Package: jaysire (via r-universe)

August 26, 2024

Type Package

Title Build jsPsych Experiments in R

Version 0.1.0

Author Danielle Navarro

Maintainer Danielle Navarro <d.navarro@unsw.edu.au>

Description The jaysire package allows the user to build browser based behavioral experiments within R by providing an interface to the jsPsych javascript library.

License MIT + file LICENSE

Encoding UTF-8

SystemRequirements libssh >= 0.6.0 (the original, not libssh2)

LazyData true

Imports jsonlite, methods, plumber, purrr, readr, tibble, magrittr, rlang, here

RoxygenNote 7.1.1

URL https://github.com/djnavarro/jaysire

BugReports https://github.com/djnavarro/jaysire/issues

Suggests rmarkdown, testthat (>= 2.1.0), covr, ssh

Repository https://djnavarro.r-universe.dev

RemoteUrl https://github.com/djnavarro/jaysire

RemoteRef HEAD

RemoteSha fd76ccbce491bc8575ac4e751d97db3d5227e327

Contents

`build_experiment` *Build the experiment files*

Description

Build the experiment files

Usage

```
build_experiment(
  timeline,
  path,
  experiment_folder = "experiment",
  data_folder = "data",
  experiment_title = NULL,
  jsPsych_path = file.path(system.file("extdata", "jsPsych", package = "jaysire")),
  resources = NULL,
  columns = NULL,
  ...
)
```

Arguments

`timeline`	A timeline object
`path`	A string specifying the path in which to build the experiment
`experiment_folder`	A string specifying the experiment subfolder
`data_folder`	A string specifying the data subfolder
`experiment_title`	A string specifying the title of the experiment
`jsPsych_path`	A string specifying the path to jsPsych
`resources`	A tibble specifying how to construct resource files, or NULL
`columns`	Additional data values (constants) to store
`...`	Arguments to pass to jsPsych.init()

Details

The `build_experiment()` function is used to build the actual jsPsych experiment from the abstract description contained in the `timeline` argument. The input for the `timeline` argument should be a timeline constructed using `build_timeline()` and the `path` argument should specify the path to the folder in which the experiment files should be created. If the experiment needs to rely on resource files (e.g., images, audio files etc) then the `resources` argument should be a tibble containing the information needed to copy those files to the appropriate location. The easiest way to do so is by using the build_resources() function: see the documentation for that function for a more detailed description of what this tibble should contain.

When called, the build_experiment() function writes all the experiment files, compiled to javascript and HTML. The file structure it creates is as follows. Within the path folder are two subfolders, data_folder and experiment_folder (named "data" and "experiment" by default). The data_folder folder is empty, but intended to serve as a location to which data files can be written. The experiment_folder folder contains the "index.html" file, which is the primary source file for the experiment page, and an "experiment.js" file that specifies the jsPsych timeline and calls the jsPsych.init() javascript function that starts the experiment running. It also contains a "resource" folder with other necessary files (see build_resources() for detail). If specified, experiment_title will set the name of the experiment as the title of the HTML file in "index.html". jsPsych_path is a string that specifies the path to jsPsych.

Because build_experiment() creates the call to jsPsych.init(), it can also be used to specify any parameters that the user may wish to pass to that function via the .... There are quite a number of parameters you can specify this way:

- display_element is a string specifying the ID of an HTML element to display the experiment in. If left blank, jsPsych will use the <body> element to display content. All keyboard event listeners are bound to this element. In order for a keyboard event to be detected, this element must have focus (be the last thing that the subject clicked on).
- on_finish is a javascript function that executes when the experiment ends. It can be constructed manually using insert_javascript(), but in many cases there is a jaysire function that will create the appropriate function for you. For example, if you want the data to be saved locally at the end of the experiment you can set on_finish = save_locally(), whereas if you want the data to be saved to the Google Datastore you can set on_finish = save_googlecloud().
- on_trial_start is a javascript function that executes when a trial begins.
- on_trial_finish is a javascript function that executes when a trial ends.
- on_data_update is a javascript function that executes every time data is stored within the jsPsych internal storage.
- on_interaction_data_update is a javascript function that executes every time a new interaction event occurs. Interaction events include clicking on a different window (blur), returning to the experiment window (focus), entering full screen mode (fullscreenenter), and exiting full screen mode (fullscreenexit).
- on_close is a javascript function that executes when the user leaves the page. This can be used, for example, to ensure that data are saved before the page is closed.
- exclusions is used to specify restrictions on the browser the subject can use to complete the experiment. See list of options below.
- show_progress_bar is a javascript logical value. If true, then a progress bar is shown at the top of the page.
- message_progress_bar is a string containing a message to display next to the progress bar. The default is 'Completion Progress'.
- auto_update_progress_bar is a javascript logical value. If true, then the progress bar at the top of the page will automatically update as every top-level timeline or trial is completed.

- show_preload_progress_bar is a javascript logical value. If true, then a progress bar is displayed while media files are automatically preloaded.
- preload_audio is a javascript array of audio files to preload before starting the experiment.
- preload_images is a javascript array of image files to preload before starting the experiment.
- preload_video is a javascript array of video files to preload before starting the experiment.
- max_load_time is a numeric value specifying the maximum number of milliseconds to wait for content to preload. If the wait time is exceeded an error message is displayed and the experiment stops. The default value is 60 seconds.
- max_preload_attempts is numeric value specifying the maximum number of attempts to preload each file in case of an error. The default value is 10. There is a small delay of 200ms between each attempt.
- use_webaudio is a javascript logical. If false, then jsPsych will not attempt to use the WebAudio API for audio playback. Instead, HTML5 Audio objects will be used. The WebAudio API offers more precise control over the timing of audio events, and should be used when possible. The default value is true.
- default_iti is a numeric value setting the default inter-trial interval in milliseeconds. The default value if none is specified is 0.
- experiment_width is a numeric value setting the desired width of the jsPsych container in pixels. If left undefined, the width will be 100% of the display element. Usually this is the <body> element, and the width will be 100% of the screen size.

Note: as of the current writing not all of these have been tested (even informally) through jaysire. Please report any unexpected behaviour by opening an issue on the GitHub page.

Value

Invisibly returns NULL

---

build_resources *Build the resource file specification from a directory path*

Description

Build the resource file specification from a directory path

Usage

```
build_resources(
  from,
  audio = c(".mp3", ".wav", ".aif", ".mid"),
  video = c(".mp4", ".mpg", ".mov", ".wmv", ".webm", ".ogg"),
  image = c(".jpg", ".png", ".bmp", ".svg", ".tiff"),
  script = c(".js"),
  style = c(".css")
)
```

Arguments

`from`	The paths to files/directories
`audio`	File extensions assumed to be audio
`video`	File extensions assumed to be video
`image`	File extensions assumed to be images
`script`	File extensions assumed to be scripts
`style`	File extensions assumed to be stylesheets

Details

Because jsPsych experiments are designed to run through the browser rather than within R, the jaysire package incorporates "resource files" in a slightly complicated way. Resource files here are divided into several categories because the experiment has to incorporate them in different ways: the code for handling images is different to the code for handling audio files or video files, and both are different to how scripts and style files are loaded. As a consequence, the `build_experiment()` function needs to know what kind of file each resource corresponds to in order to construct the experiment properly. One part of what the `build_resources()` function does is make this a little easier for the user, by scanning all files that belong to a "resource folder" (located at the path specified by the `from` argument) and using the file extension to guess the type of each resource file.

The second peculiarity is that the `build_experiment()` function will make copies of all resource files. Regardless of where the original files are taken `from`, a separate copy will be placed in an appropriate subfolder within the experiment. For example, if the primary experiment file is saved to "experiment/index.html" and it requires an image file called "picture.png", it will be copied to "experiment/resource/image/picture.png". The reason for this is to ensure that the "experiment" folder is entirely self contained, and includes *all* source files necessary to run the experiment. This is important if the experiment is designed to be deployed to a remote server (e.g., using Google App Engine), as is very often the case if one wishes to run an online experiment.

It is for this reason that the `build_experiment()` function creates copies of resource files: jaysire is designed on the presumption that the user may wish to keep the "original" versions of resource files somewhere else, and makes copies of them that can be deployed in the experiment. Viewed from this perspective, the `build_resources()` function is a helper function: as long as all the resource files your experiment requires are (at least temporarily) stored in the `from` folder, it will construct a tibble that contains all the information that `build_experiment()` needs to organise the experimental files appropriately.

There are two important details to note. First, the `from` folder should be flat: it should not contain subfolders. Second, there are various arguments (e.g., `audio`, `video`, `script` etc) that specify the file extensions that are associated with each resource type. The default values are likely to change in future as the current lists are quite restrictive.

Value

The `build_resources()` function returns a tibble with four columns, called "name", "type", "from", and "to". The "name" column lists the name of every resource file discovered in the `from` folder and the "type" column lists the kind of resource file (image, audio, video, script, style or other file). Finally, the "from" column specifies the *full* path to the existing location of the resource file, while the "to" column specifies the *relative* path to which a copy of the resource file should be copied (relative to the "index.html" file for the experiment)

See Also

insert_resource, build_experiment

build_timeline *Build a timeline from trials*

Description

Build a timeline from trials

Usage

```
build_timeline(...)
```

Arguments

...	trial objects to add to this timeline

Details

Experiments in jsPsych are specified in terms of a "timeline" object, where each timeline can consist of one or more "trial" objects and timelines can contain other timelines. In pure jsPsych it is possible to define a "bare" trial that is not contained within a timeline (the trial is essentially a timeline) but jaysire is slightly more restrictive. To build a timeline in jaysire, the output of `trial_` functions need to be passed through the `build_timeline()` function to create a properly constructed timeline object.

Once constructed, behaviour and execution of a timeline can be modified using a variety of functions. A timeline can be looped using the `display_while()` function, or executed contiionally using the `display_if()` function. Timeline variables can be attached using `set_variables()` and other parameters can be passed to the timeline using `set_parameters()`.

Value

An object of class "timeline"

`display_if` *Modify a timeline to execute if a condition is met*

Description

Modify a timeline to execute if a condition is met

Usage

```
display_if(timeline, conditional_function)
```

Arguments

`timeline`	A timeline object
`conditional_function`	A javascript function that returns true if the timeline should execute and false otherwise

Details

The `display_if()` function is used to modify an existing `timeline` object, and provides the ability for conditional branching within an experiment. To use it, the user must supply the `conditional_function`, a javascript function that executes at runtime and should evaluate to true or false. If the conditional function returns true, then the timeline object will execute; if the conditional function returns false then jsPsych will not run this timeline.

At present jaysire provides only limited tools for writing the conditional function. The `fn_data_condition()` function allows a simple approach that allows the conditional function to query the jsPsych data store, but only in a limited way. Future versions will (hopefully) provide a richer tool set for this. However, for users who are comfortable with writing javascript functions directly the `insert_javascript()` function may be useful.

Value

The modified timeline object

See Also

`display_while`, `build_timeline`, `fn_data_condition`, `insert_javascript` `display_while` *Modify a timeline to execute within a loop*

Description

Modify a timeline to execute within a loop

Usage

```
display_while(timeline, loop_function)
```

Arguments

`timeline`	The timeline object
`loop_function`	A javascript function that returns true if loop repeats, false if terminates

Details

The `display_while()` function is used to modify an existing `timeline` object, and provides the ability to include while loops within an experiment. To use it, the user must supply the `loop_function`, a javascript function that executes at runtime and should evaluate to true or false. If the loop function returns true, then the timeline object will execute for another iteration; this condinues until the loop function returns false.

At present jaysire provides only limited tools for writing the loop function. The `fn_data_condition()` function allows a simple approach that allows the loop function to query the jsPsych data store, but only in a limited way. Future versions will (hopefully) provide a richer tool set for this. However, for users who are comfortable with writing javascript functions directly the `insert_javascript()` function may be useful.

Value

The modified timeline object

See Also

`display_if`, `build_timeline`, `fn_data_condition`, `insert_javascript`

download_googlecloud *Download data from a jspsych experiment deployed on google cloud*

Description

Download data from a jspsych experiment deployed on google cloud

Usage

```
download_googlecloud()
```

Details

This function currently does nothing

See Also

save_googlecloud, run_googlecloud

download_webserver *Download data from a jspsych experiment deployed on a webserver*

Description

Download data from a jspsych experiment deployed on a webserver

Usage

```
download_webserver(ssh, keyfile = NULL, to = ".")
```

Arguments

ssh	ssh connection string to a webserver
keyfile	(optional) path to a ssh private key to log in to your webserver
to	local folder to download the data to

Details

This function assumes the default setup by run_webserver, so all it does is download the folder /var/www/server_data

See Also

save_webserver, run_webserver

fn_data_condition *Return a javascript function that checks a data value*

Description

Return a javascript function that checks a data value

Usage

```
fn_data_condition(expr, trials_back = 1)
```

Arguments

expr	An expression to be evaluated within the jsPsych data store
trials_back	The number of trials before the present one for which to query the data

Details

The `fn_data_condition()` function creates a javascript function that can query the jsPsych data store and evaluate the expression `expr` within the data store. It is (at present) very limited, and can only query the data store for a single trial (i.e., a single row in the data set). By default it queries the most recent trial (`trials_back = 1`) but this behaviour can be modified.

The intention behind this function is that it be used in conjunction with functions such as `display_if()` and `display_while()` that require a javascript function that will evaluate to true or false, in order to determine whether to continue the while loop or whether the if condition holds.

As an example, one might set `fn_data_condition(button_pressed == "0")` when calling `display_if()`. If the participant had pressed button "0" on the previous trial, then the timeline in question will be executed. Otherwise it is not.

Note that this function is a work in progress and will likely change in future versions in order to allow more flexibility.

Value

A javascript function

fn_sample *Return a javascript function that samples from an array*

Description

Return a javascript function that samples from an array

Usage

```
fn_sample(x, size, replace = FALSE, weights = NULL)
```

Arguments

`x`	A vector specifying the possible values
`size`	The number of values to sample
`replace`	Sample with replacement? (default = FALSE)
`weights`	Probability of sampling each item (ignored if replace = FALSE)

Details

The `fn_sample()` is used to return a function that, when called from within a jsPsych experiment, will mirror the behaviour of the `sample()` function from the base package using th jsPsych randomisation functions to at runtime. The input argument `x` specifies the set of values from which samples should be drawn, and the `size` argument specifies the number of samples to be drawn. When `replace = TRUE` items are sampled with replacement, and when `replace = FALSE` items are sampled without replacement. When sampling with replacement, the `weights` argument can be used to specify unequal sampling probabilities.

The current implementation is limited. It does not work when `x` is a character vector, for example. Note also that the value returned within the jsPsych experiment is always an array (not a scalar), even when `size = 1`.

Value

Returns a javascript function that samples from an array of values

---

`fullscreen` *Toggle fullscreen mode in the browser*

Description

The `fullscreen` function is used to toggle fullscreen mode in the browser.

