Behav Res (2012) 44:314–324
DOI 10.3758/s13428-011-0168-7

OpenSesame: An open-source, graphical experiment builder for the social sciences

Sebastiaan Mathôt · Daniel Schreij · Jan Theeuwes

Published online: 16 November 2011

Abstract In the present article, we introduce OpenSesame, a graphical experiment builder for the social sciences. OpenSesame is free, open-source, and cross-platform. It features a comprehensive and intuitive graphical user interface and supports Python scripting for complex tasks. Additional functionality, such as support for eyetrackers, input devices, and video playback, is available through plug-ins. OpenSesame can be used in combination with existing software for creating experiments.

Keywords Software · Stimulus presentation · Experiment builder · Python · Graphical user interface

Introduction

A little over 20 years ago, Schneider (1988) estimated that it took a professional programmer approximately 160 h to implement a new experimental paradigm. Fortunately, things have changed since then. Advances in both software and hardware have made it possible for unskilled programmers to develop their experiments rapidly, using any of the available "point-and-click" software packages (for a comparison, see Stahl, 2006). Scientists who prefer programming over the use of a graphical interface also benefit from the power of modern, high-level programming languages, which have substantially improved the readability and reduced the amount of code required to perform most tasks.

S. Mathôt (✉) · D. Schreij · J. Theeuwes
Department of Cognitive Psychology, Vrije Universiteit,
Van der Boechorststraat 1,
1081 HV Amsterdam, The Netherlands
e-mail: s.mathot@vu.nl

Another important development is the increased availability of high-quality, free experimental software. Currently, there are at least eight free software packages that are viable tools for creating experiments (see Table 1). However, these packages occupy a relatively small niche, mostly because they do not offer the type of fully graphical interface that many users have come to expect. Therefore, researchers who are most comfortable in a graphical environment generally use proprietary software.

In the present article, we introduce OpenSesame, a new experiment builder. OpenSesame is unique in that it is free, cross-platform, and arguably, offers the most intuitive and comprehensive graphical user interface (GUI) currently available. For complex tasks, which cannot be performed through the GUI, OpenSesame supports Python scripting (Van Rossum & Drake, 2011). A wide range of experiments can be created, including psychophysical experiments, speeded response time tasks, eye-tracking studies, and questionnaires.

In the first section of the present article, we provide a nontechnical description of the functionality offered by OpenSesame. In the second section, we describe how OpenSesame compares with, and can be used in combination with, existing software. The third section deals with timing considerations and is followed by the fourth section, in which the results of a benchmark experiment are described.

OpenSesame is freely available for download from http://www.cogsci.nl/opensesame. Documentation, a step-by-step tutorial, example experiments, and plug-ins can be found in the documentation area at http://osdoc.cogsci.nl/. The version of OpenSesame reviewed in the present article is 0.24. At the time of writing, OpenSesame has been downloaded more than 10,000 times (nonunique downloads from cogsci.nl and via bit-torrent), and there is an active support forum.

Table 1 An overview of software for creating experiments

Name	GUI	Free	Scripting	Platform	Reference/Vendor
DirectRT	Yes	No	Custom	Windows	Reviewed in Stahl (2006)
DMDX	No	Yes*	Custom	Windows	Forster & Forster (2003)
E-Prime	Yes	No	E-Basic	Windows	Reviewed in Stahl (2006)
Experiment Builder	Yes	No	Python	Windows	SR Research, Missisauga, ON, Canada
Inquisit	Yes	No	Custom	Windows	Reviewed in Stahl (2006)
MATLAB Psychophysics Toolbox	No	Yes**	MATLAB	Windows, Mac OS, Linux	Brainard (1997)
MEL	No***	No	Custom	IBM PC	Schneider (1988)
PEBL	No	Yes	Custom	Windows, Mac OS, Linux	Mueller (2010)
Presentation	Yes	No	Custom	Windows	Neurobehavioral Systems, Albany, CA
PsychoPy	Yes	Yes	Python	Windows, Mac OS, Linux	Peirce (2007)
PsyScope	Yes	Yes	Custom	Mac OS	Cohen, MacWhinney, Flatt, & Provost (1993)
PsyToolkit	No	Yes	Custom	Linux	Stoet (2010)
PyEPL	No	Yes	Python	Mac OS, Linux	Geller et al. (2007)
SuperLab	Yes	No	Custom	Windows	Reviewed in Stahl (2006)
Tscope	No	Yes	C/C++	Windows****	Stevens, Lammertyn, Verbruggen, & Vandierendonck (2006)
Vision Egg	No	Yes	Python	Windows, Mac OS, Linux	Straw (2008)

* Source-code is not available

** Depends on MATLAB (The MathWorks, 1998), a proprietary software package, for full functionality. Offers limited support for Octave (Eaton, 2002), an open-source MATLAB clone

*** Uses a form-based interface

**** Offers limited support for Mac OS and Linux

Usage and functionality

System requirements

OpenSesame does not impose strict system requirements. Installation packages are available for Windows XP/Vista/7 and Ubuntu/Debian Linux. OpenSesame has been extensively tested on those platforms. At the time of writing, packages for Mac OS are also available but have been labeled "experimental" pending further testing. On other platforms, users will need to manually install the software on which OpenSesame depends and run OpenSesame from the source code. Instructions for running OpenSesame from source are provided online.

The processing power required to run OpenSesame depends strongly on the type of experiment. For a typical experiment, consisting of a sequence of static stimulus displays followed by response collection, the requirements are very modest, and any relatively modern computer system will suffice (we have successfully run OpenSesame on a low-end netbook; 2-GB Ram, 1.66-GHz Intel Atom N270). When using complex stimuli, such as high-definition video, the user will need to evaluate for him- or herself whether the computer system is up to the task.

