VEPFlasher

Generated by Doxygen 1.6.3

Sun Jun 13 12:39:38 2010

Contents

1 VEP Flasher

1.1 Introduction

1.1.1 What is VEP Flasher

VEP Flasher is a program that displays stimulation grids (e.g. grids of flashing lights) for raising visually evoked potentials. Grid definition is made through script files (see Script file format).

Currently there is no user interaction.

1.1.2 Features

- Simple yet extendable architecture
- Highly configurable through simple scripting language
- Input sources: constant, sine wave, square wave, data from text file. More can be added
- Renderers: coloured rectangles with text
- Frame rate scaling

1.2 Usage

1.2.1 Invocation

From command line:

```
vepflasher <path to script file> [0]
```

The optional second argument disables fullscreen.

1.2.2 Commands

Keyboard commands recognised:

- S: start/pause video
- Q: quit

1.3 Building And Installation

1.3.1 Minimum Requirements

VEP Flasher works on almost any platform. Currently tested: MS Windows and Linux.

VEP Flasher needs a video card driver which supports vertical synchronisation (virtually any video card with 3D hardware acceleration and OpenGL support will go, except Intel Mobile)

1.3.2 Compilation Requirements

- A POSIX shell
- A C++ compiler compatible with GCC
- Development libraries: SDL v. 1.2, OpenGL, FreeType v. 2

Open the shell and type these commands:

```
./configure
```

```
make
```

```
make install
```

The last command needs administrator privileges.

1.4 Credits

1.4.1 Authors

- Giuseppe Broccio
- Davide Castellone

2 Script file format

2.1 Syntax

Each line is an instruction. Lines beginning with '@' are special commands. Lines beginning with '#' are comments.

To initialise the screen (it must be the first instruction):

@init *<width> <height> [depth]*

To create an object:

@new *<classname> <objectname>*

To set the ending frame:

@end *<frame number>*

To call a method (see VEP::ScriptObject::doCommand):

*<objectname> <command> <parameters>**

To divide frame rate by an integer:

@rate *<integer>*

To set a global font (note: `path-to-file` is relative to the script's path):

@font *<path-to-file> <size>*

2.2 Classes and methods

2.2.1 ConstantInput

A `ConstantInput` represents a constant.

Methods

- **`value`** *`float`*
  Set the constant value. Accepts a float argument.

2.2.2 SineWaveInput

A `SineWaveInput` represents a sine wave.

Methods

- **`frequency`** *`float`*
  Frequency in frames
- **`phase`** *`float`*
  Initial phase in radians
- **`box`** *`float float`*
  Set minimum and maximum (default: 0 and 1)

2.2.3 SquareWaveInput

A `SquareWaveInput` represents a square wave.

Methods

- **`times`** *`float float`*
  On-time and off-time (in frames, default: 1 1)
- **`delay`** *`float`*
  An initial offset (in frames)
- **`box`** *`float float`*
  Set minimum and maximum (default: 0 1)

2.2.4 TextFileInput

A `TextFileInput` reads data from a text file.

Methods

- **`file`** *`string`*
  Name of the file to open
- **`column`** *`integer`*
  Zero-based index of the column to use (default is 0)
- **`separator`** *`string`*
  Column separator (default is space)

- **delay** *integer*
  Initial time (row) offset

Format

File format is analogous to comma or space separated valued. Each line represents a frame. The user can choose which column to use (default is to use the first). Frames are wrapped, i.e. when the input sequence ends, values restart from the beginning.
Comment lines begin with a '#'.

2.2.5 SignalRenderer

Abstract class. Cannot be created. It represents an object which is displayed on the screen. It can be attached to an input source to show a value that can change in time.

Methods

- **attach** *string*
  Attach to the input source passed as first and only parameter

2.2.6 RectRenderer

A `RectRenderer` is a coloured rectangle with optional text, that displays a signal by changing its colour (usually the brightness).

Methods

- **attach** *string*
  See VEP::SignalRenderer
- **size** *x y width height*
  Set size and position
- **text** *text*
  Print a text inside the rectangle
- **colors** *r1 g1 b1 r2 g2 b2*
  Fill color for the rectangle (*r*, *g* and *b* are in range [0, 1]). The first triplet is the value used when input=0, the second is used when input=1. Intermediate values are iterpolated.
- **fontcolor** *r g b*
  Global: Set font color (default is black)
- **@colors** *r1 g1 b1 r2 g2 b2*
  Global: Set low-input and high-input colours for all *RectRenderer's*
- **@fontcolor** *r g b*
  Global: Set a font colour for all *RectRenderer's*.

