empty elements

# JSON to R Data frame
json.df = ldply(lapply(as.list(json), function(x) fromJSON(x, flatten=TRUE)), data
.frame)

# NA to "NULL"
json.df$firing[is.na(json.df$firing)]  =  "NULL"

# Character Strings to Factors
json.df[, (lapply(json.df, class) == "character")] = sapply(json.df[, (lapply(json
.df, class) == "character")], as.factor)
# clean up
rm(json)
# look at result
```

Shooter Data Extracted: First 5 Rows of the new Data Frame

responseID	firing	ethnicity	armed	Reaction Time	Total Points	Practice Trial	filename
R_3g7pnKNYpfDVpYx	NULL	white	FALSE	1000.00	-25	TRUE	ziwu04p2.jpg
R_3g7pnKNYpfDVpYx	TRUE	black	TRUE	939.99	-15	TRUE	zbba923.jpg
R_3g7pnKNYpfDVpYx	TRUE	white	TRUE	722.42	-5	TRUE	zjwa04d1.jpg
R_3g7pnKNYpfDVpYx	NULL	black	FALSE	1000.00	-30	TRUE	zfbu92w1.jpg
R_3g7pnKNYpfDVpYx	FALSE	black	FALSE	824.37	-25	TRUE	zkbu11w2.jpg

Note that the data is organized in a single `data frame` in a long format. That is to say that, every participant occupies multiple rows (one for every trail) and each participant is identifiable through the `responseID` column. In the following steps we will summarize the shooter data into common statistics.

Extract Count Decisions

The first summary statistics center around the count decisions in response to different types of target stimuli. For every type of the 4 types of stimuli – 2(ethnicity: black vs. white) X 2(object: armed vs. unarmed) – we get three possible response decision options: (1) shoot, (2) do not shoot, and (3) time out [null]. Additionally, the responses are separated by whether they were part of the initial practice trails or whether they were part of the experimental trials.

```
# Summarize shoot decisions by stimuli types
count = as.data.frame(with(json.df, table(responseID, ethnicity, armed, practiceTr
ial, firing)))
# Rename Factor labels
levels(count$armed) = c("unarmed", "armed")
levels(count$firing) = c("noshot", "null", "shot")
# look at result
```

Count Summary Extracted: First 5 Rows of the new Data Frame

responseID	ethnicity	armed	Practice Trial	firing	Freqency
R_0BB7yeTxeJSYZqN	black	unarmed	FALSE	noshot	1
R_0BB7yeTxeJSYZqN	black	armed	FALSE	noshot	6
R_0BB7yeTxeJSYZqN	black	unarmed	FALSE	shot	5
R_0BB7yeTxeJSYZqN	black	armed	FALSE	shot	1
R_0BB7yeTxeJSYZqN	white	unarmed	FALSE	noshot	4

Note that the current summary of the counts is still in a `long format` so that all decisions and participants are listed beneath each other. Every participant and their individual responses are clearly identified but for most types of analyses we would like to have a single row per participant with the decision patterns as individual variables (i.e., the `wide format`). We do that by `casting` (reshaping) the data by participant with `ethnicity`, `armed`, and `firing` as our measurement variables. Additionally, in this example we drop all practice trials before we re-order the data.

```
# Summarize counts per participants
Count = dcast(count[count$practiceTrial=="FALSE",],
    responseID ~ ethnicity + armed + firing,
    value.var = "Freq")
# Rename coulmns
setnames(Count,
    old = c("black_unarmed_noshot", "black_unarmed_null", "black_unarmed_shot", "b
lack_armed_noshot", "black_armed_null"," black_armed_shot", "white_unarmed_noshot"
```

