Scripting syntax

main website (http://www.psytoolkit.org) | documentation 2.0.9 (http://www.psytoolkit.org/doc2.0.9)

- sounds
- fonts

tasks

- how tasks are structured
- the concept of **variables**
- the concept of **timestamps**
- The set function
- commonly used instructions
- less often used instructions
- very advanced instructions
- instructions for special equipment

tables

blocks

- showing bitmaps, playing sounds, and waiting for keys
- setting variables
- tasklist
- feedback

blockorder

include (advanced use only)

part (for advanced user)

Links to all instructions

This syntax is merely meant to look up instructions of the PsyToolkit scripts. If you are new to PsyToolkit, it is easier to start looking at some of the examples here or more detailed examples in the experiment library (http://www.psytoolkit.org/experiment-library).

Quick link to all instructions

The complete version of this documentation is expected between October and December 2014. Some instructions are not yet completely described. Until then, click here for more details

(http://psytoolkit.gla.ac.uk/software/psyscript.html) and here click here for how PsyToolkit scripts work (http://psytoolkit.gla.ac.uk/software/scriptessence.html)

Other relevant documents:

- General information about PsyToolkit
- General information about scripting
- Detailed help about scripting and the offline version (http://psytoolkit.gla.ac.uk/software/)

How to use this manual

This manual lists all elements of a PsyToolkit script. It will also give a short syntax description. A syntax is a description of how instructions can be used. In the syntax description, the compulsory arguments of an instruction are in square brackets []. The optional arguments in normal brackets (). A | sign indicates different valid values for an argument.

For example, take the example for the instruction "show rectangle". The example below shows that there are 4 compulsory arguments: X,Y,W,H. This is the x-coordinate, the y-coordinate, the width and height. The arguments Red, Green, and Blue are optional (by default, a rectangle is white, but you can give it different colors)

show rectangle [X,Y,W,H](Red,Green,Blue)

The brackets in the syntax description are, of course, not typed in the real script; it is just a way of describing a syntax. Look carefully at the examples given for each instruction to see on how to use instructions.

structure of scripts

There are different sections in scripts, and each section is separated from others by at least one empty line.

Everything following a hash mark is a comment only for human's eyes,

the computer will ignore it.

The main sections which contain multiple lines each are:

- options describes non-default PsyToolkit settings such as screen size
- bitmaps/sounds/fonts describes which stimuli to use
- task can be used multiple times to describe paradigm trials
- table can be used multiple times to describe experimental conditions
- block can be used multiple times to run blocks of trials
- blockorder order of blocks can optionally be set in a different way (advanced users only)

options

Whenever you program an experiment, you may want to deviate from the default values. For example, the default screen size of an experiment is 800 by 600 pixels, but you can change this using the option "resolution". The available options are listed below. At the bottom of this section is a complete example.

There are many options, but they are mostly for advanced users. If you are new to PsyToolkit, you might want to ignore them first.

origin

Visual stimuli are presented on the screen, and you need to give an X and a Y coordinate to place them. The origin option tells the computer where point X=0,Y=0 (or 0,0) is.

By default, the coordinate point 0,0 is at the center of the screen. Alternatively, you can set the top-left corner of the screen as the origin (which was the default in older versions of PsyToolkit).

Example of how to use origin

```
options
  origin topleft
```

centerzero

If set, this assumes that 0,0 refers to the center of the screen, and not to the top-left point of the screen. This is now the default of PsyToolkit (since version before 1.5.0).

Example of how to use centerzero (it is very easy)

```
options
  centerzero
```

This function is no longer necessary, because it is the default.

fontdir,datadir,bitmapdir,sounddir,videodir

The options **fontdir,datadir,bitmapdir,sounddir,videodir** let you change the folders where the computer expects the stimuli to be saved. You do not need to specify this if the stimuli files are in the same directory as your scripting code.

This option makes most sense if you work in C or have large experiments.

fullscreen

Runs experiment in fullscreen mode. In Linux, this is the default, whereas in the browser it is not.

In the online version (Javascript), the view might depend on the browser. It works definitely in both Firefox and Chrome, but it might not work in other browsers. You need to test it before using it.

Example of fullscreen (with randomly jumping rectangle)

```
options
  fullscreen

task MyTask
  set $x random -300 300 # random x position
  set $y random -300 300 # random y position
  set $mysize random 10 100
  show rectangle $x $y $mysize $mysize   255 255 0 # show yellow rectangle
  delay 100
  clear -1

block MyBlock
  tasklist
    MyTask 200
  end
```

resolution

This sets the resolution of the experiment. This default resolution is 800 by 600.

When specifying the resolution, it is width by height, separated only by a space. See example below.

Example of fullscreen (with randomly jumping rectangle)

```
options
  fullscreen
  resolution 1000 800

task MyTask
  set $x random -300 300 # random x position
  set $y random -300 300 # random y position
  set $mysize random 10 100
  show rectangle $x $y $mysize $mysize   255 0 0 # show red rectangle
  delay 100
  clear -1

block MyBlock
  tasklist
    MyTask 200
  end
```

transparency

transparency on enables the use of transparency in bitmaps. By default, this is off. You want this for overlapping see through bitmaps. This is not yet fully tested in Javascript.

version

This is only for advanced users. This will only allow to compile a script with the corresponding psycc version. This is practical if you have multiple versions of psycc installed, and when you just want to make sure that the code runs. This command has no effect in the online version.

mouse

By default, the mouse cursor is not shown, because from a psychological point of view it can be a distracting stimulus. But for some experiments, the mouse is needed as a pointing device. This option makes sure the mouse cursor is visible. The only useful value is "on".

set

This lets you set a global variable. This can be handy if you want to test an experimental parameter (e.g., intertrial interval) to a specific value throughout the experiment. If you do not use the variable anywhere, this set value will just be ignored.