The graphical user interface

After starting OpenSesame, the user is presented with a GUI (see Fig. 1). The structure of the experiment is depicted graphically as a tree structure in the overview area (Fig. 1b). An experiment consists of a collection of *items*, which are self-contained parts of the experiment (i.e., conceptually distinct units). Items can be added to the experiment by dragging them from the item toolbar (Fig. 1a) onto the overview area. Items can be edited by selecting them in the overview area, after which the appropriate controls appear in the tab area (Fig. 1c).

There are 10 core items that provide the basic functionality for creating an experiment (see Table 2 for an overview; functionality can be extended as described in the Usage and functionality: Plug-ins section). Examples of items are the *sketchpad*, which handles the presentation of a single stimulus display, and the *keyboard_response*, which handles the collection of a single keyboard press. Two special items are the *loop* and the *sequence*, which control the structure of the experiment. A *sequence* item sequentially calls a number of other items. A *loop* item repeatedly calls a single other item, while varying the values of independent variables. By combining items in various

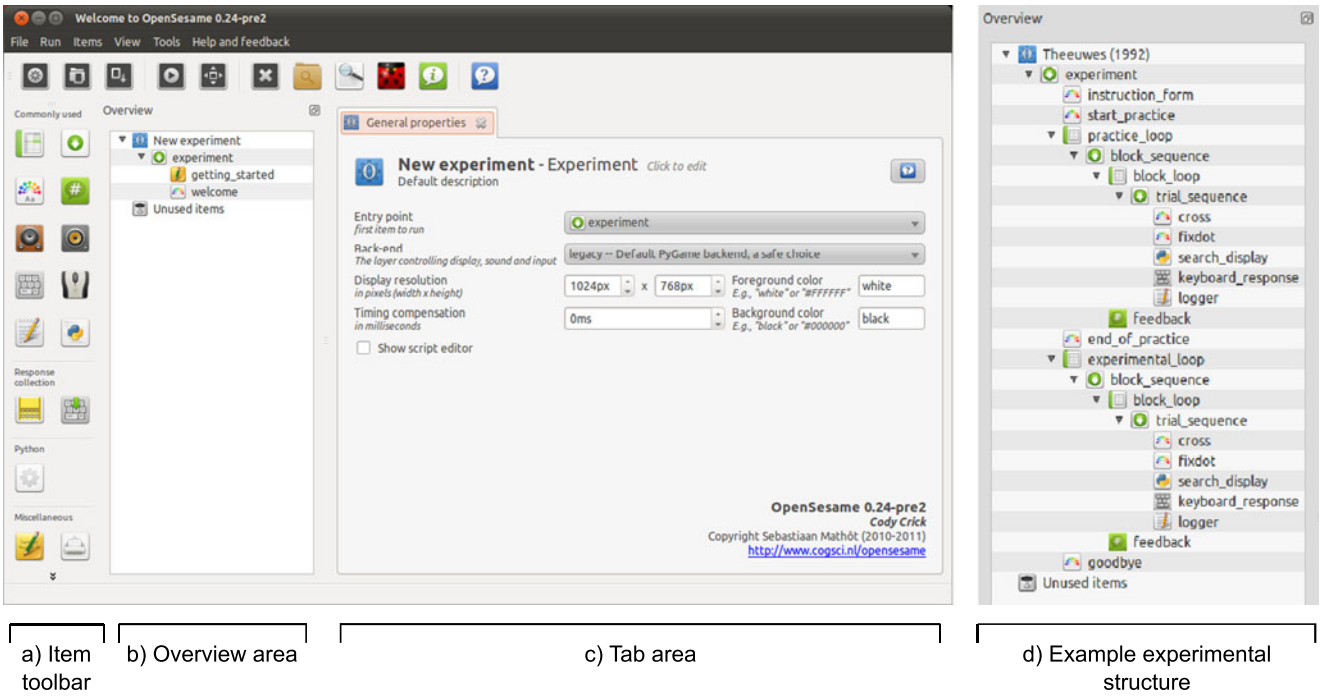


Fig. 1 The OpenSesame graphical user interface on start-up. **a** The item toolbar contains icons that can be dragged into the overview area. **b** The overview area represents the experiment as a tree-structure. **c** The tab area contains tabs for editing items and getting context-sensitive help. By clicking on an item in the overview area, a tab with the appropriate controls opens. By clicking on one of the blue "help" buttons, a context-sensitive help tab opens. **d** The structure of an example experiment shown in the overview area

ways, arbitrary experimental paradigms can be constructed (Fig. 1d).

Variables and conditional ("if") statements

One of the biggest challenges when designing a GUI is to offer sufficient flexibility. In OpenSesame, this flexibility is achieved through the support of variables and conditional ("if") statements.

Variables can be built in (e.g., *subject_nr*), set by items (e.g., *response_time*, which is set by *keyboard_response* items), or specified by the user in *loop* items (see Fig. 2a). These variables can be used throughout the GUI, by entering *[variable_name]* in places where you would normally encounter a static value. For example, if the user has defined a variable called *SOA*, this variable can be used to control the duration of a *sketchpad* item (i.e., a stimulus display) by entering *[SOA]* in the duration field (Fig. 2b). Analogously, feedback of the average response times can be given by adding the text "Your average response time was [avg_rt]msg" to a *feedback* item (Fig. 2c).

