PsyBuilder: An Open-Source, Cross-Platform Graphical Experiment Builder for Psychtoolbox With Built-In Performance Optimization

Zhicheng Lin¹,², Zhe Yang³, Chengzhi Feng¹, and Yang Zhang¹
¹Department of Psychology, Soochow University, Suzhou, Jiangsu, China; ²School of Humanities and Social Science, The Chinese University of Hong Kong, Shenzhen, China; and ³School of Computer Science & Technology, Soochow University, Suzhou, Jiangsu, China

Abstract
Psychtoolbox is among the most popular open-source software packages for stimulus presentation and response collection. It provides flexibility and power in the choice of stimuli and responses, in addition to precision in control and timing. However, Psychtoolbox requires coding in MATLAB (or its equivalent, e.g., Octave). Scripting is challenging to learn and can lead to timing inaccuracies unwittingly. It can also be time-consuming and error prone even for experienced users. We have developed the first general-purpose graphical experiment builder for Psychtoolbox, called PsyBuilder, for both new and experienced users. The builder allows users to graphically implement sophisticated experimental tasks through intuitive drag and drop without the need to script. The output codes have built-in optimized timing precision and come with detailed comments to facilitate customization. Because users can see exactly how the code changes in response to modifications in the graphical interface, PsyBuilder can also bolster the understanding of programming in ways that were not previously possible. In this tutorial, we first describe its interface, then walk the reader through the graphical building process using a concrete experiment, and finally address important issues from the perspective of potential adopters.

Keywords
experiment, PsyBuilder, Psychtoolbox, open source, graphical experiment builder

Received 6/4/21; Revision accepted 11/25/21

Experiments in behavioral and neural sciences often require generating and presenting precise visual or auditory stimuli and collecting responses from the participants. Over the last three decades, several open-source software packages have been developed—prominently, Psychtoolbox (Brainard, 1997; Klein et al., 2007; Pelli, 1997), PsychoPy (Peirce, 2007, 2009; Peirce et al., 2019; Peirce & MacAskill, 2018), and OpenSesame (Mathôt et al., 2012). These packages serve as an interface between an interpretive programming environment (e.g., MATLAB, GNU Octave, and Python) and the hardware (e.g., graphics devices, sound devices, clocks, and keyboards). Psychtoolbox, for example, provides function calls in MATLAB or GNU Octave (a free and open-source alternative to MATLAB) to control hardware, with key routines written in C code. It enables users to access the display, timing, and response-collection hardware with submillisecond timing accuracy.

Despite its popularity (Fig. 1), Psychtoolbox is script based. Learning to code is challenging for many new users. Coding and debugging can also be time-consuming and error prone even for experienced users. This issue has been widely recognized as a major weakness of
Psychtoolbox (e.g., Nuutinen et al., 2018; Schwarzbach, 2011). Indeed, although high timing precision is a key strength of Psychtoolbox, when not programmed well, Psychtoolbox scripts can lead unwittingly to poor timing accuracy (Bridges et al., 2020). Great efforts therefore have been spent toward improving its usability and efficiency. These improvements include (a) making a specific task paradigm—for example, continuous flash suppression (Tsuchiya & Koch, 2005)—easier to use by having users modify task parameters in text files with minimal requirement of MATLAB knowledge (Nuutinen et al., 2018); (b) providing a package to help transform user-provided text, image, and sound files to a multimedia slideshow that allows response collections (Schwarzbach, 2011); and (c) offering a modular organizational structure to facilitate code reuse, in which an abstraction interface helps to shorten and simplify the codes needed to run experiments (e.g., the Objective Psychophysics Toolbox; Hartmann & Weisz, 2020). Nevertheless, up until now, a comprehensive solution has been lacking.

Over the past few years, we have gone much beyond prior attempts by developing the first general-purpose graphical builder for Psychtoolbox, called PsyBuilder (see Box 1). PsyBuilder is cross-platform and runs on macOS, Windows, and Linux. It is designed to greatly flatten the learning curve for new users without programming

**Box 1. PsyBuilder in a Nutshell**

PsyBuilder is a free, open-source, cross-platform experiment builder for MATLAB (or Octave) Psychtoolbox. It provides a graphical user interface (GUI) for users to build experiments visually. The GUI was developed using Python, based on PyQt5 (a Python binding for Qt v5). It allows users to create a graphical representation of an experiment through drag and drop, and then through a simple click, convert it to a MATLAB (or Octave) script that can be edited and adapted. By design, the converted script has optimized timing precision (Table 1). It is well commented to facilitate code learning, editing, and adaptation. As a regular MATLAB (or Octave) script, it can be run to execute the experiment independently of the PsyBuilder graphical interface.
experience and to drastically cut the time needed to program an experiment, with built-in performance optimization for the output scripts. In particular, best coding practice to achieve optimal timing precision (Table 1) is automatically implemented in the output scripts of PsyBuilder.

