JASMIN: JavaScript for the Social Sciences

JASMIN is a JavaScript framework for the Social Sciences; it provides a tool for administering surveys and response time tasks in a browser. This document aims to provide a brief overview of how JASMIN works, and how to use it.

[Introduction 2](#_Toc346985682)

[JASMIN File Structure 2](#_Toc346985683)

[my\_script.js 2](#_Toc346985684)

[Package generic 3](#_Toc346985685)

[Package survey 3](#_Toc346985686)

[SurveyNavigation 4](#_Toc346985687)

[Survey 5](#_Toc346985688)

[Items 6](#_Toc346985689)

[Package task 7](#_Toc346985690)

[ScalableCanvas 7](#_Toc346985691)

[TaskManager 7](#_Toc346985692)

# Introduction

JASMIN aims to realize the following features:

|  |  |
| --- | --- |
| **Multi-lingual** | Run the same study in multiple languages |
| **Task Templates** | Easy deployment of well-known paradigms (such as Implicit Association Task and Stroop) |
| **Online/offline** | Run it on your own computer (without an Internet connection) |
| **Compatible** | Works on all major browsers |

*Table. Features of JASMIN*

JASMIN is built to run in W**eb Browsers**. All surveys and tasks are structured as **HTML** and styled with **CSS**. JASMIN is built in **JavaScript**, using **JQuery** to generate HTML and register events (such as the user pressing a key). Client-server communication for setting up an experiment (such as determining a between-subjects condition) and/or logging experiment output (such as items answers) takes place via **AJAX**. The definition of questionnaires and tasks are expressed in **JSON**. Table 1 lists all these technologies.

|  |  |
| --- | --- |
| **Technology** | **Description** |
| Web Browser | Computer programs that used to be for surfing on the Internet, but have evolved to be platforms for full-fledged applications. |
| HTML | A way to structure web pages (tables, paragraphs, headings), but not their style |
| CSS | A way to style HTML elements (give them a color, size, position, etc.) |
| JavaScript | A programming language natively supported by many web browsers |
| JQuery | A JavaScript framework that provides a unified way to interact with different kinds of web browsers |
| AJAX | A way for browsers to communicate with a web server |
| JSON | A generic way to describe data structures |

*Table 1. Technologies involved in JASMIN*

# JASMIN File Structure

JASMIN is a JavaScript framework for the Social Sciences; JASMIN provides a library of objects and functions for administering questionnaires and response time tasks. These objects and functions can be used independent of one another, but they can be used most effectively when your experiment is structured in a particular way. By default, JASMIN experiments consist of three files:

|  |  |
| --- | --- |
| **File** | **Description** |
| my\_script.js | The JavaScript that manages your experiment |
| my\_specs.js | Specifications for surveys and tasks (expressed in JSON) |
| my\_translations.txt | Translations or all the terms used in the experiment |

*Table 2. The three files that define an experiment*

# my\_script.js

In the file “my\_script.js” you setup your experiment. This file needs to implement three functions:

|  |  |
| --- | --- |
| **Function** | **Description** |
| load | Called when the page loads. Here you load all resources required (translations, settings, images). |
| start | Called when all the resources are loaded. Here you can construct and start your surveys/tasks using the specs defined in *my\_specs.js* and the translations defined in *my\_translations.txt* |
| engine | Called when something happens while the experiment is running (an item is answered, the window loses focus, an error occurs). Via the engine you can make your surveys dynamic (show or hide pages, create new items, etc.). See the Engine Callbacks section for a list of all the events that trigger a call to engine. |

*Table 3. The three functions that manage an experiment*

To perform the tasks outlined in table 3, JASMIN provides three packages: **generic**, which provides general features for web-research; **survey**, which supports surveys; and **task**, which supports response time tasks.

# Package generic

Table 4 lists all of the classes of the genericpackage. By default, this package is already setup for you with one instance of each class (that requires an instance.

|  |  |  |
| --- | --- | --- |
| **Class** | **Instance** | **Description** |
| AjaxManager | ajaxManager | Manages client server communication |
| Loader | loader | Uses AjaxManager to load text and images. Provides means to display progress |
| Logger | logger | Uses AjaxManager to (1) log the data you want to analyze later, and (2) synchronize the state of your tasks/surveys with the server |
| Dialog | dialog | Draws a small dialog screen, similar to the JavaScript alert() |
| FocusManager | focusManager | Tracks is the window has focus, and provides callbacks for warning the participant and/or pausing and restarting tasks |
| Translator | translator | Translates terms and provides callbacks for programmable translations. Use “#[term]” in surveys and tasks to have a certain term translated. |
| Dates |  | Extends the JavaScript Date object with functions for date formatting and calculating someone’s age |
| utils |  | Debugging functions such as *vardump* |
| Statistics |  | For statistics and randomizations |
|  |  |  |

