Expyriment Documentation

Release 0.7.0

Florian Krause & Oliver Lindemann

October 24, 2015

A Python library for cognitive and neuroscientific experiments. Read more ...

CHAPTER

ONE

CONTENTS

About Expyriment

1.1 Overview

Expyriment is an open-source and platform independent light-weight Python library for designing and conducting timing-critical behavioural and neuroimaging experiments. The major goal is to provide a well-structured Python library for a script-based experiment development with a high priority on the readability of the resulting programme code. It has been tested extensively under Linux and Windows.

Expyriment is an all-in-one solution, as it handles the stimulus presentation, recording of I/O events, communication with other devices and the collection and preprocessing of data. It offers furthermore a hierarchical design structure, which allows an intuitive transition from the experimental design to a running programme. It is therefore also suited for students as well as experimental psychologists and neuroscientists with little programming experience.

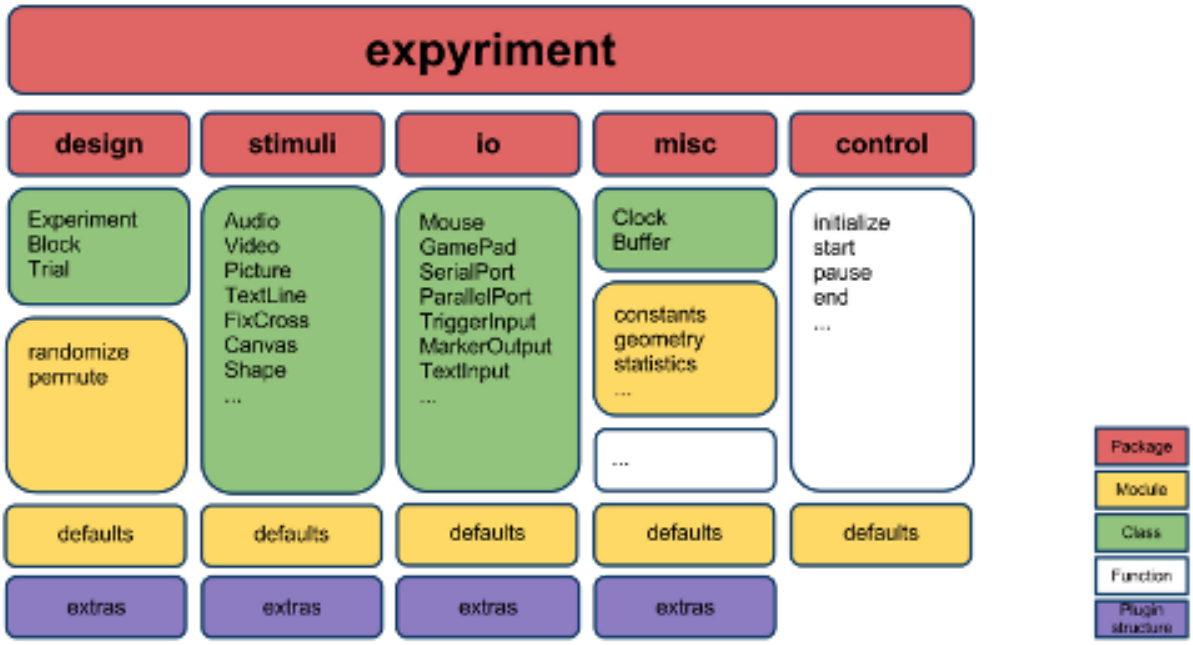
Website: http://www.expyriment.org

Authors

- Florian Krause, Radboud University Nijmegen, The Netherlands
- Oliver Lindemann, University of Potsdam, Germany

1.1.1 Main features

- Easy syntax, very readable (due to Python)
- Cross-platform (Linux, Windows, OS X)
- Allows for fast and efficient programming of experiments (do more with less code)
- A variety of standard stimuli (Text, Picture, Audio, Video etc.)
- Keyboard, Mouse, Serial Port, Parallel Port and Gamepad support
- Hierarchical structure of experimental units (Experiment, Blocks, Trials, Stimuli)



For a full documentation of all Expyriment functionality (i.e. all available modules, classes, methods, functions, constants, and attributes provided by the Expyriment Python package), please have a look at the *API reference pages*

1.1.2 Example code

Examples of full experiments can be found *here*

Creating experiments, blocks, trials and stimuli:

```
exp = expyriment.design.Experiment()
block = expyriment.design.Block()
trial = expyriment.design.Trial()
stimulus = expyriment.stimuli.TextLine(text="Hello World")
```

Building a hierarchy between various structures:

```
trial.add_stimulus(stimulus)
block.add_trial(trial)
exp.add_block(block)
```

Presenting stimuli:

```
for block in exp.blocks:
    for trial in block.trials:
        trial.stimuli[0].present()
```

Handling input devices:

```
button, rt = exp.keyboard.wait([expyriment.misc.constants.K_SPACE])
```

Logging data:

```
exp.data.add([button, rt])
```

1.1.3 Licence

Expyriment is free software and released under the Open Source GNU General Public Licence of the Free Software Foundation.

1.1.4 Publications & Citation

If you have used Expyriment in your work, please cite the follwing publication:

Krause, F. & Lindemann, O. (2013). Expyriment: A Python library for cognitive and neuroscientific experiments. *Behavior Research Methods.* doi:10.3758/s13428-013-0390-6

1.1.5 Development repository

The Expyriment development repository is currently hosted on GitHub. Get a local copy of this repository with this command:

```
git clone https://github.com/expyriment/expyriment.git
```

1.1.6 Mailing lists

Expyriment newletter: All users of Expyriment should subscribe to the Expyriment newsletter, since the project is still under development. All modifications and new versions will be announce via this mailing list. (Visit website or send an email to expyriment+subscribe@googlegroups.com).

Expyriment users mailing list: If you have questions regarding the installation, usage or documentation of Expyriment please don't hesitate to contact the Expyriment users mailing list (visit website or send an email to expyriment-users+subscribe@googlegroups.com) or contact us directly by sending an email to info@expyriment.org.

1.1.7 Suggestions and bug tracking

If you want to make suggestions to improve Expyriment or you found a bug, please post your comments to the Expyriment issues page or contact us directly by sending an email to info@expyriment.org.

1.1.8 Related Projects

If you are looking for a graphical experiment builder, we suggest OpenSesame, which uses Expyriment as the default back-end: http://www.osdoc.cogsci.nl/.

1.2 expyriment

1.2.1 Packages

expyriment.control

Modules

expyriment.control.defaults

Attributes

expyriment.control.defaults.**audiosystem_autostart**
default value: True

expyriment.control.defaults.**audiosystem_bit_depth**
default value: -16

expyriment.control.defaults.**audiosystem_buffer_size**
default value: 2048

expyriment.control.defaults.**audiosystem_channels**
default value: 2

expyriment.control.defaults.**audiosystem_sample_rate**
default value: 44100

expyriment.control.defaults.**auto_create_subject_id**
default value: False

expyriment.control.defaults.**event_logging**
default value: 1

expyriment.control.defaults.**fast_quit**
default value: False

expyriment.control.defaults.**goodbye_delay**
default value: 3000

expyriment.control.defaults.**goodbye_text**
default value: 'Ending experiment...'

expyriment.control.defaults.**initialize_delay**
default value: 10

expyriment.control.defaults.**open_gl**
default value: True

expyriment.control.defaults.**pause_key**
default value: 112

expyriment.control.defaults.**quit_key**
default value: 27

expyriment.control.defaults.**stdout_logging**
default value: True

expyriment.control.defaults.**window_mode**
default value: False

expyriment.control.defaults.**window_size**
default value: (800, 600)

Functions

expyriment.design

Modules

expyriment.design.defaults

Attributes

expyriment.design.defaults.**block_name**
default value: 'unnamed'

expyriment.design.defaults.**experiment_background_colour**
default value: (0, 0, 0)

expyriment.design.defaults.**experiment_filename_suffix**
default value: None

expyriment.design.defaults.**experiment_foreground_colour**
default value: (150, 150, 150)

expyriment.design.defaults.**experiment_name**
default value: None

expyriment.design.defaults.**experiment_text_font**
default value: 'FreeSans'

expyriment.design.defaults.**experiment_text_size**
default value: 20

expyriment.design.defaults.**max_shuffle_time**
default value: 5000

expyriment.design.defaults.**trial_list_directory**
default value: 'trials'

expyriment.design.extras

Modules

expyriment.design.extras.defaults

expyriment.design.permute

Functions

expyriment.design.randomize

Functions

Classes

expyriment.design.Block

expyriment.design.Experiment

expyriment.design.Trial

expyriment.io

Modules

expyriment.io.defaults

Attributes

expyriment.io.defaults.**datafile_delimiter**
default value: ‘,’

expyriment.io.defaults.**datafile_directory**
default value: ‘data’

expyriment.io.defaults.**eventfile_delimiter**
default value: ‘,’

expyriment.io.defaults.**eventfile_directory**
default value: ‘events’

expyriment.io.defaults.**keyboard_default_keys**
default value: None

expyriment.io.defaults.**markeroutput_default_code**
default value: 1

expyriment.io.defaults.**markeroutput_default_duration**
default value: None

expyriment.io.defaults.**mouse_show_cursor**
default value: True

expyriment.io.defaults.**mouse_track_button_events**
default value: True

expyriment.io.defaults.**mouse_track_motion_events**
default value: False

expyriment.io.defaults.**outputfile_comment_char**
default value: ‘#’

expyriment.io.defaults.**outputfile_eol**
default value: ‘n’

expyriment.io.defaults.**outputfile_time_stamp**
default value: True

expyriment.io.defaults.**serialport_baudrate**
default value: 19200

expyriment.io.defaults.**serialport_bytesize**
default value: 8

expyriment.io.defaults.**serialport_dsrdtr**
default value: 0

expyriment.io.defaults.**serialport_input_history**
default value: None

expyriment.io.defaults.**serialport_input_timing**
default value: 1

expyriment.io.defaults.**serialport_os_buffer_size**
default value: 3000

expyriment.io.defaults.**serialport_parity**
default value: 'N'

expyriment.io.defaults.**serialport_rtscts**
default value: 0

expyriment.io.defaults.**serialport_stopbits**
default value: 1

expyriment.io.defaults.**serialport_timeout**
default value: 0

expyriment.io.defaults.**serialport_xonxoff**
default value: 0

expyriment.io.defaults.**streamingbuttonbox_baseline**
default value: 0

expyriment.io.defaults.**textinput_ascii_filter**
default value: None

expyriment.io.defaults.**textinput_background_colour**
default value: None

expyriment.io.defaults.**textinput_frame_colour**
default value: None

expyriment.io.defaults.**textinput_gap**
default value: 6

expyriment.io.defaults.**textinput_length**
default value: 25

expyriment.io.defaults.**textinput_message_bold**
default value: False

expyriment.io.defaults.**textinput_message_colour**
default value: None

expyriment.io.defaults.**textinput_message_font**
default value: None

expyriment.io.defaults.**textinput_message_italic**
default value: False

expyriment.io.defaults.**textinput_message_text_size**
default value: None

expyriment.io.defaults.**textinput_position**
default value: (0, 0)

expyriment.io.defaults.**textinput_user_text_bold**
default value: False

expyriment.io.defaults.**textinput_user_text_colour**
default value: None

expyriment.io.defaults.**textinput_user_text_font**
default value: None

expyriment.io.defaults.**textinput_user_text_size**
default value: None

expyriment.io.defaults.**textmenu_background_colour**
default value: (0, 0, 0)

expyriment.io.defaults.**textmenu_gap**
default value: 2

expyriment.io.defaults.**textmenu_heading_font**
default value: None

expyriment.io.defaults.**textmenu_heading_text_colour**
default value: (255, 150, 50)

expyriment.io.defaults.**textmenu_justification**
default value: 1

expyriment.io.defaults.**textmenu_position**
default value: (0, 0)

expyriment.io.defaults.**textmenu_scroll_menu**
default value: 0

expyriment.io.defaults.**textmenu_select_background_colour**
default value: (150, 150, 150)

expyriment.io.defaults.**textmenu_select_frame_colour**
default value: (255, 150, 50)

expyriment.io.defaults.**textmenu_select_frame_line_width**
default value: 0

expyriment.io.defaults.**textmenu_select_text_colour**
default value: [0, 0, 0]

expyriment.io.defaults.**textmenu_text_colour**
default value: (200, 200, 200)

expyriment.io.defaults.**textmenu_text_font**
default value: None

expyriment.io.defaults.**textmenu_text_size**
default value: 20

expyriment.io.defaults.**triggerinput_default_code**
default value: 1

expyriment.io.extras

Modules

expyriment.io.extras.defaults

Attributes

expyriment.io.extras.defaults.**midiin_buffer_size**
default value: 1024

expyriment.io.extras.defaults.**midiout_buffer_size**
default value: 1024

expyriment.io.extras.defaults.**midiout_latency**
default value: 0

Classes

expyriment.io.extras.CedrusResponseDevice

expyriment.io.extras.MidiIn

expyriment.io.extras.MidiOut

Classes

expyriment.io.DataFile

expyriment.io.EventButtonBox

expyriment.io.EventFile

expyriment.io.GamePad

expyriment.io.InputFile

expyriment.io.Keyboard

expyriment.io.MarkerOutput

expyriment.io.Mouse

expyriment.io.OutputFile

expyriment.io.ParallelPort

expyriment.io.Screen

expyriment.io.SerialPort

expyriment.io.StreamingButtonBox

expyriment.io.TextInput

expyriment.io.TextMenu

expyriment.io.TouchScreenButtonBox

expyriment.io.TriggerInput

expyriment.misc

Modules

expyriment.misc.constants

Attributes

expyriment.misc.constants.**C_BLACK**
default value: (0, 0, 0)

expyriment.misc.constants.**C_BLUE**
default value: (0, 0, 255)