This new tool provides the much needed builder environment for Psychtoolbox, with at least three important applications. First, the graphical user interface (GUI) of PsyBuilder is easy to use and potentially can replace proprietary software (e.g., E-Prime) in student training and research. Second, the drag-and-drop functionality coupled with optimized timing and detailed comments in the output codes can improve data-collection efficiency and accuracy in research. Third, as noted by an anonymous reviewer, this GUI could also lead to exponentially faster learning of programming skills. Specifically, users can make modifications in the GUI and see exactly how the code changes (e.g., by using document comparison tools, such as: https://prettydiff.com), which helps to bolster their understanding of the code. Thus, the ability to interact and build an experiment in a visual timeline and directly see the output in code form will help new learners understand programming in MATLAB Psychtoolbox in ways that were not previously possible.

To facilitate the dissemination of the experiment builder, in this tutorial, we first describe its interface, then walk the reader through the graphical building process using a Stroop task, and end with critical consideration of its key advantages and limitations for potential adopters.

### Table 1. Best Practice to Optimize Timing Precision as Automatically Implemented in the Output Scripts of PsyBuilder

| Best coding practice to optimize timing | Reason |
|----------------------------------------|--------|
| 1. Clear prior information (e.g., variables) from computer memory at the beginning of the script. | 1. It maximizes the memory available for executing the current script. |
| 2. Set the priority level of Psychtoolbox to the highest at the beginning (and revert to the default at the end). | 2. It allows Psychtoolbox to run with higher processing priority than other processes. |
| 3. Use frame intervals (rather than milliseconds) as units for timing. | 3. Visual timing is discrete, as determined by the monitor refresh rate. |
| 4. Preallocate vectors and arrays. | 4. It speeds up operations by reducing the time cost of memory allocation to only once, with contiguous storage locations in RAM. |
| 5. Load MEX and M files and run initializations, calculations, and texture loadings (e.g., pictures) before time-critical operations (usually before the trial starts). | 5. It minimizes the processing in between stimulus presentations. |
| 6. Mark the end of drawing commands. | 6. It allows Psychtoolbox to further optimize drawing. |
| 7. Clear textures from video card memory (video RAM) when not needed. | 7. It releases the video card memory. |
| 8. Set frame flip time at half a frame interval earlier than the desired time. | 8. It accounts for potential rounding errors and delays in flip time. |

Note: RAM = random access memory.

### Disclosures

PsyBuilder is freely available for download at https://www.psybuilder.com. Furthermore, we will make the source codes open soon at https://github.com/yangzhangpsy/PsyBuilder.

All supplementary materials used in this tutorial are available at https://osf.io/xcbav/, including (a) a summary document regarding the files and the address for downloading the software package (“README.pdf”); (b) the raw data for Figure 1 (“Figure 1 data.xlsx”); (c) a supplemental tutorial with detailed explanations of the menu bar and with a chapter on how to create stimuli in PsyBuilder (“PsyBuilder 0.1 A Brief Step by Step Tutorial.pdf”); (d) screenshots elaborating on the comparisons between PsyBuilder (Version 0.1), PsychoPy Builder (Version 1.90.2), and OpenSesame (Version 3.2.8) in key GUI features (“Screenshots comparing GUIs.pdf”); and (e) final MATLAB scripts for the tutorial Stroop task (“exampleOutputTutorial.zip”).

### Tutorial

**PsyBuilder interface**

We begin the tutorial with an overview of the interface of PsyBuilder. PsyBuilder represents an experiment graphically using a horizontal timeline on the central panel (Timeline, as highlighted by ⊗ in Fig. 2), along which you can drag, drop, and arrange various experimental events. Many event types are supported, including text instruction, stimuli presentation and response collection,
and looped presentations of events. In the example in Figure 2, the main timeline has three major events: “introduction,” “blocksLoop,” and “goodbye.” Each event is simultaneously visualized in the vertical nested structure of the experiment on the upper left corner of Figure 2 (Structure, as highlighted by ②). To define, view, or edit an event in the experiment (e.g., “introduction,” “blocksLoop,” and “goodbye”), double-click on it either within the upper left Structure panel (②) or within the central Timeline panel (③). This will open the tab for that event within the central panel. Events embedded within a loop, however, are not visible on the main Timeline panel but can be accessed only through the upper left Structure panel (②). Double-click on it (e.g., “blockTime-line” within the Structure panel in Fig. 2), and it will open the associated tab for that event on the central panel.

The event tab on the central panel (③ in Fig. 2) allows you to access the detailed properties of that event. Click on the setting icon on the upper left, and it will open up a secondary, pop-out window (see Fig. 3). Depending on the event type, various parameters can be specified within this secondary window. You can input the values to the relevant variables. Note that these values do not have to be static, but can be dynamic, by using the variables available within the Variables panel (③ in Fig. 2). You can do so through drag and drop or by typing it in. The spatial layout of the event, where applicable, also appears on the tab for the event within the main window—in the current example, the text reads, “When you are ready, press any key to start!” The various variables associated with the event and their specifications are shown as a two-column list on the Properties panel (④ in Fig. 2). To generate the corresponding MATLAB script of the experiment, click on Compile option under the Building menu. The path of the script file—and error messages if any—will appear on the Output panel (⑤ in Fig. 2).