Usage

```
fullscreen(
  fullscreen_mode = TRUE,

   message = "<p>The experiment will switch to full screen mode when you press the button below</p>",
  button_label = "Continue",
  delay_after = 1000
)
```

Arguments

`fullscreen_mode`
: If TRUE, sets browser in full screen mode. Default: TRUE.

`message`
: This is the message that is displayed in the browser to inform of the switch to fullscreen mode.

`button_label`
: This is label of the button to acknowledge the switch.

`delay_after`
: Time period in ms after the switch to proceed with the following element in the timeline.

`insert_javascript` *Insert input as raw javascript*

Description

Insert input as raw javascript

Usage

```
insert_javascript(string)
```

Arguments

`string`
: a string to be interpreted as javascript code

Details

As much as possible, the jaysire package has been designed to allow the user to write a behavioural experiment from R that runs through the browser using the jsPsych javascript library, with no need to write any javascript code. However, in some cases this will not be possible and the user may need to pass raw javascript code to the experiment (e.g., when specifying an "on_finish" callback function). To do so, the javascript should be specified as a `string` that is passed to `insert_javascript()`. What this does is assign the string to the S3 class "json", which in turn means that it will be written to the "experiment.js" file as is.

Value

An object of class "json"

insert_property *Insert a property to the jsPsych data store*

Description

Insert a property to the jsPsych data store

Usage

```
insert_property(...)
```

Arguments

`...`	Name/value pairs

Details

The intention behind `insert_property()` is that it be used when adding new columns to the jsPsych data store. This can be done in two ways. First, it can occur as part of the call to `build_experiment()`. Including an argument of the form `column = insert_property(column_name = "constant value")` will insert a new column to the jsPsych data store whose value is "constant value" in every row.

The second possible way to use it is during a call to a `trial_` function. Including an argument of the form `data = insert_property(column_name = "this value")` will insert "this value" as the value for the current row only.

Note that, at present `insert_property()` simply returns a named list of its inputs. In future versions of jaysire it may have more functionality, but at the moment it is simply a call to `list()`

Value

A list of data values to add to the data store

insert_resource *Insert input as path to a resource file*

Description

Insert input as path to a resource file

Usage

```
insert_resource(file, type = NULL)
```

Arguments

`file`	A character vector of file names
`type`	A character vector of file types (if `NULL`, type is guessed from file extension)

Details

The `insert_resource()` function is designed to take a vector of filenames as input (the `file` argument), categorise files depending on their `type` ("audio", "video", "image", "script", "style" or "other") and construct the path to where those files will end up in the final experiment.

The logic for including this functionality is as follow. Because jsPsych experiments are designed to run through the browser rather than within R, the jaysire package incorporates "resource files" in a slightly complicated way. Resource files here are divided into several categories because the experiment has to incorporate them in different ways: the code for handling images is different to the code for handling audio files or video files, and both are different to how scripts and style files are loaded. As a consequence, the `build_experiment()` function needs to know what kind of file each resource corresponds to in order to construct the experiment properly.

When using jaysire, the `insert_resource()` function is generally used when building trials, and serves as a kind of "promissory note" to specify where the relevant files *will be* when the experiment is constructed using `build_experiment()`. In contrast `build_resources()` is generally used when calling `build_experiment()`, and is in essence a set of "instructions" that `build_experiment()` can use to ensure that this promise is kept.

Value

A character vector of file paths specified relative to the location of the main "index.html" file for the experiment.

See Also

`build_resources`, `build_experiment`

`insert_variable` *Insert reference to a timeline variable*

Description

Insert reference to a timeline variable

Usage

```
insert_variable(name)
```

Arguments

`name`	String specifying name of the variable to insert

Details

When creating an experiment, a common pattern is to create a series of trials that are identical in every respect except for one thing that varies across the trial (e.g., a collection of `trial_html_button_response()` trials that are the same except for the text that is displayed). A natural way to handle this in the jsPsych framework is to create the trial in the usual fashion, except that instead of specifying the *value* that needs to be included in the trial (e.g., the text itself) the code includes a reference to a *timeline variable*. This is the job of the `insert_resource()` function. As an example, instead of creating a trial in which `stimulus = "cat"` and another one that is identical except that `stimulus = "dog"`, you could create a "template" for both trials by setting `stimulus = insert_variable("animal")`. This acts as a kind promise that is filled (at runtime) by looking for an "animal" variable attached to the timeline. See the examples section for an illustration of how these functions are intended to work together.

Value

Javascript code that calls the jsPsych.timelineVariable() function

See Also

`set_variables()`, `build_timeline()`

Examples

```
# create a template from which a series of trials can be built
template <- trial_html_button_response(stimulus = insert_variable("animal"))

# create a timeline with three trials, all using the same template
# but with a different value for the "animal" variable
timeline <- build_timeline(template) %>%
  set_variables(animal = c("cat", "dog", "pig"))
```

`keycode` *Javascript key codes*

Description

Javascript key codes

Usage

```
keycode(key = NULL, code = NULL)
```

Arguments

`key`	character vector specifying keynames (default = NULL)
`code`	numeric vector specfiying keycodes (default = NULL)

Details

This function provides a mapping between the human-readable javascript `key` names and their corresponding numeric `code` values. If both input arguments are `NULL`, it returns a named numeric vector whose values correspond to the key codes and whose names correspond to the key names. If `key` is specified the return value is a vector with the corresponding numeric codes; whereas if `code` is specified the output is a character vector containing the corresponding key names. If neither argument is `NULL` the function throws an error.

Value

A numeric or character vector

`pavlovia` *Communication with pavlovia.org*

Description

The `pavlovia` plugin supports running experiments online in Pavlovia.

Usage

```
pavlovia(
  command = js_string("init"),
  participantId = NULL,
  errorCallback = NULL
)
```

Arguments

`command`	The pavlovia command: "init" (default) or "finish".
`participantId`	The participant Id: any string (NULL by default).
`errorCallback`	The callback function called whenever an error occurs (NULL by default).

`question_likert` *Create a Likert question*

Description

Create a Likert question

Usage

```
question_likert(prompt, labels, required = FALSE, name = NULL)
```

Arguments

`prompt`	the prompt for the question
`labels`	the labels on the Likert scale
`required`	is a response to the question required?
`name`	a convenient label for the question

Details

The `question_likert()` function is designed to be called when using `trial_survey_likert()` to construct a survey page that contains Likert scale response items. When rendered as part of the study, the text specified by the `prompt` argument is shown to the participant, with a set of ordered categories displayed along a horizontal line. The `labels` for these categories are shown beneath the line, and the participant responds by selecting a radio button that is placed along the line. If `required = TRUE` the participant will not be allowed to continue to the next trial unless an answer is provided.

The `name` argument should be a string that provides a convenient label for the question. If left unspecified, jsPsych defaults to labelling the questions within a survey page as "Q0", "Q1", "Q2", etc.

Value

A question object to be passed to `trial_survey_likert()`.

See Also

Survey page trials are constructed using the `trial_survey_text`, `trial_survey_likert`, `trial_survey_multi_choice` and `trial_survey_multi_select` functions. Individual questions for survey trials can be specified using `question_text`, `question_likert` and `question_multi`.

`question_multi` *Create a multiple choice/select question*

Description

Create a multiple choice/select question

Usage

```
question_multi(
  prompt,
  options,
  horizontal = FALSE,
  required = FALSE,
  name = NULL
)
```

Arguments

`prompt`	the prompt for the question
`options`	character vector of options
`horizontal`	should radio buttons be laid out horizontally?
`required`	is a response to the question required?
`name`	a convenient label for the question

Details

The `question_multi()` function is designed to be called when using `trial_survey_multi_choice()` to construct a survey page that contains multiple choice items, or `trial_survey_multi_select()` to construct one with multi-selection items.

When rendered as part of the study, the text specified by the `prompt` argument is shown to the participant, with a set of options presented either as radio buttons (for a multiple choice trial) or checkboxes (for a multiple selection trial). The text placed adjacent to the response options is specified by the `options` argument, and by default the options are laid out vertically. A horizontal arrangement can be produced by setting `horizontal = TRUE`. If `required = TRUE` the participant will not be allowed to continue to the next trial unless an answer is provided.

The `name` argument should be a string that provides a convenient label for the question. If left unspecified, jsPsych defaults to labelling the questions within a survey page as "Q0", "Q1", "Q2", etc.

Value

A question object to be passed to `trial_survey_multi_choice()` or `trial_survey_multi_select()`.

See Also

Survey page trials are constructed using the trial_survey_text, trial_survey_likert, trial_survey_multi_choice and trial_survey_multi_select functions. Individual questions for survey trials can be specified using question_text, question_likert and question_multi.

---

question_text *Create a free text response question*

Description

Create a free text response question

Usage

```
question_text(
  prompt,
  placeholder = "",
  rows = 1,
  columns = 40,
  required = FALSE,
  name = NULL
)
```

Arguments

prompt	The prompt for the question
placeholder	A string specifying the placeholder text
rows	Number of rows spanned by the text box
columns	Number of columns spanned by the text box
required	Is a response to the question required?
name	A convenient label for the question

Details

The `question_text()` function is designed to be called when using `trial_survey_text()` to construct a survey page that contains free text response items. When rendered as part of the study, the text specified by the `prompt` argument is shown to the participant, with a text box placed underneath into which a response may be typed. The size of the text box can be customised by specifying the number of text `rows` and `columns`. If a `placeholder` string is specified (e.g., `placeholder = "Type your answer here"`) it is displayed in faded text within the box, and will disappear as soon as the participant begins typing the response.

If `required = TRUE` the participant will not be allowed to continue to the next trial unless an answer is provided.

The `name` argument should be a string that provides a convenient label for the question. If left unspecified, jsPsych defaults to labelling the questions within a survey page as "Q0", "Q1", "Q2", etc.

Value

A question object to be passed to `trial_survey_text()`.

See Also

Survey page trials are constructed using the `trial_survey_text`, `trial_survey_likert`, `trial_survey_multi_choice` and `trial_survey_multi_select` functions. Individual questions for survey trials can be specified using `question_text`, `question_likert` and `question_multi`.

`respond_any_key` *Response is accepted with any key press*

Description

Response is accepted with any key press

Usage

```
respond_any_key()
```

Details

Many of the functions within the `trial_` family are designed to allow participants to respond using a key press, generally by specifying a a `choices` argument that indicates which keys will be accepted as valid responses (e.g., `choices = c("f","j")`). There are also cases where you may wish to allow every key to be a valid response (e.g., in situations where "Press any key to continue" is a sensible prompt). In those cases, specifying `choices = respond_any_key()` will produce the desired behaviour.

See Also

`respond_no_key`, `trial_html_keyboard_response`, `trial_image_keyboard_response`, `trial_audio_keyboard_response`, `trial_video_keyboard_response`

`respond_no_key` *Response is not accepted for any key press*

Description

Response is not accepted for any key press

Usage

```
respond_no_key()
```

Details

Many of the functions within the trial_ family are designed to allow participants to respond using a key press, generally by specifying a a choices argument that indicates which keys will be accepted as valid responses (e.g., choices = c("f","j")). There are also cases where you may wish to disable this, so that no key presses will be counted as valid responses (e.g., when a trial runs for a fixed duration but no response is expected). In those cases, specifying choices = respond_no_key() will produce the desired behaviour.

See Also

respond_any_key, trial_html_keyboard_response, trial_image_keyboard_response, trial_audio_keyboard_response, trial_video_keyboard_response

---

run_googlecloud *Deploy a jspsych experiment on google app engine*

Description

Deploy a jspsych experiment on google app engine

Usage

```
run_googlecloud(path, experiment_folder = "experiment", project_id)
```

Arguments

path	Path where the experiment is deployed
experiment_folder	Experiment subfolder
project_id	the google app engine project id

Details

The purpose of the run_googlecloud() function is to make it somewhat easier to deploy a jsPsych experiment to Google App Engine, so that the experiment can run in the cloud rather than on the local machine. The path and experiment_folder arguments specify where the experiment should be deployed, and should be the same that was used when calling build_experiment()) to build the experiment originally. The project_id is the name of the Google App Engine project that will host the experiment.

At present, the functionality of run_googlecloud() is quite limited. All it does is construct the appropriate command that you will need to enter at the terminal. It does not execute that command, nor does it assist you in creating the Google App Engine project itself (it is assumed that the user already has a Google Cloud account and is authorised to deploy to the project)

See Also

save_googlecloud, build_experiment

`run_locally` *Deploy a jspsych experiment locally*

Description

Deploy a jspsych experiment locally

Usage

```
run_locally(
  path,
  experiment_folder = "experiment",
  data_folder = "data",
  port = 8000
)
```

Arguments

`path`	Path where the experiment is deployed
`experiment_folder`	Experiment subfolder
`data_folder`	Data subfolder
`port`	The port to use

Details

The purpose of the `run_locally()` function is to start an R server running (using the plumber package) that will serve the experiment from the local machine. The `path`, `experiment_folder` and `data_folder` arguments specify the location specify where the experiment should be deployed, and should be the same that was used when calling `build_experiment()`) to build the experiment originally. The `port` is the numeric value of the port on which the experiment is served. Once `run_locally()` has been called, a browser window should open showing the relevant page.

There are two reasons to deploy a local experiment using `run_locally()` rather than simply opening the relevant "index.html" file in the browser. The first is for the purpose of saving data. For security reasons, browsers do not generally permit client-side javascript (e.g., the code that runs the jsPsych experiment) to save files to arbitrary locations. For this reason writing the data to file is the job of the R server, not the javascript code that is running through the browser. In other words, if the experiment is deployed locally using the `run_locally()` function, then the `save_locally()` function that used to record data locally will work properly. If, however, the "index.html" file is opened without starting the R server, data will not be saved to file.

The second reason for using `run_locally()` is that it opens up the possibility that an experiment could use server-side R code at runtime. At the moment jaysire does not have any functionality to do so, but in principle there is nothing preventing the R server from playing a more active role when the experiment is running, and future versions of the package may develop this functionality further.

See Also

save_locally, build_experiment

---

run_webserver *Deploy a jspsych experiment to a webserver*

Description

Deploy a jspsych experiment to a webserver

Usage

```
run_webserver(
  path,
  experiment_folder = "experiment",
  ssh,
  keyfile = NULL,
  webserver_configured = FALSE
)
```

Arguments

`path`	Path where the experiment is deployed
`experiment_folder`	Experiment subfolder
`ssh`	ssh connection string to a webserver
`keyfile`	(optional) path to a ssh private key to log in to your webserver
`webserver_configured`	set this to true when you've configured your webserver

Details

The purpose of the `run_webserver()` function is to make it somewhat easier to deploy a jsPsych experiment to a webserver that you control, so that the experiment can run in the cloud rather than on the local machine.

For this to work, you need to configure your webserver first. On a recent ubuntu image (available from most cloud providers), you can install apache and php with "`sudo apt install apache2 php`". You should probably also secure the connection between the webserver and the participants with https. If you have configured a domain name for your server, it is pretty simple to enable https via let's encrypt: "`sudo add-apt-repository ppa:certbot/certbot`", "`sudo apt install certbot python-certbot-apache`", and "`sudo certbot --apache`"

Used together with save_webserver, this will set up a folder on the server that apache can write to, and a little php script to receive said data (as recommended on the jspsych website).