Example of "set" in options

```
options
  fullscreen
  set &my_intertrialinterval 100

task MyTask
  set $x random -300 300 # random x position
  set $y random -300 300 # random y position
  set $mysize random 10 100
  show rectangle $x $y $mysize $mysize   255 0 0 # show red rectangle
  delay 100
  clear -1
  delay &my_intertrialinterval

block MyBlock
  tasklist
    MyTask 200
  end
```

Sometimes it is handy to have set some basic parameters in the options as global variables. That way, you can easily find them back. Good examples are intertrial intervals, maximum allowed response times, stimulus size, etc.

window (C only)

This lets you run the experiment in a window instead of in full screen. This is good for testing. Note that the full screen mode is the default mode in Linux, unless you run in test mode.

Because PsyToolkit can run in two different programming environments, some functions do not work in all environments. **C only** functions only run on the Linux version which is based on the C programming language. If you use the online version, you can fully ignore these.

render (C only)

This is quite technical and for specialists only.

In C on Linux, this option lets you set the way the graphics library SDL draws the screen. This is an advanced topic that you do not need to worry about unless you really want to. In short, the OS graphics system has different ways of drawing the screen, and PsyToolkit enables one to use those different ways, rather than deciding for you how the screen is drawn. In practice there are only minor differences. There are currently two ways to render, the default way (which is the same as in versions before 1.2.0) and using double buffering. Using "render doublebuffer" is recommended for working with moving stimuli (in Javascript, this is the default anyway). In the future, "render opengl" will allow you to use the OpenGL capacities of your computer if available (this is planned).

screensize and screendistance (C only)

This lets you set the dimensions of the screen and the eye-screen distance. Units are in millimeters. These numbers are necessary if you want to use the option **coordinates polar** or if you want to use the report stimulus size command line option. If you use a

polar coordinate system (which is rare), all xy coordinates should be specified in 100ths of degrees (thus 100 = 1 degree, 200 = 2 degrees of visual angle). This is not well implemented yet (let me know if you need this).

vsync on | off

Not relevant for the Web-based version.

Advanced users only. In Linux mode, this is "on" by default, but can be switched off, because some graphics cards do not allow to use the vsync. PsyToolkit will actually complain if it is not available.

If set to "off, the computer does not wait for synchronization of the vsync (this is the default in Javascript). This is handy for testing, since the experiment does not need the root permission during compilation. In C, if you use the **-t** option, this is the default.

Example of vsync off

```
option
  vsync off
```

egi (C only)

With the host ipname or ipaddress and the (optional) ip port sets up a connection to the Electrical Geodesics, Inc System. This has not been tested well, and is based on old code.

escape (C only)

In the Linux Desktop version, this makes it possible to end the program to by pressing the escape key. Please notice that the computer only tests whether the escape button has been pressed at the end of each trial. Hence, if you use this option, and if you want to break out your experiment, you have to hold the escape button (or whatever button you use for escaping) for the duration of at least one trial.

In the browser function, you can always escape out of your experiment, and there this option is not necessary.

parallelport (C only)

This is for advanced coding of the parallel port.

pcidio24 (C only)

This is for advanced coding of the pcidio24 port

cedrus (C only)

This indicates that you have attached a Cedrus keyboard. Optionally, you can specify a model. If you specify a model, the script will only run with that model. That strictness is only necessary in case you want to force people to use a specific Cedrus keypad.

iolab (C only)

This indicates that you have attached the IoLab device. Optionally, you can specify the voicekey parameters (see psycc -s)

executable

The filename of the executable program. Per default, this is "experiment", but it can be set with the command line option -o and with this option.

The -o option will override this option. In the online version, this option will be ignored.

Example of all options

```
options
  fontdir     /usr/lib/fonts
  datadir     /home/user/mydatadir
  bitmapdir   /usr/local/bitmaps
  sounddir    /usr/local/snd
  videodir    /usr/local/videos
  origin      topleft
  window
  resolution 1024 800
  screensize 1000 600
  screendistance 500
  coordinates polar
  vsync off
  escape
  egi 10.0.0.42
  parallelport in data out 1 5
  pcidio24 in a b out c_low c_high
  cedrus
  iolab
  executable myexperiment
  set &myinterval 100
```

You do not need to use any options if you do not want to.

loading stimuli

The computer needs to know which stimuli you use. Stimuli are typically images or sounds, and these need to be loaded. You can also load fonts and videos, although videos are not well supported and currently only work in the C mode. Below is explained on how to load bitmaps, sounds, and fonts.

bitmaps

The bitmaps command lets you define a number of bitmaps you want to use in your experiment. Bitmaps can be of common formats, png, jpg, bmp and some more.

The bitmaps line has no parameters, and is followed by lines describing the bitmap, each bitmap having a name and a file description.

Do not put anything is quotes, and don’t use spaces in the descriptions or in the filenames.

If you don't add a filename, it assume that a filename exists with the extension .png. In the example below, PsyToolkit assumes a file house.png to exist because no further bitmap info is given. This is actually the recommended way of working (it is definitely easiest).

*Example of **bitmaps** and how to refer to them in a task*

```
bitmaps
  house
  funnyface /home/user/funny.bmp
  cookie    /usr/local/cookie.jpeg
  smiley    smiley

task MyTask
  show bitmap funnyface
  show bitmap cookie 100 0
```

Note that for Javascripted experiments in a browser, there is a special feature, that lets you load bitmaps from an external web page using the http:// prefix (thus, instead of a full filename, give the URL). This makes sense under very special circumstances, and is not generally recommended, unless you really know what you are doing. The reason for this is that this can speed up download times of online experiments considerably. This because per default, all stimuli are embedded in the HTML file via data:uri, which can lead to large HTML files.