Building a Stroop-task experiment from scratch

To illustrate the experiment-building process in PsyBuilder, we now turn to the implementation of a classic experimental task, the color-word interference task, commonly referred to as the Stroop task (Stroop, 1935). Demos of additional tasks—including a visual cuing task and a rapid series visual presentation (RSVP) task—can be found under the Help menu in PsyBuilder.
stimuli creation are provided in Chapter 4 of the Supplemental tutorial (https://osf.io/xcbav/).

1. **Preparation**

Before proceeding, we need to download and install PsyBuilder on the computer first, from https://www.psybuilder.com. All major operating systems are supported, including Windows, Mac OS, and Linux. For Mac OS, we need to grant permission to the installed PsyBuilder before it can be used, by executing the following command in the Terminal window: `sudo xattr -d com.apple.quarantine /Applications/PsyBuilder.app`.

Relevant screenshots of the process are also provided in Chapter 1 of the supplemental tutorial (https://osf.io/xcbav).

Although not necessary for building the experiment using PsyBuilder, to run the eventual scripts provided by PsyBuilder, you need MATLAB (or Octave) and Psychtoolbox. For installation instructions, please follow the following link: http://psychtoolbox.org/download.

2. **Design of the task**

To implement the Stroop task—any task for that matter—we first need to think through the experimental design and procedure. In the Stroop task, a different colored word is presented each trial. The task is to respond quickly and accurately to its ink color while ignoring the semantic meaning.

Our task has one experimental manipulation, concerning the congruency between the ink color of the word and its semantic meaning. There are three different conditions: congruent, incongruent, and neutral. In the congruent condition, the ink color and the semantic color are the same (e.g., Red); in the incongruent condition, the ink color and the semantic color are of different colors (e.g., Red); in the neutral condition, the word itself is
unrelated to colors (e.g., Cat). Let us have 12 trials for each of the three conditions (congruent, incongruent, and neutral), as distributed over two blocks. In total, the task contains two blocks, each with 18 trials (six for each condition).

With this design and procedure in mind, we can proceed to draw a conceptual representation of the overall flow of the experiment (Fig. 4a). The experiment starts with a greeting display (“introduction”) that explains the task and responses—to
make a judgment regarding the ink color (which is randomly chosen from red, green, and blue) by pressing one of three keys on the main keyboard ("1," "2," or "3"). It proceeds to the two blocks ("blocksLoop"), each containing 16 trials ("trialsLoop"). In each trial, a fixation cross ("+"") is displayed at the center of the screen for 1,000 ms, and then a colored word appears at the same position until response but up to 1,500 ms. After the two blocks, the experiment ends with a goodbye display ("goodbye").

3. Implementation of the task

Let us first take a look at the overall implementation. This experimental procedure can be implemented in PsyBuilder through a graphical representation of a timeline (Fig. 4b). The events in the PsyBuilder timeline (named "timeline_0"; see Box 2 for naming conventions for variables) provides a natural correspondence with the events in the flow design described above (in Fig. 4a): The greeting and goodbye displays are implemented as text events ("introduction" and "goodbye"), and the experimental blocks are implemented as a loop event ("blocksLoop"). The loop event has two blocks ("blockTimeline"), each with its own loop of 16 trials ("trialsLoop"). Each trial is implemented as a timeline ("trialTimeline") with two events: a fixation ("fixation") and a colored word target ("stroopWord"). The properties of the trials—such as the ink color ("targetColor") and the exact word ("targetString")—are defined within the loop for the trials ("trialsLoop"), as illustrated by the table inserted on the left (Fig. 4b). Behind the scene, these graphical events are automatically converted to corresponding MATLAB/Octave codes, as illustrated by the examples at the bottom of Figure 4b.

The building process can be broken down into seven steps: (1) Set up a project, preferences, and input/output devices; (2) set up greeting display; (3) set up block loops in the main timeline; (4) set up trial loops in the block timeline; (5) set up trial events in the trial timeline; (6) set up goodbye display; and (7) save the file and

---

**Box 2. Naming Conventions for Variables**

PsyBuilder is a graphical user interface (GUI) that makes experiment building possible without coding. At the same time, by connecting a visual timeline with the output code, the GUI can also help users to learn to program. Whether using PsyBuilder just as a GUI or as a program learning tool, one routinely encounters variables (i.e., values that can change) during the process. The name of a variable consists of a string of letters, digits, and underscores. How is it chosen? There is only one rule for MATLAB variable names:

**Rule:** Names of variables must begin with a letter and can contain only letters, digits, and underscores (_).

Naming conventions—concerning style instead of rules—there are several. Understanding them helps us to better understand programming and to write codes with clarity and fewer errors. A few common naming conventions—four dos and two don’ts—are listed below.

**Convention 1: Use meaningful and unique names.** So, use “stroopColor” instead of “strpCol,” and by all means, make sure it is not already used either as a variable name elsewhere or as a function name (a function is a set of codes that accomplishes a specific task).