Particularly powerful is the possibility of combining the what-you-see-is-what-you-get *sketchpad* drawing tool (shown in Fig. 2b, c) with the use of variables. The drawing tool automatically generates a simple script that defines the elements in the *sketchpad*. By replacing the static coordinates, colors, sizes, and so forth with variables, the user can create a flexible stimulus display in an intuitive way. For example, if you insert an image (from the file *fork_left.png*) in the center of the display, OpenSesame will generate the following line of script (taken from the *affordances_orientation* example experiment, available online):

```
draw image 0 0 "fork_left.png" scale=1 center=1
show_if="always"
```

By right clicking on the object, you are given the possibility of editing this line. The static values can be replaced by variables, and thus the presented image, as well as the image's size, can be made variable (this will result in the object being hidden from the drawing tool, with the message that the sketchpad contains a variably defined object):

```
draw image 0 0 "[object]_[orientation].png" scale=[scale]
center=1 show_if="always"
```

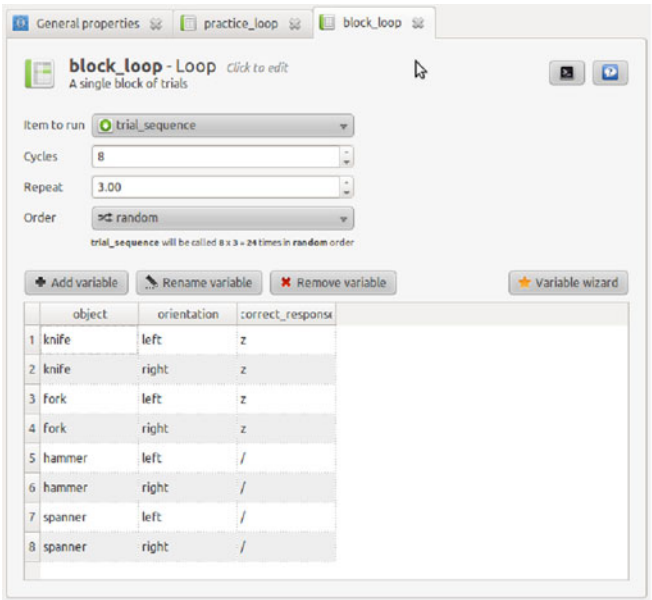
Conditional statements, commonly referred to as "if" statements, can be used to add even more flexibility to the GUI. Every item from a *sequence* item has a "Run if" parameter that is evaluated before the item is called. This can be used, for example, to show a green or red fixation dot, depending on whether the preceding response was correct (using the *correct*

Table 2 An overview of the 10 core items

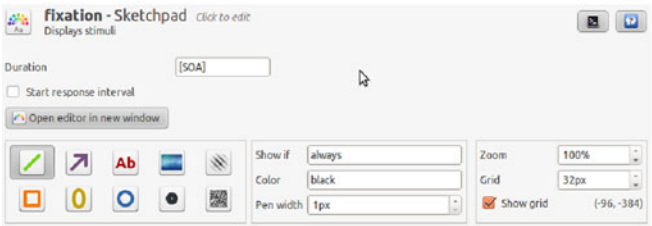
Name	Type	Description
loop	Structure	Repeatedly runs a single other item. Controls independent variables.
sequence	Structure	Runs multiple other items in sequences. Supports basic conditional statements ("Run if …").
sketchpad	Stimulus presentation	Provides a canvas for presentation of visual stimuli.
feedback	Stimulus presentation	Provides feedback to participants.
sampler	Stimulus presentation	Plays a sound from file.
synth	Stimulus presentation	Provides basic sound synthesis.
keyboard_response	Response collection	Collects keyboard responses.
mouse_response	Response collection	Collects mouse responses.
logger	Data logging	Writes variables to file.
inline_script	Inline scripting	Executes arbitrary Python code.

variable, which is automatically set by response items; see Fig. 2d). Analogously, in *sketchpad* and *feedback* items, conditional statements can be used to control which elements are actually shown, by setting the "Show if" parameter.