For more information on how data URI works see data URI on Wikipedia (http://en.wikipedia.org/wiki/Data_URI_scheme)

Here is how you do it in PsyToolkit

*Example of loading **bitmaps** from external websites*

```
bitmaps
  house http://www.my-website.com/house.png

task MyTask
  show bitmap house
```

sounds

This works like the "bitmaps" commands, just to tell the computer which sounds are being loaded. You can use any sound file format, although the **wav** file format is the one

that is most widely used and recommended for online studies.

*Example of **sounds** and how to refer to them in a task*

```
sounds
  barkingdog bark.wav
  phone      ringingphone.mp3

task MyTask
  sound phone
  delay 300
```

As for bitmaps, you can also load a sound from a web server (using http://). The advantage is that this leads to quicker loading times.

You can also use sounds in the block, for example to play some music when the participant is doing the task. This can be used to test the effect of different types of sounds, music, background noise on task performance. See sound in a block.

fonts

Here you can describe which fonts to use. In the Javascript (online) mode, only the fonts the browser has will be used, and Arial is the default. In C, you can load a font file.

Note that the same fonts go by different names. If you are not familiar with the most common types, check WikiPedia for information about Arial (https://en.wikipedia.org/wiki/Arial), information about Times New Roman (https://en.wikipedia.org/wiki/Times_New_Roman), and information about Courier (https://en.wikipedia.org/wiki/Courier_%28typeface%29).

In both C and Javascript, you can use one of the three default font types: arial, times, or courier. If you use one of these three fonts, you do not need to upload or include a font file, because these fonts are already available on computers. On Linux, Arial=FreeSans.ttf, Times=FreeSerif.ttf, and Courier=FreeMono.ttf.

*Example of **sounds** and how to refer to them in a task*

```
fonts
  arial 18
  myfont times 20
  mysmallfont arial.ttf 20
  mybigfont   arial.ttf 40

task MyTask
  font mysmallfont
  show text "hello"
  delay 1000
  font mybigfont
  show text "world"
  delay 1000
  font arial
  show text "Some text in standard arial"
  font myfont
  show text "Some text in Times New Roman"
  delay 1000
```

See also show text.

tasks

In PsyToolkit, a task (typically) describes the sequence of exactly one trial of an experiment.

Tasks are the most difficult part of the scripting language, because they involve writing real sequential computer instructions to show stimuli, wait for responses, etc. Lots of things are possible, and there are many different instruction types. Each instruction type is described below.

Before describing the instructions, a number of fundamental elements of the language are described, namely the structure of tasks, variables, and timestamps.

how tasks are structured

Tasks are really used to program what happens in one trial of a experiment paradigm. A task is just a sequence of instructions, ended with an empty line. The instructions contain the following major parts of information:

- Where a description of the different experimental conditions is. This can be done with the table statement
- How exactly the participant should respond (e.g., which keyboard keys, or with the mouse). This can be done, for example, with keys and readkey.
- The stimuli being used, and when and where they should be presented. This can be done with show and sound.
- Which data should be saved to a file for later data analysis. This is done with the instruction save

the concept of variables

Like any computer language, you can use variables. There are **global** and **local** variables. You can in principle always use global variables. Local variables are only used within one task, whereas global variables can be used anywhere in the task and block code (see also blocks). You can set initial values of global variables in the **options**.

You can set a variable with the "set" instruction.

Example of setting and using variables

```
task MyTask
  set $x 100
  set &y 10
  show bitmap FunnyFace $x &y
```

Variables can only have integer values. An integer is a whole number. For example, $x is 100, it cannot be 100.342, because the latter is not a whole number.

The equal sign is not being used to assign values to variables! That is, no "=" symbol like you would do in most programming languages.

Variables are not allowed to start with a number, are not allowed to contain spaces, minus signs, dots, or special characters except the underscore.

Good examples.

- $MyVariable1
- $my_variable_1
- $xyz

The following are **NOT** allowed!!!!!!!

- $My Variable 1 (reason: no spaces allowed)
- $My.variable.1 (reason: no dots allowed)
- $My-variable-1 (reason: no minus signs allowed)
- $My*variable (reason: no special characters such as "*" allowed)
- $123my_variable (reason: not allowed to start with a number)

There are also special types of variables written in capitals, such as RT, STATUS, TABLEROW, BLOCKNAME, and BLOCKNUMBER. They just give you access to important information about a response or the task condition.

the concept of timestamps

Timestamps are a special type of variable, but they are only used in very advanced scripts and are normally not necessary. Whenever you set a timestamp, the current time is being saved.

The set function

set [$&][variablename] [new value, variable, or expression]

As explained about in "variables", the set function can set the value of variables.

A variable is a basic element in any programming language. A variable is simply a symbol which holds a value. For example, x = 10. PsyToolkit can only work with whole numbers as values.

set is a relatively advanced function, and when learning PsyToolkit, you do not immedeately need it, and you can write scripts without **set**.

set a local variable to a value (the $ indicates that it is local)

```
set $x 10
```

set a global variable to a value (the & indicates that it is local)

```
set &x 10
```