expyriment.misc.constants.**C_DARKGREY**
default value: (150, 150, 150)

expyriment.misc.constants.**C_EXPYRIMENT_ORANGE**
default value: (255, 150, 50)

expyriment.misc.constants.**C_EXPYRIMENT_PURPLE**
default value: (160, 70, 250)

expyriment.misc.constants.**C_GREEN**
default value: (0, 255, 0)

expyriment.misc.constants.**C_GREY**
default value: (200, 200, 200)

expyriment.misc.constants.**C_RED**
default value: (255, 0, 0)

expyriment.misc.constants.**C_WHITE**
default value: (255, 255, 255)

expyriment.misc.constants.**C_YELLOW**
default value: (255, 255, 0)

expyriment.misc.constants.**EXPYRIMENT_LOGO_FILE**
default value: u'/build/python-expyriment-iaBzlQ/python-expyriment-0.7.0+git34-g55a4e7e/documentation/sphinx/expyriment/expyriment_logo.png'

expyriment.misc.constants.**K_0**
default value: 48

expyriment.misc.constants.**K_1**
default value: 49

expyriment.misc.constants.**K_2**
default value: 50

expyriment.misc.constants.**K_3**
default value: 51

expyriment.misc.constants.**K_4**
default value: 52

expyriment.misc.constants.**K_5**
default value: 53

expyriment.misc.constants.**K_6**
default value: 54

expyriment.misc.constants.**K_7**
default value: 55

expyriment.misc.constants.**K_8**
default value: 56

expyriment.misc.constants.**K_9**
default value: 57

expyriment.misc.constants.**K_ALL_DIGITS**
default value: [48, 49, 50, 51, 52, 53, 54, 55, 56, 57]

expyriment.misc.constants.**K_ALL_LETTERS**
default value: [97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122]

expyriment.misc.constants.**K_AMPERSAND**
default value: 38

expyriment.misc.constants.**K_ASTERISK**
default value: 42

expyriment.misc.constants.**K_AT**
default value: 64

expyriment.misc.constants.**K_BACKQUOTE**
default value: 96

expyriment.misc.constants.**K_BACKSLASH**
default value: 92

expyriment.misc.constants.**K_BACKSPACE**
default value: 8

expyriment.misc.constants.**K_BREAK**
default value: 318

expyriment.misc.constants.**K_CAPSLOCK**
default value: 301

expyriment.misc.constants.**K_CARET**
default value: 94

expyriment.misc.constants.**K_CLEAR**
default value: 12

expyriment.misc.constants.**K_COLON**
default value: 58

expyriment.misc.constants.**K_COMMA**
default value: 44

expyriment.misc.constants.**K_DELETE**
default value: 127

expyriment.misc.constants.**K_DOLLAR**
default value: 36

expyriment.misc.constants.**K_DOWN**
default value: 274

expyriment.misc.constants.**K_END**
default value: 279

expyriment.misc.constants.**K_EQUALS**
default value: 61

expyriment.misc.constants.**K_ESCAPE**
default value: 27

expyriment.misc.constants.**K_EURO**
default value: 321

expyriment.misc.constants.**K_EXCLAIM**
default value: 33

expyriment.misc.constants.**K_F1**
default value: 282

expyriment.misc.constants.**K_F10**
default value: 291

expyriment.misc.constants.**K_F11**
default value: 292

expyriment.misc.constants.**K_F12**
default value: 293

expyriment.misc.constants.**K_F13**
default value: 294

expyriment.misc.constants.**K_F14**
default value: 295

expyriment.misc.constants.**K_F15**
default value: 296

expyriment.misc.constants.**K_F2**
default value: 283

expyriment.misc.constants.**K_F3**
default value: 284

expyriment.misc.constants.**K_F4**
default value: 285

expyriment.misc.constants.**K_F5**
default value: 286

expyriment.misc.constants.**K_F6**
default value: 287

expyriment.misc.constants.**K_F7**
default value: 288

expyriment.misc.constants.**K_F8**
default value: 289

expyriment.misc.constants.**K_F9**
default value: 290

expyriment.misc.constants.**K_GREATER**
default value: 62

expyriment.misc.constants.**K_HASH**
default value: 35

expyriment.misc.constants.**K_HELP**
default value: 315

expyriment.misc.constants.**K_HOME**
default value: 278

expyriment.misc.constants.**K_INSERT**
default value: 277

expyriment.misc.constants.**K_KP0**
default value: 256

expyriment.misc.constants.**K_KP1**
default value: 257

expyriment.misc.constants.**K_KP2**
default value: 258

expyriment.misc.constants.**K_KP3**
default value: 259

expyriment.misc.constants.**K_KP4**
default value: 260

expyriment.misc.constants.**K_KP5**
default value: 261

expyriment.misc.constants.**K_KP6**
default value: 262

expyriment.misc.constants.**K_KP7**
default value: 263

expyriment.misc.constants.**K_KP8**
default value: 264

expyriment.misc.constants.**K_KP9**
default value: 265

expyriment.misc.constants.**K_KP_DIVIDE**
default value: 267

expyriment.misc.constants.**K_KP_ENTER**
default value: 271

expyriment.misc.constants.**K_KP_EQUALS**
default value: 272

expyriment.misc.constants.**K_KP_MINUS**
default value: 269

expyriment.misc.constants.**K_KP_MULTIPLY**
default value: 268

expyriment.misc.constants.**K_KP_PERIOD**
default value: 266

expyriment.misc.constants.**K_KP_PLUS**
default value: 270

expyriment.misc.constants.**K_LALT**
default value: 308

expyriment.misc.constants.**K_LCTRL**
default value: 306

expyriment.misc.constants.**K_LEFT**
default value: 276

expyriment.misc.constants.**K_LEFTBRACKET**
default value: 91

expyriment.misc.constants.**K_LEFTPAREN**
default value: 40

expyriment.misc.constants.**K_LESS**
default value: 60

expyriment.misc.constants.**K_LMETA**
default value: 310

expyriment.misc.constants.**K_LSHIFT**
default value: 304

expyriment.misc.constants.**K_LSUPER**
default value: 311

expyriment.misc.constants.**K_MENU**
default value: 319

expyriment.misc.constants.**K_MINUS**
default value: 45

expyriment.misc.constants.**K_MODE**
default value: 313

expyriment.misc.constants.**K_NUMLOCK**
default value: 300

expyriment.misc.constants.**K_PAGEDOWN**
default value: 281

expyriment.misc.constants.**K_PAGEUP**
default value: 280

expyriment.misc.constants.**K_PAUSE**
default value: 19

expyriment.misc.constants.**K_PERIOD**
default value: 46

expyriment.misc.constants.**K_PLUS**
default value: 43

expyriment.misc.constants.**K_POWER**
default value: 320

expyriment.misc.constants.**K_PRINT**
default value: 316

expyriment.misc.constants.**K_QUESTION**
default value: 63

expyriment.misc.constants.**K_QUOTE**
default value: 39

expyriment.misc.constants.**K_QUOTEDBL**
default value: 34

expyriment.misc.constants.**K_RALT**
default value: 307

expyriment.misc.constants.**K_RCTRL**
default value: 305

expyriment.misc.constants.**K_RETURN**
default value: 13

expyriment.misc.constants.**K_RIGHT**
default value: 275

expyriment.misc.constants.**K_RIGHTBRACKET**
default value: 93

expyriment.misc.constants.**K_RIGHTPAREN**
default value: 41

expyriment.misc.constants.**K_RMETA**
default value: 309

expyriment.misc.constants.**K_RSHIFT**
default value: 303

expyriment.misc.constants.**K_RSUPER**
default value: 312

expyriment.misc.constants.**K_SCROLLOCK**
default value: 302

expyriment.misc.constants.**K_SEMICOLON**
default value: 59

expyriment.misc.constants.**K_SLASH**
default value: 47

expyriment.misc.constants.**K_SPACE**
default value: 32

expyriment.misc.constants.**K_SYSREQ**
default value: 317

expyriment.misc.constants.**K_TAB**
default value: 9

expyriment.misc.constants.**K_UNDERSCORE**
default value: 95

expyriment.misc.constants.**K_UP**
default value: 273

expyriment.misc.constants.**K_a**
default value: 97

expyriment.misc.constants.**K_b**
default value: 98

expyriment.misc.constants.**K_c**
default value: 99

expyriment.misc.constants.**K_d**
default value: 100

expyriment.misc.constants.**K_e**
default value: 101

expyriment.misc.constants.**K_f**
default value: 102

expyriment.misc.constants.**K_g**
default value: 103

expyriment.misc.constants.**K_h**
default value: 104

expyriment.misc.constants.**K_i**
default value: 105

expyriment.misc.constants.**K_j**
default value: 106

expyriment.misc.constants.**K_k**
default value: 107

expyriment.misc.constants.**K_l**
default value: 108

expyriment.misc.constants.**K_m**
default value: 109

expyriment.misc.constants.**K_n**
default value: 110

expyriment.misc.constants.**K_o**
default value: 111

expyriment.misc.constants.**K_p**
default value: 112

expyriment.misc.constants.**K_q**
default value: 113

expyriment.misc.constants.**K_r**
default value: 114

expyriment.misc.constants.**K_s**
default value: 115

expyriment.misc.constants.**K_t**
default value: 116

expyriment.misc.constants.**K_u**
default value: 117

expyriment.misc.constants.**K_v**
default value: 118

expyriment.misc.constants.**K_w**
default value: 119

expyriment.misc.constants.**K_x**
default value: 120

expyriment.misc.constants.**K_y**
default value: 121

expyriment.misc.constants.**K_z**
default value: 122

expyriment.misc.constants.**P_BALANCED_LATIN_SQUARE**
default value: 'balanced-latin-square'

expyriment.misc.constants.**P_CYCLED_LATIN_SQUARE**
default value: 'cycled-latin-square'

expyriment.misc.constants.**P_RANDOM**
default value: 'random'

expyriment.misc.data_preprocessing

Classes

expyriment.misc.data_preprocessing.Aggregator

Functions

expyriment.misc.defaults

expyriment.misc.extras

Modules

expyriment.misc.extras.defaults

expyriment.misc.geometry

Classes

expyriment.misc.geometry.XYPoint

Functions

expyriment.misc.statistics

Functions

Classes

expyriment.misc.Buffer

expyriment.misc.ByteBuffer

expyriment.misc.Clock

Functions

expyriment.stimuli

Modules

expyriment.stimuli.defaults

Attributes

expyriment.stimuli.defaults.**canvas_colour**
default value: None

expyriment.stimuli.defaults.**canvas_position**
default value: (0, 0)

expyriment.stimuli.defaults.**circle_anti_aliasing**
default value: 0

expyriment.stimuli.defaults.**circle_colour**
default value: None

expyriment.stimuli.defaults.**circle_line_width**
default value: 0

expyriment.stimuli.defaults.**circle_position**
default value: (0, 0)

expyriment.stimuli.defaults.**dot_anti_aliasing**
default value: 0

expyriment.stimuli.defaults.**dot_colour**
default value: None

expyriment.stimuli.defaults.**dot_position**
default value: (0, 0)

expyriment.stimuli.defaults.**ellipse_anti_aliasing**
default value: 0

expyriment.stimuli.defaults.**ellipse_colour**
default value: None

expyriment.stimuli.defaults.**ellipse_line_width**
default value: 0

expyriment.stimuli.defaults.**ellipse_position**
default value: (0, 0)

expyriment.stimuli.defaults.**fixcross_anti_aliasing**
default value: 0

expyriment.stimuli.defaults.**fixcross_colour**
default value: None

expyriment.stimuli.defaults.**fixcross_line_width**
default value: 1

expyriment.stimuli.defaults.**fixcross_position**
default value: (0, 0)

expyriment.stimuli.defaults.**fixcross_size**
default value: (20, 20)

expyriment.stimuli.defaults.**frame_anti_aliasing**
default value: 0

expyriment.stimuli.defaults.**frame_colour**
default value: None

expyriment.stimuli.defaults.**frame_frame_line_width**
default value: 5

expyriment.stimuli.defaults.**frame_position**
default value: (0, 0)

expyriment.stimuli.defaults.**line_anti_aliasing**
default value: 0

expyriment.stimuli.defaults.**line_colour**
default value: None

expyriment.stimuli.defaults.**picture_position**
default value: (0, 0)

expyriment.stimuli.defaults.**rectangle_colour**
default value: None

expyriment.stimuli.defaults.**rectangle_line_width**
default value: 0

expyriment.stimuli.defaults.**rectangle_position**
default value: (0, 0)

expyriment.stimuli.defaults.**shape_anti_aliasing**
default value: 0

expyriment.stimuli.defaults.**shape_colour**
default value: None

expyriment.stimuli.defaults.**shape_line_width**
default value: 0

expyriment.stimuli.defaults.**shape_position**
default value: (0, 0)

expyriment.stimuli.defaults.**tempdir**
default value: '/tmp/expyriment'

expyriment.stimuli.defaults.**textbox_background_colour**
default value: None

expyriment.stimuli.defaults.**textbox_position**
default value: (0, 0)

expyriment.stimuli.defaults.**textbox_text_bold**
default value: False

expyriment.stimuli.defaults.**textbox_text_colour**
default value: None

expyriment.stimuli.defaults.**textbox_text_font**
default value: None

expyriment.stimuli.defaults.**textbox_text_italic**
default value: False

expyriment.stimuli.defaults.**textbox_text_justification**
default value: 1

expyriment.stimuli.defaults.**textbox_text_size**
default value: None

expyriment.stimuli.defaults.**textbox_text_underline**
default value: False

expyriment.stimuli.defaults.**textline_background_colour**
default value: None

expyriment.stimuli.defaults.**textline_position**
default value: (0, 0)