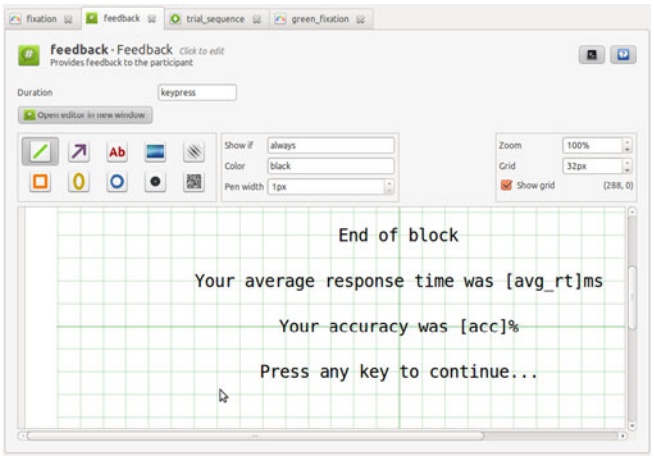
a) variables defined in a *loop*



b) using a variable to specify the duration of a *sketchpad*



c) using variables to provide feedback



d) conditional statements in a *sequence*

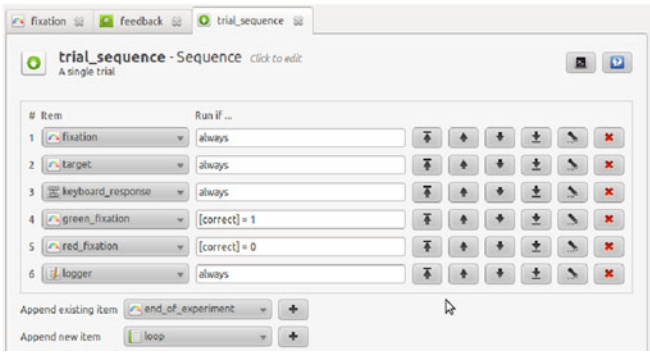


Fig. 2 Using variables and conditional ("if") statements. **a** Independent variables can be defined in a *loop* item. **b** By entering *[SOA]* in the duration field of a *sketchpad* item, the variable *SOA* is used to control the presentation duration of the sketchpad. This assumes that *SOA* has been defined elsewhere in the experiment. **c** By using *[avg_rt]* and *[acc]* as part of the text in a feedback item, appropriate feedback can be given to the user. The variables *avg_rt* and *acc* are set automatically by the various response items (e.g., *keyboard_response*). **d** Conditional statements can be used to control which items from a *sequence* will be called. By entering *[correct]= 1* in the "Run if…" field of "green_fixation," the item will be called only if the variable *correct* has been set to *1*. The variable *correct* is set automatically by the various response items

Data output format

Usually, data logging will be handled by the *logger* item. Every time that a *logger* item is called, a single row of data is written to the output file. The output file is a plain-text comma-separated spreadsheet (.csv), in which each row typically corresponds to a trial and each column corresponds to a variable. This format is compatible with all commonly used spreadsheet software.

Alternatively or in addition, users can write arbitrary messages to the output file, using Python code in *inline_script* items. In this case, the format of the output file is determined by the user.

Python inline coding

Despite the flexibility of the GUI, there will sometimes be situations that require a full-fledged programming language. For this reason, OpenSesame supports Python scripting (Van Rossum & Drake, 2011). Python is a powerful programming language that is widely used in the scientific community and the computing world in general (the seventh most widely used programming language, according to Tiobe.com, 2011).

To use Python scripting, you add an *inline_script* item to the experiment. Rather than the knobs, buttons, and input fields that you will see when editing other items, the *inline_script* item offers an embedded programming editor. You can use the OpenSesame Python modules, which offer simple routines for handling display presentation, sound generation, response collection, and so forth. Alternatively, you can interact directly with the selected back-end (see the Usage and functionality: The back-end layer section). The latter option is mostly useful for advanced users who are familiar with one of the back-ends (i.e., PsychoPy [Peirce, 2007], PyGame, and/ or OpenGL) or if the native OpenSesame modules do not offer the desired functionality.

File format and the file pool

External files, such as images, sound files, and videos, are stored in the file pool (Fig. 3). Optionally, OpenSesame also copies participant data files to the file pool after an experiment has finished. The file pool is saved along with the experiment in a single *.opensesame.tar.gz* file. This extension is somewhat ungainly but accurately reflects that the file is a *.tar.gz* archive, which can be opened in most archiving tools (e.g., WinRar or FileRoller). Therefore, in the unlikely event that OpenSesame fails to open the experiment, you can still access the experiment script and any files that have been stored in the file pool.

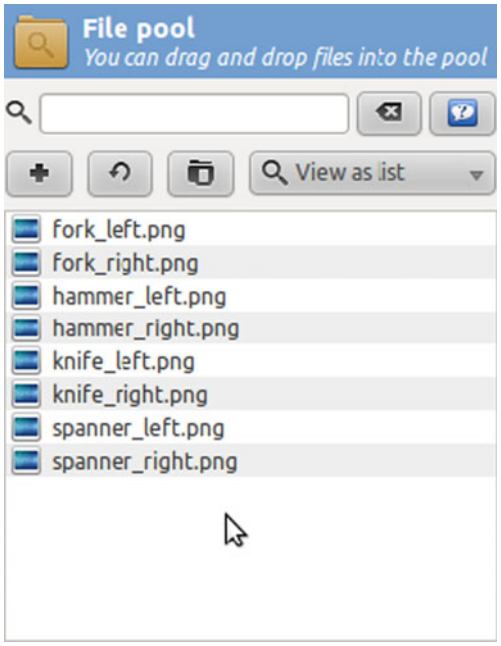


Fig. 3 The file pool. All files that are used in the experiment, such as images, sounds, and videos, are stored in the file pool. The file pool is saved along with the experiment, allowing for maximum portability

Plug-ins

OpenSesame can be extended through plug-ins. Plug-ins offer graphical controls, appear in the item toolbar, and can be dragged into the experiment just like the built-in items. Therefore, from the perspective of the user, there is little difference between using a plug-in and using any of the 10 core items.

Plug-ins offer arbitrary functionality, such as support for specific devices or handling particular tasks that would otherwise have to be implemented through Python inline code. Currently, there are plug-ins available that support the Eyelink series of eyetrackers (SR Research, Mississauga, ON, Canada), the Mantra object tracker (Mathôt & Theeuwes, 2011), the Serial Response Box (Psychology Software Tools, Sharpsburg, PA), input devices connected to the parallel port, and video playback.

The back-end layer

OpenSesame consists of three distinct layers. The top layer handles the GUI and offers the functionality needed to create an experiment. The middle layer handles the execution of an experiment. The bottom, or back-end, layer handles the interaction with the display, sound, and input devices.

There are many different Python libraries available that could, in principle, be used in the back-end layer. Some, such as PyGame, Pyglet, and PyOpenGL, are general

purpose, mostly oriented toward development of games. Others, such as PsychoPy (Peirce, 2007), VisionEgg (Straw, 2008), and PyEPL (Geller, Schleifer, Sederberg, Jacobs, & Kahana, 2007), have been developed specifically for creating psychological experiments.