*Table 4a. Main classes of the generic package*

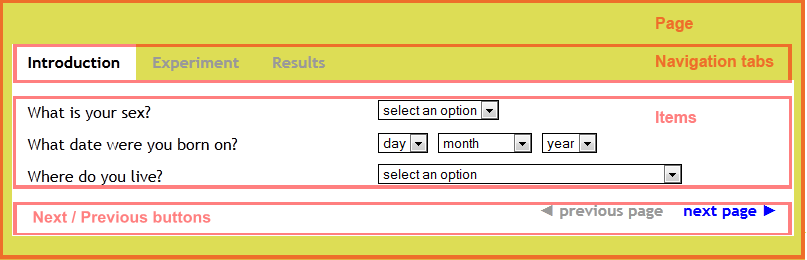
These classes are not used anymore:

|  |  |  |
| --- | --- | --- |
| **Class** | **Instance** | **Description** |
| Popup | popup | Manages a window for the PopupManager. If the window closes, it attempts to synchronize the data in the Logger with the server. |
| PopupManager | popupManager | Manages popups |

*Table 4b. Obsolete classes of the generic package*

# Package survey

A survey is organized in tabs, pages, and items. Survey pages and items are managed by the Survey class, while navigation over pages and tabs is managed by the SurveyNavigation class. Each page in the survey contains three parts: (1) a list of navigation tabs; (2) a list of items, (3) next/previous buttons. See picture 1 for an visual impression of these three parts.

*Picture 1. The three parts of a survey page*

## SurveyNavigation

SurveyNavigation determines the order in which the pages of a Survey are administered, and the progress of the participant. SurveyNavigation is setup with a navigationSpecs array, which organizes a Survey in tabs and pages. The tabs can be used to indicate progress in the survey and navigation between pages. Each tab can contain one or more pages, to be administered in sequence. Table 5 lists the properties of a tab.

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Example** | **Description** |
| id | string | “my\_tab” | Tab ID (handier than the tab index) |
| name | string | "#[tab\_name]" | Name of navigation tab displayed to participant |
| pages | index array | [ "page1", "page2" ] | Pages that belong to this tab |

*Table 5. Properties of a tab.*

SurveyNavigation maintains the following state variables:

|  |  |
| --- | --- |
| **Property** | **Description** |
| tabs | An array structured likenavigationSpecs, properties of *tabs* override thos of navigationSpecs, allowing one to change the name of a tab and/or the pages that belong to it. |
| currentTab, currentPage | The tab & page that the participant is currently at |
| furthestTab, furthestPage | The furthest tab & page that the participant has been so far |

*Table 6. SurveyNavigation state variables*

In determining which page a participant should visit, SurveyNavigation applies the following rules:

* The next button sends participants to the next page of a tab, or to the first page of the next tab, if no pages are left on the current tab.
* The previous button sends participants to the previous page of a tab, or to the last page of the previous tab, if no pages are left on the current tab.
* The next and previous buttons are only visible if a previous and next page are available
* If a participants clicks a tab, then the following happens:
  + If this tab is the furthest tab we’ve been so far, then send the participant to the currentPage of this tab
  + If this tab is an earlier tab, then send the participant to the first page of this tab

SurveyNavigation also determines if all the required items on the current page should be answered before allowing the participant to go to another page. Only when the participant is moving backwards from the furthest tab and page, is he allowed to leave required items unanswered.

## Survey

A Survey can be setup with the surveySpecs array, which contains three elements: “general”, for general setting, “pages”, for the pages, and “items” for the items. The tables below list the properties of “general” and each page in “pages”. The properties of the items array are described in the section section.

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Example** | **Description** |
| source | string | "my\_survey" | Name used as source for logged events |
| itemspacing | string | "6px" | Vertical space between items |

*Table 7. Survey general properties*

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Example** | **Description** |
| navigation | bool | true | Show navigation bar |
| prevNext | bool | false | Show "previous" and "next" buttons |
| class | string | "PageDefault" | Name of a CSS class for styling the page ("PageDefault" for normal, "PageFull" for maximum size) |
| rounded | bool | false | Use rounded corners? (warning: this option may be removed) |
| items | index array | [ "age", "gender" ] | Array with names of items displayed on this page |

*Table 8. Survey page properties*

Survey maintains the following state variables:

|  |  |
| --- | --- |
| **Property** | **Description** |
| pages | An array structured like“pages” in surveySpecs, overrides surveySpecs |
| items | An array structured like“items” in surveySpecs, overrides surveySpecs. By setting the “answer” property an item can be preset with a certain answer. |

*Table 9. SurveyNavigation state variables*

## Items

Items are managed by Survey. A set of prefab items are available for most common situations, and these can be automatically constructed based on the item specs provided to the Survey. Each item has two default properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Example** | **Description** |
| required | bool | true | If true, then item needs to be answered before going to the next page |
| type | string | "likert" | Item type |
| specs | assoc array | { … } | Specifications of item (different specifications apply to different item types; see below |