expyriment.stimuli.defaults.**textline_text_bold**
default value: False

expyriment.stimuli.defaults.**textline_text_colour**
default value: None

expyriment.stimuli.defaults.**textline_text_font**
default value: None

expyriment.stimuli.defaults.**textline_text_italic**
default value: False

expyriment.stimuli.defaults.**textline_text_size**
default value: None

expyriment.stimuli.defaults.**textline_text_underline**
default value: False

expyriment.stimuli.defaults.**textscreen_background_colour**
default value: None

expyriment.stimuli.defaults.**textscreen_heading_bold**
default value: False

expyriment.stimuli.defaults.**textscreen_heading_colour**
default value: None

expyriment.stimuli.defaults.**textscreen_heading_font**
default value: None

expyriment.stimuli.defaults.**textscreen_heading_italic**
default value: False

expyriment.stimuli.defaults.**textscreen_heading_size**
default value: None

expyriment.stimuli.defaults.**textscreen_heading_underline**
default value: False

expyriment.stimuli.defaults.**textscreen_position**
default value: (0, 0)

expyriment.stimuli.defaults.**textscreen_size**
default value: None

expyriment.stimuli.defaults.**textscreen_text_bold**
default value: False

expyriment.stimuli.defaults.**textscreen_text_colour**
default value: None

expyriment.stimuli.defaults.**textscreen_text_font**
default value: None

expyriment.stimuli.defaults.**textscreen_text_italic**
default value: False

expyriment.stimuli.defaults.**textscreen_text_justification**
default value: 1

expyriment.stimuli.defaults.**textscreen_text_size**
default value: None

expyriment.stimuli.defaults.**textscreen_text_underline**
default value: False

expyriment.stimuli.defaults.**tone_amplitude**
default value: 0.5

expyriment.stimuli.defaults.**tone_bitdepth**
default value: 16

expyriment.stimuli.defaults.**tone_frequency**
default value: 440

expyriment.stimuli.defaults.**tone_samplerate**
default value: 44100

expyriment.stimuli.defaults.**video_position**
default value: [0, 0]

expyriment.stimuli.defaults.**visual_position**
default value: (0, 0)

expyriment.stimuli.extras

Modules

expyriment.stimuli.extras.defaults

Attributes

expyriment.stimuli.extras.defaults.**dotcloud_background_colour**
default value: None

expyriment.stimuli.extras.defaults.**dotcloud_dot_colour**
default value: None

expyriment.stimuli.extras.defaults.**dotcloud_flipping**
default value: None

expyriment.stimuli.extras.defaults.**dotcloud_position**
default value: (0, 0)

expyriment.stimuli.extras.defaults.**dotcloud_radius**
default value: None

expyriment.stimuli.extras.defaults.**dotcloud_rotation**
default value: None

expyriment.stimuli.extras.defaults.**dotcloud_scaling**
default value: None

expyriment.stimuli.extras.defaults.**lcdsymbol_background_colour**
default value: None

expyriment.stimuli.extras.defaults.**lcdsymbol_colour**
default value: None

expyriment.stimuli.extras.defaults.**lcdsymbol_flipping**
default value: None

expyriment.stimuli.extras.defaults.**lcdsymbol_gap**
default value: 5

expyriment.stimuli.extras.defaults.**lcdsymbol_inactive_colour**
default value: None

expyriment.stimuli.extras.defaults.**lcdsymbol_line_width**
default value: 1

expyriment.stimuli.extras.defaults.**lcdsymbol_position**
default value: (0, 0)

expyriment.stimuli.extras.defaults.**lcdsymbol_rotation**
default value: None

expyriment.stimuli.extras.defaults.**lcdsymbol_scaling**
default value: None

expyriment.stimuli.extras.defaults.**lcdsymbol_simple_lines**
default value: False

expyriment.stimuli.extras.defaults.**lcdsymbol_size**
default value: None

expyriment.stimuli.extras.defaults.**noisetone_amplitude**
default value: 0.5

expyriment.stimuli.extras.defaults.**noisetone_bitdepth**
default value: 16

expyriment.stimuli.extras.defaults.**noisetone_samplerate**
default value: 44100

expyriment.stimuli.extras.defaults.**polygondot_anti_aliasing**
default value: 10

expyriment.stimuli.extras.defaults.**polygondot_colour**
default value: None

expyriment.stimuli.extras.defaults.**polygondot_position**
default value: (0, 0)

expyriment.stimuli.extras.defaults.**polygondot_resolution_factor**
default value: 1

expyriment.stimuli.extras.defaults.**polygonellipse_anti_aliasing**
default value: 5

expyriment.stimuli.extras.defaults.**polygonellipse_colour**
default value: None

expyriment.stimuli.extras.defaults.**polygonellipse_line_width**
default value: 0

expyriment.stimuli.extras.defaults.**polygonellipse_position**
default value: (0, 0)

expyriment.stimuli.extras.defaults.**polygonellipse_resolution_factor**
default value: 1

expyriment.stimuli.extras.defaults.**polygonline_anti_aliasing**
default value: 0

expyriment.stimuli.extras.defaults.**polygonline_colour**
default value: None

expyriment.stimuli.extras.defaults.**polygonrectangle_anti_aliasing**
default value: 0

expyriment.stimuli.extras.defaults.**polygonrectangle_colour**
default value: None

expyriment.stimuli.extras.defaults.**polygonrectangle_position**
default value: (0, 0)

expyriment.stimuli.extras.defaults.**randomdotkinematogram_background_colour**
default value: None

expyriment.stimuli.extras.defaults.**randomdotkinematogram_dot_colour**
default value: None

expyriment.stimuli.extras.defaults.**randomdotkinematogram_dot_diameter**
default value: 5

expyriment.stimuli.extras.defaults.**randomdotkinematogram_dot_lifetime**
default value: 400

expyriment.stimuli.extras.defaults.**randomdotkinematogram_dot_speed**
default value: 100

expyriment.stimuli.extras.defaults.**randomdotkinematogram_north_up_clockwise**
default value: True

expyriment.stimuli.extras.defaults.**randomdotkinematogram_position**
default value: (0, 0)

expyriment.stimuli.extras.defaults.**stimuluscircle_background_colour**
default value: None

expyriment.stimuli.extras.defaults.**stimuluscloud_background_colour**
default value: None

expyriment.stimuli.extras.defaults.**stimuluscloud_flipping**
default value: None

expyriment.stimuli.extras.defaults.**stimuluscloud_position**
default value: (0, 0)

expyriment.stimuli.extras.defaults.**stimuluscloud_rotation**
default value: None

expyriment.stimuli.extras.defaults.**stimuluscloud_scaling**
default value: None

expyriment.stimuli.extras.defaults.**stimuluscloud_size**
default value: None

expyriment.stimuli.extras.defaults.**visualmask_background_colour**
default value: None

expyriment.stimuli.extras.defaults.**visualmask_dot_colour**
default value: None

expyriment.stimuli.extras.defaults.**visualmask_dot_percentage**
default value: 50

expyriment.stimuli.extras.defaults.**visualmask_dot_size**
default value: (5, 5)

expyriment.stimuli.extras.defaults.**visualmask_smoothing**
default value: 3

Classes

expyriment.stimuli.extras.DotCloud

expyriment.stimuli.extras.LcdSymbol

expyriment.stimuli.extras.NoiseTone

expyriment.stimuli.extras.PolygonDot

expyriment.stimuli.extras.PolygonEllipse

expyriment.stimuli.extras.PolygonLine

expyriment.stimuli.extras.PolygonRectangle

expyriment.stimuli.extras.RandomDotKinematogram

expyriment.stimuli.extras.StimulusCircle

expyriment.stimuli.extras.StimulusCloud

expyriment.stimuli.extras.VisualMask

Classes

expyriment.stimuli.Audio

expyriment.stimuli.BlankScreen

expyriment.stimuli.Canvas

expyriment.stimuli.Circle

expyriment.stimuli.Dot

expyriment.stimuli.Ellipse

expyriment.stimuli.FixCross

expyriment.stimuli.Frame

expyriment.stimuli.Line

expyriment.stimuli.Picture

expyriment.stimuli.Rectangle

expyriment.stimuli.Shape

expyriment.stimuli.TextBox

expyriment.stimuli.TextLine

expyriment.stimuli.TextScreen

expyriment.stimuli.Tone

expyriment.stimuli.Video

1.2.2 Functions

1.3 Advanced

1.3.1 API reference tool

Besides this online documentation, Expyriment includes a full offline API reference tool which will allow you to browse and search the API offline, using a graphical user interface. For more information on offline documentation, please call:

```
expyriment.show_documentation()
```

1.3.2 Expyriment test suite

The Expyriment test suite is a guided tool for testing your computer's abilities/performance. This includes timing accuracy of visual stimulus presentation, audio playback functionality, mouse functionality and serial port functionality/usage.

Eventually, all test results can be saved as a protocol, together with some information about the system.

Starting the test suite

The test suite can either be started from within an experiment, or from an interactive Python session (for instance with IPython).

To start the test suite, just call:

```
expyriment.control.run_test_suite()
```

Menu overview

Here is a brief explanation of the available options:

1. *Visual stimulus presentation*
   - Tests if stimuli can be presented timing accurately
   - Tests if stimulus presentation is synchronized to the refresh rate of the screen
   - Tests the video card's settings for buffering
2. *Auditory stimulus presentation*
   - Tests functionality of audio playback
3. *Font viewer*
   - Test installed fonts
4. *Mouse test*
   - Tests mouse accuracy (polling time)
   - Tests functionality of mouse buttons
5. *Serial port test*
   - Tests functionality of devices connected via the serial port
6. *Write protocol*
   - Saves all test results, as well as information about the system, as a text file.

7. *Quit*

- Quits the test suite

1.3.3 Command line interface

The command line interface can be used to start the Expyriment Reference Tool, or to run the Expyriment test suite or any Python script with predefined Expyriment default settings.

Usage:

```
python -m expyriment.cli [EXPYRIMENT SCRIPT] [OPTIONS]

    OPTIONS:
          -g              No OpenGL
          -t              No time stamps for output files
          -w              Window mode
          -f              Fast mode (no initialize delay and fast quitting)
          -a              Auto create subject ID
          -i              Intensive logging (log level 2)
          -d              Develop mode (equivalent to -gwfat)
          -T              Run Test Suite
          -A              Start API Reference Tool
          -h              Show this help
```

1.3.4 Data preprocessing

In most cases, data acquired by Expyriment needs to be further processed before a statistical analysis can be performed. This processing entails an aggregation of the dependent variables over all factor-level combinations of the experimental design. Expyriment provides an easy, but flexible way to automatize this process with the included data preprocessing module of the misc package (*expyriment.misc.data_preprocessing*).

1.3.5 Import Expyriment data into R

The function expyriment_data.R concatenates all data and returns an R data frame with all subjects. Between subject factors will be added as variables to the data matrix.

1.3.6 Using non-English characters

Expyriment has full unicode support. This means that, in principle, non-English characters (such as umlaut, accent, special character) can be used in strings throughout the library. Two different forms of using non-English characters have to be dissociated:

Non-English characters in strings in the Expyriment script file

When attempting to use non-English characters in strings in your Expyriment script file, the following three conditions have to be met:

1. **Only use non-English charactes in unicode strings!**

   For example: Use `u"Überexperiment"` instead of `"Überexperiment"`.

2. **Know the encoding used by your editor!**

   For example: IDLE will automatically suggest to save in utf-8 encoding when non-English characters are found in the script file. We suggest to always save in utf-8.

3. **Define the encoding in your Expyriment script file!**:

   ```
   # -*- coding: <encoding> -*-
   ```

   The line has to be one of the first two lines in the file, where <encoding> is the encoding used!

   For example:

   ```
   # -*- coding: utf-8 -*-
   ```

Non-English characters in other text files (e.g. stimuli lists)

When an Expyriment method saves a text file, it will always automatically add a header line specifying the encoding with which the file was saved. Which encoding this is depends on the system Expyriment is running on (it uses the default encoding defined by the locale settings).

When an Expyriment method reads in a text file, it will always read the header line first and decode the text automatically (into unicode strings). If no such header is found, the encoding set by the system locale will be used (and if this fails, utf-8).

1.3.7 The Expyriment plugin system (extras)

Usage

The design, stimuli, io and misc packages can be extended with plugins (additional classes) that can be accessed via the 'extras' namespace of each package. There are two locations Expyriment will look for installed plugins:

1. In the 'extras' directories of the corresponding packages of the Expyriment installation.
2. In the 'design', 'stmuli', 'io' and 'misc' direcories within a '.expyriment' or '~expyriment' directory located in the current user's home directory

In both cases, plugins will be integrated into the '.extras' namespace of each package (e.g. expyriment.stimuli.extras.DotCloud).

Development

Basically, extra plugins are a simple python module with a single class, where the filename is the class name in lowercase. Additionally a file called 'classname_defaults.py' can be created which will hold the default values for all parameters given when initializing the class. The naming convention is 'classname_parameter'. For design and misc extras this is all there is, but for io and stimuli plugins, additional conventions need to be taken care of.

io.extras

IO plugins have to inherit from 'expyriment.io.input' or 'expyriment.io.output' or both. This means they can also inherit from any other io class.

stimuli.extras

Stimulus plugins have to inherit from 'expyriment.stimuli.Stimulus'. This means they can also inherit from any other stimulus class. Additionally, every extra stimulus class needs a '_create_surface' method that defines what happens when the stimulus is preloaded, plotted or presented.

1.4 Technical issues

1.4.1 Hardware compatibility

Video cards

We generally have good experiences with recent NVIDIA or ATI cards. OpenGL mode should work with all drivers that use an OpenGL specification >= 2.0. Drivers implementing an older OpenGL specification (>= 1.4) should work when the 'GL_ARB_texture_non_power_of_two' extension is present.

On some some integrated Intel cards syncing to the vertical retrace does not seem to work!