**Convention 2: Start with a lower case letter and use mixed case to separate different parts.** So, use “stroopColor” instead of “stroopcolor” and “dataUsaMale” instead of “dataUSAMale.”

**Convention 3: Use the prefix “n” to represent the number of objects and “i” (or “j,” “k,” etc.) to indicate iterators.** So, use “nBlocks” to represent the total number of blocks and “iBlock” to represent a block iterator (an iterator is an object that traverses a container, in this case “nBlocks”).

**Convention 4: Use the suffix “No” to represent a single entity number.** So, use “blockNo” to represent the specific entity number of the block (e.g., the third block).

**Convention 5: Avoid MATLAB reserved keywords and built-in special values.** Thus, avoid special values, such as “ans,” “pi,” and “eps,” and the names listed by the command iskeyword, such as “break,” “case,” and “catch.”

**Convention 6: Avoid negated Boolean names.** Thus, use “isCorrect” instead of “isNotCorrect.”
convert it to an M file script. To make it easy to follow, we have included screenshots right next to the tutorial instruction below (Figs. 5a–5o).

**Step 1: Set Up a Project, Preferences, and Input/Output Devices**

In Step 1, we will give the project a name and define its directory—which is where the final MATLAB/Octave scripts will be saved to and where any external files (images, audio, etc.) needed for the experiment should be included.

![Fig. 5a. Actions 1 to 3 in Step 1.](image)

1. Click “Create New File” on the greeting screen to create a new file for the project.
2. Give the project a name: “Stroop” for our tutorial. You can change the folder location if you like.
3. Then save the file.

**(continued)**

4–5. To set up system preferences, on the top Menu bar (beneath the title bar), under Preferences, click Font to set default text font, including Font Family to Times New Roman and Font Size to 22. Note: Not all fonts are supported by Psychtoolbox; please confirm whether your selection is supported by Psychtoolbox.

6. Under Preferences, click Psychtoolbox to set default warning and debug settings; use the default settings or modify to your preferences.

7–9. To set up input/output devices, under Devices, click Input to define the response-collection device and click Output there to define the stimulus-presentation device. Here, because our tutorial task requires only keyboard responses (Keyboard) and needs only one stimuli-presentation device (Screen) and because these are selected by default, no additional operations are needed. (For a detailed explanation of the top menu bar, please refer to Chapter 2 of the supplemental tutorial, https://osf.io/xchav/.)

![Fig. 5b. Actions 4 to 9 in Step 1.](image)

**(continued)**
Step 2: Set Up Greeting Display

In Step 2, we will define what the participant sees when he or she begins the study. It is generally good practice to provide the participant with some instructions about what to do during the study. In our example—the Stroop task—we will instruct the participant on which keys to press throughout the study and let them know how they can begin the task.

1. In the Timeline window, select and drag the Text icon from the Events tab onto the main timeline space below (named “Timeline_0” as shown in the Structure window). Double-click the name “Text_0” to change it to “introduction” (alternatively, select the name and press “F2”). For naming conventions, see Box 2.

2. Double-click the Text icon in the timeline to open it for editing. Enter the instruction for the experiment in the opened interface (“Welcome to . . . ”).

3. Click the Setting icon on the upper left to open the Text property box.
Step 3: Set Up Block Loops in the Main Timeline

In Step 3, we will define the main structure of the experiment—the blocks—using a key feature called *loops*. Unlike a *timeline*, which plays a fixed sequence of events, a loop plays timelines repetitively, which is suitable for defining blocks and trials. Typically, an experiment has multiple blocks. Here, we have two blocks, and we will define them in Step 3.

1. To define blocks, navigate back to the “Timeline_0” event tab. In the *Events* tab, select and drag the *Loop* icon to the timeline and place it behind the “introduction” event we just created. Double-click the name to change it to “blocksLoop.”

(continued)
**Step 4: Set Up Trial Loops in the Block Timeline**

In Step 4, we will define the main unit of the experiment—trials—using loops. Typically, each block has many trials. Here, we have 18 trials for each block, and we will define them in Step 4, including the specific word, ink color, and the correct response for each trial.

1. Similar to defining blocks, we define trials as a loop within the block timeline in which the trial loop plays the trial timeline in a repetitive way. In the Structure window, double-click “blockTimeline” to open it.

2. In the Events tab under Timeline, drag the Loop icon to the “blockTimeline” below.

3. Double-click the name of the loop and change it to “trialsLoop.”

4. In the Structure window, double-click “trialsLoop” to open it and define the looping for the trials.

(continued)
The figure shows a graphical representation of actions 13 to 17 in Step 4.