OpenSesame is back-end independent, in the sense that different libraries can be used in the back-end layer. New back-ends can be created and added, much like plug-ins. Currently, there are three back-ends available: the *legacy* back-end, which uses PyGame; the *psycho* back-end, which uses PsychoPy (Peirce, 2007); and the *opengl* back-end, which uses PyGame in combination with PyOpenGL (see Table 3).

From the perspective of the GUI user, there is little noticeable difference between the back-ends (although there may be small differences in anti-aliasing and font rendering). A red square will always be a red square, regardless of which library is used to draw it. Nevertheless, there are compelling reasons for having different back-ends to choose from. The first is that not all back-ends may be equally well supported on all platforms. Second, back-ends may differ in their timing properties (see the Benchmark experiment section). Third, each back-end offers unique functionality that the user can exploit when writing Python inline code. For example, if the *psycho* back-end is enabled, you can use the PsychoPy routines for creating visual stimuli. More generally, users can select the back-end that they are most familiar with and best suits their own approach to creating experiments.

Comparison with and interoperability with existing software

Table 1 provides a list of the most popular software for creating experiments. There are already many software packages to choose from, yet the functionality offered by OpenSesame is unique. In this section, we will focus on two packages with which OpenSesame arguably has the most in common: E-Prime (Psychology Software Tools, Sharpsburg, PA) and PsychoPy (Peirce, 2007).

E-Prime has been built on the legacy of the Micro Experimental Library (MEL; Schneider, 1988). Largely because it was one of the first programs to offer a graphical environment for creating experiments, E-Prime has, in our experience, become the de facto standard in many psychological labs around the world. There are at least four important differences between E-Prime and OpenSesame. The most obvious difference is that OpenSesame is free and open-source, whereas E-Prime is nonfree and proprietary. Because of the open character of OpenSesame and the availability of a plug-in framework, it is easy for third parties to add and modify functionality. Second, OpenSesame is cross-platform, whereas E-Prime is exclusively available for Windows. A final, crucial difference is the language that is used for inline coding. Whereas E-Prime uses E-Basic, a dialect of the well-known BASIC language, OpenSesame uses Python (see the Usage and functionality: Python inline coding section). The advantage of using Python over E-Basic is the availability of a large number of libraries, many of which are oriented toward the scientific community (e.g., Jones, Oliphant, & Peterson, 2001).

PsychoPy is an open-source project that has gained considerable momentum as a comprehensive and well-maintained Python library for creating psychological experiments (Peirce, 2007). Like OpenSesame, PsychoPy is cross-platform and open-source. And like OpenSesame, PsychoPy offers both a GUI (the "builder view") and a set of Python libraries. However, there are substantial differences between the graphical interfaces offered by both packages. In the builder view of PsychoPy, the temporal progression of the experiment is shown, but the spatial arrangement of the stimuli is not readily apparent. In OpenSesame, the temporal progression is shown as well (in the overview area; see Fig. 1b, d), but in addition, the user can get a visual preview of the stimulus arrangements through the *sketchpad* item (Fig. 1c). Depending, of course, on the prior experience and personal preference of the user, such a preview can be very helpful. OpenSesame also offers a number of advanced features, which are not available in the PsychoPy builder view, such as integrated drawing tools, a more advanced sound synthesizer, and GUI support for widely used devices such as the Serial Response Box (Psychology Software Tools, Sharpsburg, PA) and the Eyelink series of eyetrackers (SR Research, Mississauga, ON, Canada). More generally, OpenSesame and PsychoPy differ in their target audience and core functionality.

Table 3 An overview of OpenSesame back-ends

Back-End Name	Hardware Accelerated	Underlying Technology
legacy	No	PyGame in non-OpenGL mode
opengl	Yes	PyGame in OpenGL mode
psycho	Yes	PsychoPy in Pyglet window mode (Peirce, 2007)

Although PsychoPy provides a GUI, the focus is on the specialized set of Python libraries, which offer high-level routines for creating complex visual stimuli, particularly those that are frequently used in psychophysical experiments (e.g., Gabor patches and random dot patterns). For this reason, PsychoPy will appeal mostly to people who prefer to code their experiments, such as MATLAB users who are looking for a viable open-source alternative. In contrast, OpenSesame offers only basic Python libraries but has a comprehensive GUI and will, therefore, appeal to users who are mostly at home in a graphical environment. As was discussed previously (see the Usage: The back-end layer section), OpenSesame and PsychoPy can be used in combination. This allows users to get "the best of both worlds" by combining the OpenSesame GUI with the PsychoPy Python libraries.

Timing

What is "millisecond precision timing"?

A common and valid question that applies to all experiment-building software is whether the timing is sufficiently precise. In our view, the best measure of a system's timing precision is the interval between the timestamp of a stimulus' presentation and the timestamp of a response, given the fastest possible responder that is not directly part of the system itself. Put more simply, the lowest possible response time is a good indication of a system's timing precision. The reason for this is that any inaccuracies in the timing of display presentation and response collection will add to the lowest possible response time. Perhaps even more importantly, the lowest possible response time should be consistent, so that any delay is fixed and does not constitute a source of noise (see also the Benchmark experiment section).

However, even when a system has "millisecond precision timing" in this sense, as most modern systems do, there are other factors that should be taken into account. First, many psychological labs, including our own, often use garden variety keyboards as input devices. Such keyboards have been designed to keep up with typing, and not to record responses with millisecond precision. As such, keyboards have been reported to have a relatively high and variable latency of up to 20 ms (however, it is debatable whether this constitutes a significant source of noise, when the much larger variability in human response times is considered; for a discussion, see Damian, 2010).