In the rest, the local or global nature of the variable is irrelevant for explaining set, and we just work with **$x**

take the value of other variable

```
set $x &y
```

increase the value of a variable

```
set $x 10
set $x increase 2  # after this, $x has value 11
```

decrease the value of a variable (default is 1)

```
set $x 10
set $x decrease # after this, $x has value 9
```

use a mathematics expression

```
set $x expression $x / 2 + $a + 1
```

set to random value

```
set $x random 1 10
```

set to random value in steps of 2

```
set $x random 2 100 2
```

set to random value from specific set of values

```
set $x random from 1 2 3 5 10
```

set to value from the pcidio24 IO card (if you have one!) (for advanced users)

```
set $x pcidio24 a # set to value of register A
```

set to the milliseconds that have been passed since start of experiment (for advanced users)

```
set $x time-since-start
```

set to the millisecond difference of two timestamps (for advanced user)

```
timestamp MyTime1
dely 500
timestamp MyTime2
set $x timestamp-diff MyTime1 MyTime2 # now x should be 500
```

get the timestamp in seconds and milliseconds (for advanced users)

```
timestamp MyTime1
dely 2500
timestamp MyTime2
set $x timestamp-seconds MyTime2 # value should be 2
set $x timestamp-milliseconds MyTime2 # value should be 2500
```

get bitmap number of the bitmap currently under the mouse (for advanced users)

```
bitmaps
  myface
  yourface
  smileyface

task test
  show bitmap yourface -100 0    ## first bitmap
  show bitmap smileyface 200 200 ## second bitmap
  readmouse l 1 1000
  set $myMouseX MOUSE_X # current mouse x-coordinate
  set $myMouseY MOUSE_Y # current mouse x-coordinate
  set $x bitmap-under-mouse $myMouseX $myMouseY ## $x should be 1
```

commonly used instructions

There are many different instructions for showing stimuli, recording and saving responses. Below they are all listed with a short example of their use. This section describes the ones that are most often used, and that you really need to understand.

Before we start, here is an alphabetical list of the most commonly used instructions:

- presenting stimuli
  - show shows a stimulus on the screen
  - clear removes a stimulus from screen
  - sound plays a sound
- measuring responses and response times
  - keys tells the computer which keys are being used in a task
  - readkey waits for a keyboard press
  - readmouse waits for a mouse click or mouse movement
  - choose fancy multiple stimuli selection with mouse
- waiting for some time
  - delay wait period in milliseconds
- saving data
  - save saves data to file
- telling the computer about experimental conditions
  - table specifies where the experiment conditions are
- conditionals
  - if do something depending on variable value(s)
  - while do something depending on variable value(s)
  - set set a variable

Below, each of these commands is given in more detail.

keys

keys [list of keycodes]

At the beginning of a task description, you should tell the computer which keys from the keyboard participants might have to press. This line should be one the first lines in the task description.

Keys given will be associated a keyboard numbers, starting with one. You can use this when saving data. The variable KEY will contain this value when a key has been pressed.

*Example **keys** using only two keys*

```
keys a z
```

List of all the keys available on all platforms:

- letts: a to z
- numbers: 0 to 9
- special keys: enter capslock tab space end home insert
- special keys: escape slash backslash quote comma period
- arrow keys: up down right left
- numerical keyboard (keypad): kp0 kp1 kp2 kp3 kp4 kp5 kp6 kp7 kp8 kp9
- number keyboard: kp_period kp_slash kp_star kp_minus kp_plus kp_enter

The following keys are not availble in the Javascript mode

- shift keys: lshift rshift
- control keys: lcontrol rcontrol
- alt keys: lalt ralt
- logo keys: lsuper rsuper

table

table [name of table]

The table simply refers to a table instruction (which is separately defined from the task, and it will contain information about the different experimental instructions). Note that you must define a table elsewhere in your script, which is explained elsewhere in this document. Each line in a table will be considered the description of a different experimental condition; therefore, tables allow you to have one task description for different conditions. This way you can vary the color, position, or size of stimuli. It is a key element in PsyToolkit scripts, although you can write scripts without tables as well!

The table row chosen on a given trial is in the variable **TABLEROW**.

*Example **table***

```
table MyTable
```

show

show [bitmap|text|rectangle](X,Y)(...)

This is a key instruction, which can show a rectangle, a bitmap, or a text anywhere on the screen. It can also change the background color (show background). The number of parameters given to a show instruction can vary. The show command for bitmap, rectangle, and text, and text, will now be given separately.

Read more here for an indepth explanation of how you present your stimuli on screen where you want them to be.

show bitmap

show bitmap [bitmap](X,Y)

Imagine you have loaded a bitmap in the bitmaps section of your task description. Now you can show it. Just specify the bitmap, and the bitmap will be shown at screen center. You can also specify the X and Y coordinates (optional).

*Example of **show bitmap***

```
show bitmap MyBitmap
show bitmap MyBitmap 200 10
```

show rectangle

show rectangle [bitmap](X,Y,Width,Height)(Red,Green,Blue)

You can display colored rectangles, specifying the X and Y position as well as the Width and Height. You also need to specify the Red, Green, and Blue values. That means you will have 7 more values!

In the example below, a rectangle is show at position 0,0 and it has both a width and a height of 100 pixels. The red/green/blue values are set to 255/0/0. The maximum value in each of these three color channels is 255. Given that only red is specified, the rectangle will be red.

PsyToolkit uses the Red-Green-Blue (RGB) color model. Each color can be specified as a combination of red, green, and blue. For each of these three basic colors, you need to give a value between zero and 255. You can create any color, see RGB model on wikipedia (http://en.wikipedia.org/wiki/Rgb). There are websites (http://www.rapidtables.com/web/color/RGB_Color.htm) where you can create your own colors:

Table 1. Examples of colors in the RGB model:

Color	Red	Green	Blue
White	255	255	255
Pure red	255	0	0
Pure blue	0	0	255
Pure green	0	255	0
Yellow	255	255	0

Grey	128	128	128
Orange	255	128	0
Pink	255	100	180

*Example of **show rectangle***

```
show rectangle 0 0 100 100 255 0 0
```

show background

show background (Red,Green,Blue)

Sometimes you want the whole screen to have a specific color. In principle, you could just draw a rectangle the size of the screen. The "show background" function does exactly that. You just give the three color parameters and the whole screen will be filled with that color. Otherwise, it is treated like a normal screen stimulus and it has a number and it can be cleared.