5. In the Timeline window, click the Add Rows icon.
6. In the pop-up box Add Rows, fill in 11 and then click OK to add 11 rows for a total of 12 rows.
7. In the first row, change the value for the Repetitions column from 1 to 2; do the same for rows 2 to 6 (either type in separately or drag down the value 2 from the first row to the end of the sixth row while holding down the “Alt”/“Option” key).
8. In the first row, fill in the value for the Timeline column as “trialTimeline” and press the “Enter” key to create a new timeline for the trial. Notice that in the Structure window (left), an icon named “trialTimeline” also appears there.
9. Do the last step for the remaining trials (either type in separately or use the drag-down method mentioned above).
10. Click the Setting button (top left) to open the Properties box for loops as follows.
11. In the Properties box, in the drop-down menu for Order, select “Random without Replacement” to randomize the presentation of the trials within “trialsLoop.”
12. Click OK to confirm the choice.
13. Although the timeline is the same for all the trials, they may differ in specific values in relevant variables. Therefore, we need to specify these values. In the Timeline menu bar, click the Add Variables icon to define these variables first.
14. Click the Add button in the pop-up box Add Variables to add a new variable; in total, click four times to add four variables.
15. In Variable Name, for each of the four variables, type in “targetString,” “targetColor,” “correctResponse,” and “congruency,” which are used to specify the text, color, correct response key, and congruency condition for the trials, respectively.
16. Click OK to confirm the operation. Each variable now creates a column in the “trialsLoop” structure, four in total.
17. Fill in the details for the four new variables: Fill in the text under “targetString,” fill in the corresponding ink color under “targetColor,” fill in the correct button under “correctResponse,” and finally, fill in the congruency condition in “congruency” (“congruent,” “neutral,” and “incongruent”), as detailed in the left picture. In general, typical variable types other than structure—such as string, array, cell, numeric, logical—are supported. The formats for string, array, and cell variables are shown in the above example. For numeric, any specific number can be used, and for logical, 0 and 1 can be used.
**Step 5: Set Up Trial Events in the Trial Timeline**

In Step 5, we will define the detail of each trial using a timeline. Typically, each trial starts with a fixation, followed by some stimuli and response. These events, timing, and response collection are defined here using the variables (and their values) created in Step 4.

1. To define the trial timeline, in the *Structure* window, double-click the “trialTimeline” icon to open it in the *Timeline* window.
2. First, we define the fixation. In the *Events* tab, select and drag the *Text* icon to the timeline space; double-click the name “Text_1” to change it to “fixation.”

(continued)
12. Click the “X” icon in the “fixation” tab to close it.
13. In the Structure window, double-click the “trialTimeline” icon to switch back to the trial timeline interface (alternatively, click the “trialTimeline” tab in the Timeline window).
14. After defining the fixation, we proceed to the target. In the Events tab, select and drag the Text icon to the timeline below.
15. Double-click the name “Text_2” to change it to “stroopWord.”
16. Double-click the “stroopWord” icon in the timeline to open it for editing, as follows.

(continued)

17. Recall that trials differ in the specific values in the relevant variables defined in the trial loop previously. These variables determine the properties of the target. To use these variables to define the target, enter “[“ in the input interface (in PsyBuilder, variables are enclosed within a pair of “[ ]” symbols). Then select “trialLoop.var.targetString” from the dropdown variables using the “Up” and “Down”
(continued)

arrow keys and press “Enter” to confirm (or
drag the “trialsLoop.var.targetString” variable
from the Variables window on the right to the
input interface). This variable determines the
text of the word.

18. Click the Setting button (top left) to define
text properties, as follows.

19. To define the ink color of the word, drag and
drop the “trialsLoop.var.targetColor” variable
from the Variables window (right) to the
Foreground Color box (alternatively, type in
the “[]” symbol and then select this variable
from the drop-down menu or type in the
variable directly).

20. In Font Family, choose the default text font.
21. In Font Size, choose the default font size.
22. Click Apply to confirm.
23. Click the Duration tab to set timing
information, as follows.
24. In Duration (ms), set the presentation
duration to “1500” (ms).
25. In the Input Devices box, click Add to set the
response-collection device.
26. Select “keyboard_0.”
27. Click OK to confirm.
28. To define response parameters, first, for the
selected “keyboard_0” input device, fill in the
box Allowable with “[1]!{2}@{3}#,” which
specifies the keys to be recognized (that is,
“1,” “2,” and “3” on the keyboard).
29. To define the correct response key, from the
Variables window, drag the variable “trialsLoop.
var.correctResponse” to the box Correct.
30. In the box RT Window, select “(Same as
duration),” which means that only the
response during the current event will be
recognized.
31. In the box Response Action, select
“Terminate,” which ends the presentation of
the current event when a key press is
recognized.
32. Press OK to confirm.

**Step 6: Set Up Goodbye Display**

In Step 6, we will define the goodbye display as the end
of the experiment. The purpose is to indicate to the
participant that the study has finished and to provide
instructions if needed.

---

1. The setup is similar to that for the
“introduction” event. In the Structure window
(left), double-click the “Timeline_0” icon to
open it in the Timeline window for editing.
2. In the Events tab, drag the Text icon to
“Timeline_0” and place it behind the
“blocksLoop” event.

(continued)
Fig. 5n. Actions 9 to 10 in Step 6.