See Also

save_webserver, build_experiment, download_webserver

Examples

```
## Not run:
build_experiment(..., on_finish = save_webserver())
run_webserver(ssh = "user@server.com", keyfile = "~/.ssh/id_rsa")
download_webserver(ssh = "user@server.com", keyfile = "~/.ssh/id_rsa")

## End(Not run)
```

---

save_googlecloud *Return a javascript function to save data to Google datastore*

Description

Return a javascript function to save data to Google datastore

Usage

```
save_googlecloud()
```

Details

The purpose of the `save_googlecloud()` is to return a javascript function that, when called from within the jsPsych experiment, will write the data to the Google datastore. The intention is that when an experiment is to be deployed on Google App Engine (i.e., using the `run_googlecloud()` function to deploy the experiment), the `save_googlecloud()` function provides the mechanism for saving the data. If the goal is simply to save the data set at the end of the experiment, the easiest way to do this is when building the experiment using `build_experiment()`. Specifically, the method for doing this is to include the argument `on_finish = save_googlecloud()` as part of the call to `build_experiment()`.

Value

A javascript function to write data to the Google datastore

See Also

run_googlecloud, build_experiment

save_locally — *Return a javascript function to save data locally*

Description

Return a javascript function to save data locally

Usage

```
save_locally()
```

Details

The purpose of the `save_locally()` is to return a javascript function that, when called from within the jsPsych experiment, will write the data to a CSV file on the local machine (in the data folder associated with the experiment). The intention is that when an experiment is to be deployed locally (i.e., using the `run_locally()` function to run the experiment using an R server on the local machine), the `save_locally()` function provides the mechanism for saving the data. If the goal is simply to save the data set at the end of the experiment, the easiest way to do this is when building the experiment using `build_experiment()`. Specifically, the method for doing this is to include the argument `on_finish = save_locally()` as part of the call to `build_experiment()`.

Value

A javascript function to save data locally

See Also

`run_locally`, `build_experiment`

save_webserver — *Return a javascript function to save data via a script on the webserver*

Description

Return a javascript function to save data via a script on the webserver

Usage

```
save_webserver()
```

Details

The purpose of the `save_webserver()` is to return a javascript function that, when called from within the jsPsych experiment, will write the data to the server. This assumes that the experiment will be run on a (php)script-enabled webserver. This way, you know the data will never touch any other computer than the server you've presumably secured and have data processing agreements in place for.

Value

Return a javascript function to save data via a script on the webserver

See Also

run_webserver, download_webserver

set_parameters *Modify a timeline to set parameter values*

Description

Modify a timeline to set parameter values

Usage

```
set_parameters(timeline, ...)
```

Arguments

timeline	The timeline object
...	A set of name/value pairs defining the parameters

Details

The set_parameters() function provides a general purpose method of adding arbitrary parameters to an existing timeline. Anything that jsPsych recognises as a possible timeline parameter can be inserted using this method. Some possibilities are shown in the examples section.

Value

The modified timeline object

See Also

build_timeline, set_variables

Examples

```
# typically we begin with a trial template:
trial_template <- trial_html_button_response(
     stimulus = insert_variable(name = "my_stimulus"),
     choices = c("true", "false")
 )

 # then we fill it out so that there is now a "block" of trials:
 equations <- c("13 + 23 = 36",  "17 - 9 = 6", "125 / 5 = 25")
 trials <- build_timeline(trial_template) %>%
```

```
    set_variables(my_stimulus = equations)

# we can randomise presentation order and repeat the block:
trials <- trials %>%
    set_parameters(randomize_order = TRUE, repetitions = 2)
```

---

set_variables *Modify a timeline to set possible values for variables*

Description

Modify a timeline to set possible values for variables

Usage

```
set_variables(timeline, ...)
```

Arguments

timeline	The timeline object
...	A set of name/value pairs defining the timeline variables

Details

When creating an experiment, a common pattern is to create a series of trials that are identical in every respect except for one thing that varies across the trial (e.g., a collection of trial_html_button_response() trials that are the same except for the text that is displayed). A natural way to handle this in the jsPsych framework is to create the trial in the usual fashion, except that instead of specifying the *value* that needs to be included in the trial (e.g., the text itself) the code includes a reference to a *timeline variable*. Inserting the *reference* to the variable is the job of the insert_variable() function; *attaching* that variable to the timeline and specifying its possible values is the job of set_variables. This is most easily explained by using an example, as shown below.

Value

The modified timeline object

See Also

build_timeline, insert_variable

Examples

```
# create a template from which a series of trials can be built
template <- trial_html_button_response(stimulus = insert_variable("animal"))

# create a timeline with three trials, all using the same template
# but with a different value for the "animal" variable
timeline <- build_timeline(template) %>%
  set_variables(animal = c("cat", "dog", "pig"))
```

`temporary_folder` *Creates a temporary folder*

Description

Creates a temporary folder

Usage

```
temporary_folder()
```

Details

The `temporary_folder()` function is a convenience function used to create a new temporary folder inside the temporary directory (see `tempdir`) for the current R session. The name of the subfolder is always "jaysire_" followed by a 5-character alphanumeric string.

The purpose of this function is mostly expository: it makes it a little easier to create easy-to-follow tutorials on the package website. It is not expected that users of the jaysire package would have much need for this function

Value

A string specifying the path to the folder

`trial_animation` *Specify an animation trial*

Description

The `trial_animation` function is used to display a sequence of images at a fixed rate.

Usage

```
trial_animation(
  stimuli,
  frame_time = 250,
  frame_isi = 0,
  sequence_reps = 1,
  choices = respond_any_key(),
  prompt = NULL,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimuli`	A vector of paths to the image files
`frame_time`	How long to display each image, in milliseconds
`frame_isi`	How long is the gap between images, in milliseconds
`sequence_reps`	How many times to repeat the sequence
`choices`	A character vector of keycodes (either numeric values or the characters themselves). Alternatively, respond_any_key() and respond_no_key() can be used
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).
`post_trial_gap`	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
`on_finish`	A javascript callback function to execute when the trial finishes
`on_load`	A javascript callback function to execute when the trial begins, before any loading has occurred
`data`	An object containing additional data to store for the trial

Details

This function is used to specify an "animation" trial in a jsPsych experiment. An animation trial displays a sequence of images at a fixed frame rate and the sequence can be looped a specified number of times. The participant is free to respond at any point during the animation, and the time of the response is recorded.

Stimulus display:

The only required argument is `stimulus`, which should be a vector of paths to the image files, one per frame. The file paths should refer to the locations of the image files at the time the experiment is *deployed*, so it is often convenient to use the `insert_resource` function to construct these file paths automatically. The images will be displayed in the order that they appear in the `stimulus` vector.

The behaviour of an animation trial can be customised in various ways. The `frame_time` parameter specifies the length of time (in milliseconds) that each image stays on screen, and the frame_isi parameter controls the inter-stimulus interval (that is, the gap between successive images) during which a blank screen is shown. The sequence_reps argument specifies the number of times the sequence repeats.

Response mechanism:

Because animation trials typically require precise timing, they are designed to accept key press responses only, and choices argument is used to control which keys will register a valid response. The default value choices = respond_any_key() is to allow the participant to press any key to register their response. Alternatively it is possible to set choices = respond_no_key(), which prevents all keys from registering a response: this can be useful if the trial is designed to run for a fixed duration, regardless of what the participant presses.

In many situations it is preferable to require the participant to respond using specific keys (e.g., for a binary choice tasks, it may be desirable to require participants to press F for one response or J for the other). This can be achieved in two ways. One possibility is to use a character vector as input (e.g., choices = c("f","j")). The other is to use the numeric code that specifies the desired key in javascript, which in this case would be choices = c(70, 74). To make it a little easier to work with numeric codes, the jaysire package includes the keycode() function to make it easier to convert from one format to the other.

Other behaviour:

The prompt argument is used to specify text that remains on screen while the animation displays. The intended use is to remind participants of the valid response keys, but it allows HTML markup to be included and so can be used for more general purposes.

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The animation_sequence value is an array encoded in JSON format. Each element of the array is an object that represents a stimulus in the animation sequence. Each object has a stimulus property, which is the image that was displayed, and a time property, which is the time in ms, measured from when the sequence began, that the stimulus was displayed.
- The responses value is an array encoded in JSON format. Each element of the array is an object representing a response given by the subject. Each object has a stimulus property,

indicating which image was displayed when the key was pressed, an rt property, indicating the time of the key press relative to the start of the animation, and a key_press property, indicating which key was pressed.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

`trial_audio_button_response` *Specify an audio trial with button response*

Description

The `trial_audio_button_response` function is used to play an audio stimulus and collect a response using on screen buttons.

Usage

```
trial_audio_button_response(
  stimulus,
  choices = c("button 0", "button 1", "button 2"),
  button_html = NULL,
  margin_vertical = "0px",
  margin_horizontal = "8px",
  prompt = NULL,
  trial_ends_after_audio = FALSE,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

- `stimulus` Path to the audio file to be played
- `choices` Labels for the buttons. Each element of the character vector will generate a different button.
- `button_html` A template of HTML for generating the button elements (see details)
- `margin_vertical` Vertical margin of the buttons
- `margin_horizontal` Horizontal margin of the buttons
- `prompt` A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).
- `trial_ends_after_audio` Does the trial end after the audio finishes playing? (default = FALSE)
- `trial_duration` How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.
- `response_ends_trial` If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.
- `post_trial_gap` The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
- `on_finish` A javascript callback function to execute when the trial finishes
- `on_load` A javascript callback function to execute when the trial begins, before any loading has occurred
- `data` An object containing additional data to store for the trial

Details

The `trial_audio_button_response` function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one plays audio files and records responses generated with a button click.

Stimulus display: If the browser supports it, audio files are played using the WebAudio API. This allows for reasonably precise timing of the playback. The timing of responses generated is measured against the WebAudio specific clock, improving the measurement of response times. If the browser does not support the WebAudio API, then the audio file is played with HTML5 audio.

Response mechanism: The response buttons can be customized using HTML formatting, via the `button_html` argument. This argument allows the user to specify an HTML used to generating the button elements. If this argument is a vector of the same length as `choices` then the i-th element of `button_html` will be used to define the i-th response button. If `button_html` is a single string then the same template will be applied to every button. The templating is defined using a special string "%choice%" that will be replaced by the corresponding element of the `choices` vector. By default the jsPsych library creates an HTML button of class "jspsych-btn" and the styling is governed by the corresponding CSS.

Other behaviour: The trial can end when the subject responds (`response_ends_trial = TRUE`), when the audio file has finished playing (`trial_ends_after_audio = TRUE`), or if the subject has failed to respond within a fixed length of time (specified using the `trial_duration` argument).

Like all functions in the `trial_` family it contains four additional arguments:

- The `post_trial_gap` argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the `build_experiment` function, and a blank screen is displayed during this gap period.
- The `on_load` and `on_finish` arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the `insert_javascript` function. However, the `fn_` family of functions supplies a variety of functions that may be useful in many cases.
- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when `build_experiment` is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `rt` value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The `button_pressed` variable is a numeric value indicating which button was pressed. The first button in the choices array is recorded as value 0, the second is value 1, and so on.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are `trial_image_button_response`, `trial_image_keyboard_response`, `trial_image_slider_response`, `trial_audio_button_response`, `trial_audio_keyboard_response`, `trial_audio_slider_response`, `trial_video_button_response`, `trial_video_keyboard_response`, `trial_video_slider_response`, `trial_html_button_response`, `trial_html_keyboard_response` and `trial_html_slider_response`.

`trial_audio_keyboard_response`
Specify an audio trial with keyboard response

Description

The `trial_audio_keyboard_response` function is used to play an audio stimulus and collect a response using a key press.

Usage

```
trial_audio_keyboard_response(
  stimulus,
  choices = respond_any_key(),
  prompt = NULL,
  trial_ends_after_audio = FALSE,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimulus`	Path to the audio file to be played
`choices`	A character vector of keycodes (either numeric values or the characters themselves). Alternatively, respond_any_key() and respond_no_key() can be used
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).
`trial_ends_after_audio`	Does the trial end after the audio finishes playing? (default = FALSE)
`trial_duration`	How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.
`response_ends_trial`	If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.
`post_trial_gap`	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
`on_finish`	A javascript callback function to execute when the trial finishes
`on_load`	A javascript callback function to execute when the trial begins, before any loading has occurred
`data`	An object containing additional data to store for the trial

Details

The trial_audio_keyboard_response function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). It plays audio files and records responses generated with a key press.

Stimulus display: If the browser supports it, audio files are played using the WebAudio API. This allows for reasonably precise timing of the playback. The timing of responses generated is measured against the WebAudio specific clock, improving the measurement of response times. If the browser does not support the WebAudio API, then the audio file is played with HTML5 audio.

Response mechanism: For this kind of trial, participants can make a response by pressing a key, and the choices argument is used to control which keys will register a valid response. The default value choices = respond_any_key() is to allow the participant to press any key to register their response. Alternatively it is possible to set choices = respond_no_key(), which prevents all keys from registering a response: this can be useful if the trial is designed to run for a fixed duration, regardless of what the participant presses.

In many situations it is preferable to require the participant to respond using specific keys (e.g., for a binary choice tasks, it may be desirable to require participants to press F for one response or J for the other). This can be achieved in two ways. One possibility is to use a character vector as input (e.g., choices = c("f","j")). The other is to use the numeric code that specifies the desired key in javascript, which in this case would be choices = c(70, 74). To make it a little easier to work with numeric codes, the jaysire package includes the keycode() function to make it easier to convert from one format to the other.

Other behaviour: The trial can end when the subject responds (response_ends_trial = TRUE), when the audio file has finished playing (trial_ends_after_audio = TRUE), or if the subject has failed to respond within a fixed length of time (specified using the trial_duration argument).

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data: When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The rt value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.

- The `key_press` variable is the numeric javascript key code corresponding to the response.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are `trial_image_button_response`, `trial_image_keyboard_response`, `trial_image_slider_response`, `trial_audio_button_response`, `trial_audio_keyboard_response`, `trial_audio_slider_response`, `trial_video_button_response`, `trial_video_keyboard_response`, `trial_video_slider_response`, `trial_html_button_response`, `trial_html_keyboard_response` and `trial_html_slider_response`.

`trial_audio_slider_response`

Specify an audio trial with slider bar response

Description

The `trial_audio_slider_response` function is used to play an audio stimulus and collect a response using a slider bar.

Usage

```
trial_audio_slider_response(
  stimulus,
  labels = c("0%", "25%", "50%", "75%", "100%"),
  button_label = "Continue",
  min = 0,
  max = 100,
  start = 50,
  step = 1,
  slider_width = NULL,
  require_movement = FALSE,
```

```
  prompt = NULL,
  trial_ends_after_audio = FALSE,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimulus`	Path to the audio file to be played
`labels`	Labels displayed at equidistant locations on the slider.
`button_label`	Label placed on the "continue" button
`min`	Minimum value of the slider
`max`	Maximum value of the slider
`start`	Initial value of the slider
`step`	Step size of the slider
`slider_width`	Horizontal width of the slider (defaults to display width)
`require_movement`	Does the user need to move the slider before clicking the continue button?
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).
`trial_ends_after_audio`	If TRUE the trial will end as soon as the audio file finishes playing.
`trial_duration`	How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.
`response_ends_trial`	If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.
`post_trial_gap`	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
`on_finish`	A javascript callback function to execute when the trial finishes
`on_load`	A javascript callback function to execute when the trial begins, before any loading has occurred
`data`	An object containing additional data to store for the trial

Details

The `trial_audio_button_response` belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). It plays audio files and records responses generated with a slider bar.