*Example of **show rectangle***

```
show background 100 100 100 ## creates a grey background
delay 500
show rectangle 100 200 200 0 0 ## draw red rectangle
delay 1000
clear -1 ## erase the the rectangle
```

You do not need to clear the background and the background will stay on until next trial (which is most likely what people want)

show text

The show text instruction shows text. You need to specify a font and font size as well in the fonts **section**

In the desktop version, you can load True Type Files (TTF). In the web-based version, you can only use three standard types of fonts, namely *arial*, *times*, and *courier*.

show text [text](X,Y)(Red,Green,Blue)

*Example of **show text** for Linux*

```
fonts
  MyArial arial.ttf 20

table MyFirstTable
  10 "some text"
  15 "some other text"

task MyTask
  font MyArial
  table MyFirstTable
  show text "Some green text"  0 0   0 255 0
  show text @2  0 0   0 255 0
```

The default alignment is "center". That means that if you ask something to presented at, say, position 10,10, that is where the center of the word is. See the example below on how to set the alignment. Also, note that you can set the text color for all subsequent "show text" commands, and here you can use a color word.

While "show text ..." is for actually displaying the text, you can set the alignment of color separately. That is easy if you for example want to present multiple texts in the same color.

*Example of **text** and **show text** with alignment and color*

```
fonts
  arial 20

task MyTask
  text color yellow
  text align center
  show text "Some text with alignment: center"  0 -100
  text align left
  show text "Some text with alignment: left" 0 0
  text align right
  show text "Some text with alignment: right"  0 100
  delay 10000

block test
  tasklist
    MyTask 1
  end
```

The following color words are currently implemented: white, yellow, red, green, blue, pink, purple, black, grey, orange, pink.

For "text color", you can also specify the color in three decimal values or as 6 digit hex code. See example below for the same statement of specifying yellow in three different ways. This example makes no sense in a real experiment (because why would you specify the same color 3x in a row?), but just demonstrates how to use the "text color" statement.

*Example of **text color***

```
  # specify color as word:
  text color yellow
  # specify color as Red Green Blue code:
  text color 255 255 0
  # specify color as Hexadecimal value:
  text color FFFF00
```

clear

clear [bitmap occurrence](list of bitmap occurrences)

To erase a stimulus you put on the screen with "show", you can erase with clear. The call "clear" takes at least one parameter. The number corresponds to the count of show calls. For example, if you want to erase a particular bitmap which was shown as the second show event in your task, use "clear 2". Alternatively, you can use negative referral numbers to refer to preceding stimuli: -1 refers to the last presented bitmap! Using the negative referrals is actually much easier to use, because that way you do not need to count bitmaps.

The bitmap counter is reset every trial!

Example using clear

```
show bitmap redcircle
show bitmap greencircle
delay 500
clear 2 # clears the second presented bitmap
# or
show bitmap redcircle
show bitmap greencircle
delay 500
clear -1 # clears last presented bitmap, that is "green circle"
# or
show bitmap redcircle
show bitmap greencircle
delay 500
clear 1 2 # clears both bitmaps
```

Rarely, you want to clear the whole screen. This is not recommended for time critical things, because it can be slower that changing small parts of the screen. Sometimes you want to clear the whole screen at the beginning or end of trial. You can use **clear screen** to do so.

delay

The delay instruction pauses the program for the specified number of milliseconds. You will need this function often, for example, if you want to show a stimulus for a specific time interval, or for waiting between trials.

delay [milliseconds]

Examples of delay that shows how to show a stimulus for 200 ms

```
show bitmap MyBitmap
delay 200
clear -1
```

The parameter of delay (in the example 200) can be a variable or a table entry as well. Look at the examples to learn how to do that.

readkey

The readkey instruction tells the computer to wait for a keyboard response. The first argument, the correct key, corresponds to the keys in the keys instruction.

readkey [correct key number][maximum RT]

In the following example, there are two possible keys in the task (named MyTask), the a and the z key. Imagine that a stimulus requires the z button to be pressed, that is the second key (with a being the first in the line "keys"). The readkey command will now way 3 seconds for a key press. If the participants presses the z key, the STATUS will be set to CORRECT (which equals numerical value 1). If the participants presses the wrong key, it will be WRONG (numerical value 2), and if there is no response at all within 3000 ms, the STATUS code will be TIMEOUT (numerical value 3).

Example of readkey

```
task MyTask
  keys a z
  show bitmap PressTheZkey
  readkey 2 3000
```

After a call to **readkey**, the user can use the following variables:

- RT (with the response time in milliseconds, that is the time of the key down event)
- TT (the time the key has been released)
- STATUS (with values CORRECT,WRONG, or TIMEOUT, or 1,2, or 3)

readmouse

Similar to readkey, you can check if the participant clicked the mousekey, and if so, if the mouse was where it should be. You can also just wait for the mouse to be moved into a certain area of the screen. In fact, you can check if the location of a specific stimulus has been touched with the mouse (bitmap, or rectangle, or text).

Here are the different options

- Wait until the mouse is clicked **and** the mouse is in the correct location
- Wait until the participant moves the mouse to the correct location

In the following example, we show two rectangles, and we want that the participant mouse the mouse over the green one (the first bitmap), which is positioned on the left (-200). We give maximally 5 seconds, that is 5000 milliseconds.

Example of readmouse

```
task checkMouse
  show rectangle -200 0 40 40  0 255 0 # green rect, left
  show rectangle  200 0 40 40  255 0 0 # red rect, right
  readmouse 1 5000
  save STATUS RT
```

In the following example, we have exactly the same task, but we want that the mouse is being clicked as well as in the first stimulus. We add the "l" argument, with "l" standing for the left mousebutton. In Javascript, it is recommended only to use the left mousebutton, because the right mouse button might show a browser-specific menu, which is obviously not something you want.