Working configurations

Here is a list of configurations we observed to work:

- Nvidia GTX 650 (Linux-x86; NVIDIA driver 310.14)
- Nvidia Quadro NVS 290 (Linux-x86; NVIDIA driver 295.40)
- Nvidia Quadro NVS 290 (Windows XP SP3; NVIDIA driver)

We recommend to always use the Expyriment test suite to check the performance of your specific configuration!

External devices

Besides standard serial and parallel port communication, there is special support for:

- Event button box
- Streaming button box
- Trigger input
- Marker output
- Cedrus response devices

Event button box

An event button box is a simple device which sends out values (bytes) whenever a button is pressed (or released).

Event button boxes can be used by initializing an *expyriment.io.EventButtonBox* object:

```
bb = expyriment.io.EventButtonBox(expyriment.io.SerialPort("COM1"))
key, rt = bb.wait() # Wait for any value
```

Streaming button box

A streaming button box constantly sends out a certain baseline value (e.g. 0) in predefined intervals (e.g. each 1 ms). Button press (or release) events (if present) are added to the baseline.

Streaming button boxes can be used by initializing an *expyriment.io.StreamingButtonBox* object:

```
bb = expyriment.io.StreamingButtonBox(expyriment.io.SerialPort("COM1"),
                              baseline=128)
key, rt = bb.wait() # Wait for any value other than 128
```

This allows for instance for calculating the response timing without relying on the computers internal clock, but by "counting" the incoming bytes from the button box:

```
bb = expyriment.io.StreamingButtonBox(
            expyriment.io.SerialPort("COM1"), baseline=128)
bb.clear()
exp.clock.wait(1000)
rt = bb.interface.read_input().index(129)    # Get reaction time by counting
                                             # input events since last clear
```

It is important to notice that operating systems only buffer a certain amount of bytes (usually 4096). To prevent an overflow of this buffer, the button box has to be checked regularly. Additionally, an `input_history` can be used on the *expyriment.io.SerialPort* object which is automatically updated whenever the serial port is polled or cleared. By setting the `os_buffer_size` correctly, a warning will be logged whenever the amount of bytes in the OS serial port buffer reaches maximum capacity. **The important part now is to update the input_history regularly**. To gain maximal control, this should be done manually at Sending to external deviceappropriate places in the code. However, Expyriment provides also the possibility to register a callback function which will be called regularly during all waiting methods in the library. By registering the `check()` method of the streaming button box, the `input_history` will be updated fairly regular, which should suffice for most use cases:

```
expyriment.control.register_wait_callback_function(bb.check)
bb.interface.input_history.check_value(129) # Check if 129 was
                                            # received at any time
# RT by counting elements in input history
start = bb.interface.input_history.get_size() - 1
exp.clock.wait(1000)
rt = bb.interface.input_nput_history.check_value(129,
                                 search_start_position=start) - start
```

Trigger input

Expyriment can wait for triggers from external devices, like for instance an MR scanner.

When updated regularly, Expyriment can also keep track of the amount of triggers that have been received. Importantly, this has to be done manually

Trigger inputs can be used by initializing an *expyriment.io.TriggerInput* object.

Basic usage

In most of the cases, a researcher knows when a trigger is to be expected and he can wait for it explicitly. Code execution will be blocked until the trigger is received:

```
trigger = exyriment.io.TriggerInput(expyriment.io.SerialPort("COM1"))
trigger.wait(1) # Wait for code 1
```

Advanced usage

In some cases, code blocking might not be a solution, since a trial has to continue while waiting for the trigger. For instance, in an fMRI study, a trial might consist of several components and span several TR. One way to solve this would be logging constantly all input events in a separate thread. However, this will introduce timing uncertainties, since the operating system is in charge of how and when threads communicate. We thus decided against an implementation with threads for the same reasons Expyriment does not implement a main event loop: Maximal control by the user. Nevertheless, input events can still be buffered without introducing timing uncertainties, given the following two conditions:

1. Incoming events are streaming, either by sending some baseline in regular intervals (e.g. a 0 each millisecond), or by a regular incoming signal of interest (e.g. a constant TR from the MR scanner).
2. The input device is polled regularly, such that the serial port OS buffer does not overflow. (Most implementations use an OS buffer of 4096 bytes).

If those two conditions are met, an `input_history` can be used on the *expyriment.io.SerialPort* object which is automatically updated whenever the serial port is polled or cleared. By setting the `os_buffer_size` correctly, a warning will be logged whenever the amount of bytes in the OS serial port buffer reaches maximum capacity. **The important part now is to update the input_history regularly**. To gain maximal control, this should be done manually at appropriate places in the code. However, Expyriment provides also the possibility to register a callback function which will be called regularly during all waiting methods in the library. By registering the `get_trigger()` method of the input trigger, the `input_history` will be updated fairly regular, which should suffice for most use cases:

```
trigger = exyriment.io.TriggerInput(expyriment.io.SerialPort(external"COM1",
                input_history=True, os_buffer_size=3000))
expyriment.control.register_wait_callback_function(trigger.get_triggers)
print trigger.trigger_count
```

Marker output

Expyriment can send markers to external devices, like for instance EEG computers.

Marker outputs can be used by creating an *expyriment.io.MarkerOutput* object.

Basic usage

Sending out markers is straight forward. Some devices (e.g. EEG systems) expect a 0 to be send after the code. We can specify this by telling the output marker at what duration this 0 is supposed to be sent:

```
marker = expyriment.io.MarkerOutput(expyriment.io.SerialPort("COM1"), duration=20)
marker.send(1) # Send code 1
```

Cedrus response devices

Expyriment comes with a high-level wrapper for Cedrus response devices **expyriment.io.extras.CedrusResponseDevice_**, which allows you to easily use all Cedrus response devices.

To use these devices, however, the third-party Python package pyxid needs to be installed on the system.

Installing pyxid

- Download pyxid
- Install as described here.

1.4.2 Timing and empirical testing of Expyriment

How accurate is the timing in Expyriment?

In general, the Expyriment clock can feature up to 1 ms accuracy. The exact timing, however, is subject to various factors, which we discuss here.

Time and compatibility issues can be conveniently tested using the *Expyriment test suite*.

Stimulus presentation

Visual

Computer screens are updated according to their refresh rate. What happens then is that the whole screen is redrawn line by line all the time. For example, with a refresh rate of 60Hz, the screen is redrawn 60 times per second (1000ms) and the duration it takes to redraw it line by line is 16.66 ms (1000/60). When attempting to redraw the screen while it is currently already being updated (the lines are drawn) the result might lead to artifacts, since the update occurs immediately, leading to parts of both, the new and the old screen content, being visible. What is even worse is that you will never know in which phase of the redrawing the new redraw was started. Thus, you cannot be sure when exactly the new content is fully visible on screen.

The first step towards getting around this problem is to synchronize the actual redraw to the vertical retrace of the screen. This means that a change in content will never happen immediately, but always only when the retrace is at the top left position. When synchronizing to the vertical retrace, the graphic card is told to update the screen the next time it starts redrawing the first line. While this will solve the problem of artifacts, you will still face the problem of not knowing when exactly something was visible on the screen, since the graphic card handles this synchronization itself in the background.

Solving this problem is the exact (and only) reason why Expyriment uses OpenGL by default. It allows to wait for the vertical retrace to actually happen before proceeding with the code that tells the graphics card to update the screen (this is also known as blocking on the vertical retrace). This means that whenever something should be presented on screen, no matter in which line the redraw is at this point in time, the graphic card will wait for the redraw to be in the first line and then present the stimulus. Since the code is blocking, the time Expyriment reports the stimulus to be presented on screen will always be the time when the redraw is starting at the first line. Coming back to the example of the small dot in the center of the screen: Expyriment will correctly report a longer presentation time when the redraw has been just over the center line when the screen update was issued.

It is important to set your graphic card's driver settings to support synchronizing to the vertical retrace ("Sync to VBlank" or "V-sync") and to switch off any power saving schemes on the graphic card.

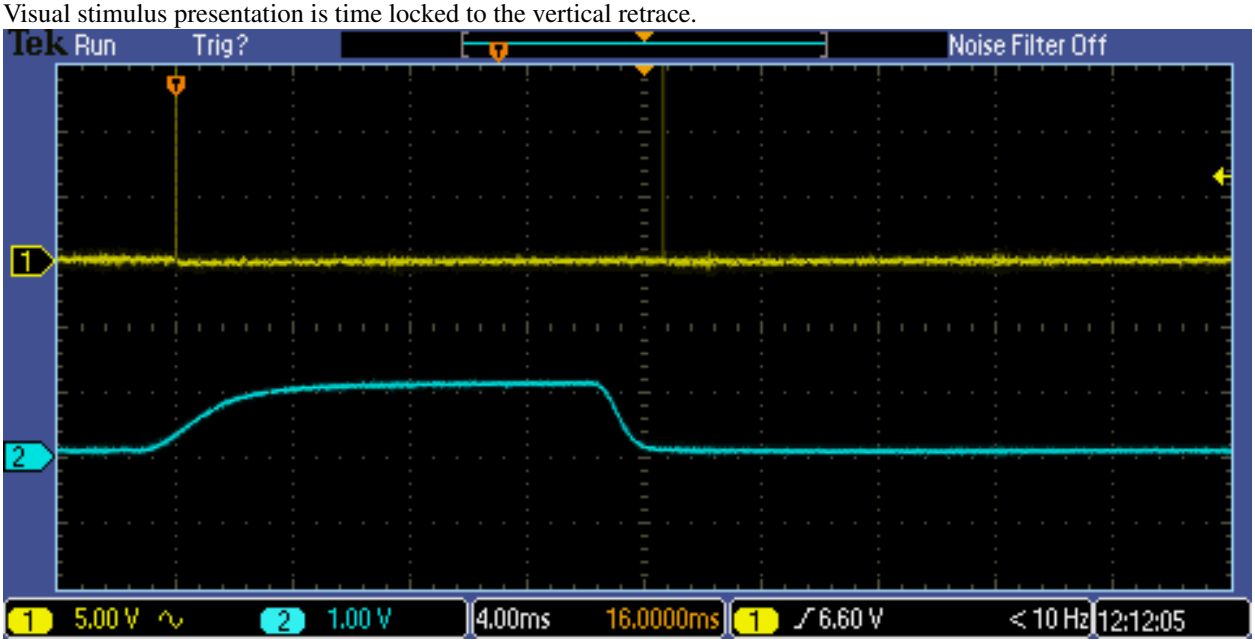
Test results

We tested the visual timing by presented a white and a black fullscreen rectangle directly after each other. The brightness of the left upper edge of the screen was recorded using an optic sensor attached to an oscilloscope. After each screen presentation, a marker was send via the serial port to the oscilloscope. Testing was done on an Intel Core Duo PC with an Nvidia Quadro NVS 290 graphics card, running Microsoft Windows XP SP3. The monitor used was a Samsung SyncMaster 2233. Expyriment was running in OpenGL mode (default).

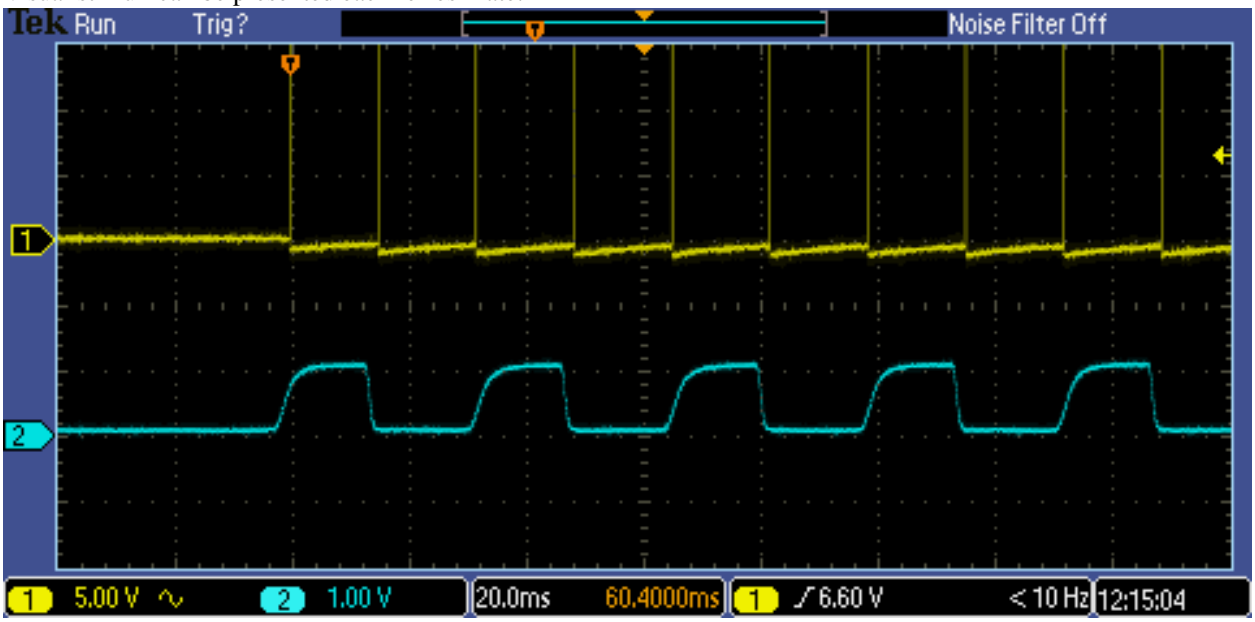
The results revealed:

- When presenting a stimulus, updating the screen does successfully block code execution until the vertical retrace actually happens. This can be seen in Figure 1, where the marker (yellow) lines up with the onset and offset of an increase in brightness (turquoise) which represent the onset of the white and the onset of the black screen, respectively.
- Presenting (preloaded) stimuli can accurately be done each refresh rate (Figure 2).

Visual stimulus presentation is time locked to the vertical retrace.



Visual stimuli can be presented each refresh rate.