Stimulus display:

If the browser supports it, audio files are played using the WebAudio API. This allows for reasonably precise timing of the playback. The timing of responses generated is measured against the WebAudio specific clock, improving the measurement of response times. If the browser does not support the WebAudio API, then the audio file is played with HTML5 audio.

Response mechanism:

Participant responses for this trial type are collected using a slider bar that the participant can move using the mouse. Once the participant is happy with this positioning they can click a button at the bottom of the page to move on to the next trial. This response method can be customised in several ways depending on the following arguments:

- The `min` and `max` arguments are numeric values that specify the minimum value (leftmost point on the slider) and the maximum value (rightmost point on the slider) that a participant can respond with.
- The `start` parameter is a numeric value that indicates where the value of the the slider is initially position. By default this is set to the middle of the scale, but there are many cases where it may be sensible to have the slider bar start at one end of the scale.
- The movement of the slider is discretised, and the granularity of this movement can be customised using the `step` parameter. This should be a numeric value that specifies the smallest possible increment that the participant can move the slider in either direction.
- The text labels displayed below the slider bar can also be customised by specifying the `labels` parameter. This argument should be a character vector that contains the labels to be displaed. Labels will be displayed at equally spaced intervals along the slider, though it is possible to include blank labels to create the impression of unequal spacing if that is required.
- The `slider_width` controls the horizontal width of the slider bar: the default value of `NULL` creates a slider that occupies 100% of the width of the jsPsych display. Note that this may not be 100% of the screen width.
- To ensure that participants do engage with the slider, it is possible to set `require_movement = TRUE` which forces the participant to move the slider at least once in order to be permitted to move onto the next trial.
- The `button_label` argument specifies the text displayed on the button that participants click to move to the next trial.

Other behaviour:

As is the case for most `trial_` functions there is a `prompt` argument, a string that specifies additional text that is displayed on screen during the trial. The value of `prompt` can contain HTML markup, allowing it to be used quite flexibly if needed.

The trial can end when the subject responds (`response_ends_trial = TRUE`), when the audio file has finished playing (`trial_ends_after_audio = TRUE`), or if the subject has failed to respond within a fixed length of time (specified using the `trial_duration` argument).

Like all functions in the `trial_` family it contains four additional arguments:

- The `post_trial_gap` argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the `build_experiment` function, and a blank screen is displayed during this gap period.

- The `on_load` and `on_finish` arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the `insert_javascript` function. However, the `fn_` family of functions supplies a variety of functions that may be useful in many cases.
- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when `build_experiment` is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `rt` value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The `response` is the numeric value of the slider bar.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are `trial_image_button_response`, `trial_image_keyboard_response`, `trial_image_slider_response`, `trial_audio_button_response`, `trial_audio_keyboard_response`, `trial_audio_slider_response`, `trial_video_button_response`, `trial_video_keyboard_response`, `trial_video_slider_response`, `trial_html_button_response`, `trial_html_keyboard_response` and `trial_html_slider_response`.

trial_categorize_animation

Specify a categorization trial with an animated stimulus

Description

The `trial_categorize_animation` function is used to display a sequence of images at a fixed rate, collect a categorization response with the keyboard, and provide feedback.

Usage

```
trial_categorize_animation(
  stimuli,
  key_answer,
  choices = respond_any_key(),
  text_answer = "",
  correct_text = "Correct",
  incorrect_text = "Wrong",
  frame_time = 500,
  sequence_reps = 1,
  allow_response_before_complete = FALSE,
  prompt = NULL,
  feedback_duration = 2000,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimuli`	Character vector of paths to image files
`key_answer`	The numeric key code indicating the correct response
`choices`	A character vector of keycodes (either numeric values or the characters themselves). Alternatively, respond_any_key() and respond_no_key() can be used
`text_answer`	A label associated with the correct answer
`correct_text`	Text to display when correct answer given ('%ANS%' substitutes text_answer)
`incorrect_text`	Text to display when wrong answer given ('%ANS%' substitutes text_answer)
`frame_time`	How long to display each image, in milliseconds
`sequence_reps`	How many times to show the entire sequence
`allow_response_before_complete`	If TRUE the user can respond before the animation sequence finishes
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).

`feedback_duration`
: How long to show the feedback, in milliseconds

`post_trial_gap`
: The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.

`on_finish`
: A javascript callback function to execute when the trial finishes

`on_load`
: A javascript callback function to execute when the trial begins, before any loading has occurred

`data`
: An object containing additional data to store for the trial

Details

The `trial_categorize_animation` function is used to show a sequence of images at a specified frame rate. The subject responds by pressing a key. Feedback indicating the correctness of the response is given.

Stimulus display:

The `stimulus` argument should be a vector of paths to the image files, one per frame. The file paths should refer to the locations of the image files at the time the experiment is *deployed*, so it is often convenient to use the `insert_resource` function to construct these file paths automatically. The images will be displayed in the order that they appear in the `stimulus` vector.

The behaviour of an animation trial can be customised in various ways. The `frame_time` parameter specifies the length of time (in milliseconds) that each image stays on screen. The `sequence_reps` argument specifies the number of times the sequence repeats.

Response mechanism: For this kind of trial, participants can make a response by pressing a key, and the `choices` argument is used to control which keys will register a valid response. The default value `choices = respond_any_key()` is to allow the participant to press any key to register their response. Alternatively it is possible to set `choices = respond_no_key()`, which prevents all keys from registering a response: this can be useful if the trial is designed to run for a fixed duration, regardless of what the participant presses.

In many situations it is preferable to require the participant to respond using specific keys (e.g., for a binary choice tasks, it may be desirable to require participants to press F for one response or J for the other). This can be achieved in two ways. One possibility is to use a character vector as input (e.g., `choices = c("f","j")`). The other is to use the numeric code that specifies the desired key in javascript, which in this case would be `choices = c(70, 74)`. To make it a little easier to work with numeric codes, the jaysire package includes the `keycode()` function to make it easier to convert from one format to the other.

If `allow_response_before_correct = TRUE` the participant is permitted to make a response before the stimulus display completes.

Feedback:

In a categorisation trial, there is always presumed to be a "correct" response for any given stimulus, and the participant is presented with feedback after the response is given. This feedback can be customised in several ways:

- The `key_answer` argument specifies the numeric `keycode` that corresponds to the correct response for the current trial.

- The `correct_text` and `incorrect_text` arguments are used to customise the feedback text that is presented to the participant after a response is given. In both cases, there is a special value `"%ANS%"` that can be used, and will be substituted with the value of `text_answer`. For example if we set `text_answer = "WUG"`, we could then set `correct_text = "Correct! This is a %ANS%"` and `incorrect_text = "Wrong. This is a %ANS%"`. This functionality can be particularly useful if the values of `text_answer` and `stimulus` are specified using timeline variables (see `insert_variable()` and `set_variables()`).
- The `feedback_duration` argument specifies the length of time the feedback is displayed, in milliseconds.

Other behaviour:

The `prompt` argument is used to specify text that remains on screen while the animation displays. The intended use is to remind participants of the valid response keys, but it allows HTML markup to be included and so can be used for more general purposes.

Like all functions in the `trial_` family it contains four additional arguments:

- The `post_trial_gap` argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the `build_experiment` function, and a blank screen is displayed during this gap period.
- The `on_load` and `on_finish` arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the `insert_javascript` function. However, the `fn_` family of functions supplies a variety of functions that may be useful in many cases.
- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when `build_experiment` is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `stimulus` value is a JSON encoded representation of the array of stimuli displayed in the trial.
- The `key_press` value indicates which key the subject pressed. The value is the numeric key code corresponding to the subject's response.
- The `rt` value is the response time in milliseconds for the subject to make a response. The time is measured from when the stimulus first appears on the screen until the subject's response.
- The `correct` value is true if the subject got the correct answer, false otherwise.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

There are three types of categorization trial, corresponding to the `trial_categorize_animation`, `trial_categorize_html` and `trial_categorize_image` functions.

`trial_categorize_html` *Specify a categorization trial with an HTML stimulus*

Description

The `trial_categorize_html` function is used to display an HTML stimulus, collect a categorization response with the keyboard, and provide feedback.

Usage

```
trial_categorize_html(
  stimulus,
  key_answer,
  choices = respond_any_key(),
  text_answer = "",
  correct_text = "Correct",
  incorrect_text = "Wrong",
  prompt = NULL,
  force_correct_button_press = FALSE,
  show_stim_with_feedback = TRUE,
  show_feedback_on_timeout = FALSE,
  timeout_message = "Please respond faster",
  stimulus_duration = NULL,
  feedback_duration = 2000,
  trial_duration = NULL,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimulus`	The HTML to be displayed.
`key_answer`	The numeric key code indicating the correct response

`choices`	A character vector of keycodes (either numeric values or the characters themselves). Alternatively, respond_any_key() and respond_no_key() can be used
`text_answer`	A label associated with the correct answer
`correct_text`	Text to display when correct answer given ('%ANS%' substitutes text_answer)
`incorrect_text`	Text to display when wrong answer given ('%ANS%' substitutes text_answer)
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).
`force_correct_button_press`	If TRUE the correct button must be pressed after feedback in order to advance
`show_stim_with_feedback`	If TRUE the stimulus image will be displayed as part of the feedback. Otherwise only text is shown
`show_feedback_on_timeout`	If TRUE the "wrong answer" feedback will be presented on timeout. If FALSE, a timeout message is shown
`timeout_message`	The message to show on a timeout non-response
`stimulus_duration`	How long to show the stimulus, in milliseconds. If NULL, then the stimulus will be shown until the subject makes a response
`feedback_duration`	How long to show the feedback, in milliseconds
`trial_duration`	How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.
`post_trial_gap`	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
`on_finish`	A javascript callback function to execute when the trial finishes
`on_load`	A javascript callback function to execute when the trial begins, before any loading has occurred
`data`	An object containing additional data to store for the trial

Details

The `trial_categorize_html` function is used to show an HTML object on the screen. The subject responds by pressing a key. Feedback indicating the correctness of the response is given.

Stimulus display: The `stimulus` argument is a string specifying the text to be displayed as the stimulus. It can include HTML markup, meaning that it can be used to any stimulus that can be specified using HTML. It remains on screen for a length of time corresponding to the `stimulus_duration` parameter in milliseconds (or indefinitely if the parameter is `NULL`).

Response mechanism: For this kind of trial, participants can make a response by pressing a key, and the `choices` argument is used to control which keys will register a valid response. The default value `choices = respond_any_key()` is to allow the participant to press any key to register their response. Alternatively it is possible to set choices = respond_no_key(), which prevents all keys from registering a response: this can be useful if the trial is designed to run for a fixed duration, regardless of what the participant presses.

In many situations it is preferable to require the participant to respond using specific keys (e.g., for a binary choice tasks, it may be desirable to require participants to press F for one response or J for the other). This can be achieved in two ways. One possibility is to use a character vector as input (e.g., choices = c("f","j")). The other is to use the numeric code that specifies the desired key in javascript, which in this case would be choices = c(70, 74). To make it a little easier to work with numeric codes, the jaysire package includes the keycode() function to make it easier to convert from one format to the other.

Feedback:

In a categorisation trial, there is always presumed to be a "correct" response for any given stimulus, and the participant is presented with feedback after the response is given. This feedback can be customised in several ways:

- The key_answer argument specifies the numeric keycode that corresponds to the correct response for the current trial.
- The correct_text and incorrect_text arguments are used to customise the feedback text that is presented to the participant after a response is given. In both cases, there is a special value "%ANS%" that can be used, and will be substituted with the value of text_answer. For example if we set text_answer = "WUG", we could then set correct_text = "Correct! This is a %ANS%" and incorrect_text = "Wrong. This is a %ANS%". This functionality can be particularly useful if the values of text_answer and stimulus are specified using timeline variables (see insert_variable() and set_variables()).
- The force_correct_button_press argument is a logical variable. If set to TRUE the participant cannot move forward to the next trial until the correct response is given.
- When show_stim_with_feedback = TRUE, the stimulus remains on screen while the feedback is presented. If it is set to FALSE the stimulus is not visible.
- Sometimes a categorisation trial has a deadline, specified by the value of trial_duration. If a response is not given by that time, the trial ends. Optionally, a feedback screen can be presented whenever this occurs, by setting show_feedback_on_timeout = FALSE, and the text of this feedback is specified by using the timeout_message argument.

Other behaviour:

The prompt argument is used to specify text that remains on screen while the animation displays. The intended use is to remind participants of the valid response keys, but it allows HTML markup to be included and so can be used for more general purposes.

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.

- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when `build_experiment` is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `stimulus` value is the HTML displayed on the trial.
- The `key_press` value indicates which key the subject pressed. The value is the numeric key code corresponding to the subject's response.
- The `rt` value is the response time in milliseconds for the subject to make a response. The time is measured from when the stimulus first appears on the screen until the subject's response.
- The `correct` value is true if the subject got the correct answer, false otherwise.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

There are three types of categorization trial, corresponding to the `trial_categorize_animation`, `trial_categorize_html` and `trial_categorize_image` functions.

`trial_categorize_image` *Specify a categorization trial with an image stimulus*

Description

The `trial_categorize_image` function is used to display an image stimulus, collect a categorization response with the keyboard, and provide feedback.