Example of readmouse

```
task checkMouse
  show rectangle -200 0 40 40  0 255 0 # green rect, left
  show rectangle  200 0 40 40  255 0 0 # red rect, right
  readmouse l 1 5000
  save STATUS RT
```

There is one issue that arises if you have stimuli that overlap eachother. Imagine the following situation. You have one big rectangle that is just their to show a clear yellow rectangle, and on top of that is a much smaller rectangle. If you want that people click

that second smaller rectangle, you need to tell the computer that you do not care about the first one. The way to do that is to specify the range of bitmaps that you are interested in. See example below:

Example of readmouse

```
task checkMouse
  show rectangle -200 0 400 400  255 255 0 # big yellow rectangle
  show rectangle  50 0 40 40  255 0 0 # small red one
  readmouse l 2 5000 range 2 2 # wait for second one being clicked
  save STATUS RT
```

choose

This is a new function under development. Use at your own risk. Currently only implemented in Javascript.

Under certain circumstances, you want to ask the participant to click multiple stimuli on the screen. Ideally, you would want to allow the participant to select objects and deselect as well. This is all possible with the relatively complex function **choose**.

In short, **choose** will let the participant click on a range stimuli, show a symbol on top of the stimuli, and when clicked again the symbol will disappear. The functions **savechoose** can save the information collected. Here is a full example.

In the following example, there are three symbols shown on the screen that can be selected with a selector bitmap. Options for this command can be set before **choose** is called with **choose option**. The participant has 60 seconds (60,000 milliseconds).

Example of choose

```
bitmaps
  markingsymbol
  house
  ball
  car

task clickMysymbols
  show bitmap house # bitmap 1
  show bitmap ball  # bitmap 2
  show bitmap car   # bitmap 3
  choose option select markingsymbol
  choose 60000 1 3
  savechoose selected # save which symbols were clicked
  savechoose rts # save when symbol first clicked if at
```

The savechoose command might soon be replaced. Currently, it shows for each stimulus in the stimulus range (here 1 to 3) whether it was selected or not (1 or 0), and when a symbol was clicked for the first time.

save

Saves variables. Typically, a task ends with a save line.

The save instruction is of critical importance, because it makes sure that the information you need for your data analysis is being stored. By default, PsyToolkit does not keep any information (unlike some other experiment software). The user needs to tell PsyToolkit exactly which information is being stored to the data file.

Typically, you will want to save what the current condition is, the current block (if you have more than one block), and at least the response time (RT), and whether the participant responded correctly or not (STATUS). The examples on this website can help you to understand this.

The best place for the save command is at the end of your task description.

save [list of variables]

Example of save

```
save BLOCKNAME RT STATUS
```

sound

Used to play sounds. A sound just starts playing the code continues. If you want to do nothing during the sound, you need to have it followed by a delay. You can also stop the sound at any time using the **silence** instruction.

sound [sound name (as defined in 'sounds']

Example of sound

```
sound MySoundFile
delay 200
silence MySoundFile
```

if

You can use **if** to only carry out some commands. This is often needed for showing a feedback message if people make a mistake. The opposite of **if** is **fi**.

Examples of if

```
task MyTask
  show bitmap stimulus
  readkey 1 1000
  if STATUS == CORRECT
    show bitmap WellDone
    delay 1000
    clear -1
  fi
  if STATUS != CORRECT
    show bitmap Mistake
    delay 2000
    clear -1
  fi
  if $x == &y
    show bitmap SmileyFace $x $y
  fi
```

while

Similar to **if**, you can put code in a "while loop". The while loop is ended with the **while-end** statement.

Examples of while loop

```
task MyTask
  set $mycounter 0
  while $mycounter < 10
    set $x random -200 200
    set $y random -200 200
    show bitmap SmileyFace $x $y
    set $mycounter increase
  while-end
```

> If you make a mistake, the computer might be stuck in the while loop. For example, if you do not increase the value of $mycounter in the above example, the code will be stuck and keep drawing new SmileyFace symbols at random positions.

set

Set is a fairly complex function and should not be used by absolute beginners. There are the following ways to set a variable. Below are examples for setting the variable $a.

- set $a 10 (just set variable to value)
- set $a $x (give $a the value of $x)
- set $a random 1 10 (choose random number)
- set $a random 1 10 2 (choose random number in steps, like 1,3,5,7,9)
- set $a expression $x * 5 (set $a to value of $x times 5
- set $a increase (increase $a by 1)
- set $a increase 10 (increase $a by 10)
- set $a decrease
- set $a decrease 5

> timestamps and bitmap-under-mouse require good experience with

PsyToolkit

Assume you have set the time of time1 and time2 using the timestamp function, and you want to know what the difference between the following is:

- set $a time-since-start (time since start in milliseconds)
- set $a timestamp-diff time1 time2 (difference between two [[timestamp]timestamps)
- set $a timestamp seconds time1
- set $a timestamp milliseconds time1

The following three set commands set the value of variable $a to the bitmap number under the mouse. This is handy for figuring out which image a participant clicked with the mouse:

- set $a bitmap-under-mouse $x $y
- set $a bitmap-under-mouse $x $y up range 1 5
- set $a bitmap-under-mouse $x $y down range 1 5

less often used instructions

Some functions are rarely used or necessary, yet they add to the potential of PsyToolkit scripting.

while

Everything between a while and while-end statement will be carried out until the condition on the **while** line is satisfied.

Examples of while

```
while $x < 100
  increase $x
  save $x
while-end
```

font

You can set the font of the next *show text" instruction.

Examples of font

```
fonts
  MySmallFont arial.ttf 20
  MyBigFont arial.ttf 50

task Mytask
  font MySmallFont
  show text "small" 0 0
  font MyBigFont
  show text "small" 0 100
```

very advanced instructions

end

You can end the task at any point in your list of statements in a task. You can even end the current tasklist or the whole experiment. All you need to do is **end task** , **end tasklist** , or **end experiment**

This can be useful, for example, when people are training and you want to stop the block of trials when they made a mistake.