Audio

Playing back audio is handled by PyGame. The present() and play() methods of auditory stimuli will return immediately. Since the audio stream has to be sent to the hardware, there will still be a delay before the audio can be heard. Unfortunately, the latency of the sound onset is not known by Expyriment. However, it is assumed to be relatively stable over time. Setting the audio buffersize to a smaller value than the default can decrease the delay, but might result in distorted audio.

It is important to set your samplerate, bitdepth and audio buffersize correctly. Setting the buffersize too low will result in distorted audio!

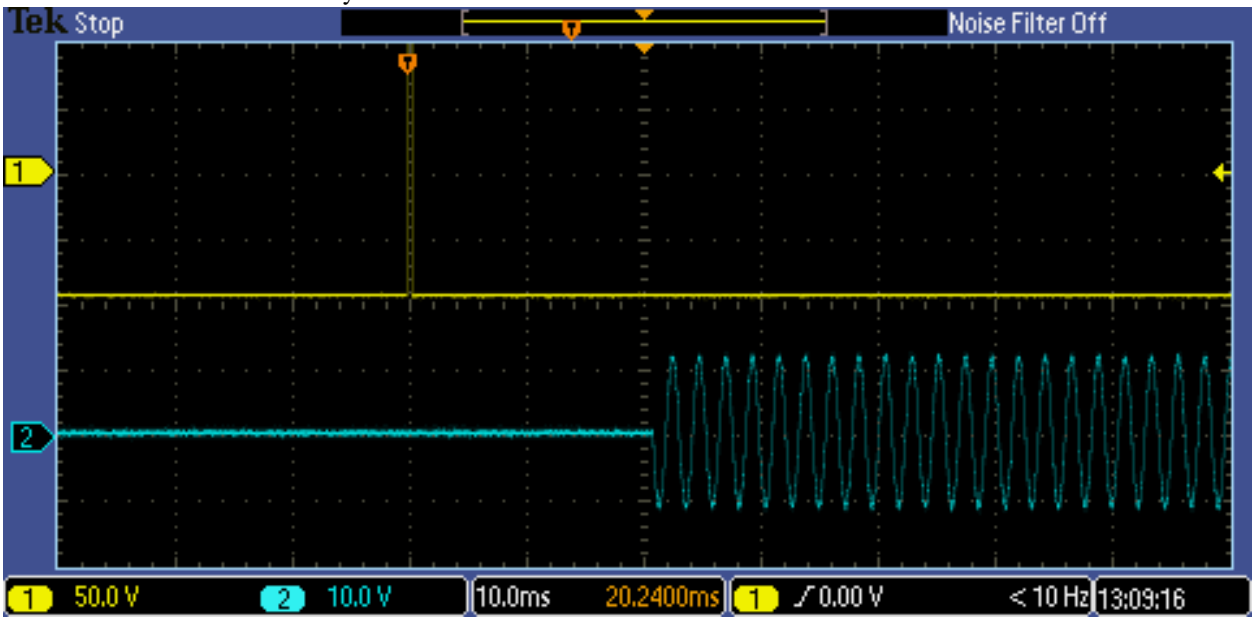
Test results

We tested the audio timing by repeatedly playing back a beep tone (a 1 second sine wave). The output of the sound card was measured by an oscilloscope. Before starting playback of the beep, a marker was send via the serial port to the oscilloscope. Testing was done on an Intel Core Duo PC with a Soundblaster Audigy sound card, running Microsoft Windows XP SP3. In Expyriment, the samplerate was set to 44100 Hz, bitdepth to 16 bit and the buffersize equaled 128.

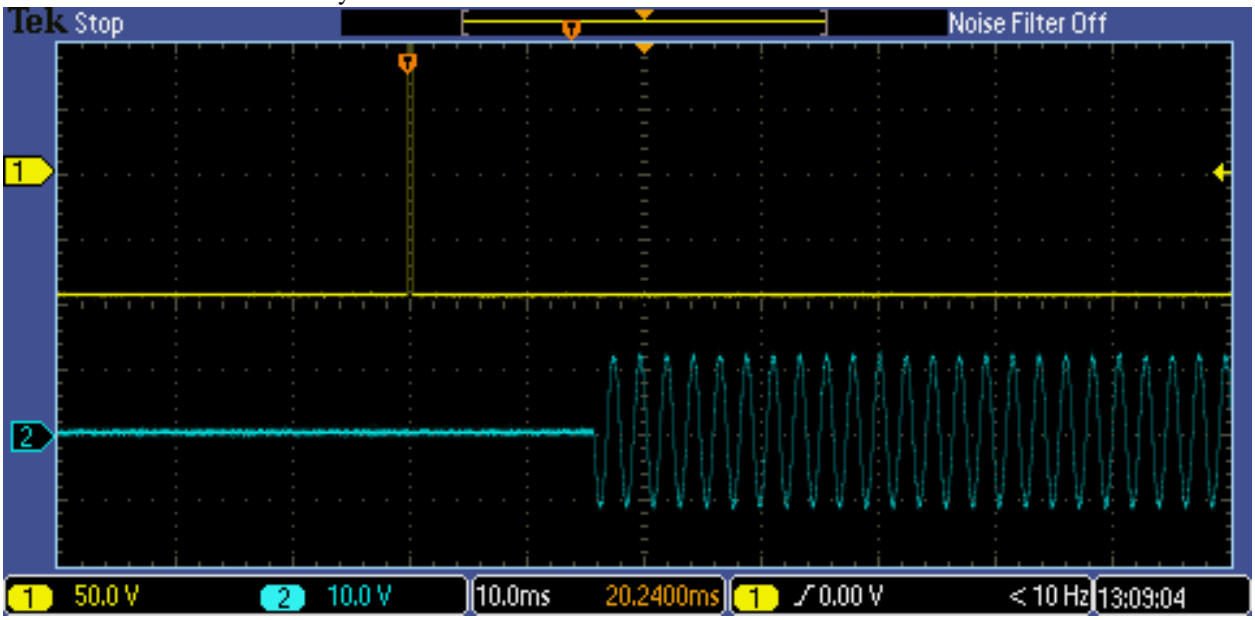
The results revealed:

- Audio playback was subject to a latency of maximally 20 ms. Figure 1 shows the maximal measured latency between the start of the playback (yellow) and the onset of the sound (turquoise).
- This latency was relatively stable with a jitter of 5 ms. Figure 2 shows the minimal latency we could measure.

Maximal measured audio latency.



Minimal measured audio latency



Video

Video presentation is a tricky subject. In Expyriment, the present() method of a video stimulus will start playback and present the first (current) frame on the screen. Thus, visual onset of this frame will be synchronized with the vertical retrace (see visual stimulus presentation above). Each following frame has to be plotted on the screen and the screen has to be updated. The wait_end() method of a video stimulus will automatically present each frame on the screen until the video is over. When Expyriment is in OpenGL mode, the process of plotting each frame might take longer than one refresh rate which will result in dropping frames (e.g. frames not being presented at all). To control for this, the wait_end() method will report and log if any frames were dropped during video playback.

Unfortunetly, right now, Expyriment can only handle MPEG-1 encoded videos!

Measuring user input

In Expyriment all inputs (keybard, mouse, gameport, serial port, parallel port) can be checked by directly polling them (via the wait() methods of the corresponding io object). This allows for the most accurate timing possible. Since Python wraps C functions for getting the system time, the accuracy is even more precise than milliseconds (which is the unit Expyriment uses).

Expyriment does *not* have a main event loop (i.e. it will not automatically check for any incoming events in the background)! This was a design decision, since we think that in 99% of all cases the time of the user input is specified in the design and thus know beforehand (e.g. a response after a stimulus onset). Adding an event loop would make things unnecessarily more complicated for those 99%.

However, we also thought of those cases that need to check user input during other operations: All events can manually be pushed from either Pygame's event cue (keyboard, mouse, joystick) or the operating system's buffer (serial port, parallel port) into an EventBuffer object. Doing this regularly is up to the user.

Keyboard

Keyboards (PS2 and USB) are known to have poor timing accuracy. Usually these are in the range of several 100th of a second.

Test results

We tested the timing of a Logitec USB keyboard in Windows XP SP3 using optical tracking.

Our results revealed:

- A timing accuracy between 20 and 26 ms.

Mouse

On most operating systems, USB mice are polled at a rate of 8 ms. Mice with special drivers might be set to poll more often.

Test results

We tested the mouse accuracy of a standard USB mouse on Windows XP SP3 by measuring the time between reported position changes.

Our results revealed:

- The expected standard accuracy of 8 ms.
- Using a Logitec G700 USB mouse with a dedicated driver, polling rates could A be reduced, leading to an increased accuracy of 1 ms.

Serial port

The serial port is very accurate and thus suited for timing accurate measurements. If a computer does not have a serial port, USB-to-serial converter can be used (e.g. from Sweex or Keyspan). However, the timing accuracy of these depends on the implementation and drivers used!

It is important to deactivate any additional FIFO buffers or delays, provided by the port driver!

Test results

We tested the timing of a UART 16550A serial port (a real one, not a USB-to-serial converter!) on Windows XP SP3 by sending a byte to a connected loopback device which immediately sends the byte back. We then measured the time between sending and receiving. We repeated this process 1000 times.

Our results revealed:

- With a baudrate of 115200, the maximal measured time between sending and receiving a byte was 0.283894736842 ms.
- With a baudrate of 19200, the maximal measured time between sending and receiving a byte was 0.689593984962 ms.

Parallel port

The parallel port works by directly applying a current (writing) and measuring if a current is applied (sending) to several pins on the connector. Expyriment is only able to read from Acknowledge, Paper-Out and Selected pins!

1.4.3 Currently known problems and limitations

Here are some current problems and limitations of Expyriment you should be aware of. Where possible, we include suggestions on how to deal with or work around the issue.

MPEG-1 video playback only

At the moment, only MPEG-1 video files can be played back.

This is a limitation of the underlying video system of Pygame. On the long run, we are planning to move to a different Python video package.

No native 3D stimuli

Right now Expyriment only offers static 2D visual stimuli.

While PyOpenGL can be used direclty to create dynamic 3D stimuli, we are planning to add a dedicated 3D stimulus class in the future, to facilitate the creation of 3D stimuli.

No support for multiple monitors

It is not possible to run Expyriment in fullscreen mode on a specific monitor, since the underlying Pygame package is not aware of multiple monitors.

If the additional monitors are set to extend the desktop, then Expyriment will treat everything as one big display (spanned over all monitors).

If you simply want to run an experiment on a different monitor (e.g. an external monitor on a laptop), we suggest to set the additional monitor to clone the primary one.

1.5 Changelog

1.5.1 Version 0.7.0 (2 Mar 2014)

New Features:

- new feature in testsuite: Font viewer
- new extra stimulus: stimuli.extras.RandomDotKinematogram
- new timer and experiment clock to ensure monotonic timing
- Clock: new method (static) monotonic_time (this time should be always used)
- data_preprocessing: new exclusion rule, which allows removing trials depending on their deviation (std) from mean (e.g., 'RT > 1.5*std')
- improvements for OS X in get_system_info()
- proper unicode handling: use unicode strings whenever unicode characters are needed
- files: the character encoding is now written to files and used when opening them
- FreeFonts are now part of the Expyriment distribution to guarantee the same fonts across platforms
- new io class: TouchScreenButtonBox
- new options for control.start(): skip_ready_screen and subject_id to start with predefined subject id
- experiments now also have a global mouse object: experiment.mouse
- new property for io.Mouse: is_visible
- Secure hashes for experiments help to ensure that the correct version is running in the lab. Secure hashes will be displayed at the start and printed in all output files as well as in the command line output.

Fixed:

- experiment clock now with monotonic timing
- bug in extras.CedrusResponseDevice
- several bugs in documentation
- incompatibility with multiprocessing.Pool
- bug in Visual.add_noise()
- bug in io.SerialPort.read_line()
- bugfix: stimuli.shapes can now be used as background stimuli for io.TextInput & io.TextMenu

Changed:

- several Android related changes (have no impact for normal use of Expyriment)
- overlapping methods of stimuli now work on absolute_position by default

1.5.2 Version 0.6.4 (5 Aug 2013)

New Features:

- log levels can be changed while running experiment via the method Experiment.set_logging. Access current via Experiment.loglevel
- Modification the logging of individual objects while runtime. Most classes have now the method set_logging(onoff), to switch on or off the logging.
- design.randomize.rand_element returns a random element from a list
- blocks and trails have the property 'factor_dict', which is a dictionary with with all factors
- experimental Android support

Fixed:

- incorrect background colour for TextInput
- Font in TextScreen
- several fixed in documentation
- switching off logging via "expyriment.control.defaults.event_logging = 0" not now working
- "numpy version bug" in data.preprocessing
- unicode bug for picture, audio and video filenames
- polling of parallel port
- io.TextMenu font was hardcoded

1.5.3 Version 0.6.3 (14 Apr 2013)

New Features:

- misc.geometry contains function to convert between pixel and degrees of visual angle: position2visual_angle & visual_angle2position
- io.TextInput has now a position and can be plotted on a background stimulus
- misc.find_font
- misc.list_fonts

Fixed:

- Initializing experiments in the IDLE shell
- TextInput user_text_font and user_text_bold can now be changed
- bugs in font selection
- API reference tool should now also open when there are whitespaces in Python executable path

Changed:

- renamed TextInput.user_colour –> user_text_colour
- FixCross.cross_size has been renamed to FixCross.size. FixCross.size is now a tuple (int, int) and defines both dimensions (x, y) separately.
- Expyriment is checking also the version of all required packages and dependencies
- all doc string are now in line with the numpy-doc conventions

1.5.4 Version 0.6.2 (12 Dec 2012)

New Features:

- new stimuli.extras.PolygonLine class

Fixed:

- Expyriment could not be imported on Windows 7 64
- misc.geometry.position2coordinate bug
- io.Mouse.self_test bug is fixed

Changed:

- stimuli.Line was rewritten to not depend on PolygonRectangle anymore; the old Line stimulus is now stimuli.extras.PolygonLine

1.5.5 Version 0.6.1 (9 Dec 2012)

Fixed:

- Testsuite wouldn't start anymore
- API reference tool would not start on Windows XP in some cases

1.5.6 Version 0.6.0 (4 Dec 2012)

New Features:

- new stimuli.Circle class
- new stimuli.Ellipse class
- new stimuli.extra.PolygonDot class
- new stimuli.extra.PolygonEllipse class
- new stimuli.extra.PolygonRectangle class
- new method: stimuli.Shape.add_vertices to add multiple vertices at once
- an additional suffix for the main data and event files can be set when creating an Experiment
- Unfilled Shapes by setting a line_width, when creating a shape
- Shape: new property line_width

Fixed:

- stimuli.Shape: several fixes, related to surface creation and xy point calculation
- Logging of unicode text in TextLine stimulus
- stimuli.TextLine and stimuli.TextBox can now also receive a font file as text_font argument
- stimuli.TextLine.copy()
- Bug fixes in self tester of stimuli
- Fixed segmentation fault when repeatedly initalizing and ending an experiment
- Fixed surface size shapes
- Fixed incorrect line_width plotting for scaled shapes

- Copying preloaded stimuli in OpenGL mode
- Bug fixed in io.InputFile.get_line()

Changed:

- io.DataFile: variable "Subject" is now called "subject_id"
- io.Screen.update() should be even more accurate now (about 0.5 milliseconds)
- misc.data_preprocessing: argument and property "experiment_name" in all objects is now called "file_name"
- misc.data_preprocessing.Aggregator can handle files with different suffixes: see __inti__ and reset
- stimuli.Dot: is_center_inside, is_overlapping and is_inside are deprecated now
- stimuli.Dot is deprecated now, use stimuli.Circle instead
- stimuli.Shape: is_point_inside and is_shape_overlapping are deprecated now
- stimuli.Shape.native_scaling does now optionally scale also the line_width
- stimuli.Frame: property line_width is now renmes to frame_line_width. Since line_width is a property the underlying shape and always 0 (filled shapes)
- stimuli.Frame is deprecated now, use stimuli.Rectangle with line_width > 0
- stimuli.Rectangle was rewritten and is not inherited from Shape anymore; the old Rectangle stimulus is now knwon as stimuli.extras.PolygonRectangle

1.5.7 Version 0.5.2 (13 Jun 2012)

New Features:

- data_preprocessing.print_n_trials(variables)
- data_preporcessing.get_variable_data: get data as numpy arrays.
- data_preporcessing.add_variable: add data from numpy.
- read trials from csv file into a block design.block.add_trials_from_csv_file
- block.read_design (counterpart to save_design)
- the main event_file logs now also the standard output and Python errors
- statistic functions are now robust against type violations (like nan_mean)
- design will be automatically saved to event file when experiment starts
- functions to check if IDLE of IPython are running (in control)
- several further new minor features

Fixed:

- Serial Port test can now be quit without quitting the test suite
- FixCross, width vertical line
- Serial Port will be closed when script crashes in IDLE
- Fix for stimuli.extras.VisualMask
- Fix for io.TextInput
- Fixes and adjustments for default logging
- API browser now works on OS X

- API browser fonts on Windows
- several bug fixes in data_preporcessing

1.5.8 Version 0.5.1 (07 Mar 2012)

Fixed:

- Bug in Serial Port test when no input is received
- Bug for get_verrsion() under Python 2.6 under OS X

Changed:

- Text colour for API HTML reference

1.5.9 Version 0.5.0 (06 Mar 2012)

New Features:

- new io class: TextMenu
- new function: expyriment.get_system_info()
- new function in control: get_defaults()
- new method in ButtonBox: check()
- new methods in io.Mouse: wait_event, wait_motion
- new misc modules: geometry and statistics
- new Cedrus Response Devices support in io.extras
- new test suite:
  - new method in control: run_testsuite()
  - the testsuite can write a protocol with all results and information about the system
- folder for settings and extra plugins: $HOME/.expyriment/ or $HOME/~expyriment/
  - if the file post_import.py exist in this folder it will be executed automatically after the import of expyriment
  - extra plugins can be now also included in the folder $HOME/.expyriment/<module_name>/ (or ~expyriment)
- command line interface:
  - see "python -m expyriment.cli -h" for help
- better on- and offline documentation:
  - new function: expyriment.show_documentation()
  - new Api Reference Tool (API browse and search GUI)
- new function in misc.data_preprocessing: write_concatenated_data
- ButtonBox and TriggerInput work now optional with bitwise comparisons
- SerialPort and ParallelPort have a get_available_ports() method
- wait callback functions can now also be registered via the experiment
- some new constants

Changed:

- ButtonBox has been replaced by StreamingButtonBox and EventButtonBox
- the experiment is now THE central structure in the framework. Importantly, start does not require an experiment anymore and starts instead the currently initialized experiment.
- textinput.filter is rename to textinput.ascii_filter
- stimuli.TextBox: Size is now a mandatory parameter
- ending expyriment will now only optionally call sys.exti()
- stimuli.Audio.is_playing() and Audio.wait_end() are replaced by control.get_audiosystem_is_playing() and control.wait_end_audiosystem()
- stimuli.Audio.play() now returns a pygame.mixer.Channel object
- control.run_in_develop_mode() is renamed to control.set_develop_mode()
- no config files will be supported anymore

Fixed:

- the Windows installer will now remove all files from an old installation before installing
- IDLE will not freeze anymore, when a script crashes
- several attributes/properties were did not appear in the API reference
- major bug in keyboard.check()
- (possibly) fixed is_playing() method in Audio
- ordering of serial ports in SerialPort.get_available_ports()
- visual problems when graphics card is set to do flipping with tripple buffer

1.5.10 Version 0.4.0 (22 Nov 2011)

New Features:

- saving and loading designs, new functions in experiment class (save_design and load_design)
- new module: expyriment.misc.data_preprocessing with functions for data handling and a new tool to preprocess and aggregate expyriment data (class Aggregator). Note: Preliminary version. Use with caution.
- new extra stimulus class: visual mask (depends on PIL)
- new extra io class: Webcam (depends in PIL and OpenCV)
- new extra io class: MidiIn
- new extra io class: MidiOut
- the repository and the expyriment source code zip file contain examples
- 'setup.py install' removes old installation
- new function: block.save_trials

Changed:

- Extra modules are now hidden
- skipped function experiment.save_trial_list (please use the new experiment.save_design instead)
- rename property block.all_factors_as_text –> block.factors_as_text

- rename property experiment.trials_as_text –> experiment.design_as_text
- rename experiment.bws_factor_permutation_randomize –> experiment.bws_factor_randomized
- Factor conditions/levels have to be a number (int, float) or a string. No other data types are allowed.

Fixed:

- Bug in testing function of visual extra stimuli
- Bug fix, unbalanced between subject factor permutation for hierarchical designs by subject_ID
- Bug fix, playing short tones (duration<1 sec.)

1.5.11 Version 0.3.3 (19 Oct 2011)

New Features:

- stimuli.Video.wait_frame() stops playback and screen updating after the given frame

Fixed:

- Printing experiments with no block factors defined will work now

1.5.12 Version 0.3.2 (12 Oct 2011)

New Features:

- stimuli.Audio: wait_end(), is_playing

Changed:

- stimuli.Video: present() will now block until the first frame is presented.
- stimuli.Video: play() will not render anything anymore
- stimuli.Tone and stimuli.extras.NoiseTone: duration is now set in ms
- design.Block.get_a_random_trial() is now called get_random_trial()

Fixed:

- Visual stimuli: picture() method works now
- Visual stimuli: copy() method was erroneous
- design.Block.get_summary(): Ordering of trial factors
- design.Block and design.Trial: Factor values can now be dictionaries as well
- stimuli.Tone and stimuli.extras.NoiseTone: Works correctly on Windows now
- design.Block.get_random_trial(): Could crash in some occasions
- Fix underscore at the end of filenames

1.5.13 Version 0.3.1 (8 Sep 2011)

Changed:

- SerialPort: byte_buffer is now input_history
- ParallelPort: byte_buffer removed (just did not make any sense)

- ButtonBox: buffer and has_buffer attributes are gone
- Buttons of Mouse and GamePad now start from 0
- register_wait_function renamed to register_wait_callback_function

Fixed:

- Critical bug on Windows about parsing of expyriment installation folder
- Critical bug in Block.copy() which would destroy Pygame surface
- Mouse.check_button_pressed(): Mismatch in button numbering
- Dot: Fixed calculation for setting polar coordinates
- ParallelPort: Sending data
- MarkerOutput: Duration computation

1.5.14 Version 0.3.0 (31 Aug 2011)

New Features:

- expyriment.control.register_wait_function(): A function registered here will be called in each wait method in the framework at least once
- expyriment.control.run_in_develop_mode(): Automatically sets some defaults (window_mode=True, open_gl=False, fast_quit=True)
- SerialPort: read_line() will wait for and return full lines
- SerialPort: os_buffer_size attribute will affect the warning behaviour of the byte buffer
- ByteBuffer: add_events() can be used to add multiple events at once

Changed:

- defaults, constants as well as initialize(), start(), pause() and end() will no longer be available via expyriment but only via expyriment.control
- SerialPort: Updating the byte_buffer is now faster and warnings are more precise

Fixed:

- GamePad.wait_press(): Can now also check for first button (button 0)

1.5.15 Version 0.2.1 (19 Aug 2011)

New Features:

- MarkerOutput can now send with a specified duration (needed for EEG systems connected via parallel port)
- Advanced trial shuffling

Fixed:

- Critical bug in Trial.copy() which leads to broken surfaces
- Blocking mode for serial port
- Unicode in TextBox? and TextScreen?
- wait_press() in GamePad does now check for more than one button

1.5.16 Version 0.2.0 (26 May 2011)

New Features:

- Overall structure has changed quite a bit. There are now only 5 submodules (control, design, io, stimuli, misc). Things like initialize() and start() are now in control. Constants are now in misc. Each module has its own defaults now.

Changed:

- Adding blocks and trexpyrimentials will always add a copy. There is no option for adding a reference anymore.
- Block and Trial IDs are now relative to where they are added. For instance, two blocks can contain 10 unique trials each, but for both blocks the trial IDs will go from 0 to 9.
- Adding stimuli will always add a reference!
- Stimuli have still an absolute unique ID

Fixed:

- A variety of small bugs have been fixed

1.5.17 Version 0.1.4 (22 May 2011)

Fixed:

- Getting a picture from a stimulus was broken

1.5.18 Version 0.1.3 (12 May 2011)

Fixed:

- Tempfiles of surfaces are now closed after creation (critical on Windows!)

1.5.19 Version 0.1.2 (11 May 2011)

Changed:

- expyriment version now printed on import, not on Experiment creation anymore

Fixed:

- Setup script will not try to check mercurial information by default anymore

1.5.20 Version 0.1.1 (11 May 2011)

New Features:

- Throws a usefull exception on old or integrated Intel graphics cards that do not support OpenGL properly

Fixed:

- Experiment.permute_blocks() will not destroy the surfaces of the stimuli anymore

1.5.21 Version 0.1.0 (10 May 2011)

First public release

Getting Started

1.6 Installation

1.6.1 How to install Expyriment?

The latest releases of Expyriment can be downloaded from GitHub. Note, that Expyriment depends on the following software packages that have to be installed on your system:

- Python 2 (>=2.6),
- Pygame (>=1.9)
- PyOpenGL (>=3.0).

Additional packages, which are optional and only required for some features of Expyriment are PySerial (>=2.5) (to use serial port communication), PyParallel (>=0.2) (to use parallel port communication) and NumPy (>=1.6) (to use data preprocessing). Please be aware that Expyriment plugins (extras) might have additional dependencies.

Importantly, Expyriment relies on 32-bit versions of all these packages!

We provide more detailed platform-specific instructions for installing Expyriment here:

Platform-specific instructions: Windows

Dependencies

If you are using Windows, download the following installers and follow their instructions:

- Python 2
- Pygame
- PyOpenGL

and, if needed:

- PySerial
- PyParallel and giveio
- NumPy

Installing Expyriment

To install the latest version of Expyriment, download "expyriment-0.7.0-win32.exe" from the Release page and execute it.

Notes

Do not start your experiments out of IDLE

If you are using the IDLE editor that comes with the Python installation, be aware that IDLE itself is written in Python. Starting your Expyriment programme out of IDLE (by clicking on "Run" or by pressing F5), might thus lead to improper timing!

We therefore strongly suggest to run Expyriment programmes from the command line when testing participants.

Platform-specific instructions: Linux

Dependencies

If you are in the lucky position of working on a Linux system, installing the required packages can be easily done via your package manager. On Debian-based systems (e.g. Ubuntu) the following command will install everything in one go:

```
sudo apt-get install python python-pygame python-opengl python-serial python-parallel python-numpy
```

Installing Expyriment

You can then install Expyriment with the online installer:

```
wget -P /tmp 'https://raw.github.com/expyriment/expyriment-tools/master/expyriment_online_install_lin
```

Alternatively, you can download "expyriment-0.7.0.zip from the Release page and install as described here.

(For Ubuntu, there is furthermore an Expyriment package available through the following third-party PPA: https://launchpad.net/~smathot/+archive/cogscinl. Please note that we do not provide support for this package.)

Notes

Switch off desktop effects, when running an experiment

Several window managers nowadays come with a compositing engine to produce 3D desktop effects. To get accurate timing of the visual stimulus presentation it is important to switch off desktop effects in your window manager!

Platform-specific instructions: Mac OS X

Dependencies

If you are using OS X, download the following installers and follow their instructions:

- Python 2
- Tcl
- Pygame

and, if needed:

- NumPy

- PyOpenGL and PySerial (has to be installed as described here).