Usage

```
trial_categorize_image(
  stimulus,
  key_answer,
  choices = respond_any_key(),
  text_answer = "",
  correct_text = "Correct",
  incorrect_text = "Wrong",
  prompt = NULL,
  force_correct_button_press = FALSE,
  show_stim_with_feedback = TRUE,
  show_feedback_on_timeout = FALSE,
  timeout_message = "Please respond faster",
  stimulus_duration = NULL,
  feedback_duration = 2000,
  trial_duration = NULL,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimulus` The path to the image file to be displayed.

`key_answer` The numeric key code indicating the correct response

`choices` A character vector of keycodes (either numeric values or the characters themselves). Alternatively, respond_any_key() and respond_no_key() can be used

`text_answer` A label associated with the correct answer

`correct_text` Text to display when correct answer given ('%ANS%' substitutes text_answer)

`incorrect_text` Text to display when wrong answer given ('%ANS%' substitutes text_answer)

`prompt` A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).

`force_correct_button_press`
If TRUE the correct button must be pressed after feedback in order to advance

`show_stim_with_feedback`
If TRUE the stimulus image will be displayed as part of the feedback. Otherwise only text is shown

`show_feedback_on_timeout`
If TRUE the "wrong answer" feedback will be presented on timeout. If FALSE, a timeout message is shown

`timeout_message`
The message to show on a timeout non-response

`stimulus_duration`
How long to show the stimulus, in milliseconds. If NULL, then the stimulus will be shown until the subject makes a response

feedback_duration	How long to show the feedback, in milliseconds
trial_duration	How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.
post_trial_gap	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
on_finish	A javascript callback function to execute when the trial finishes
on_load	A javascript callback function to execute when the trial begins, before any loading has occurred
data	An object containing additional data to store for the trial

Details

The `trial_categorize_image` function is used to show an image object on the screen. The subject responds by pressing a key. Feedback indicating the correctness of the response is given.

Stimulus display: For trials that display an image, the `stimulus` argument is a string that specifies the path to the image file. More precisely, it must specify the path to where the image file will be located at the time the experiment runs. Typically, if an experiment is deployed using the `build_experiment()` function all resource files will be stored in a "resource" folder, and the images will be copied to the "image" subfolder. So if the image to be displayed is a file called "picture.png", the `stimulus` path on a Mac or Linux machine would likely be "resource/image/picture.png". Note that this path is specified relative to the location of the primary experiment file "image.html". To make this a little easier, the `insert_resource()` function can be used to construct resource paths automatically. In the example above, `stimulus = insert_resource("picture.png")` would suffice.

Response mechanism: For this kind of trial, participants can make a response by pressing a key, and the `choices` argument is used to control which keys will register a valid response. The default value `choices = respond_any_key()` is to allow the participant to press any key to register their response. Alternatively it is possible to set `choices = respond_no_key()`, which prevents all keys from registering a response: this can be useful if the trial is designed to run for a fixed duration, regardless of what the participant presses.

In many situations it is preferable to require the participant to respond using specific keys (e.g., for a binary choice tasks, it may be desirable to require participants to press F for one response or J for the other). This can be achieved in two ways. One possibility is to use a character vector as input (e.g., `choices = c("f","j")`). The other is to use the numeric code that specifies the desired key in javascript, which in this case would be `choices = c(70, 74)`. To make it a little easier to work with numeric codes, the jaysire package includes the `keycode()` function to make it easier to convert from one format to the other.

Feedback:

In a categorisation trial, there is always presumed to be a "correct" response for any given stimulus, and the participant is presented with feedback after the response is given. This feedback can be customised in several ways:

- The `key_answer` argument specifies the numeric `keycode` that corresponds to the correct response for the current trial.

- The correct_text and incorrect_text arguments are used to customise the feedback text that is presented to the participant after a response is given. In both cases, there is a special value "%ANS%" that can be used, and will be substituted with the value of text_answer. For example if we set text_answer = "WUG", we could then set correct_text = "Correct! This is a %ANS%" and incorrect_text = "Wrong. This is a %ANS%". This functionality can be particularly useful if the values of text_answer and stimulus are specified using timeline variables (see insert_variable() and set_variables()).
- The force_correct_button_press argument is a logical variable. If set to TRUE the participant cannot move forward to the next trial until the correct response is given.
- When show_stim_with_feedback = TRUE, the stimulus remains on screen while the feedback is presented. If it is set to FALSE the stimulus is not visible.
- Sometimes a categorisation trial has a deadline, specified by the value of trial_duration. If a response is not given by that time, the trial ends. Optionally, a feedback screen can be presented whenever this occurs, by setting show_feedback_on_timeout = FALSE, and the text of this feedback is specified by using the timeout_message argument.

Other behaviour:

The prompt argument is used to specify text that remains on screen while the animation displays. The intended use is to remind participants of the valid response keys, but it allows HTML markup to be included and so can be used for more general purposes.

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The stimulus value is the path to the image file.
- The key_press value indicates which key the subject pressed. The value is the numeric key code corresponding to the subject's response.
- The rt value is the response time in milliseconds for the subject to make a response. The time is measured from when the stimulus first appears on the screen until the subject's response.
- The correct value is true if the subject got the correct answer, false otherwise.

In addition, it records default variables that are recorded by all trials:

- trial_type is a string that records the name of the plugin used to run the trial.

- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

There are three types of categorization trial, corresponding to the `trial_categorize_animation`, `trial_categorize_html` and `trial_categorize_image` functions.

`trial_generic` *Specify a trial using any plugin*

Description

The `trial_generic` function is used to create a trial with an arbitrary jsPsych plugin.

Usage

```
trial_generic(type, ...)
```

Arguments

`type`	the type of trial
`...`	arguments passed to the trial plugin

Details

The `trial_generic()` function is the most flexible of all the functions within the `trial_` family, and can be use to insert a trial of any `type`. For example, by setting `type = "image-keyboard-response"`, it will create an image trial using a keyboard response, precisely analogous to trials created using the `trial_image_keyboard_response()` function. More generally the `type` value should be a string that specifies the name of the corresponding jsPsych plugin file: in this case, the file name for the plugin "jspsych-image-keyboard-response.js" so the corresponding `type` value is "image-keyboard-response".

While the advantage to `trial_generic()` is flexibility, the disadvantage is that all arguments to the plugin must be specified as named arguments passed via `...`, and it can take some trial and error to get a novel plugin to behave in the expected fashion. For example, if a particular argument to the jsPsych plugin takes a logical value, it may not always be sufficient to use logical values `TRUE` or

FALSE when the trial is constructed from within R. The reason for this is that when the R code is converted to javascript (using the jsonlite package), it *does* correctly convert the R logicals TRUE and FALSE to the corresponding javascript logical values true and false, but by default this value is written to a javascript array of length one rather than recorded as a scalar value (i.e., the javascript code becomes [true] rather than true). When this occurs, jsPsych often does not produce the desired behaviour as these two entities are not considered equivalent in javascript.

In future versions of jaysire there may be better support for arbitary plugins, but for the moment users should be aware that trial_generic() can be somewhat finicky to work with.

Value

Functions with a trial_ prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with NULL elements removed. Logical values in the input (TRUE and FALSE) are transformed to character vectors "true" and "false" and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, true and false.

---

trial_html_button_response

Specify an HTML trial with button response

Description

The trial_html_button_response function is used to display an HTML stimulus and collect a response using on screen buttons.

Usage

```
trial_html_button_response(
  stimulus,
  choices = c("button 0", "button 1", "button 2"),
  button_html = NULL,
  margin_vertical = "0px",
  margin_horizontal = "8px",
  prompt = NULL,
  stimulus_duration = NULL,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimulus`	The HTML content to be displayed.
`choices`	Labels for the buttons. Each element of the character vector will generate a different button.
`button_html`	A template of HTML for generating the button elements (see details)
`margin_vertical`	Vertical margin of the buttons
`margin_horizontal`	Horizontal margin of the buttons
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).
`stimulus_duration`	How long to show the stimulus, in milliseconds. If NULL, then the stimulus will be shown until the subject makes a response
`trial_duration`	How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.
`response_ends_trial`	If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.
`post_trial_gap`	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
`on_finish`	A javascript callback function to execute when the trial finishes
`on_load`	A javascript callback function to execute when the trial begins, before any loading has occurred
`data`	An object containing additional data to store for the trial

Details

The `trial_html_button_response` function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one displays HTML and records responses generated with a button click.

Stimulus display: The `stimulus` argument is a string specifying the text to be displayed as the stimulus. It can include HTML markup, meaning that it can be used to any stimulus that can be specified using HTML. It remains on screen for a length of time corresponding to the `stimulus_duration` parameter in milliseconds (or indefinitely if the parameter is `NULL`).

Response mechanism: The response buttons can be customized using HTML formatting, via the `button_html` argument. This argument allows the user to specify an HTML used to generating the button elements. If this argument is a vector of the same length as `choices` then the i-th element of `button_html` will be used to define the i-th response button. If `button_html` is a single string then the same template will be applied to every button. The templating is defined using a special string "%choice%" that will be replaced by the corresponding element of the `choices` vector.

By default the jsPsych library creates an HTML button of class "jspsych-btn" and the styling is governed by the corresponding CSS.

Like all functions in the `trial_` family it contains four additional arguments:

- The `post_trial_gap` argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the `build_experiment` function, and a blank screen is displayed during this gap period.
- The `on_load` and `on_finish` arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the `insert_javascript` function. However, the `fn_` family of functions supplies a variety of functions that may be useful in many cases.
- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial

Other behaviour: Depending on parameter settings, the trial can end when the subject responds (`response_ends_trial = TRUE`), or after a fixed amount of time (specified using the `trial_duration` argument) has elapsed. The length of time that the stimulus remains visible can also be customized using the (`stimulus_duration`) argument.

Like all functions in the `trial_` family it contains four additional arguments:

- The `post_trial_gap` argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the `build_experiment` function, and a blank screen is displayed during this gap period.
- The `on_load` and `on_finish` arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the `insert_javascript` function. However, the `fn_` family of functions supplies a variety of functions that may be useful in many cases.
- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial

Data: When this function is called from R it returns the trial object that will later be inserted into the experiment when `build_experiment` is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `rt` value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The `button_pressed` variable is a numeric value indicating which button was pressed. The first button in the choices array is recorded as value 0, the second is value 1, and so on.
- The `stimulus` variable records the HTML content that was displayed on this trial.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are `trial_image_button_response`, `trial_image_keyboard_response`, `trial_image_slider_response`, `trial_audio_button_response`, `trial_audio_keyboard_response`, `trial_audio_slider_response`, `trial_video_button_response`, `trial_video_keyboard_response`, `trial_video_slider_response`, `trial_html_button_response`, `trial_html_keyboard_response` and `trial_html_slider_response`.

---

trial_html_keyboard_response

Specify an HTML trial with keyboard response

Description

The `trial_html_keyboard_response` function is used to display an HTML stimulus and collect a response using a key press.

Usage

```
trial_html_keyboard_response(
  stimulus,
  choices = respond_any_key(),
  prompt = NULL,
  stimulus_duration = NULL,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimulus`	The HTML content to be displayed.
`choices`	A character vector of keycodes (either numeric values or the characters themselves). Alternatively, respond_any_key() and respond_no_key() can be used
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).

`stimulus_duration`
: How long to show the stimulus, in milliseconds. If NULL, then the stimulus will be shown until the subject makes a response

`trial_duration`
: How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.

`response_ends_trial`
: If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.

`post_trial_gap`
: The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.

`on_finish`
: A javascript callback function to execute when the trial finishes

`on_load`
: A javascript callback function to execute when the trial begins, before any loading has occurred

`data`
: An object containing additional data to store for the trial

Details

The `trial_html_keyboard_response` function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one displays HTML and records responses generated with a key press.

Stimulus display: The `stimulus` argument is a string specifying the text to be displayed as the stimulus. It can include HTML markup, meaning that it can be used to any stimulus that can be specified using HTML. It remains on screen for a length of time corresponding to the `stimulus_duration` parameter in milliseconds (or indefinitely if the parameter is `NULL`).

Response mechanism: For this kind of trial, participants can make a response by pressing a key, and the `choices` argument is used to control which keys will register a valid response. The default value `choices = respond_any_key()` is to allow the participant to press any key to register their response. Alternatively it is possible to set `choices = respond_no_key()`, which prevents all keys from registering a response: this can be useful if the trial is designed to run for a fixed duration, regardless of what the participant presses.

In many situations it is preferable to require the participant to respond using specific keys (e.g., for a binary choice tasks, it may be desirable to require participants to press F for one response or J for the other). This can be achieved in two ways. One possibility is to use a character vector as input (e.g., `choices = c("f","j")`). The other is to use the numeric code that specifies the desired key in javascript, which in this case would be `choices = c(70, 74)`. To make it a little easier to work with numeric codes, the jaysire package includes the `keycode()` function to make it easier to convert from one format to the other.

Other behaviour: Depending on parameter settings, the trial can end when the subject responds (`response_ends_trial = TRUE`), or after a fixed amount of time (specified using the `trial_duration` argument) has elapsed. The length of time that the stimulus remains visible can also be customized using the (`stimulus_duration`) argument.

Like all functions in the `trial_` family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data: When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The rt value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The key_press variable is the numeric javascript key code corresponding to the response.
- The stimulus variable records the HTML content that was displayed on this trial.

In addition, it records default variables that are recorded by all trials:

- trial_type is a string that records the name of the plugin used to run the trial.
- trial_index is a number that records the index of the current trial across the whole experiment.
- time_elapsed counts the number of milliseconds since the start of the experiment when the trial ended.
- internal_node_id is a string identifier for the current "node" in the timeline.

Value

Functions with a trial_ prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with NULL elements removed. Logical values in the input (TRUE and FALSE) are transformed to character vectors "true" and "false" and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, true and false.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are trial_image_button_response, trial_image_keyboard_response, trial_image_slider_response, trial_audio_button_response, trial_audio_keyboard_response, trial_audio_slider_response, trial_video_button_response, trial_video_keyboard_response, trial_video_slider_response, trial_html_button_response, trial_html_keyboard_response and trial_html_slider_response.

trial_html_slider_response

Specify an HTML trial with slider bar response

Description

The `trial_html_slider_response` function is used to display an HTML stimulus and collect a response using a slider bar.

Usage

```
trial_html_slider_response(
  stimulus,
  labels = c("0%", "25%", "50%", "75%", "100%"),
  button_label = "Continue",
  min = 0,
  max = 100,
  start = 50,
  step = 1,
  slider_width = NULL,
  require_movement = FALSE,
  prompt = NULL,
  stimulus_duration = NULL,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimulus`	The HTML content to be displayed.
`labels`	Labels displayed at equidistant locations on the slider.
`button_label`	Label placed on the "continue" button
`min`	Minimum value of the slider
`max`	Maximum value of the slider
`start`	Initial value of the slider
`step`	Step size of the slider
`slider_width`	Horizontal width of the slider (defaults to display width)
`require_movement`	Does the user need to move the slider before clicking the continue button?
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).

stimulus_duration
: How long to show the stimulus, in milliseconds. If NULL, then the stimulus will be shown until the subject makes a response

trial_duration
: How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.

response_ends_trial
: If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.

post_trial_gap
: The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.

on_finish
: A javascript callback function to execute when the trial finishes

on_load
: A javascript callback function to execute when the trial begins, before any loading has occurred

data
: An object containing additional data to store for the trial

Details

The `trial_html_slider_response` function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one displays HTML and records responses generated with a slider.

Stimulus display: The `stimulus` argument is a string specifying the text to be displayed as the stimulus. It can include HTML markup, meaning that it can be used to any stimulus that can be specified using HTML. It remains on screen for a length of time corresponding to the `stimulus_duration` parameter in milliseconds (or indefinitely if the parameter is `NULL`).