(continued)

3. Rename it to “Goodbye” and double-click the icon on the timeline to edit its contents, as follows.
4. Click the Setting button (top left).
5. In the General tab, modify the content in the Text box to: “Thank you very much for your participation!”
6. Use the default for Font Family and Font Size.
7. Click Apply to confirm the operation.
8. Click the Duration tab to set the timing properties, as follows.
9. In the Duration (ms) box, set the presentation duration to “2000” (in ms).
10. Click OK to confirm the operation.

Step 7: Save the File and Convert It to M File Scripts

Now, we are ready to compile the script. These scripts are saved to the same folder as the PsyBuilder project set up at the beginning.

Choosing PsyBuilder Versus Other Software Packages

So far, we have described the interface of PsyBuilder and walked the reader through the graphical building process using a Stroop task. Here, we consider key advantages of PsyBuilder and its limitations from the perspective of a potential adopter.

Major advantages of PsyBuilder

A key advantage of PsyBuilder is that by providing an easy-to-use GUI, it resolves the long-standing usability limitation of Psychtoolbox without compromising its power or flexibility. This makes PsyBuilder suitable for new users and can potentially replace proprietary software
(e.g., E-Prime) in teaching and training. Moreover, because users can see exactly how the code changes in response to modifications in the GUI, this functionality can help bolster their understanding of programming in ways that were not previously possible. On the other hand, for intermediate and advanced users, the drag-and-drop functionality coupled with optimized timing (see Box 3) and detailed comments in the output codes can improve data-collection efficiency and accuracy in research. This is important because even experienced users are not immune to the proneness to unwitting mistakes in manual coding.

Other major features include the following. First, most common stimuli are readily available in PsyBuilder, including images, text, sounds, videos, shapes, gratings, random-dot kinematograms, and any of their combinations (see Chapter 4 of the supplemental tutorial). These stimuli can be presented with any user-defined level of visual transparency as enabled by alpha blending. Second, various inputs are supported, including eye tracker (e.g., from EyeLink), gamepad, keyboard, mouse, and response box. Third, several modes of communications with external devices are available, including screen (with support for multiple displays), sound, network port, parallel port (e.g., for sending electroencephalography triggers to mark the time stamp for events), and serial port. Fourth, common complex procedures, such as Bayesian adaptive staircases (e.g., QUEST; Watson & Pelli, 1983), are supported. Fifth,

Box 3. Timing Accuracy: What It Is, Why It Is Important, and How to Achieve It

In this tutorial, we have mentioned timing accuracy (precision) several times. What is timing accuracy, after all? Timing accuracy is evaluated based on two dimensions: the average deviation from the true value (bias) and trial-to-trial variability (variance). Both bias and variance can be critically important depending on the question. For example, when the question of interest concerns absolute time—questions such as how long it takes to detect the appearance of an object, recognize a face, and categorize a scene—then the magnitude of bias needs to be measured and corrected accordingly. When the question of interest concerns the difference in time between conditions or groups, then timing bias can be canceled out by their subtraction, but only if the bias is stable, which should be measured and verified, particularly when the comparison involves a systematic difference in testing time or across different hardware setups.

But in both cases, timing variance introduces noise to measurement, which reduces measurement reliability and statistical power and hence should be minimized. For example, in electroencephalography and event-related-potential studies, even small timing variance in the triggers that mark the time stamp for critical events (e.g., a stimulus) can strongly affect the validity of the results and conclusions. In typical behavioral tasks, trial-to-trial response time variability is usually large enough to overshadow small timing variance, but in cases in which the number of trials or participants is small, timing variance can still be critical because there are not enough trials or participants to average out the noise.

Timing accuracy critically depends on hardware, software, and software-to-hardware communications. For example, Psychtoolbox uses OpenGL to draw visual stimuli, PortAudio for auditory stimuli, and java for response collection. It serves as an interface between MATLAB (or GNU Octave) and low-level libraries to enable submillisecond timing accuracy. PsyBuilder automatically implements best coding practices to optimize timing precision (Table 1). It also provides further optimization, including (a) allowing users to choose to load image textures before the event, the trial, or the experiment; (b) making only one texture for one image even when it is used for different events; and (c) clearing image textures from video card memory only when the images are not needed subsequently. PsyBuilder also automatically saves event onset times for later verification. To verify timing accuracy, one can use a photodiode to measure the actual presentation time and the intended presentation time.

Timing accuracy also depends on best practices that must be implemented by the user. The central principle is to maximize the processing resources for executing the scripts (particularly time-sensitive codes). Therefore, it is recommended to (a) use sufficient RAM (random access memory) and dedicated hardware with the latest drivers (e.g., dedicated audio cards) and (b) disconnect from the Internet and not run applications and software that are not strictly necessary for the experiment (e.g., software updates or antivirus software). For licensed software (e.g., MATLAB), it ideally should be run using a local license instead of a network license to prevent periodic checks for network licenses.
experimental and event parameters can vary dynamically on every loop. Finally, the support of loops that use different timelines (e.g., for different trials), conditional statements (if and switch), and variables afford users the ability to create a wide range of procedures flexibly. In short, PsyBuilder can help new and experienced users to create experiments both simple and complex with fewer errors and higher efficiency compared with coding in Psychtoolbox.