*Table 10. Default item properties*

The following item types are available. Listed with each item, are the properties that can be setup in the specs array.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Specs** | **Type** | **Example** | **Description** |
| date |  |  |  | *Creates drop-down menus in which a date (day, month, year) can be entered. Uses the term "year", "month", and "date" as empty options for the drowpdowns.* |
|  | question | string | "How old are you?" | *Question* |
|  | order | string | *"d m y"* | *The order in which day (d), month (m) and year (y) are presented.* |
| dropdown |  |  |  | *Creates a drop-down menu* |
|  | question | string | "Which flavor would you like?" | *Question* |
|  | empty | string | "Pick an answer…" | *Text for initial (empty) answer option* |
|  | options | assoc array | { "b" : "banana", "c" : "chocolate" } | *Array of answer options, the key is the code used for an answer option, the value the text dispayed to the participant* |
| likert |  |  |  | *Creates a list of questions with likert-scale answer options* |
|  | questions | index array | [ "Do you like it?", "Can you dig it?" ] | *Array of questions* |
|  | options | int | 4 | *Number of answer options* |
|  | scale width | int | 400 | *Width of likert scale in pixels* |
|  | labels options | index array | [ "1", "2", "3", "4" ] | *(optional) Array labels for individual answer options; number of elements should match options* |
|  | labels ends | assoc array | { "left" : "hardly", "right" : "very much" } | *(optional) Array labels for scale ends, "left" for left end, and "right" for right end* |
| open |  |  |  | *A single line text input* |
|  | question | string | "Which flavor would you like?" | *Question* |
| paragraph |  |  |  | *A paragraph with text (not actually an item that requires any response). If the itemData.answer of a paragraph item has a value, then this value is displayed instead of the "text" property.* |
|  | question | string | "Some supportive text" | *Question* |
| custom |  |  |  | *A custom item* |
|  | constructor |  | null | *The item constructor function* |

*Table 11. Item types and specs*

# Package task

The task package provides modules for response time tasks. The table below lists these modules.

|  |  |
| --- | --- |
| **Class** | **Description** |
| EventManager | Times trial events and registers responses/response times |
| ScalableCanvas | Scales a set of sprites on a vistual canvas to uniformly fit to a container DIV |
| SlideShow | Presents slideshows |

*Table 4. Main classes of the generic package*

## Task engines

For several tasks there are *task engines* available. A task engine runs a particular paradigm (such as an IAT) in a standardized way. To do this, each task has some standard properties, as listed below. These properties are described per task in a separate document, see “CBM Tasks Manual.docx” for descriptions of the VPT, AAT, BIAT, and Stroop tasks.

|  |  |
| --- | --- |
| **Type** | **Description** |
| Terms | A set of terms used for instructions (such as invalid key feedback) |
| Eventlogs | A set of events that are logged for later analysis |
| Engine | A module that creates a configuration variable for the engine |
| Config | A configuration variable that is used by the engine to customize the task |

The next sections describe the terms, eventlog, engine and config of each task engine. Some properties are shared across tasks. These properties are listed below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Type** | **Example** | **Description** |
| responseWindow | string | 4000 | *Number of ms that the task waits for a response* |
| buttons | assoc array | *{ 73 : "up", 78 : "down" }* | *Keycodes and labels of response buttons* |
| stimuliType | string | *"picture”* | *Type of stimuli: "text" or "picture"* |
| translationCallbacks | assoc array |  | *Array of the form { term: function() }; any translation in which term is used, the translation of this term is provided by the value returned by function()* |
| … |  |  | *Additional configuration keys that depend on the particular task* |
| blocks | index array |  | *Array of the form [ { “intro”: [], “keys” : [], “trials” [] } ]. Each block starts with a set of introduction slides as specified in “intro”; trial settings as specified in “trials”; and a text reminding the participant of the buttons as specified in “keys”. A block may contain additional values that depend on the particular task.* |

# Engine callbacks

An experiment can be made dynamic by changing the state of Survey or Task in response to a call to your engine function. The engine function receives one argument, an associative array containing information about the event that triggered the engine callback. The “action” property in this array identifies what kind of event happened.

|  |  |
| --- | --- |
| **action** | **Description** |
| next | Participant clicked the “next” button (but did not yet leave the page). Return false to disallow the participant leaving the page. |
| previous | Participant clicked the “previous” button (but did not yet leave the page). Return false to disallow the participant leaving the page. |
| drawn | Called after a page has been drawn |
|  |  |
| focus | Called when the window received focus (e.g. the user clicked in a window that did not yet have focus) |
| blur | Called when the window lost focus (e.g. the user clicked outside of a window that had focus) |
| error | An error occurred, the “message” property contains an error message. |

# Default logs

The following are a list of events that are automatically logged.

|  |  |  |
| --- | --- | --- |
| **source** | **type** | **Description** |
| FocusManager | focus | Window received focus |
| FocusManager | blur | Window lost focus |
| Loader | loading start | Loading of resources started |
| Loader | loading done | Loading of resources completed |
| Item | change | The answer of an itemchanged. name = item name, params = current answer |
|  |  |  |
|  |  |  |