Response mechanism: Participant responses for this trial type are collected using a slider bar that the participant can move using the mouse. Once the participant is happy with this positioning they can click a button at the bottom of the page to move on to the next trial. This response method can be customised in several ways depending on the following arguments:

- The `min` and `max` arguments are numeric values that specify the minimum value (leftmost point on the slider) and the maximum value (rightmost point on the slider) that a participant can respond with.
- The `start` parameter is a numeric value that indicates where the value of the the slider is initially position. By default this is set to the middle of the scale, but there are many cases where it may be sensible to have the slider bar start at one end of the scale.
- The movement of the slider is discretised, and the granularity of this movement can be customised using the `step` parameter. This should be a numeric value that specifies the smallest possible increment that the participant can move the slider in either direction.
- The text labels displayed below the slider bar can also be customised by specifying the `labels` parameter. This argument should be a character vector that contains the labels to be displaed. Labels will be displayed at equally spaced intervals along the slider, though it is possible to include blank labels to create the impression of unequal spacing if that is required.

- The slider_width controls the horizontal width of the slider bar: the default value of NULL creates a slider that occupies 100% of the width of the jsPsych display. Note that this may not be 100% of the screen width.
- To ensure that participants do engage with the slider, it is possible to set require_movement = TRUE which forces the participant to move the slider at least once in order to be permitted to move onto the next trial.
- The button_label argument specifies the text displayed on the button that participants click to move to the next trial.

Other behaviour: As is the case for most trial_ functions there is a prompt argument, a string that specifies additional text that is displayed on screen during the trial. The value of prompt can contain HTML markup, allowing it to be used quite flexibly if needed.

Depending on parameter settings, the trial can end when the subject responds (response_ends_trial = TRUE), or after a fixed amount of time (specified using the trial_duration argument) has elapsed. The length of time that the stimulus remains visible can also be customized using the (stimulus_duration) argument.

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data: When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The rt value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The response is the numeric value of the slider bar.
- The stimulus variable records the HTML content that was displayed on this trial.

In addition, it records default variables that are recorded by all trials:

- trial_type is a string that records the name of the plugin used to run the trial.
- trial_index is a number that records the index of the current trial across the whole experiment.
- time_elapsed counts the number of milliseconds since the start of the experiment when the trial ended.
- internal_node_id is a string identifier for the current "node" in the timeline.

Value

Functions with a trial_ prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with NULL elements removed. Logical values in the input (TRUE and FALSE) are transformed to character vectors "true" and "false" and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, true and false.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are trial_image_button_response, trial_image_keyboard_response, trial_image_slider_response, trial_audio_button_response, trial_audio_keyboard_response, trial_audio_slider_response, trial_video_button_response, trial_video_keyboard_response, trial_video_slider_response, trial_html_button_response, trial_html_keyboard_response and trial_html_slider_response.

---

trial_image_button_response *Specify an image trial with button response*

Description

The trial_image_button_response function is used to display an image stimulus and collect a response using on screen buttons.

Usage

```
trial_image_button_response(
  stimulus,
  stimulus_height = NULL,
  stimulus_width = NULL,
  maintain_aspect_ratio = TRUE,
  choices = c("button 0", "button 1", "button 2"),
  button_html = NULL,
  margin_vertical = "0px",
  margin_horizontal = "8px",
  prompt = NULL,
  stimulus_duration = NULL,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

- `stimulus` The path of the image file to be displayed.
- `stimulus_height` Set the height of the image in pixels. If NULL, then the image will display at its natural height.
- `stimulus_width` Set the width of the image in pixels. If NULL, then the image will display at its natural width.
- `maintain_aspect_ratio` If setting only the width or only the height and this parameter is TRUE, then the other dimension will be scaled to maintain the image's aspect ratio.
- `choices` Labels for the buttons. Each element of the character vector will generate a different button.
- `button_html` A template of HTML for generating the button elements (see details)
- `margin_vertical` Vertical margin of the buttons
- `margin_horizontal` Horizontal margin of the buttons
- `prompt` A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).
- `stimulus_duration` How long to show the stimulus, in milliseconds. If NULL, then the stimulus will be shown until the subject makes a response
- `trial_duration` How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.
- `response_ends_trial` If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.
- `post_trial_gap` The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
- `on_finish` A javascript callback function to execute when the trial finishes
- `on_load` A javascript callback function to execute when the trial begins, before any loading has occurred
- `data` An object containing additional data to store for the trial

Details

The `trial_image_button_response` function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one displays an image and records responses generated with a button click.

Stimulus display: For trials that display an image, the `stimulus` argument is a string that specifies the path to the image file. More precisely, it must specify the path to where the image file will be located at the time the experiment runs. Typically, if an experiment is deployed using the build_experiment() function all resource files will be stored in a "resource" folder, and the images will be copied to the "image" subfolder. So if the image to be displayed is a file called "picture.png", the stimulus path on a Mac or Linux machine would likely be "resource/image/picture.png". Note that this path is specified relative to the location of the primary experiment file "image.html". To make this a little easier, the insert_resource() function can be used to construct resource paths automatically. In the example above, stimulus = insert_resource("picture.png") would suffice.

Other aspects to the stimulus display can be controlled with other arguments. The stimulus_height and stimulus_width arguments can be used to manually control the image display size by specifying the height/width in pixels. If only one of these two arguments is specified, but the maintain_aspect_ratio value is set to TRUE, the other dimension of the image will automatically be scaled appropriately.

The length of time that the image remains on screen can also be customised by setting the stimulus_duration argument: this should be a numeric value indicating the number of milliseconds before the image disappears. Alternatively, a value of NULL (the default) ensures that the image remains visible until the trial ends.

Response mechanism: The response buttons can be customized using HTML formatting, via the button_html argument. This argument allows the user to specify an HTML used to generating the button elements. If this argument is a vector of the same length as choices then the i-th element of button_html will be used to define the i-th response button. If button_html is a single string then the same template will be applied to every button. The templating is defined using a special string "%choice%" that will be replaced by the corresponding element of the choices vector. By default the jsPsych library creates an HTML button of class "jspsych-btn" and the styling is governed by the corresponding CSS.

Other behaviour: Depending on parameter settings, the trial can end when the subject responds (response_ends_trial = TRUE), or after a fixed amount of time (specified using the trial_duration argument) has elapsed. The length of time that the stimulus remains visible can also be customized using the (stimulus_duration) argument.

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data: When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `rt` value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The `button_pressed` variable is a numeric value indicating which button was pressed. The first button in the choices array is recorded as value 0, the second is value 1, and so on.
- The `stimulus` variable records the path to the image that was displayed on this trial.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are `trial_image_button_response`, `trial_image_keyboard_response`, `trial_image_slider_response`, `trial_audio_button_response`, `trial_audio_keyboard_response`, `trial_audio_slider_response`, `trial_video_button_response`, `trial_video_keyboard_response`, `trial_video_slider_response`, `trial_html_button_response`, `trial_html_keyboard_response` and `trial_html_slider_response`.

---

trial_image_keyboard_response

Specify an image trial with keyboard response

Description

The `trial_image_keyboard_response` function is used to display an image stimulus and collect a response using a key press.

Usage

```
trial_image_keyboard_response(
  stimulus,
  stimulus_height = NULL,
  stimulus_width = NULL,
  maintain_aspect_ratio = TRUE,
  choices = respond_any_key(),
```

```
  prompt = NULL,
  stimulus_duration = NULL,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimulus` The path of the image file to be displayed.

`stimulus_height` Set the height of the image in pixels. If NULL, then the image will display at its natural height.

`stimulus_width` Set the width of the image in pixels. If NULL, then the image will display at its natural width.

`maintain_aspect_ratio` If setting only the width or only the height and this parameter is TRUE, then the other dimension will be scaled to maintain the image's aspect ratio.

`choices` A character vector of keycodes (either numeric values or the characters themselves). Alternatively, respond_any_key() and respond_no_key() can be used

`prompt` A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).

`stimulus_duration` How long to show the stimulus, in milliseconds. If NULL, then the stimulus will be shown until the subject makes a response

`trial_duration` How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.

`response_ends_trial` If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.

`post_trial_gap` The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.

`on_finish` A javascript callback function to execute when the trial finishes

`on_load` A javascript callback function to execute when the trial begins, before any loading has occurred

`data` An object containing additional data to store for the trial

Details

The `trial_image_keyboard_response` function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one displays an image and records responses generated with a key press.

Stimulus display: For trials that display an image, the stimulus argument is a string that specifies the path to the image file. More precisely, it must specify the path to where the image file will be located at the time the experiment runs. Typically, if an experiment is deployed using the build_experiment() function all resource files will be stored in a "resource" folder, and the images will be copied to the "image" subfolder. So if the image to be displayed is a file called "picture.png", the stimulus path on a Mac or Linux machine would likely be "resource/image/picture.png". Note that this path is specified relative to the location of the primary experiment file "image.html". To make this a little easier, the insert_resource() function can be used to construct resource paths automatically. In the example above, stimulus = insert_resource("picture.png") would suffice.

Other aspects to the stimulus display can be controlled with other arguments. The stimulus_height and stimulus_width arguments can be used to manually control the image display size by specifying the height/width in pixels. If only one of these two arguments is specified, but the maintain_aspect_ratio value is set to TRUE, the other dimension of the image will automatically be scaled appropriately.

The length of time that the image remains on screen can also be customised by setting the stimulus_duration argument: this should be a numeric value indicating the number of milliseconds before the image disappears. Alternatively, a value of NULL (the default) ensures that the image remains visible until the trial ends.

Response mechanism: For this kind of trial, participants can make a response by pressing a key, and the choices argument is used to control which keys will register a valid response. The default value choices = respond_any_key() is to allow the participant to press any key to register their response. Alternatively it is possible to set choices = respond_no_key(), which prevents all keys from registering a response: this can be useful if the trial is designed to run for a fixed duration, regardless of what the participant presses.

In many situations it is preferable to require the participant to respond using specific keys (e.g., for a binary choice tasks, it may be desirable to require participants to press F for one response or J for the other). This can be achieved in two ways. One possibility is to use a character vector as input (e.g., choices = c("f","j")). The other is to use the numeric code that specifies the desired key in javascript, which in this case would be choices = c(70, 74). To make it a little easier to work with numeric codes, the jaysire package includes the keycode() function to make it easier to convert from one format to the other.

Other behaviour: Depending on parameter settings, the trial can end when the subject responds (response_ends_trial = TRUE), or after a fixed amount of time (specified using the trial_duration argument) has elapsed. The length of time that the stimulus remains visible can also be customized using the (stimulus_duration) argument.

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.

- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial.

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when `build_experiment` is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `rt` value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The `key_press` variable is the numeric javascript key code corresponding to the response.
- The `stimulus` variable records the path to the image that was displayed on this trial.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are `trial_image_button_response`, `trial_image_keyboard_response`, `trial_image_slider_response`, `trial_audio_button_response`, `trial_audio_keyboard_response`, `trial_audio_slider_response`, `trial_video_button_response`, `trial_video_keyboard_response`, `trial_video_slider_response`, `trial_html_button_response`, `trial_html_keyboard_response` and `trial_html_slider_response`.

`trial_image_slider_response` *Specify an image trial with slider bar response*

Description

The `trial_image_slider_response` function is used to display an image stimulus and collect a response using a slider bar.

Usage

```
trial_image_slider_response(
  stimulus,
  stimulus_height = NULL,
  stimulus_width = NULL,
  maintain_aspect_ratio = TRUE,
  labels = c("0%", "25%", "50%", "75%", "100%"),
  button_label = "Continue",
  min = 0,
  max = 100,
  start = 50,
  step = 1,
  slider_width = NULL,
  require_movement = FALSE,
  prompt = NULL,
  stimulus_duration = NULL,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`stimulus`	The path of the image file to be displayed.
`stimulus_height`	Set the height of the image in pixels. If NULL, then the image will display at its natural height.
`stimulus_width`	Set the width of the image in pixels. If NULL, then the image will display at its natural width.
`maintain_aspect_ratio`	If setting only the width or only the height and this parameter is TRUE, then the other dimension will be scaled to maintain the image's aspect ratio.
`labels`	Labels displayed at equidistant locations on the slider.
`button_label`	Label placed on the "continue" button
`min`	Minimum value of the slider
`max`	Maximum value of the slider
`start`	Initial value of the slider
`step`	Step size of the slider
`slider_width`	Horizontal width of the slider (defaults to display width)
`require_movement`	Does the user need to move the slider before clicking the continue button?
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).

`stimulus_duration`
: How long to show the stimulus, in milliseconds. If NULL, then the stimulus will be shown until the subject makes a response

`trial_duration`
: How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.

`response_ends_trial`
: If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.

`post_trial_gap`
: The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.

`on_finish`
: A javascript callback function to execute when the trial finishes

`on_load`
: A javascript callback function to execute when the trial begins, before any loading has occurred

`data`
: An object containing additional data to store for the trial

Details

The `trial_image_slider_response` function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one displays an image and records responses generated with a slider.

Stimulus display: For trials that display an image, the `stimulus` argument is a string that specifies the path to the image file. More precisely, it must specify the path to where the image file will be located at the time the experiment runs. Typically, if an experiment is deployed using the `build_experiment()` function all resource files will be stored in a "resource" folder, and the images will be copied to the "image" subfolder. So if the image to be displayed is a file called "picture.png", the `stimulus` path on a Mac or Linux machine would likely be "resource/image/picture.png". Note that this path is specified relative to the location of the primary experiment file "image.html". To make this a little easier, the `insert_resource()` function can be used to construct resource paths automatically. In the example above, `stimulus = insert_resource("picture.png")` would suffice.

Other aspects to the stimulus display can be controlled with other arguments. The `stimulus_height` and `stimulus_width` arguments can be used to manually control the image display size by specifying the height/width in pixels. If only one of these two arguments is specified, but the `maintain_aspect_ratio` value is set to `TRUE`, the other dimension of the image will automatically be scaled appropriately.

The length of time that the image remains on screen can also be customised by setting the `stimulus_duration` argument: this should be a numeric value indicating the number of milliseconds before the image disappears. Alternatively, a value of `NULL` (the default) ensures that the image remains visible until the trial ends.

Response mechanism:

Participant responses for this trial type are collected using a slider bar that the participant can move using the mouse. Once the participant is happy with this positioning they can click a button at the bottom of the page to move on to the next trial. This response method can be customised in several ways depending on the following arguments:

- The min and max arguments are numeric values that specify the minimum value (leftmost point on the slider) and the maximum value (rightmost point on the slider) that a participant can respond with.
- The start parameter is a numeric value that indicates where the value of the the slider is initially position. By default this is set to the middle of the scale, but there are many cases where it may be sensible to have the slider bar start at one end of the scale.
- The movement of the slider is discretised, and the granularity of this movement can be customised using the step parameter. This should be a numeric value that specifies the smallest possible increment that the participant can move the slider in either direction.
- The text labels displayed below the slider bar can also be customised by specifying the labels parameter. This argument should be a character vector that contains the labels to be displaed. Labels will be displayed at equally spaced intervals along the slider, though it is possible to include blank labels to create the impression of unequal spacing if that is required.
- The slider_width controls the horizontal width of the slider bar: the default value of NULL creates a slider that occupies 100% of the width of the jsPsych display. Note that this may not be 100% of the screen width.
- To ensure that participants do engage with the slider, it is possible to set require_movement = TRUE which forces the participant to move the slider at least once in order to be permitted to move onto the next trial.
- The button_label argument specifies the text displayed on the button that participants click to move to the next trial.

Other behaviour:

As is the case for most trial_ functions there is a prompt argument, a string that specifies additional text that is displayed on screen during the trial. The value of prompt can contain HTML markup, allowing it to be used quite flexibly if needed.

Depending on parameter settings, the trial can end when the subject responds (response_ends_trial = TRUE), or after a fixed amount of time (specified using the trial_duration argument) has elapsed. The length of time that the stimulus remains visible can also be customized using the (stimulus_duration) argument.

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial.

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `rt` value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The `response` is the numeric value of the slider bar.
- The `stimulus` variable records the path to the image that was displayed on this trial.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are `trial_image_button_response`, `trial_image_keyboard_response`, `trial_image_slider_response`, `trial_audio_button_response`, `trial_audio_keyboard_response`, `trial_audio_slider_response`, `trial_video_button_response`, `trial_video_keyboard_response`, `trial_video_slider_response`, `trial_html_button_response`, `trial_html_keyboard_response` and `trial_html_slider_response`.

`trial_instructions` *Specify pages of instructions to display*

Description

The `trial_instructions` function is used to display one or more pages of instructions that a participant can browse.

Usage