Computer monitors also have a number of peculiar properties. Rather than being refreshed instantaneously, monitors are refreshed line by line from the top down and, for each line, pixel by pixel from left to right. On CRT (nonflat screen) monitors, there is only a single active pixel at any given time, so pixels are "fading out" most of the time (which is why you can observe a flickering at low refresh rates). On TFT (flat screen) monitors, pixels remain active, so a refresh resembles a "flood fill" from top to bottom (which is why you do not observe flickering on a TFT monitor). Most researchers are aware that it is best to synchronize the presentation of a new display with the moment that the monitor starts refreshing from the top (*synchronization to vertical refresh*, or *v-sync*). However, even if this is done, displays do not appear instantaneously. With a refresh rate of 100 Hz, a stimulus presented at the top of the display will appear up to 10 ms before a stimulus presented at the bottom of the display. Another consequence of the monitor's refresh cycle is that displays cannot be presented at arbitrary points in time. Again, with a refresh rate of 100 Hz, the interval between two stimulus displays is always a multiple of 10 ms. Attempting to use different intervals will lead to dropped frames or aberrant timing.

The prepare–run strategy

Psychological experiments typically consist of short intervals, called *trials*, during which a participant perceives stimuli and performs a task. Timing should be controlled during a trial, but some unpredictable variation in the duration of the interval between trials is acceptable (cf. Schneider, 1988). Therefore, the best strategy for experimental software is to perform time-consuming tasks before a trial and to keep the operations that need to be performed during a trial to a bare minimum.

OpenSesame implements this strategy by calling each element from a *sequence* item twice. During the *prepare* phase, items are given time to prepare—for example, by generating a sound in the case of a *synth* item, or by creating an "offline canvas" in the case of a *sketchpad*. During the *run* phase, items simply execute a very limited number of trivial functions, such as showing a previously constructed canvas. This strategy substantially reduces the risk of timing glitches. The prepare–run strategy is implemented at the level of *sequence* items, which will typically contain the time-critical parts of an experiment. If a *sequence* is called multiple times by a *loop*, the loop as a whole is not prepared. Doing so would quickly lead to excessive memory usage and potentially cause, rather than prevent, timing issues.

Testing your own system

The timing properties of OpenSesame, or any other experiment builder, depend on the computer setup that is used. Therefore, OpenSesame comes with a basic test

experiment (*test_suite*) that allows you to run a number of checks. Most importantly, the test experiment checks whether the reported interval between two presented *sketchpads* matches the specified interval. The same test allows you to verify that synchronization to the vertical refresh is enabled (see the Timing: What is "millisecond precision timing"? section). If not, "tearing," in the form of horizontal lines running through the display, will be readily apparent.

Benchmark experiment

Even though the included test experiment allows you to run some useful checks (see the Timing: Testing your own system section), it is important to note that this form of testing relies on the computer's self-report. And this can be misleading. Specifically, the computer may report that a display has been presented when, in fact, it has only been queued and will be presented some time later. The aim of the present experiment was, therefore, to check timing precision more rigorously and to provide a list of benchmarks that allow the reader to evaluate whether OpenSesame is sufficiently precise to be useful in his or her experimental setting.

We used an artificial responder that responds, for all intents and purposes, instantaneously to an increase in light intensity. We measured the response times to a white display. Both inaccuracies in the reported time of display presentation (assuming that inaccuracies are always such that the display is presented after the reported time, and never before) and inaccuracies in the reported time of the response (assuming that inaccuracies are always such that the reported time of the response is after the actual response) add to the lowest possible response time. The lowest possible response time can therefore be used as a measure of the combined timing precision of display presentation and response collection.

Method

Two different test systems were used (the system specifications are listed in Table 4). A black display was presented for 100 ms, followed by a white display. Automated responses to the white display were collected using a modified button-box that responds to an increase in light intensity (i.e., the presentation of the white display) through a photo-resistor. This photo-resistor was attached to the top-left of the monitor. To ensure that we obtained realistic results (i.e., not tweaked to optimize performance), the test experiment was created entirely using the GUI, and no Python inline code was used. The serial response box plug-in was used to interface with the button-box.

The experiment was run under Windows XP (Systems 1 and 2) and Ubuntu Linux 10.04 (System 2). The following combinations of resolution and refresh rate were tested: 1,680 × 1,050 at 120 Hz (System 1); 1,280 × 1,024 at 85 Gz (System 2); 1,024 × 768 at 60 Hz (System 2). A color depth of 32 bits was used in all cases. All three currently

Table 4 Results of the benchmark experiment

Operating System	Test System	Display Mode	Back-End	Average Response Time (ms)	Standard Deviation of Response Time (ms)	Sync to Vertical Refresh Enabled
Windows XP	System 1 *	1,680×1,050 @ 120 Hz	psycho	3.28	0.53	Yes
			legacy	11.94	2.45	Yes
	System 2 **	1,024×768 @ 60 Hz	psycho	3.18	0.53	Yes
			legacy	16.01	2.59	Yes
		1,280×1,024 @ 85 Hz	psycho	3.02	0.53	Yes
			legacy	9.24	1.37	Yes
Ubuntu Linux 10.04 LTS ***		1,024×768 @ 60 Hz	psycho	1.56	0.41	Yes
			opengl	1.45	0.51	Yes
			legacy	3.59	4.30	No
		1,280×1,024 @ 85 Hz	psycho	1.55	0.44	Yes
			opengl	1.87	0.55	Yes
			legacy	3.33	2.48	No

* Computer: Intel Core 2 DUO E8400, 3Ghz, 2Gb; graphics adapter: ATI Radeon EA H4350 (discrete); monitor: Samsung 2233RZ, TFT, 22 in