```
, "white_unarmed_null", "white_unarmed_shot", "white_armed_noshot", "white_armed_n
ull", "white_armed_shot"),
     new = c("ct.bk.unarmed.noshot", "ct.bk.unarmed.null", "ct.bk.unarmed.shot", "c
  t.bk.armed.noshot", "ct.bk.armed.null", "ct.bk.armed.shot", "ct.wt.unarmed.noshot"
, "ct.wt.unarmed.null", "ct.wt.unarmed.shot", "ct.wt.armed.noshot", "ct.wt.armed.n
ull", "ct.wt.armed.shot"))
# Add total counts of timeouts
Count$null_sum = rowSums(Count[,grep("null",names(Count))])
rm(count)
# look at result
```

Count Summary per participant: First 5 Rows of the new Data Frame

responseID	ct.bk.unarmed.noshot	ct.bk.unarmed.null	ct.bk.unarmed.shot	[…]	ct.wt.armed.null	ct.wt.armed.shot	Null sum
R_0BB7yeTxeJSYZqN	1	8	1	[…]	4	3	22
R_0f7qOEsyUE0voNb	6	3	1	[…]	2	8	6
R_0H5wt2iiugKwzXr	5	2	3	[…]	1	4	4
R_0HyH0Hsdfg2dsf	1	2	7	[…]	0	10	2

Signal Detection Statistics

In the next step we convert the count statistics into more common signal detection statistics. While they are conceptually the same, many researchers and readers are more comfortable with decision counts in the form of proportions and with the common labels of 'correct rejection', 'false alarm', 'miss', and 'hit' (*Note that we divide every count by 10 because participants, in our case, saw 10 targets of every stimuli option*). This also offers the option to further summarize the decisions more broadly and irrespective of the targets ethnicity (then dividing by 20 for a total of 10 Black and 10 White targets).

```
# Signal Detection Basics Counts (CR = correct rejection, FA = false alarm, M = mi
ss, H = hit)
Signal = Count[,grep("shot",names(Count))]/10
setnames(Signal,
    old = c("ct.bk.unarmed.noshot", "ct.bk.unarmed.shot", "ct.bk.armed.noshot", "c
t.bk.armed.shot", "ct.wt.unarmed.noshot", "ct.wt.unarmed.shot", "ct.wt.armed.nosho
t", "ct.wt.armed.shot"),
    new = c("CR.bk", "FA.bk", "M.bk", "H.bk", "CR.wt", "FA.wt", "M.wt", "H.wt"))

# Signal Detection Basics overall percentage (irrespective of ethnicity)
Signal$CR.tot = rowSums(Count[,grep("unarmed.noshot",names(Count))])/20
```

```
Signal$FA.tot = rowSums(Count[,grep("unarmed.shot",names(Count))])/20
Signal$M.tot = rowSums(Count[,grep(".armed.noshot",names(Count))])/20
Signal$H.tot = rowSums(Count[,grep(".armed.shot",names(Count))])/20
Signal$responseID = Count$responseID
```

We also offer more complex signal detection measures (of sensitivity and response bias). A detailed description of their uses and formulas would, however, go beyond the scope of this manual and can be found elsewhere (e.g., Stanislaw & Todorov, 1999).

```
#  Sensitivity  Analysis
## Whites
### Sensitivity Statistics
Signal$dprime.wt = qnorm(Signal$H.wt)-qnorm(Signal$FA.wt)
Signal$Aprime.wt = (0.5+(sign(Signal$H.wt-Signal$FA.wt)*
   (((Signal$H.wt-Signal$FA.wt)^2+abs(Signal$H.wt-Signal$FA.wt))/
   (4*pmax(Signal$H.wt, Signal$FA.wt)-4*Signal$H.wt*Signal$FA.wt))))
Signal$Adprime.wt = pnorm(Signal$dprime.wt/sqrt(2))

### Response Bias
Signal$Beta.wt = exp((qnorm(Signal$FA.wt)^2-qnorm(Signal$H.wt)^2)/2)
Signal$c.wt = -(qnorm(Signal$H.wt)+qnorm(Signal$FA.wt))/2
Signal$Bdbprime.wt = (sign(Signal$H.wt-Signal$FA.wt)*
    ((Signal$H.wt*(1-Signal$H.wt)-Signal$FA.wt*(1-Signal$FA.wt))/
    (Signal$H.wt*(1-Signal$H.wt)+Signal$FA.wt*(1-Signal$FA.wt))))