```
trial_instructions(
  pages,
  key_forward = keycode("right arrow"),
  key_backward = keycode("left arrow"),
  allow_backward = TRUE,
  allow_keys = TRUE,
  show_clickable_nav = FALSE,
  button_label_previous = "Previous",
```

```
  button_label_next = "Next",
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

pages	Character vector. Each element should be an HTML-formatted string specifying a page
key_forward	This is the key that the subject can press in order to advance to the next page, specified as their numeric key code or as characters
key_backward	This is the key that the subject can press in order to return to the previous page.
allow_backward	If TRUE, participants can navigate backwards
allow_keys	If TRUE, participants can use keyboard keys to navigate
show_clickable_nav	If TRUE, buttons will be shown to allow navigation
button_label_previous	Text on the "previous" button
button_label_next	Text on the "next" button
post_trial_gap	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
on_finish	A javascript callback function to execute when the trial finishes
on_load	A javascript callback function to execute when the trial begins, before any loading has occurred
data	An object containing additional data to store for the trial

Details

The `trial_instructions()` function is used to generate trials that show instruction to the participant. It allows participants to navigate through multiple pages of instructions at their own pace, recording how long thet spend on each page. Navigation can be done using the mouse or keyboard. Participants can be allowed to navigate forwards and backwards through pages, if desired.

Specifying instructions:

- The `pages` argument is a required argument, and should be a character vector. Each element of the vector specifies the text to be displayed on a single instruction page, and can include HTML markup. Depending on parameter values, instruction pages can be navigated by clicking on screen buttons or else by using key presses.
- To navigate using buttons, the `show_clickable_nav` argument must be set to `TRUE`. If `allow_backward` is also set to `TRUE` this will create two buttons on screen, one allowing the participant to move forward to the next page, and another allowing them to move backward to the previous page. If `allow_backward = FALSE`, only the forward button is shown. The text labels for these buttons can be customised using the `button_label_previous` and `button_label_next` arguments.

- To navigate using key presses, the `allow_keys` argument must be set to `TRUE`. By default, the keyboard navigation uses the right arrow to move forward, and the the left arrow key to move back, but these can be customised using the `key_forward` and `key_backward` arguments. The values for these arguments should either be a numeric keycode or the corresponding character code. For an overview of these codes, see the `keycode()` function documentation.

Other behaviour:

Like all functions in the `trial_` family it contains four additional arguments:

- The `post_trial_gap` argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the `build_experiment` function, and a blank screen is displayed during this gap period.
- The `on_load` and `on_finish` arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the `insert_javascript` function. However, the `fn_` family of functions supplies a variety of functions that may be useful in many cases.
- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when `build_experiment` is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `view_history` value is a JSON string containing the order of pages the subject viewed (including when the subject returned to previous pages) and the time spent viewing each page.
- The `rt` value is the response time in milliseconds for the subject to view all of the pages.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

`trial_survey_likert` *Specify a survey page with Likert scale items*

Description

The `trial_survey_likert` function is used to display a survey page with one or more items with Likert scale responses.

Usage

```
trial_survey_likert(
  questions,
  preamble = "",
  scale_width = NULL,
  randomize_question_order = FALSE,
  button_label = "Continue",
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`questions`	A question or list of questions
`preamble`	Text to appear above the questions
`scale_width`	Width of the scale in pixels (NULL is the display width)
`randomize_question_order`	Should order be randomised?
`button_label`	Text for the continue button
`post_trial_gap`	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
`on_finish`	A javascript callback function to execute when the trial finishes
`on_load`	A javascript callback function to execute when the trial begins, before any loading has occurred
`data`	An object containing additional data to store for the trial

Details

The `trial_survey_likert` function creates a trial that displays a set of questions with Likert scale responses.

Survey construction:

There are five arguments that are relevant to the survey itself:

- The main argument is questions, which can and can either consist of a single question object generated b yquestion_likert or a list of such objects. The Likert scale items are laid out on an ordered scale with radio buttons spaced at equal intervals, whose labels are specified when calling question_likert. See the documentation for the question function for details of what this entails.
- The preamble argument is used to specify introductory text that appears about the survey page. It accepts HTML markup and so can be used quite flexibly.
- The scale_width parameter controls the horizontal width of the Likert scale, in pixels. By default, this is set to 100% of the width of the jsPsych container (which may not be 100% of the screen width).
- The randomize_question_order argument is a logical value that indicates whether or not the survey items should appear in a random order.
- The button_label specifies text to appear on the button displayed at the bottom of the page, and which the participant must click before moving on to the next trial.

Other behaviour:

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The responses value is a an array containing all selected choices in JSON format for each question. The encoded object will have a separate variable for the response to each question, with the first question in the trial being recorded in Q0, the second in Q1, and so on. The responses are recorded as the name of the option label. If the name parameter is defined for the question, then the response will use the value of name as the key for the response in the responses object.
- The rt value is the response time in milliseconds for the subject to make a response. The time is measured from when the questions first appear on the screen until the subject's response.
- The question_order value is a string in JSON format containing an array with the order of questions. For example [2,0,1] would indicate that the first question was trial.questions[2] (the third item in the questions parameter), the second question was trial.questions[0], and the final question was trial.questions[1].

In addition, it records default variables that are recorded by all trials:

- trial_type is a string that records the name of the plugin used to run the trial.
- trial_index is a number that records the index of the current trial across the whole experiment.
- time_elapsed counts the number of milliseconds since the start of the experiment when the trial ended.
- internal_node_id is a string identifier for the current "node" in the timeline.

Value

Functions with a trial_ prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with NULL elements removed. Logical values in the input (TRUE and FALSE) are transformed to character vectors "true" and "false" and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, true and false.

See Also

Survey page trials are constructed using the trial_survey_text, trial_survey_likert, trial_survey_multi_choice and trial_survey_multi_select functions. Individual questions for survey trials can be specified using question_text, question_likert and question_multi.

---

trial_survey_multi_choice

Specify a survey page with multiple choice items

Description

The trial_survey_multi_choice function is used to display a survey page with one or more multiple choice questions

Usage

```
trial_survey_multi_choice(
  questions,
  preamble = "",
  randomize_question_order = FALSE,
  button_label = "Continue",
  required_message = "You must choose at least one response for this question",
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`questions`	A question or list of questions
`preamble`	Text to appear above the questions
`randomize_question_order`	Should order be randomised?
`button_label`	Text for the continue button
`required_message`	Message to display if required response is not given.
`post_trial_gap`	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
`on_finish`	A javascript callback function to execute when the trial finishes
`on_load`	A javascript callback function to execute when the trial begins, before any loading has occurred
`data`	An object containing additional data to store for the trial

Details

The `trial_survey_multi_choice` function creates a trial that displays a set of questions with multiple choice responses.

Survey construction:

There are five arguments that are relevant to the survey itself:

- The main argument is `questions`, which can and can either consist of a single question object generated by `question_multi` or a list of such objects. The multiple choice items are laid out as a set of radio buttons with labels specified when calling `question_multi`, and the participant can select only one possible response. See the documentation for the question function for details of what this entails.
- The `preamble` argument is used to specify introductory text that appears about the survey page. It accepts HTML markup and so can be used quite flexibly.
- The `required_message` parameter specifies the text of the message to be displayed if a participant attempts to move to the next trial without answering a required question.
- The `randomize_question_order` argument is a logical value that indicates whether or not the survey items should appear in a random order.
- The `button_label` specifies text to appear on the button displayed at the bottom of the page, and which the participant must click before moving on to the next trial.

Other behaviour:

Like all functions in the `trial_` family it contains four additional arguments:

- The `post_trial_gap` argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the `build_experiment` function, and a blank screen is displayed during this gap period.
- The `on_load` and `on_finish` arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the `insert_javascript` function. However, the `fn_` family of functions supplies a variety of functions that may be useful in many cases.

- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The responses value is a an array containing all selected choices in JSON format for each question. The encoded object will have a separate variable for the response to each question, with the first question in the trial being recorded in Q0, the second in Q1, and so on. The responses are recorded as the name of the option label. If the name parameter is defined for the question, then the response will use the value of name as the key for the response in the responses object.
- The rt value is the response time in milliseconds for the subject to make a response. The time is measured from when the questions first appear on the screen until the subject's response.
- The question_order value is a string in JSON format containing an array with the order of questions. For example [2,0,1] would indicate that the first question was trial.questions[2] (the third item in the questions parameter), the second question was trial.questions[0], and the final question was trial.questions[1].

In addition, it records default variables that are recorded by all trials:

- trial_type is a string that records the name of the plugin used to run the trial.
- trial_index is a number that records the index of the current trial across the whole experiment.
- time_elapsed counts the number of milliseconds since the start of the experiment when the trial ended.
- internal_node_id is a string identifier for the current "node" in the timeline.

Value

Functions with a trial_ prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with NULL elements removed. Logical values in the input (TRUE and FALSE) are transformed to character vectors "true" and "false" and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, true and false.

See Also

Survey page trials are constructed using the trial_survey_text, trial_survey_likert, trial_survey_multi_choice and trial_survey_multi_select functions. Individual questions for survey trials can be specified using question_text, question_likert and question_multi.

`trial_survey_multi_select`
Specify a survey page with multiple selection items

Description

The `trial_survey_multi_select` function is used to display a survey page with one or more multiple selection questions

Usage

```
trial_survey_multi_select(
  questions,
  preamble = "",
  randomize_question_order = FALSE,
  button_label = "Continue",
  required_message = "You must choose at least one response for this question",
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`questions`	A question or list of questions
`preamble`	Text to appear above the questions
`randomize_question_order`	Should order be randomised?
`button_label`	Text for the continue button
`required_message`	Message to display if required response is not given.
`post_trial_gap`	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
`on_finish`	A javascript callback function to execute when the trial finishes
`on_load`	A javascript callback function to execute when the trial begins, before any loading has occurred
`data`	An object containing additional data to store for the trial

Details

The `trial_survey_multi_select` function creates a trial that displays a set of questions with multiple selection responses.

Survey construction:

There are five arguments that are relevant to the survey itself:

- The main argument is questions, which can and can either consist of a single question object generated by question_multi or a list of such objects. The multiple choice items are laid out as a set of check boxes with labels specified when calling question_multi, and the participant can select as many boxes as they like. See the documentation for the question function for details of what this entails.
- The preamble argument is used to specify introductory text that appears about the survey page. It accepts HTML markup and so can be used quite flexibly.
- The required_message parameter specifies the text of the message to be displayed if a participant attempts to move to the next trial without answering a required question.
- The randomize_question_order argument is a logical value that indicates whether or not the survey items should appear in a random order.
- The button_label specifies text to appear on the button displayed at the bottom of the page, and which the participant must click before moving on to the next trial.

Other behaviour:

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The responses value is a an array containing all selected choices in JSON format for each question. The encoded object will have a separate variable for the response to each question, with the first question in the trial being recorded in Q0, the second in Q1, and so on. The responses are recorded as the name of the option label. If the name parameter is defined for the question, then the response will use the value of name as the key for the response in the responses object.
- The rt value is the response time in milliseconds for the subject to make a response. The time is measured from when the questions first appear on the screen until the subject's response.
- The question_order value is a string in JSON format containing an array with the order of questions. For example [2,0,1] would indicate that the first question was trial.questions[2] (the third item in the questions parameter), the second question was trial.questions[0], and the final question was trial.questions[1].

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Survey page trials are constructed using the `trial_survey_text`, `trial_survey_likert`, `trial_survey_multi_choice` and `trial_survey_multi_select` functions. Individual questions for survey trials can be specified using `question_text`, `question_likert` and `question_multi`.

`trial_survey_text` *Specify a survey page with free text responding*

Description

The `trial_survey_text` function is used to display a survey page that allows free text responding.

Usage

```
trial_survey_text(
  questions,
  preamble = "",
  randomize_question_order = FALSE,
  button_label = "Continue",
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`questions` A question or list of questions

`preamble` Text to appear above the questions

`randomize_question_order`
Should order be randomised?

`button_label`	Text for the continue button
`post_trial_gap`	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
`on_finish`	A javascript callback function to execute when the trial finishes
`on_load`	A javascript callback function to execute when the trial begins, before any loading has occurred
`data`	An object containing additional data to store for the trial

Details

The `trial_survey_text` function creates a trial that displays a set of questions with free text responses.

Survey construction:
There are four arguments that are relevant to the survey itself:

- The main argument is `questions`, which can and can either consist of a single question object generated by `question_text` or a list of such objects. See the documentation for the question function for details of what this entails.
- The `preamble` argument is used to specify introductory text that appears about the survey page. It accepts HTML markup and so can be used quite flexibly.
- The `randomize_question_order` argument is a logical value that indicates whether or not the survey items should appear in a random order.
- The `button_label` specifies text to appear on the button displayed at the bottom of the page, and which the participant must click before moving on to the next trial.

Other behaviour:
Like all functions in the `trial_` family it contains four additional arguments:

- The `post_trial_gap` argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the `build_experiment` function, and a blank screen is displayed during this gap period.
- The `on_load` and `on_finish` arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the `insert_javascript` function. However, the `fn_` family of functions supplies a variety of functions that may be useful in many cases.
- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial

Data: When this function is called from R it returns the trial object that will later be inserted into the experiment when `build_experiment` is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `responses` value is a string in JSON format containing the response for each question. The encoded object will have a separate variable for the response to each question, with the first question in the trial being recorded in Q0, the second in Q1, and so on. Each response is a string containing whatever the subject typed into the associated text box. If the name parameter is defined for the question, then the response will use the value of name as the key for the response in the responses object.