** Computer: Intel Core 2 DUO E8400, 3Ghz, 2Gb; graphics adapter: Intel GMA 4500, Intel Q45/Q43 Express chipset (integrated); monitor: Llyama vision master pro 454, CRT, 17 in

*** Running Gnome 2.30 with the Metacity window manager. The compositing layer (i.e., "Compiz" or "Desktop Effects") was disabled

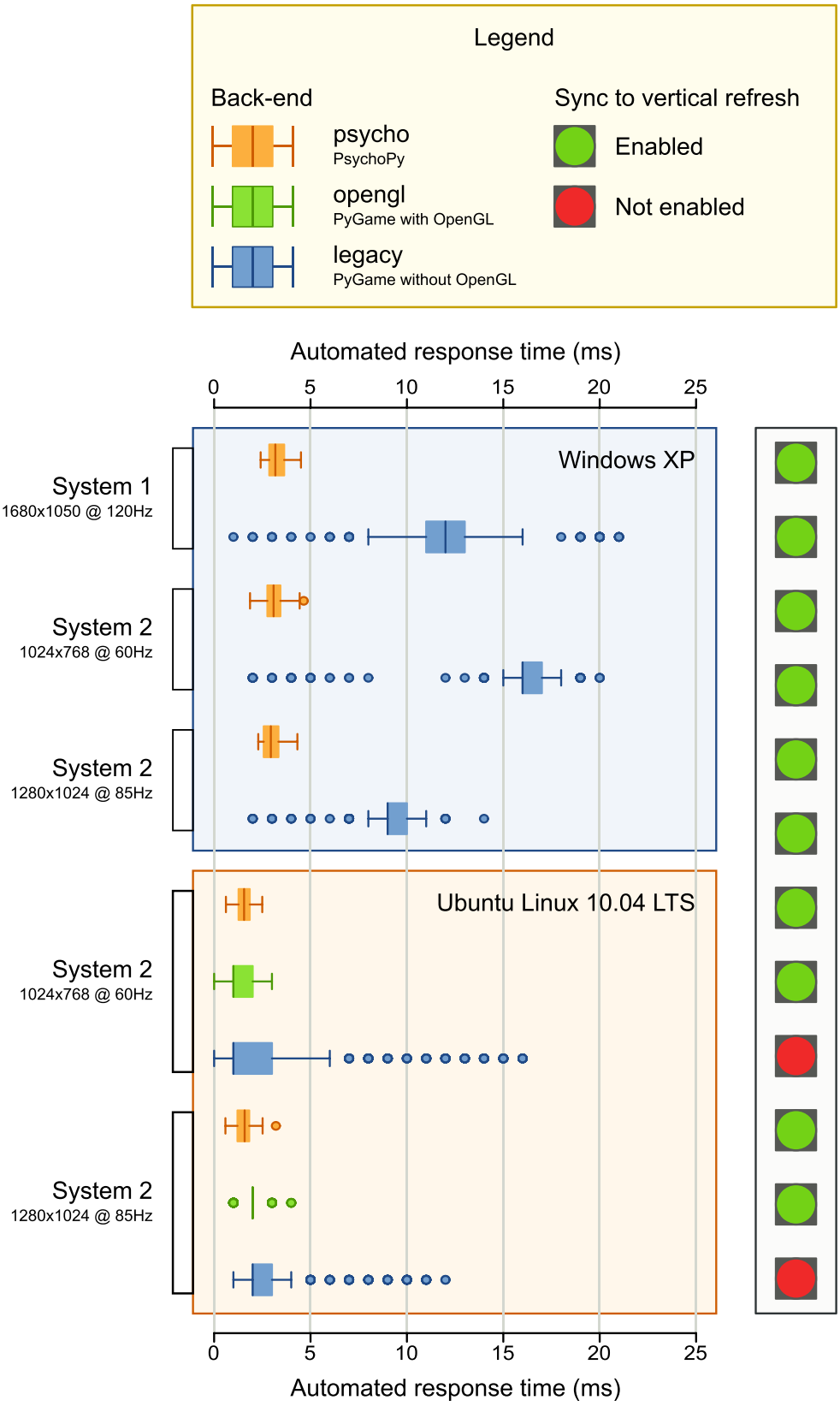


Fig. 4 Results of the benchmark experiment. The automated response times are shown as a Tukey box plot conform Robbins (2004). The central line reflects the median value. The rectangle shows the interquartile range (the 25th percentile to the 75th percentile). The whiskers reflect the minimum and maximum values that fall within 1.5 times the interquartile range. Dots correspond to individual observations that fall outside of 1.5 times the interquartile range

available back-ends were tested: *legacy*, *opengl* (Linux only), and *psycho* (see the Usage and functionality: The back-end layer section). For each test, 1,000 responses were collected. During the test, we checked visually for "tearing," which indicates that the synchronization to the vertical refresh is not enabled.

Results and discussion

The results of the experiment are shown in Fig. 4 and, in more detail, in Table 4. No further statistics were performed, since the data are essentially descriptive. Synchronization to the vertical refresh was enabled in all cases except for the *legacy* back-end on Ubuntu Linux 10.04. The results clearly show that for time-critical experiments, it is advisable to use one of the hardware accelerated back-ends, either *psycho* or *opengl*. With these, the response times are consistently below 4 ms on Windows XP and 2 ms on Ubuntu Linux 10.04. This negligible delay should be acceptable in even the most time-critical experiments.

As an aside, the results do not show any evidence that display mode has an impact on performance (preparing a large canvas obviously takes more time, but this cost is hidden in the prepare phase, as is discussed in the Timing: The prepare–run strategy section). This means that there is no reason to choose a low resolution, color depth, or refresh rate, unless many of the drawing operations are performed "on the fly," rather than prepared beforehand.