This is, however, a statement that you will rarely need.

timestamp

timestamp can be used to capture the current time. This can be handy if you are checking the time past in a while..while-end loop. It can also be handy if you are trying to debug code and check if the timing is as expected.

You can get the timestamp-diff between two timestamp variables using the set command as in the example below. Timestamps cannot be directly accessed or used other than through **set timestamp-diff**

Examples of timestamp

```
task MyTask
  timestamp MyFirstTimestamp
  delay 1000
  timestamp MySecondTimestamp
  set $x timestamp-diff MyFirstTimestamp MySecondTimestamp
  save $x
```

sprite

> The speed of moving sprites is based on various factors. Speed depends on the screen update frequency and on the rendering type (on some systems, double buffering is takes more time than default buffering, this is not due to the time it takes to draw, but due to more complex issues of how graphics cards draw screens). It just means that if you use experiments on different systems, you might have slightly different sprite moving speeds.

instructions for special equipment

Special equipment are primarily IO cards and special keyboards. I recommend Cedrus keyboards. If you like, you can build your own keyboard. This saves a lot of money. You can attach it to the parallel port (if you have one). This keyboard is called "ultra" (designed by Felix Frey, University of Leipzig), and a document how to build this is freely available (http://psytoolkit.gla.ac.uk/download/manual/ultra-keyboard-howto.1.pdf).

cedrus readkey

Wait for a specific key of your USB Cedrus keyboard. Make sure you know which key has which numerical value (you might want to experiment a bit with this, and there is an example programming that shows you the number of each key). On Linux, you can call this using the "testcedrus" command, which comes with PsyToolkit.

tables

Tables have lines and rows. Each time a table is being used in a **task**, one of its rows is chosen. Each column can be referred to using the @ sign. For example, **@2** refers to the second column on the row that is being chosen for a given task trial. In the block, the user can specify how table lines are selected. The default is randomly, but there are alternative ways (for example: fixed order, repeat on trial).

Example of a table

```
table MyTable
 10  2  bitmap1
-10  1  bitmap2
```

Example of a table with strings

```
table MyTable
 10  2  bitmap1  "condition one"
-10  1  bitmap2  "condition two"
```

In a task that uses a table (as instructed with the table instruction), each column can be referred to with the @ sign. Thus @2 refers to the second column of the table row that has been selected.

Each time that a task is being carried out, only one table-row of the associated table is being selected. By default, a row is chosen randomly. There are other ways to choose table rows (using the tasklist instruction in blocks).

In each task trial with a table, you have access not only access to the columns, but you can also get the number of the tablerow. The variable is called TABLEROW. This can be handy for saving data. If there is a lot of important data in each tablerow that you want to use in your analysis, you want to make sure which TABLEROW was being used.

blocks

In experiments, trials come in blocks. A block of trials means that the participant does, for example, 100 trials of the Stroop task. Instead, you might want to have a break in the middle, so you could create instead 2 blocks of 50 trials. Blocks call tasks, so you really must have at least one block in your PsyToolkit script. Blocks can be complex, but they can also be very easy, like this example below.

Example of simple block, calling the stroop task 100 times

```
block MyBlock
  tasklist
    strooptask 100
  end
```

showing bitmaps, playing sounds, and waiting for keys

Often, you want to show instructions to participants. You can show bitmaps, and you can show series of bitmaps participants can scroll forward and backwards through (pagers).

bitmap

The bitmap is simple. You often want to show a bitmap with an instruction and then wait for a keypress.

Example of bitmap in block

```
block Myblock
  bitmap MyInstruction
  wait_for_key
  tasklist
    strooptask 100
  end
```

> Showing a bitmap in a task and in a block is different. Here in blocks, do not put the word **show** before the bitmap. In tasks you have to. There are good reasons for this distinction (in tasks, stimuli are fast paced, whereas in blocks, they are thought of as instructions).

sound

As in the task, you can start a sound with **sound** and stop it with **silence**

Example of sound in block

```
block Myblock
  sound welcometune
  bitmap MyInstruction
  wait_for_key
  silence welcometune
  tasklist
    strooptask 100
  end
```

delay

As in tasks, you can put in a delay. Sometimes this is nice for countdowns.

Example of message in block

```
block Myblock
  bitmap number3
  delay 500
  bitmap number2
  delay 500
  bitmap number1
  delay 500
  tasklist
    mytask 100
  end
```

message

Instead of showing a bitmap and waiting for a key, you can combine this in one command, called message.

Example of message in block

```
block Myblock
  message MyInstruction
  tasklist
    strooptask 100
  end
  message ThankYouBitmap
```

pager

Sometimes you want a series of instructions and let the browser use through it. The pager command does that. Users can use the arrow keys and use the Q key to get out of it. The space bar will go to the next image. Make sure that all these bitmaps are in the bitmap section.

Make sure you tell in the instructions how users browse through it and how they get out of it.

Example of message in block

```
block Myblock
  pager MyInstruction1 MyInstruction2 MyInstruction3 MyInstruction4
  tasklist
    strooptask 100
  end
```

setting variables

Sometimes you want to set a global variable. You can do that just like you would do in tasks. For example, you might set the maximum response time to a higher value during training.

Example of message in block

```
block MyTraining
  pager MyInstruction1 MyInstruction2 MyInstruction3 MyInstruction4
  set &MaxResponseTime 2000
  tasklist
    strooptask 20
  end

block MyRealDatablock
  set &MaxResponseTime 1000
  pager MyInstruction1 MyInstruction2 MyInstruction3 MyInstruction4
  tasklist
    strooptask 100
  end
```

tasklist

Each block has a task list. This describes which tasks are being called, how many times, and in what order. You might have one or more tasks. It is rare to have multiple tasks, but in taskswitching paradigms this might be the case.