- The `rt` value is the response time in milliseconds for the subject to make a response. The time is measured from when the questions first appear on the screen until the subject's response.
- The `question_order` value is a string in JSON format containing an array with the order of questions. For example [2,0,1] would indicate that the first question was trial.questions[2] (the third item in the questions parameter), the second question was trial.questions[0], and the final question was trial.questions[1].

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Survey page trials are constructed using the `trial_survey_text`, `trial_survey_likert`, `trial_survey_multi_choice` and `trial_survey_multi_select` functions. Individual questions for survey trials can be specified using `question_text`, `question_likert` and `question_multi`.

`trial_video_button_response`

Specify a video trial with button response

Description

The `trial_video_button_response` function is used to play a video stimulus and collect a response using on screen buttons.

Usage

```
trial_video_button_response(
  sources,
  trial_ends_after_video = FALSE,
  width = NULL,
  height = NULL,
```

```
  autoplay = TRUE,
  controls = FALSE,
  start = NULL,
  stop = NULL,
  rate = 1,
  choices = c("button 0", "button 1", "button 2"),
  button_html = NULL,
  margin_vertical = "0px",
  margin_horizontal = "8px",
  prompt = NULL,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`sources`	Path(s) to the video file. Videos may be specified in multiple formats (e.g., .mp4, .ogg, .webm)
`trial_ends_after_video`	If TRUE the trial will end as soon as the video finishes playing.
`width`	The width of the video display in pixels (if NULL, natural width is used)
`height`	The height of the video display in pixels (if NULL, natural height is used)
`autoplay`	Does the video play automatically?
`controls`	Should the video controls be shown?
`start`	Time point in seconds to start video (NULL starts at the beginning)
`stop`	Time point in seconds to stop video (NULL stops at the end)
`rate`	What rate to play the video (1 = normal, <1 slower, >1 faster)
`choices`	Labels for the buttons. Each element of the character vector will generate a different button.
`button_html`	A template of HTML for generating the button elements (see details)
`margin_vertical`	Vertical margin of the buttons
`margin_horizontal`	Horizontal margin of the buttons
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).
`trial_duration`	How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.

`response_ends_trial`
: If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.

`post_trial_gap`
: The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.

`on_finish`
: A javascript callback function to execute when the trial finishes

`on_load`
: A javascript callback function to execute when the trial begins, before any loading has occurred

`data`
: An object containing additional data to store for the trial

Details

The `trial_video_button_response` function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one plays a video and records responses generated with a button click.

Stimulus display: TBA

Response mechanism:

The response buttons can be customized using HTML formatting, via the `button_html` argument. This argument allows the user to specify an HTML used to generating the button elements. If this argument is a vector of the same length as `choices` then the i-th element of `button_html` will be used to define the i-th response button. If `button_html` is a single string then the same template will be applied to every button. The templating is defined using a special string "%choice%" that will be replaced by the corresponding element of the `choices` vector. By default the jsPsych library creates an HTML button of class "jspsych-btn" and the styling is governed by the corresponding CSS.

Other behaviour: Depending on parameter settings, the trial can end when the subject responds (`response_ends_trial = TRUE`), or after a fixed amount of time (specified using the `trial_duration` argument) has elapsed. The trial can also be made to end automatically at the end of the video using the `trial_ends_after_video` argument.

Like all functions in the `trial_` family it contains four additional arguments:

- The `post_trial_gap` argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the `build_experiment` function, and a blank screen is displayed during this gap period.
- The `on_load` and `on_finish` arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the `insert_javascript` function. However, the `fn_` family of functions supplies a variety of functions that may be useful in many cases.
- The `data` argument can be used to insert custom data values into the jsPsych data storage for this trial

Data:

When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `rt` value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The `button_pressed` variable is a numeric value indicating which button was pressed. The first button in the choices array is recorded as value 0, the second is value 1, and so on.
- The `stimulus` variable records a JSON encoding of the sources array.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are trial_image_button_response, trial_image_keyboard_response, trial_image_slider_response, trial_audio_button_response, trial_audio_keyboard_response, trial_audio_slider_response, trial_video_button_response, trial_video_keyboard_response, trial_video_slider_response, trial_html_button_response, trial_html_keyboard_response and trial_html_slider_response.

`trial_video_keyboard_response`

Specify a video trial with keyboard response

Description

The `trial_video_keyboard_response` function is used to play a video stimulus and collect a response using a key press.

Usage

```
trial_video_keyboard_response(
  sources,
  trial_ends_after_video = FALSE,
  width = NULL,
  height = NULL,
  autoplay = TRUE,
  controls = FALSE,
  start = NULL,
  stop = NULL,
  rate = 1,
  choices = respond_any_key(),
  prompt = NULL,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`sources`	Path(s) to the video file. Videos may be specified in multiple formats (e.g., .mp4, .ogg, .webm)
`trial_ends_after_video`	If TRUE the trial will end as soon as the video finishes playing.
`width`	The width of the video display in pixels (if NULL, natural width is used)
`height`	The height of the video display in pixels (if NULL, natural height is used)
`autoplay`	Does the video play automatically?
`controls`	Should the video controls be made available to the user?
`start`	Specifies the time in seconds to start the video (if NULL, start at beginning)
`stop`	Specifies the time in seconds to stop the video (if NULL, stop at end)
`rate`	What rate to play the video (1 = normal, <1 slower, >1 faster)
`choices`	A character vector of keycodes (either numeric values or the characters themselves). Alternatively, respond_any_key() and respond_no_key() can be used
`prompt`	A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).
`trial_duration`	How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.
`response_ends_trial`	If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.

post_trial_gap	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.
on_finish	A javascript callback function to execute when the trial finishes
on_load	A javascript callback function to execute when the trial begins, before any loading has occurred
data	An object containing additional data to store for the trial

Details

The trial_video_keyboard_response function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one plays a video and records responses generated with a key press.

Stimulus display: TBA

Response mechanism: For this kind of trial, participants can make a response by pressing a key, and the choices argument is used to control which keys will register a valid response. The default value choices = respond_any_key() is to allow the participant to press any key to register their response. Alternatively it is possible to set choices = respond_no_key(), which prevents all keys from registering a response: this can be useful if the trial is designed to run for a fixed duration, regardless of what the participant presses.

In many situations it is preferable to require the participant to respond using specific keys (e.g., for a binary choice tasks, it may be desirable to require participants to press F for one response or J for the other). This can be achieved in two ways. One possibility is to use a character vector as input (e.g., choices = c("f","j")). The other is to use the numeric code that specifies the desired key in javascript, which in this case would be choices = c(70, 74). To make it a little easier to work with numeric codes, the jaysire package includes the keycode() function to make it easier to convert from one format to the other.

Other behaviour: Depending on parameter settings, the trial can end when the subject responds (response_ends_trial = TRUE), or after a fixed amount of time (specified using the trial_duration argument) has elapsed. The trial can also be made to end automatically at the end of the video using the trial_ends_after_video argument.

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data: When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The `rt` value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The `key_press` variable is the numeric javascript key code corresponding to the response.
- The `stimulus` variable records a JSON encoding of the sources array.

In addition, it records default variables that are recorded by all trials:

- `trial_type` is a string that records the name of the plugin used to run the trial.
- `trial_index` is a number that records the index of the current trial across the whole experiment.
- `time_elapsed` counts the number of milliseconds since the start of the experiment when the trial ended.
- `internal_node_id` is a string identifier for the current "node" in the timeline.

Value

Functions with a `trial_` prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with `NULL` elements removed. Logical values in the input (`TRUE` and `FALSE`) are transformed to character vectors `"true"` and `"false"` and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, `true` and `false`.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are `trial_image_button_response`, `trial_image_keyboard_response`, `trial_image_slider_response`, `trial_audio_button_response`, `trial_audio_keyboard_response`, `trial_audio_slider_response`, `trial_video_button_response`, `trial_video_keyboard_response`, `trial_video_slider_response`, `trial_html_button_response`, `trial_html_keyboard_response` and `trial_html_slider_response`.

`trial_video_slider_response`

Specify a video trial with slider bar response

Description

The `trial_video_slider_response` function is used to play a video stimulus and collect a response using a slider bar.

Usage

```
trial_video_slider_response(
  sources,
  trial_ends_after_video = FALSE,
  width = NULL,
  height = NULL,
  autoplay = TRUE,
  controls = FALSE,
  start = NULL,
  stop = NULL,
  rate = 1,
  labels = c("0%", "25%", "50%", "75%", "100%"),
  min = 0,
  max = 100,
  slider_start = 50,
  step = 1,
  slider_width = NULL,
  require_movement = FALSE,
  button_label = "Continue",
  prompt = NULL,
  trial_duration = NULL,
  response_ends_trial = TRUE,
  post_trial_gap = 0,
  on_finish = NULL,
  on_load = NULL,
  data = NULL
)
```

Arguments

`sources`	Path(s) to the video file. Videos may be specified in multiple formats (e.g., .mp4, .ogg, .webm)
`trial_ends_after_video`	If TRUE the trial will end as soon as the video finishes playing.
`width`	The width of the video display in pixels (if NULL, natural width is used)
`height`	The height of the video display in pixels (if NULL, natural height is used)
`autoplay`	Does the video play automatically?
`controls`	Should the video controls be shown?
`start`	Time point in seconds to start video (NULL starts at the beginning)
`stop`	Time point in seconds to stop video (NULL stops at the end)
`rate`	What rate to play the video (1 = normal, <1 slower, >1 faster)
`labels`	Labels displayed at equidistant locations on the slider.
`min`	Minimum value of the slider
`max`	Maximum value of the slider
`slider_start`	Initial value of the slider

`step` Step size of the slider

`slider_width` Horizontal width of the slider (defaults to display width)

`require_movement` Does the user need to move the slider before clicking the continue button?

`button_label` Label placed on the "continue" button

`prompt` A string (may contain HTML) that will be displayed below the stimulus, intended as a reminder about the actions to take (e.g., which key to press).

`trial_duration` How long to wait for a response before ending trial in milliseconds. If NULL, the trial will wait indefinitely. If no response is made before the deadline is reached, the response will be recorded as NULL.

`response_ends_trial` If TRUE, then the trial will end when a response is made (or the trial_duration expires). If FALSE, the trial continues until the deadline expires.

`post_trial_gap` The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap.

`on_finish` A javascript callback function to execute when the trial finishes

`on_load` A javascript callback function to execute when the trial begins, before any loading has occurred

`data` An object containing additional data to store for the trial

Details

The `trial_video_slider_response` function belongs to the "stimulus-response" family of trials, all of which display a stimulus of a particular type (image, audio, video or HTML) and collect responses using a particular mechanism (button, keyboard or slider). This one plays a video and records responses generated with a slider.

Stmulus display: TBA

Response mechanism: Participant responses for this trial type are collected using a slider bar that the participant can move using the mouse. Once the participant is happy with this positioning they can click a button at the bottom of the page to move on to the next trial. This response method can be customised in several ways depending on the following arguments:

- The `min` and `max` arguments are numeric values that specify the minimum value (leftmost point on the slider) and the maximum value (rightmost point on the slider) that a participant can respond with.
- The `start` parameter is a numeric value that indicates where the value of the the slider is initially position. By default this is set to the middle of the scale, but there are many cases where it may be sensible to have the slider bar start at one end of the scale.
- The movement of the slider is discretised, and the granularity of this movement can be customised using the `step` parameter. This should be a numeric value that specifies the smallest possible increment that the participant can move the slider in either direction.
- The text labels displayed below the slider bar can also be customised by specifying the `labels` parameter. This argument should be a character vector that contains the labels to be displaed. Labels will be displayed at equally spaced intervals along the slider, though it is possible to include blank labels to create the impression of unequal spacing if that is required.

- The slider_width controls the horizontal width of the slider bar: the default value of NULL creates a slider that occupies 100% of the width of the jsPsych display. Note that this may not be 100% of the screen width.
- To ensure that participants do engage with the slider, it is possible to set require_movement = TRUE which forces the participant to move the slider at least once in order to be permitted to move onto the next trial.
- The button_label argument specifies the text displayed on the button that participants click to move to the next trial.

Other behaviour: As is the case for most trial_ functions there is a prompt argument, a string that specifies additional text that is displayed on screen during the trial. The value of prompt can contain HTML markup, allowing it to be used quite flexibly if needed.

Depending on parameter settings, the trial can end when the subject responds (response_ends_trial = TRUE), or after a fixed amount of time (specified using the trial_duration argument) has elapsed. The trial can also be made to end automatically at the end of the video using the trial_ends_after_video argument.

Like all functions in the trial_ family it contains four additional arguments:

- The post_trial_gap argument is a numeric value specifying the length of the pause between the current trial ending and the next one beginning. This parameter overrides any default values defined using the build_experiment function, and a blank screen is displayed during this gap period.
- The on_load and on_finish arguments can be used to specify javascript functions that will execute before the trial begins or after it ends. The javascript code can be written manually and inserted *as* javascript by using the insert_javascript function. However, the fn_ family of functions supplies a variety of functions that may be useful in many cases.
- The data argument can be used to insert custom data values into the jsPsych data storage for this trial

Data: When this function is called from R it returns the trial object that will later be inserted into the experiment when build_experiment is called. However, when the trial runs as part of the experiment it returns values that are recorded in the jsPsych data store and eventually form part of the data set for the experiment.

The data recorded by this trial is as follows:

- The rt value is the response time in milliseconds taken for the user to make a response. The time is measured from when the stimulus first appears on the screen until the response.
- The response is the numeric value of the slider bar.
- The stimulus variable records a JSON encoding of the sources array.

In addition, it records default variables that are recorded by all trials:

- trial_type is a string that records the name of the plugin used to run the trial.
- trial_index is a number that records the index of the current trial across the whole experiment.
- time_elapsed counts the number of milliseconds since the start of the experiment when the trial ended.
- internal_node_id is a string identifier for the current "node" in the timeline.

Value

Functions with a trial_ prefix always return a "trial" object. A trial object is simply a list containing the input arguments, with NULL elements removed. Logical values in the input (TRUE and FALSE) are transformed to character vectors "true" and "false" and are specified to be objects of class "json", ensuring that they will be written to file as the javascript logicals, true and false.

See Also

Within the "stimulus-response" family of trials, there are four types of stimuli (image, audio, video and HTML) and three types of response options (button, keyboard, slider). The corresponding functions are trial_image_button_response, trial_image_keyboard_response, trial_image_slider_response, trial_audio_button_response, trial_audio_keyboard_response, trial_audio_slider_response, trial_video_button_response, trial_video_keyboard_response, trial_video_slider_response, trial_html_button_response, trial_html_keyboard_response and trial_html_slider_response.

Index