Discussion

In the present article, we have introduced OpenSesame, a graphical, open-source experiment builder for the social sciences. OpenSesame complements existing experiment-building software in a number of ways.

First, OpenSesame offers the kind of fully graphical environment that, until now, was offered only by proprietary, nonfree software.

Second, OpenSesame is extensible through plug-ins (see the Usage and functionality: Plug-ins section). Among other things, this means that support for external devices can be added and, once a plug-in has been created, this novel functionality will integrate seamlessly with the user interface. Currently available plug-ins offer support for the Eyelink series of eyetrackers (SR Research, Mississauga, ON, Canada), the Mantra object tracker (Mathot & Theeuwes, 2011), the Serial Response Box (Psychology Software Tools, Sharpsburg, PA), input devices connected to the parallel port, and video playback.

Third, OpenSesame supports Python scripting (see the Usage and functionality: Python inline coding section). Even though the aim of OpenSesame is to have a flexible user interface that allows you to create complex experimental paradigms in a graphical way, there will sometimes be occasions where there is a need for a full programming language. Python (Van Rossum & Drake, 2011) is a widely used language and has excellent support for scientific applications in general (Jones et al., 2001) and for the design of psychological experiments in particular (Geller et al., 2007; Peirce, 2007; Straw, 2008).

Fourth, OpenSesame aims for interoperability with existing software. Specifically, this means that OpenSesame can use different back-ends to handle display and input operations (see the Usage and functionality: The back-end layer section). If, for example, you want to use the PsychoPy (Peirce, 2007) routines for creating visual stimuli, you can select the psycho back-end. When this back-end is selected, all display and input operations will be handled by PsychoPy, and you will be able to use the PsychoPy routines in your experiment. Other back-ends can be created and added in much the same way as plug-ins.

Fifth, OpenSesame is cross-platform (see the Usage and functionality: System requirements section). This is particularly useful in environments where different operating systems are being used. For example, our own lab contains a mixture of Windows XP, Mac OS, and Linux computers. With OpenSesame, experiments are fully portable between operating systems.

In summary, OpenSesame is a new graphical experiment builder for the social sciences. OpenSesame is unique in that it makes creating psychological experiments easy and accessible for everyone. And perhaps even fun.

Author Note We would like to thank Per Sederberg for his code contributions (opengl back-end and advanced_delay plug-in), Jarik den Hartog and Cor Stoof for their technical support, the NeuroDebian team (Michael Hanke and Yaroslav Halchenko) for their expertise in Debian packaging, and Lotje van der Linden for her ideas and help with the tutorial. This research was funded by Netherlands Organization for Scientific Research Grant 463-06-014 to Jan Theeuwes.

References

Brainard, D. H. (1997). The psychophysics toolbox. *Spatial Vision, 10,* 433–436.

Cohen, J. D., MacWhinney, B., Flatt, M., & Provost, J. (1993). PsyScope: An interactive graphic system for designing and controlling experiments in the psychology laboratory using Macintosh computers. *Behavior Research Methods, Instruments, & Computers, 25,* 257–271.

Damian, M. F. (2010). Does variability in human performance outweigh imprecision in response devices such as computer keyboards? *Behavior Research Methods, 42,* 205–2011.

Eaton, J. W. (2002). *GNU Octave manual*. Bristol, U.K.: Network Theory Ltd.

Forster, K. I., & Forster, J. C. (2003). DMDX: A windows display program with millisecond accuracy. *Behavior Research Methods, Instruments, & Computers, 35,* 116–124.

Geller, A. S., Schleifer, I. K., Sederberg, P. B., Jacobs, J., & Kahana, M. J. (2007). PyEPL: A cross-platform experiment-programming library. *Behavior Research Methods, 39,* 950–958.

Jones, E., Oliphant, T., & Peterson, P. (2001). *Scipy: Open source scientific tools for Python*. Retrieved from http://www.scipy.org/

Mathôt, S., & Theeuwes, J. (2011). Mantra: An open method for object and movement tracking. *Behavior Research Methods*. doi:10.3758/s13428-011-0105-9

The MathWorks. (1998). *MATLAB user's guide*. Natick, MA: Author

Mueller, S. T. (2010). *The PEBL manual*. Retrieved from http://pebl.sourceforge.net/

Peirce, J. W. (2007). PsychoPy: Psychophysics software in Python. *Journal of Neuroscience Methods, 162,* 8–13.

Robbins, N. B. (2004). *Creating more effective graphs* (1st ed.). Hoboken, NJ: Wiley-Interscience.

Schneider, W. (1988). Micro experimental laboratory: An integrated system for IBM PC compatibles. *Behavior Research Methods, Instruments, & Computers, 20,* 206–217.

Stahl, C. (2006). Software for generating psychological experiments. *Experimental Psychology, 53,* 218–232.

Stevens, M., Lammertyn, J., Verbruggen, F., & Vandierendonck, A. (2006). Tscope: A C library for programming cognitive experiments on the MS Windows platform. *Behavior Research Methods, 38,* 280–286.

Stoet, G. (2010). PsyToolkit: A software package for programming psychological experiments using Linux. *Behavior Research Methods, 42,* 1096–1104.

Straw, A. D. (2008). Vision Egg: An open-source library for realtime visual stimulus generation. *Frontiers in Neuroinformatics, 2*(4), 1–10.

Tiobe.com. (2011, May). *Most popular programming languages*. Retrieved from http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Van Rossum, G., & Drake, F. L. (2011). *Python language reference manual*. Bristol, U.K.: Network Theory Ltd.