## Blacks
### Sensitivity Statistics
Signal$dprime.bk = qnorm(Signal$H.bk)-qnorm(Signal$FA.bk)
Signal$Aprime.bk = (0.5+(sign(Signal$H.bk-Signal$FA.bk)*
   (((Signal$H.bk-Signal$FA.bk)^2+abs(Signal$H.bk-Signal$FA.bk))/
   (4*pmax(Signal$H.bk, Signal$FA.bk)-4*Signal$H.bk*Signal$FA.bk))))
Signal$Adprime.bk = pnorm(Signal$dprime.bk/sqrt(2))

### Response Bias
Signal$Beta.bk = exp((qnorm(Signal$FA.bk)^2-qnorm(Signal$H.bk)^2)/2)
Signal$c.bk = -(qnorm(Signal$H.bk)+qnorm(Signal$FA.bk))/2
Signal$Bdbprime.bk = (sign(Signal$H.bk-Signal$FA.bk)*
    ((Signal$H.bk*(1-Signal$H.bk)-Signal$FA.bk*(1-Signal$FA.bk))/
    (Signal$H.bk*(1-Signal$H.bk)+Signal$FA.bk*(1-Signal$FA.bk))))

## Overall
### Sensitivity Statistics
Signal$dprime.tot = qnorm(Signal$H.tot)-qnorm(Signal$FA.tot)
Signal$Aprime.tot = (0.5+(sign(Signal$H.tot-Signal$FA.tot)*
(((Signal$H.tot-Signal$FA.tot)^2+abs(Signal$H.tot-Signal$FA.tot))/
(4*pmax(Signal$H.tot,Signal$FA.tot)-4*Signal$H.tot*Signal$FA.tot))))
Signal$Adprime.tot = pnorm(Signal$dprime.tot/sqrt(2))

### Response Bias
Signal$Beta.tot = exp((qnorm(Signal$FA.tot)^2-qnorm(Signal$H.tot)^2)/2)
Signal$c.tot = -(qnorm(Signal$H.tot)+qnorm(Signal$FA.tot))/2
Signal$Bdbprime.tot = (sign(Signal$H.tot-Signal$FA.tot)*
   ((Signal$H.tot*(1-Signal$H.tot)-Signal$FA.tot*(1-Signal$FA.tot))/
   (Signal$H.tot*(1-Signal$H.tot)+Signal$FA.tot*(1-Signal$FA.tot))))
```

If you decide to calculate (any of) these more complex measures as described above make sure to merge them with the other count and simple signal detection summaries.

```
# Merge Counts with Signal Detection Statistics
Decisions = merge(x=Count, y=Signal, by.x="responseID", by.y="responseID")
rm(Signal)
```

Reaction Times

In our last step, we will also calculate mean reaction times for the different decision patterns. Importantly, the decision of whether null responses (time outs are saved as the maximum reaction time) are included in the mean calculation should be considered. The code below includes the reaction times of null responses in the calculation because decisions of data cleaning and missing data have to be made by the individual researcher and all have an influence of which data-points you want to include for the summary statistics. The below code, consequently, offers a framework to be built upon. The easiest way to adapt the current version is to create a separate `data frame` with the participants relevant for summary before extracting the reaction time means.

```
pacman::p_load(magrittr,dplyr)
# Summarize Mean Reaction Times by Target Stimuli, Ethnicity, and Shoot Decisions
rt = json.df %>%
   group_by(responseID, ethnicity, armed, practiceTrial, firing) %>%
   summarise_at(vars("reactionTime"), mean)

# Rename Factor Lables
rt$ethnicity = factor(rt$ethnicity, labels = c("black", "white"))
rt$armed = factor(rt$armed, labels = c("unarmed", "armed"))
rt$firing = factor(rt$firing, labels = c("noshot", "null", "shot"))

# Aggregate per Participant
RT = dcast(rt[which(rt$practiceTrial=="FALSE" & rt$firing!="null"),], responseID ~
ethnicity + armed + firing, value.var = "reactionTime")