Installing Expyriment

Download "expyriment-0.7.0.zip from the Release page and install as described here.

Notes

Do not start your experiments out of IDLE

If you are using the IDLE editor that comes with the Python installation, be aware that IDLE itself is written in Python. Starting your Expyriment programme out of IDLE (by clicking on "Run" or by pressing F5), might thus lead to improper timing!

We therefore strongly suggest to run Expyriment programmes from the command line when testing participants.

Platform-specific instructions: Android

Introduction

With Python and Pygame being ported to Android (PGS4A), it is in principle possible to use Expyriment on Android devices, however, without OpenGL support. This can be achieved by compiling Python, Pygame and Expyriment into a Java app, using PGS4A. For ease of use, we provide the "Expyriment Android Runtime", an Android application which can be used to directly run experiments on an Andoroid device (with Android > 2.2).

Installing Expyriment

The easiest way to run experiments on Android devices is to use our "Expyriment Android Runtime" appplication. You can only download the current version from our Android download page. In the future it will also be available in the Google Play Store.

Installing Expyriment scripts

Once installed, the application will look for Expyriment scripts (each in its own subdirectory) in a directory called 'expyriment', located at the root level of either storage device under 'mnt' (i.e. the internal or external SD card). Examples of correctly located Expyriment scripts include:

/mnt/sdcard0/expyriment/exp1/exp1.py

/mnt/sdcard0/expyriment/exp2/exp2.py

/mnt/extSdCard/expyriment/exp3/exp3.py

/mnt/extSdCard/expyriment/exp4/exp4.py

Notes

Extra plugins not supported The current version of the "Expyriment Android Runtime" does not support extras plugins.

1.7 Beginner's tutorial to get started with Expyriment

1.7.1 How to get started with Expyriment?

To start programming a new experiment you can take any text editor. Preferably one with syntax highlighting for Python of course. (If you are on Windows you might want to use *IDLE* which is part of the Python installation).

Let's start right away with a very basic example!

Write the following code into an empty text file and save the file as first_experiment.py:

```
import expyriment

exp = expyriment.design.Experiment(name="First Experiment")
expyriment.control.initialize(exp)

expyriment.control.start(exp)

expyriment.control.end()
```

Now run the file by either double clicking on it (if you are on Windows) or by typing the following into a command line:

```
python first_experiment.py
```

The following should happen:

- Expyriment will start up, showing the startup screen and a countdown of 10 seconds
- "Preparing experiment..." will be presented on the screen (very briefly)
- You will be asked to enter the subject number
- "Ready" will be presented on the screen
- After pressing ENTER "Quitting experiment..." will be presented on the screen

Let's see what we just did in more detail:

`import expyriment`

We imported the Expyriment package into Python, such that we can use it there.

`exp = expyriment.design.Experiment(name="First Experiment")`

`expyriment.control.initialize(exp)`

We created a new Experiment object by calling the Experiment class in the submodule design and named it "First Experiment". Immediately after we initialized this experiment to be the active one. This does the following:

- Present the startup screen with the countdown (which is there to ensure that the Python interpreter has enough time to start up properly and will be time accurate afterwards)
- Start an experimental clock (which thereafter will be available as `exp.clock`)
- Create the screen (which thereafter will be available as `exp.screen`)
- Create an event file (which thereafter will be available as `exp.events`)
- Present the "Preparing experiment..." screen

`expyriment.control.start(exp)`

We started running the experiment we just created. This does the following:

- Present a screen to ask for the subject number (which thereafter will be available as `exp.subject`) and wait for the RETURN key to be pressed
- Create a data file (which thereafter will be available as `exp.data`)
- Present the "Ready" screen

```
expyriment.control.end()
```

We finished our experiment, so we quit Expyriment. This will automatically save the data as well as the event file and show the "Ending experiment..." screen.

Okay great, now let's actually do something in this experiment. Let's say we want to present a stimulus. Change the code to look like this:

```
import expyriment

exp = expyriment.design.Experiment(name="First Experiment")
expyriment.control.initialize(exp)

stim = expyriment.stimuli.TextLine(text="Hello World")
stim.preload()

expyriment.control.start(exp)

stim.present()
exp.clock.wait(1000)

expyriment.control.end()
```

If you run the programme now the following should happen:

- Expyriment will start up, showing the startup screen and a countdown of 10 seconds
- "Preparing experiment..." will be presented on the screen (very briefly)
- You will be asked to enter the subject number
- "Ready" will be presented on the screen
- After pressing ENTER, "Hello World" will be presented on the screen for 1000 ms
- "Ending experiment..." will be presented on the screen

Again, let's look into the new things we added in more detail:

```
stim = expyriment.stimuli.TextLine(text="Hello World")
```

We created a text stimulus with the text "Hello World".

```
stim.preload()
```

We preloaded the stimulus into memory (to ensure that this does not happen when presenting it later, since this may take some time).

```
stim.present()
```

We presented the stimulus on the screen.

```
exp.clock.wait(1000)
```

We waited for 1000 ms, while the stimulus is still on the screen (since we did not present something else afterwards).

Let's add some common experimental design structures to get a bit more organized. Modify the code to look like this:

```
import expyriment

exp = expyriment.design.Experiment(name="First Experiment")
expyriment.control.initialize(exp)

block = expyriment.design.Block(name="A name for the block")
trial = expyriment.design.Trial()
stim = expyriment.stimuli.TextLine(text="Hello World")
stim.preload()
trial.add_stimulus(stim)
block.add_trial(trial)
exp.add_block(block)

expyriment.control.start(exp)

stim.present()
exp.clock.wait(1000)

expyriment.control.end()
```

Running this will show you the same as before. This is, because we only made changes in the experimental design, but not in the experiment conduction!

Here is what we added in detail:

`block = expyriment.design.Block("A name for the block")`

We created an experimental block by calling the Block class in the design submodule and gave the block then name "Block One"

`trial = expyriment.design.Trial()`

We created an experimental trial by calling the Trial class in the design submodule.

`trial.add_stimulus(stim)`

We added our stimulus to the trial.

`block.add_trial(trial)`

We added our trial to the block.

`exp.add_block(block)`

We added our block to the experiment.

We now have a nice hierarchical structure:

- The experiment with one block
- The block has one trial
- The trial includes one stimulus

Of course this is only makes sense when more blocks and trials are used. Let's now create two blocks with 2 Trials each. Each of those trials will have exactly one stimulus. Change the code to look like this:

```
import expyriment

exp = expyriment.design.Experiment(name="First Experiment")
expyriment.control.initialize(exp)

block_one = expyriment.design.Block(name="A name for the first block")
trial_one = expyriment.design.Trial()
```

```
stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 1, Trial 1")
stim.preload()
trial_one.add_stimulus(stim)
trial_two = expyriment.design.Trial()
stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 1, Trial 2")
stim.preload()
trial_two.add_stimulus(stim)
block_one.add_trial(trial_one)
block_one.add_trial(trial_two)
exp.add_block(block_one)

block_two = expyriment.design.Block(name="A name for the second block")
trial_one = expyriment.design.Trial()
stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 2, Trial 1")
stim.preload()
trial_one.add_stimulus(stim)
trial_two = expyriment.design.Trial()
stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 2, Trial 2")
stim.preload()
trial_two.add_stimulus(stim)
block_two.add_trial(trial_one)
block_two.add_trial(trial_two)
exp.add_block(block_two)


expyriment.control.start(exp)

for block in exp.blocks:
    for trial in block.trials:
        trial.stimuli[0].present()
        exp.clock.wait(1000)

expyriment.control.end()
```

When running this the following happens:

- Expyriment will start up, showing the startup screen and a countdown of 10 seconds
- "Preparing experiment..." will be presented on the screen
- You will be asked to enter the subject number
- "Ready" will be presented on the screen
- After pressing ENTER, the stimuli are presented in the order: stimuli in trial_one and trial_two of block_one followed by the stimuli in trial_one and trial_two of block_two. All four are presented for 1000 ms
- "Ending experiment..." will be presented on the screen

Let's see what we did exactly:

```
block_one = expyriment.design.Block(name="A name for the first
block")
```

```
trial_one = expyriment.design.Trial()
```

```
sim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 1,
Trial 1")
```

```
stim.preload()
```

```
trial_one.add_stimulus(stim)
```

```
trial_two = expyriment.design.Trial()

stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 1,
Trial 2)

trial_two.add_stimulus(stim)

block_one.add_trial(trial_one)

block_one.add_trial(trial_two)
```

We created a block, two trials and two stimuli. We put one of the stimuli in each of the trials, the trials into the block and the block into the experiment.

```
block_two = expyriment.design.Block(name="A name for the second
block")

trial_one = expyriment.design.Trial()

stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 2,
Trial 1

stim.preload()

trial_one.add_stimulus(stim)

trial_two = expyriment.design.Trial()

stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 2,
Trial 2")

trial_two.add_stimulus(stim)

block_two.add_trial(trial_one)

block_two.add_trial(trial_two)

exp.add_block(block_two)
```

We created another block with again two trials and two stimuli and connected them like the first one.

```
for block in exp.blocks:

    for trial in block.trials:

        trial.stimuli[0].present()

        exp.clock.wait(1000)
```

We loop over all blocks in the experiment (two in our case). For each of the blocks, we loop again over all trials in that block (again two in our case). For each trial we present the first stimulus (because we only added one to each trial). After each stimulus presentation we wait for 1000 ms.

We now want to measure some reaction times after each stimulus presentation. Modify the code to look like this:

```
import expyriment

exp = expyriment.design.Experiment(name="Text Experiment")
expyriment.control.initialize(exp)

block_one = expyriment.design.Block(name="A name for the first block")
trial_one = expyriment.design.Trial()
stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 1, Trial 1")
stim.preload()
trial_one.add_stimulus(stim)
trial_two = expyriment.design.Trial()
```

```
stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 1, Trial 2")
trial_two.add_stimulus(stim)
block_one.add_trial(trial_one)
block_one.add_trial(trial_two)
exp.add_block(block_one)

block_two = expyriment.design.Block(name="A name for the second block")
trial_one = expyriment.design.Trial()
stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 2, Trial 1")
stim.preload()
trial_one.add_stimulus(stim)
trial_two = expyriment.design.Trial()
stim = expyriment.stimuli.TextLine(text="I am a stimulus in Block 2, Trial 2")
trial_two.add_stimulus(stim)
block_two.add_trial(trial_one)
block_two.add_trial(trial_two)
exp.add_block(block_two)

expyriment.control.start(exp)

for block in exp.blocks:
    for trial in block.trials:
        trial.stimuli[0].present()
        key, rt = exp.keyboard.wait([expyriment.misc.constants.K_LEFT,
                                     expyriment.misc.constants.K_RIGHT])
        exp.data.add([block.name, trial.id, key, rt])

expyriment.control.end()
```

When you run this code, the following happens:

- Expyriment will start up, showing the startup screen and a countdown of 10 seconds
- "Preparing experiment..." will be presented on the screen
- You will be asked to enter the subject numtrial_one.add_stimulusber
- "Ready" will be presented on the screen
- After pressing ENTER the stimuli are presented in the order: stimuli in trial_one and trial_two of block_one followed by the stimuli in trial_one and trial_two of block_two. After each presentation the programme waits for the LEFT or RIGHT arrow key to be pressed until it proceeds.
- "Ending experiment..." will be presented on the screen

Let's see why this is:

```
key, rt = exp.keyboard.wait([expyriment.misc.constants.K_LEFT,
    expyriment.misc.constants.K_RIGHT])
```

We waited for a keyboards response which is either the LEFT or the RIGHT arrow key (as defined by a list with those two keys as elements). This function returns the key that was pressed as well as the reaction time.

```
exp.data.add([block.name, trial.id, key, rt])
```

We added the name of the block, the id of the trial, the pressed key and the reaction time to the data file (by adding a list with those two as elements). The id of a trial is automatically set when the trial is added to a block.

Now have a look at the "data" and "events" directories (in the same directory where your first_example.py is located). The "data" directory contains data log files, named according to the experiment name, the subject number and a timestamp. The file ending is .xpd. The event directory contains event log files with the ending .xpe. Open the latest data file to see the data we just logged. Notice that the first rows are a header with some information about the file. However, it would be nice to also have the variable names of what is logged in there. To do this, add the following lines above where you start the experiment:

```
exp.data_variable_names = ["Block", "Trial", "Key", "RT"]
```

What this does is to add the given names into the data file header, separated by commas.

The last thing to mention in this brief tutorial are the default settings. Each module (control, design, io, stimuli, misc) has its own defaults. Changing these defaults will only have an effect before the corresponding object is created. Thus, a safe place is right at the beginning of your file, just above creating an experiment. Note also that it is handy to overwrite other default settings in the beginning as well, to have one central place for important settings. It might also shorten calls to the classes later on. For instance, the `experiment_name` can also be set as `mysettings.experiment_name` and the `name="Test Experiment"` parameter is not needed anymore. However, using explicit parameters in the call to classes will overwrite any previous default settings! One of the most common things to do, while developing is to change the default presentation mode from fullscreen to a window:

```
expyriment.control.window_mode = True
```

```
expyriment.control.window_size = (800,600)
```

Also, when using older machines with very old video cards, you might want to run in fullscreen, but without using OpenGL:

```
expyriment.control.open_gl = False
```

That's it so far. We are at the end of the getting started tutorial. As a summary, have a look at the following code, which again show the overall structure of an Expyriment file with the 3 main parts:

```
import expyriment

# Any global settings go here

exp = expyriment.control.initialize()

# Create design (blocks and trials)
# Create stimuli (and put them into trials)
# Create input/output devices (like button boxes etc.)

expyriment.control.start(exp)

# Experiment conduction
# Loop over blocks and trials, present stimuli and get user input

expyriment.control.end()
```

1.8 Example Experiments

Here you can find some code examples to see Expyriment in action. All examples are fully working experiments.