**Potential limitations of PsyBuilder**

Potential limitations of PsyBuilder have to do with it being a GUI interface. That is, GUI interfaces are inherently limited in their functionality compared with full-fledged scripting. Although many sophisticated features are supported out of the box in PsyBuilder, it obviously cannot implement every possible stimulus and procedure. Fortunately, this limitation is mitigated by the fact that PsyBuilder outputs executable scripts that are well commented and optimized—users can readily modify them to suit the particular needs not fulfilled by PsyBuilder. Moreover, advanced features currently unavailable in PsyBuilder are being actively developed, including (a) the implementation of complex procedures such as multiple object tracking and stimulation to record steady-state visual-evoked potential, (b) support for more devices (e.g., currently only response boxes from Cedrus and eye trackers from EyeLink are supported), (c) improved support of texts, and (d) direct support for questionnaires and online (over the Internet) data collection.

**Maintenance and development of PsyBuilder**

Currently, PsyBuilder is being developed and maintained by the authors. With the large user base of MATLAB Psychtoolbox, we are excited for the community to contribute to the development of PsyBuilder. Indeed, one of the Demos provided in the Help menu—an RSVP task—was contributed by Wenfeng Chen from Renmin University of China during the testing phase of this software. We are committed to the success of PsyBuilder, and we have set up a forum to facilitate bug reporting, community connections, and collaborations. Users are welcome to ask questions and report potential bugs on this public forum for PsyBuilder: https://www.psybuilder.com.

**MATLAB or Python?**

Whereas Psychtoolbox and PsyBuilder use MATLAB (Octave), PsychoPy and OpenSesame, both with builder environments, rely on Python. MATLAB/Octave and Python are both popular programming languages, and Psychtoolbox and PsychoPy also enjoy a large user base. Ultimately, the choice of software packages is a personal one, depending on the purpose, circumstances, and personal preference.

Therefore, for the purpose of this tutorial, it might be useful to focus the comparison on the builder environment. Table 2 summarizes the comparisons of PsyBuilder, PsychoPy Builder (Peirce et al., 2019), and OpenSesame (Mathot et al., 2012) in a number of major features. Screenshots that elaborate on these comparisons are provided at https://osf.io/xcbav/. Therefore, for example, if you intend to use online data collection, then this feature is currently not supported in PsyBuilder (Psychtoolbox) but is available in PsychoPy and OpenSesame, although we do plan to work on this feature in future releases. In terms of timing (see also Box 3), in a recent evaluation of a range of popular free and commercial software packages—Psychtoolbox, PsychoPy, OpenSesame, E-Prime, Presentation—the lowest timing variability was obtained with Psychtoolbox in Ubuntu, with a mean intertrial standard deviation of 0.18 ms in various lab-based tests, including reaction times, visual durations, visual onset, audio onset, and audiovisual synchronization (Bridges et al., 2020). But generally, for most purposes, timing accuracy is adequate for modern software packages, including PsychoPy.

**Summary**

We have developed the first general-purpose graphical experiment builder for Psychtoolbox, PsyBuilder. PsyBuilder can be used as a tool to build experiments without the need to code. It can also be used to facilitate exponentially faster learning of programming. The output scripts are automatically optimized in timing precision and with detailed comments to facilitate customization. In this tutorial, we introduced its interface and walked the reader through a Stroop task as an example. We also considered its major advantages and limitations for potential adopters. By flattening the learning curve for new users, cutting the time needed for programming an experiment, and optimizing the performance of the output codes in terms of timing precision and detailed comments, we find that PsyBuilder represents a major new development for Psychtoolbox. It is poised to contribute to the open-science movement by facilitating the wide adoption of Psychtoolbox in student training and research in lieu of proprietary software (e.g., E-Prime). It can also help accelerate scientific progress by improving data-collection efficiency and accuracy in behavioral and neural sciences.
Table 2. Comparisons of PsyBuilder (Version 0.1), PsychoPy Builder (Version 1.90.2), and OpenSesame (Version 3.2.8) in Key GUI Features

| Feature                                                                 | Software                                      |
|------------------------------------------------------------------------|-----------------------------------------------|
| Timing precision (intertrial standard deviation, in ms)                | 0.18                                          |
| Device communication                                                   | Parallel, serial, and network port            |
| Drag and drop                                                          | Parallel                                      |
| WYSIWYG for multiple items in the GUI                                  | Parallel (and only via a third-party plugin)  |
| Simultaneous presentation of video with other stimuli (e.g., a fixation)| Some (e.g., not for variables and stimuli drawing) |
| GUI for setting parameters                                             | Yes                                           |
| Different timelines (sequences, routines) under one loop (cycle), such as different trial timelines in one block | No                                            |
| Autocomplete for variables when typing                                  | Yes                                           |
| Conditional statements (if/switch) in the GUI                          | No                                            |
| A list of variables that can be used for the current event             | Yes                                           |
| Multiple QUESTs                                                        | Yes                                           |
| Eye-movement-based response                                             | Yes                                           |
| Eye-tracker support                                                    | Yes                                           |
| Monitor gamma correction                                               | Yes                                           |
| Questionnaires                                                         | No                                            |
| Run online                                                             | Yes (via pavlovia.org)                        |
|                                                                        | Yes (via OSWeb extension)                     |

Note: The data for intertrial standard deviations are based on Bridges et al. (2020). See also supplementary screenshots at https://osf.io/xcbav/, titled “Screenshots comparing GUIs.pdf.” GUI = graphical user interface; WYSIWYG = what you see is what you get.