### Rename Columns
setnames(RT,
    old = c("black_unarmed_noshot", "black_unarmed_shot", "black_armed_noshot", "b
lack_armed_shot", "white_unarmed_noshot", "white_unarmed_shot", "white_armed_nosho
t", "white_armed_shot"),
    new = c("rt.bk.unarmed.noshot", "rt.bk.unarmed.shot", "rt.bk.armed.noshot", "r
t.bk.armed.shot", "rt.wt.unarmed.noshot", "rt.wt.unarmed.shot", "rt.wt.armed.nosho
t", "rt.wt.armed.shot"))
rm(rt)

# Summarize Mean Reaction Times by Decision Correctness
## Correct Decisions
```

```
rt.cor= subset(json.df, (practiceTrial==FALSE & armed==TRUE & firing=="TRUE") | (p
racticeTrial==FALSE & armed==FALSE & firing=="FALSE")) %>%
    group_by(responseID, ethnicity) %>%
    summarise_at(vars("reactionTime"), mean)
### Aggregate per Participant
rt.cor.t = dcast(rt.cor, responseID ~ ethnicity, value.var = "reactionTime")
### Add mean correct reaction time (irrespective of Ethnicity)
rt.cor.t$total= (subset(json.df, (practiceTrial==FALSE & armed==TRUE & firing=="TR
UE") | (practiceTrial==FALSE & armed==FALSE & firing=="FALSE")) %>%
    group_by(responseID) %>%
    summarise_at(vars("reactionTime"), mean))$reactionTime

### Rename Columns
setnames(rt.cor.t,
    old = c("black","white","total"),
    new = c("rt.cor.bk","rt.cor.wt","rt.cor.all"))

## Incorrect Decisions
rt.incor = subset(json.df, (practiceTrial==FALSE & armed==TRUE & firing=="FALSE")
| (practiceTrial==FALSE & armed==FALSE & firing=="TRUE")) %>%
    group_by(responseID, ethnicity) %>%
    summarise_at(vars("reactionTime"), mean)
### Aggregate per Participant
rt.incor.t = dcast(rt.incor, responseID ~ ethnicity, value.var = "reactionTime")
### Add mean incorrect reaction time (irrespective of Ethnicity)
rt.incor.t$total= (subset(json.df, (practiceTrial==FALSE & armed==TRUE & firing=="
FALSE") | (practiceTrial==FALSE & armed==FALSE & firing=="TRUE")) %>%
    group_by(responseID) %>%
    summarise_at(vars("reactionTime"), mean))$reactionTime

### Rename Columns
setnames(rt.incor.t,
    old = c("black", "white", "total"),
    new = c("rt.cor.bk", "rt.cor.wt", "rt.cor.all"))
```

We, finally, merge the count decisions (including the signal detection measures, if necessary), with the different reaction time data frames using the common `response ID` of the participants.

```
# Merge Reaction Times and Decisions
RT2 = merge(RT, rt.cor.t, by="responseID",all=T)
RT3 = merge(RT2, rt.incor.t, by="responseID",all=T)
DV = merge(Decisions, RT3, by="responseID",all=T)

# Clean-up Environment
rm(RT,RT2,RT3,rt.cor,rt.cor.t,rt.incor,rt.incor.t,Count,Decisions)
```

References

Correll, J., Park, B., Judd, C. M., & Wittenbrink, B. (2002). The police officer's dilemma: Using ethnicity to disambiguate potentially threatening individuals. *Journal of Personality and Social Psychology*, *83*(6), 1314–1329. https://doi.org/10.1037//0022-3514.83.6.1314

Hine, C. (2016). The internet and research methods. In G. N. Gilbert & P. Stoneman (Eds.), *Researching Social Life* (pp. 339–356). Los Angeles, CA: Sage Publications.

Stanislaw, H., & Todorov, N. (1999). Calculating of signal detection theory measures. *Behavior Research Methods, Instruments, & Computers*, *31*(1), 137–149. https://doi.org/10.3758/BF03207704