1.8.1 Simon task

An experiment to asses a spatial stimulus-response compatibility effect (see wikipedia).

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
A simple behavioural task to asses a Simon effect.

See also:
http://en.wikipedia.org/wiki/Simon_effect

"""

from expyriment import design, control, stimuli, io, misc


# Create and initialize an Experiment
exp = design.Experiment("Simon Task")
control.initialize(exp)

# Define and preload standard stimuli
fixcross = stimuli.FixCross()
fixcross.preload()

# Create IO
#response_device = io.EventButtonBox(io.SerialPort("/dev/ttyS1"))
response_device = exp.keyboard

# Create design
for task in ["left key for green", "left key for red"]:
    b = design.Block()
    b.set_factor("Response", task)
    for where in [["left", -300], ["right", 300]]:
        for what in [["red", misc.constants.C_RED],
                     ["green", misc.constants.C_GREEN]]:
            t = design.Trial()
            t.set_factor("Position", where[0])
            t.set_factor("Colour", what[0])
            s = stimuli.Rectangle([50, 50], position=[where[1], 0],
                                  colour=what[1])
            t.add_stimulus(s)
            b.add_trial(t, copies=20)
    b.shuffle_trials()
    exp.add_block(b)
exp.add_bws_factor("ResponseMapping", [1, 2])
exp.data_variable_names = ["Position", "Button", "RT"]

# Start Experiment
control.start()
exp.permute_blocks(misc.constants.P_BALANCED_LATIN_SQUARE)
for block in exp.blocks:
    stimuli.TextScreen("Instructions", block.get_factor("Response")).present()
    response_device.wait()
    for trial in block.trials:
        fixcross.present()
        exp.clock.wait(1000 - trial.stimuli[0].preload())
        trial.stimuli[0].present()
        button, rt = response_device.wait()
        exp.data.add([trial.get_factor("Position"), button, rt])
```

```
# End Experiment
control.end()
```

1.8.2 Word fragment completion task

Task as used for instance in Weldon, 1991. The script read in a stimulus list file (`demo stimulus list`).

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Word fragment completion task as used in the study of Weldon (1991).

Stimulus list: "word_fragment_completion_stimuluslist.csv"!

Weldon, M. S. (1991). Mechanisms underlying priming on
perceptual tests. Journal of Experimental Psychology: Learning, Memory, and
Cognition, 17, 526-541.

"""

import csv
from expyriment import design, control, stimuli, io, misc

control.set_develop_mode(True)

#### read in wordlist file and make design
exp = design.Experiment("word fragment completion test")
block = design.Block()
with open("word_fragment_completion_stimuluslist.csv", "rb") as f:
    reader = csv.reader(f)
    for row in reader:
        trial = design.Trial()
        trial.set_factor("word", row[0].strip())
        trial.set_factor("fragment", row[1].strip())
        block.add_trial(trial)
block.shuffle_trials()
exp.add_block(block)
exp.add_data_variable_names(["word", "fragment", "RT", "RT2", "answer"])

control.initialize(exp)

#prepare some stimuli
fixcross = stimuli.FixCross(line_width=1)
fixcross.preload()
blank = stimuli.BlankScreen()
blank.preload()
txt_input = io.TextInput("")
control.start(exp)

#run experiment
for trial in exp.blocks[0].trials:
    #present blank inter-trial-screen and prepare stimulus
    blank.present()
    fragment = ""
    for c in trial.get_factor("fragment").upper():
        fragment += c + " "
```

```
    target = stimuli.TextLine(fragment.strip())
    target.preload()
    exp.clock.wait(1000)
    #present fixcross
    fixcross.present()
    exp.clock.wait(500)
    #present target
    target.present()
    key, rt = exp.keyboard.wait(misc.constants.K_SPACE)
    #ask response
    exp.clock.reset_stopwatch()
    answer = txt_input.get()
    rt2 = exp.clock.stopwatch_time
    #process answer and save data
    blank.present()
    answer = answer.strip()
    exp.data.add([trial.get_factor("word"), trial.get_factor("fragment"),
                  rt, rt2, answer])
    target.unload()

control.end()
```

1.8.3 Number classification task

A full experiment to access SNARC and SNARC-like effects in a number and a letter classification task (e.g., Gevers, Reynvoer, & Fias (2003)) with two response mappings, error feedback and between-subject factors.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
A parity judgment task to assess the SNARC effect.

See e.g.:
Gevers, W., Reynvoet, B., & Fias, W. (2003). The mental representation of
ordinal sequences is spatially organized. Cognition, 87(3), B87-95.

"""

from expyriment import design, control, stimuli
from expyriment.misc import constants


control.set_develop_mode(False)

########### DESIGN ###################
exp = design.Experiment(name="SNARC")

# Design: 2 response mappings x 8 stimuli x 10 repetitions
for response_mapping in ["left_odd", "right_odd"]:
    block = design.Block()
    block.set_factor("mapping", response_mapping)
    #add trials to block
    for digit in [1, 2, 3, 4, 6, 7, 8, 9]:
        trial = design.Trial()
        trial.set_factor("digit", digit)
        block.add_trial(trial, copies=10)
```

```
    block.shuffle_trials()
    exp.add_block(block)

exp.add_experiment_info("This a just a SNARC experiment.")
#add between subject factors
exp.add_bws_factor('mapping_order', ['left_odd_first', 'right_odd_first'])
#prepare data output
exp.data_variable_names = ["block", "mapping", "trial", "digit", "ISI",
                           "btn", "RT", "error"]

#set further variables
t_fixcross = 500
min_max_ISI = [200, 750] # [min, max] inter_stimulus interval
ITI = 1000
t_error_screen = 1000
no_training_trials = 10

######### INITIALIZE ##############
control.initialize(exp)

# Prepare and preload some stimuli
blankscreen = stimuli.BlankScreen()
blankscreen.preload()
fixcross = stimuli.FixCross()
fixcross.preload()
error_beep = stimuli.Tone(duration=200, frequency=2000)
error_beep.preload()

#define a trial
def run_trial(cnt, trial):
    # present Fixation cross and prepare trial in the meantime
    fixcross.present()
    exp.clock.reset_stopwatch()
    ISI = design.randomize.rand_int(min_max_ISI[0], min_max_ISI[1])
    digit = trial.get_factor("digit")
    target = stimuli.TextLine(text=str(digit), text_size=60)
    target.preload()
    exp.clock.wait(t_fixcross - exp.clock.stopwatch_time)
    #present blankscreen for a random interval
    blankscreen.present()
    exp.clock.wait(ISI)
    # Present target & record button response
    target.present()
    btn, rt = exp.keyboard.wait([constants.K_LEFT, constants.K_RIGHT])
    #Error feedback if required
    if block.get_factor("mapping") == "left_odd":
        error = (digit % 2 == 0 and btn == constants.K_LEFT) or \
                (digit % 2 == 1 and btn == constants.K_RIGHT)
    else:
        error = (digit % 2 == 1 and btn == constants.K_LEFT) or \
                (digit % 2 == 0 and btn == constants.K_RIGHT)
    if error:
        error_beep.present()
    #write data and clean up while inter-trial-interval
    blankscreen.present()
    exp.clock.reset_stopwatch()
    exp.data.add([block.id, block.get_factor("mapping"),
                  cnt, target.text, ISI,
```

```
                btn, rt, int(error)])
    exp.data.save()
    target.unload()
    exp.clock.wait(ITI - exp.clock.stopwatch_time)


######### START ##############
control.start(exp)

# permute block order across subjects
if exp.get_permuted_bws_factor_condition('mapping_order') == "right_odd_first":
    exp.swap_blocks(0, 1)

# Run the actual experiment
for block in exp.blocks:
    # Show instruction screen
    if block.get_factor("mapping") == "left_odd":
        instruction = "Press LEFT arrow key for ODD\n" + \
                      "and RIGHT arrow key for EVEN numbers."
    else:
        instruction = "Press RIGHT arrow key for ODD\n" + \
                      "and LEFT arrow key for EVEN numbers."
    stimuli.TextScreen("Indicate the parity of the numbers", instruction +
                       "\n\nPress space bar to start training.").present()
    exp.keyboard.wait(constants.K_SPACE)
    #training trials
    for cnt in range(0, no_training_trials):
        trial = block.get_random_trial()
        run_trial(-1 * (1 + cnt), trial) #training trails has negative trial numbers
    # Show instruction screen
    stimuli.TextScreen("Attention!", instruction +
                       "\n\nThe experimental block starts now.").present()
    exp.keyboard.wait(constants.K_SPACE)
    # experimental trials
    for cnt, trial in enumerate(block.trials):
        run_trial(cnt, trial)


####### END EXPERIMENT ########
control.end(goodbye_text="Thank you very much for participating in our experiment",
            goodbye_delay=5000)
```

1.8.4 Line bisection task

Example of a line bisection task that is optimized for the use of touchscreens and the Expyriment Android Runtime.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
A line bisection task.

This example is appropriate to illustrates the use of the Android runtime environment for Exypriment

"""

from expyriment import control, stimuli, io, design, misc
```

```
# settings
design.defaults.experiment_background_colour = misc.constants.C_GREY
design.defaults.experiment_foreground_colour = misc.constants.C_BLACK
line_length = 200

def line_bisection_task(line_length, position):
    # make button
    button = stimuli.Rectangle(size=(40,20),
                position=(exp.screen.size[0]/2-25, 15-exp.screen.size[1]/2))
    button_text = stimuli.TextLine(text="ok", position=button.position,
                                text_colour=misc.constants.C_WHITE)
    mark_position = None
    while True:
        canvas = stimuli.BlankScreen()
        line = stimuli.Rectangle(size=(line_length,3), position=position,
                    colour=misc.constants.C_BLACK)
        line.plot(canvas)
        if mark_position is not None:
            # plot button and mark line on canvas
            button.plot(canvas)
            button_text.plot(canvas)
            markline = stimuli.Rectangle(size=(1,25),
                        position=(mark_position, line.position[1]),
                        colour=misc.constants.C_RED)
            markline.plot(canvas)
        # present stimulus
        canvas.present()
        # wait for mouse or touch screen response
        _id, pos, _rt = exp.mouse.wait_press()
        # process clicked position position
        if abs(pos[1]-line.position[1])<=50 and\
                    abs(pos[0]-line.position[0])<=line_length/2:
            mark_position = pos[0]
        else:
            if button.overlapping_with_position(pos): # is button pressed
                return mark_position - line.position[0]

### init ###
exp = control.initialize()

# create touch button box
buttonA = stimuli.Rectangle(size=(80, 40), position=(-60, 0))
buttonB = stimuli.Rectangle(size=(80, 40), position=(60, 0))
textA = stimuli.TextLine(text="quit", position=buttonA.position,
            text_colour=misc.constants.C_WHITE)
textB = stimuli.TextLine(text="next", position=buttonB.position,
            text_colour=misc.constants.C_WHITE)
touchButtonBox = io.TouchScreenButtonBox(button_fields=[buttonA, buttonB],
                stimuli=[textA, textB])
### start ###
control.start(exp)
exp.mouse.show_cursor()

# trial loop
while True:
    # find random position
    rx, ry = ((exp.screen.size[0]-line_length)/2, (exp.screen.size[1]-50)/2)
    pos = [design.randomize.rand_int(-rx, rx), design.randomize.rand_int(-ry, ry)]
```

```
    # present task
    judgment = line_bisection_task(line_length, position=pos)
    # save data
    exp.data.add(pos +[judgment])
    # ask for new trail
    touchButtonBox.show()
    btn, _rt = touchButtonBox.wait()
    if btn==buttonA:
        break

## end##
control.end()
```

1.8.5 Really short example

Expyriment is efficient!. See here a very short example of an functioning experiment in less than 20 lines of pure code.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
A very short example experiment in 16 lines of pure code.

Participants have to indicate the parity of digits by pressing
the left arrow key for odd and the right arrow key for even numbers.

"""

from expyriment import control, stimuli, design, misc

digit_list = [1, 2, 3, 4, 6, 7, 8, 9]*12
design.randomize.shuffle_list(digit_list)

exp = control.initialize()
exp.data_variable_names = ["digit", "btn", "rt", "error"]

control.start(exp)

for digit in digit_list:
    target = stimuli.TextLine(text=str(digit), text_size=80)
    exp.clock.wait(500 - stimuli.FixCross().present() - target.preload())
    target.present()
    button, rt = exp.keyboard.wait([misc.constants.K_LEFT, misc.constants.K_RIGHT])
    error = (button == misc.constants.K_LEFT) == digit%2
    if error: stimuli.Tone(duration=200, frequency=2000).play()
    exp.data.add([digit, button, rt, int(error)])
    exp.clock.wait(1000 - stimuli.BlankScreen().present() - target.unload())

control.end(goodbye_text="Thank you very much...", goodbye_delay=2000)
```

1.8.6 Data preprocessing

Preprocessing the data of the SNARC experiment for further statistical analysis.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Example analysis script for snarc_experiment.py

The current script produces two files for different analysis of the SNARC
effect (ANOVA vs. slopes analysis) using mean and median RTs

"""

from expyriment.misc import data_preprocessing, constants


agg = data_preprocessing.Aggregator(data_folder="./data/",
                                    file_name="snarc_experiment")

agg.set_subject_variables(["mapping_order"])
agg.set_computed_variables(["parity = digit % 2", #0:odd, 1:even
                        "size = digit > 5", #0:small, 1:large
                        "space = btn == {0}".format(constants.K_RIGHT) #0:left, 1:right
                        ])
# RTs: space x size
agg.set_exclusions(["RT > 1000", "RT < 200", "error == 1", "trial<0"])
agg.set_dependent_variables(["mean(RT)"])
agg.set_independent_variables(["size", "space"])
print agg
agg.aggregate(output_file="rt_size_space.csv")
# RTs: slopes analysis
agg.set_independent_variables(["digit"])
agg.aggregate(output_file="rt_digits.csv")
```

E