Transparency

Action Editor: Julia Strand
Editor: Daniel J. Simons
Author Contributions
Z. Lin and Z. Yang are joint first authors with equal contributions. Y. Zhang conceived and supervised the project, designed and implemented the compiler. Y. Zhang, Z. Yang, and Z. Lin designed the graphical user interface. Y. Zhang and Z. Yang implemented the graphical user interface and maintained the support website. Z. Lin and Y. Zhang wrote the tutorial and made the figures. Y. Zhang, Z. Yang, and Z. Feng provided managerial and financial support.

Declaration of Conflicting Interests
The author(s) declared that there were no conflicts of interest with respect to the authorship or the publication of this article.

Funding
The study was supported by National Natural Science Foundation of China (32171049, 32071045), Natural Science Foundation of Jiangsu Province (BK20201411), MOE (Ministry of Education in China) Project of Humanities and Social Sciences (19YJC190030), Guangdong Basic and Applied Basic Research Foundation (2019A1515110574), and Shenzhen Fundamental Research Program (JCYJ20210324134603010).
Open Practices

Open Data: not applicable
Open Materials: not applicable
Preregistration: not applicable

ORCID iDs

Zhicheng Lin https://orcid.org/0000-0002-6864-6559
Yang Zhang https://orcid.org/0000-0002-1827-9383

References

Brainard, D. H. (1997). The psychophysics toolbox. *Spatial Vision*, 10(4), 433–436. https://doi.org/10.1163/156856897X00357

Bridges, D., Pittot, A., MacAskill, M. R., & Peirce, J. W. (2020). The timing mega-study: Comparing a range of experiment generators, both lab-based and online. *PeerJ*, 8, Article e9414. https://doi.org/10.7717/peerj.9414

Hartmann, T., & Weisz, N. (2020). An Introduction to the objective psychophysics toolbox. *Frontiers in Psychology*, 11, Article 585437. https://doi.org/10.3389/fpsyg.2020.585437

Klein, M., Brainard, D. H., & Pelli, D. G. (2007). What’s new in Psychtoolbox-3? *Perception*, 36(14), 1–16.

Mathot, S., Schreij, D., & Theeuwes, J. (2012). OpenSesame: An open-source, graphical experiment builder for the social sciences. *Behavior Research Methods*, 44(2), 314–324. https://doi.org/10.3758/s13428-011-0168-7

Nuutinen, M., Mustonen, T., & Hakkinen, J. (2018). CFS MATLAB toolbox: An experiment builder for continuous flash suppression (CFS) task. *Behavior Research Methods*, 50(5), 1933–1942. https://doi.org/10.3758/s13428-017-0961-z

Peirce, J. W. (2007). PsychoPy—Psychophysics software in Python. *Journal of Neuroscience Methods*, 162(1–2), 8–13. https://doi.org/10.1016/j.jneumeth.2006.11.017

Peirce, J. W. (2009). Generating stimuli for neuroscience using PsychoPy. *Frontiers in Neuroinformatics*, 2, Article 10. https://doi.org/10.3389/neuro.11.010.2008

Peirce, J. W., Gray, J. R., Simpson, S., MacAskill, M., Hochenberger, R., Sogo, H., Kastman, E., & Lindelov, J. K. (2019). PsychoPy2: Experiments in behavior made easy. *Behavior Research Methods*, 51(1), 195–203. https://doi.org/10.3758/s13428-018-01193-y

Peirce, J. W., & MacAskill, M. (2018). *Building experiments in PsychoPy*. SAGE.

Pelli, D. G. (1997). The VideoToolbox software for visual psychophysics: Transforming numbers into movies. *Spatial Vision*, 10(4), 437–442. https://doi.org/10.1163/156856897X00366

Schwarzbach, J. (2011). A simple framework (ASF) for behavioral and neuroimaging experiments based on the psychophysics toolbox for MATLAB. *Behavior Research Methods*, 43(4), 1194–1201. https://doi.org/10.3758/s13428-011-0106-8

Stroop, J. (1935). Studies of interference in serial verbal reactions. *Journal of Experimental Psychology*, 18, 643–662.

Tsuchiya, N., & Koch, C. (2005). Continuous flash suppression reduces negative afterimages. *Nature Neuroscience*, 8(8), 1096–1101. https://doi.org/10.1038/nn1500

Watson, A. B., & Pelli, D. G. (1983). QUEST: A Bayesian adaptive psychometric method. *Perception & Psychophysics*, 33(2), 113–120. https://doi.org/10.3758/bf03202828