On each line of the task list, you need at the very least provide the name of the task and the number of trials.

Example of simple tasklist

```
block MyTraining
  tasklist
    strooptask 20
  end
```

You can specify that you want that whenever people make a mistake, the same trial should be repeated, using **repeat_on_error**. If you use this, you need to specify what is considered to be an error using the error statement (see example below).

Repeat_on_error only works when you use tables, random variables are not set to the same values.

Example of simple tasklist.

```
table MyTable
  "condition1" 255 0 0 1 ## respond with "r" to red rectangle
  "condition2" 0 255 0 2 ## respond with "g" to green rectangle

task Mytask
  table MyTable
  keys r g
  show rectangle 0 0 @2 @3 @4
  readkey @5 2000
  if STATUS == WRONG
    error ## this tells PsyTookit that this trial is considered an error
  fi
  save @1 STATUS RT

block MyTraining
  tasklist
    Mytask 10 repeat_on_error
  end
```

In the example above, you see a red or green rectangle, and you need to respond with the "r" or "g" key. You might wonder why you would need to explicitly specify what an error is if you use "repeat_on_error". This is a good question, because in the above example it is indeed an error if the person responds with the wrong key. But there are experiments in which you actually want that people do not respond (that is STATUS is TIMEOUT). And in some experiments, there is more than one response per trial. The "error" statement gives you full control over what an error is and what not.

In some experiments, you might want that all participants carry out the trials in exactly the same order. You can just run through each line of your task table. You use the **fixed** option for this.

Example of simple tasklist

```
block MyTraining
  tasklist
    strooptask 20 fixed
  end
```

In some experiments, you might want that participants do at least a certain number of trials correct. You use the **correct** option for this. In the example below, the participant needs to have done 10 trials correct, but after 100 trials it will stop no matter how many the participant did correct. In the **allcorrect** variant, they must be consecutively correct.

Example of simple tasklist

```
block MyTraining
  tasklist
    strooptask 10 correct 100
  end

block MyTraining
  tasklist
    strooptask 10 allcorrect 100
  end
```

Finally, you sometimes want that trials do never repeat. There are three ways to get this in the tasklist statement

- *all_before_repeat* : do all trials as in the table, randomly selected, but run them all before chosen again
- *no_repeat* : never immedeatly repeat a condition from the table (i.e., the same row of a table will not be repeated in next trial)
- *fixed* : run trials in the order as they are in the table

Note that the **fixed** option has been described elsewhere in this document.

In the following example, there are only three entries on the following list. They will not be repeated until they are all done. You can explicitly add that you never want any repeat.

Example of tasklist

```
table MyTable
  bitmapBlue
  bitmapRed
  bitmapGreen

task strooptask
  table MyTable
  keys space
  show bitmap @1
  readkey 1

block MyTraining
  tasklist
    strooptask 10 all_before_repeat no_repeat
  end
```

Make sure that you always have an **end** at the end of your tasklist

feedback

Feedback can be used to create feedback to the participant, for example about the average response speed. Creating feedback is part of the "block" structure. Feedback is described in a separate document, because it is of advanced use: how to use feedback

blockorder

Advanced use

By default, blocks are carried out in the order they are described in your code. You can alter this order by rearranging the block code, but there is an easier and quicker way: Blockorder.

Using the **blockorder** statement, you can list the blocks you want to do and the order you want them in.

The idea behind this function is just speed of coding, and ease of changing the order (for example for counterbalancing).

In the following example, imagine there is code for three blocks, called *training*, *test1*, and *test2*. The following example show how to set the order of these blocks in different ways. In future versions of PsyToolkit, this functionality will be expanded.

Example of how to use blockorder

```
blockorder
  training
  test1
  test2
```

include (advanced use only)

You can include another file into your script. This can be useful if you work with very large tables which you want to store in a separate file. Any line starting with *include* followed by a filename will use that filename at that place in the script.

Included files must be in the same directory as the main script file.

Example of how to use include

```
table My_Large_Table
  include tablefile.txt

task some_task
  table My_Large_Table
  show bitmap @1
  delay 100

block test
  tasklist
    some_task 10
  end
```

Included files (like *tablefile.txt* in the example) should not include other files using *include*.

part (for advanced user)

You can write "snippets" of a few lines and include them elsewhere in your script. This is handy if you have multiple tasks or blocks which are broadly the same except for a few lines. This way can use a part instead of re-typing the whole script.

part just replaces the text into your code. The main aim is to have shorter more efficient code without repetitions of the same stuff.

Example of how to use "part"

```
part showAnimatedSquare
  show rectangle 0 0 50 50  255 255 0
  delay 100
  show rectangle 0 0 100 100  255 255 0
  delay 100
  show rectangle 0 0 150 150  255 255 0
  delay 100

part removeSquare
  clear -1 -2 -3

task some_task
  keys space
  part showAnimatedSquare
  readkey 1 1000
  part removeSquare
  delay 100

task another_task
  keys a
  part showAnimatedSquare
  readkey 1 5000
  part removeSquare
  delay 200

block test
  tasklist
    some_task 10
  end

block test
  tasklist
    another_task 10
  end
```

Links to all instructions

(this list is not yet complete)

Options: origin| bitmapdir| sounddir| videodir| fontdir| fullscreen| resolution| transparency| version| mouse| variable| window| render| screensize| screendistance| vsync| egi|escape| parallelport| pcidio24|cedrus| iolab| executable

Task instructions: keys| show| text| clear| sound| set| keys| readkey| readmouse| choose| delay| save| table| if while

Block instructions: bitmap| sound| delay| message| pager| set| tasklist| feedback

Special instructions: include| part

Last updated 2016-08-16 10:43:03 CEST