Note

Global parameters are effective only for objects created after they are set and the object on which they are called. If both global and local parameters are set for a `RectRenderer`, only the local one is taken into account.

2.3 Script Example

```
#This is a test script for VEP Flasher
#Comments begin with a hash
#Each line is an instruction
#Special instruction begin with '@', all other instructions are method calls
    #Indentation is recognised

#Initialise the screen
@init 1024 768
@end 140

#Typical usage:
#1. create and configure an input source
@new SineWaveInput si
si frequency 10
si box 0 1

#2. Create and configure a signal renderer
@new RectRenderer rect
rect size 100 100 20 20

#3. attach an input source to a signal renderer
rect attach si

#An input source can be attached to several renderers!
@new RectRenderer rect2
rect2 attach si
rect2 size 100 150 20 20
@new RectRenderer rect3
rect3 size 150 100 20 20
rect3 attach si

#Test reading data from a text file (columns start from 0)
@new TextFileInput ti
ti file testdata.txt
ti column 2
@new RectRenderer rti
rti attach ti

#Another source
@new SquareWaveInput sqi
sqi times 3 10
@new RectRenderer rect4
rect4 attach sqi
rect4 size 150 150 20 20

#ALL POSSIBLE ERRORS FOLLOW

#The following line should raise an error: @init cannot be called twice
@init 640 480 32

#A non-existent object
abcd value 10

#A non-existent class
@new abcd i

#Non-terminated instructions
@new
@end
@new SineWaveInput
si
si frequency
```

3 Class Diagram

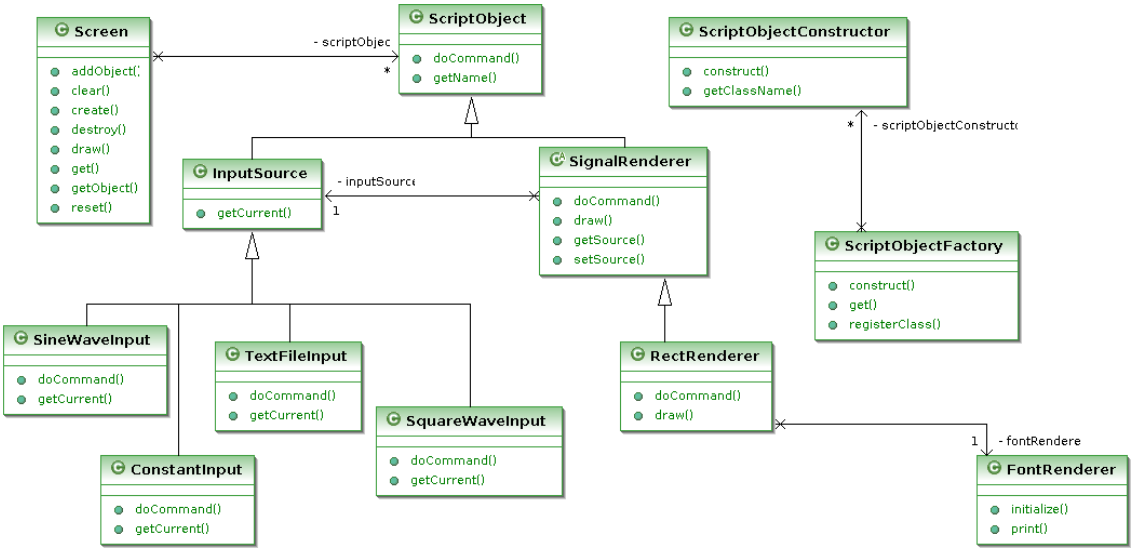

Figure 1: Complete class diagram

4 Class Documentation

4.1 ConstantInput Class Reference

Class representing a constant input.

`#include <ConstantInput.h>`

Public Member Functions

- virtual double getCurrent ()

  Return the current value. It usually depends on the current frame number.

- virtual string doCommand (const string &command, const vector< string > &args)

  Execute a command.

4.1.1 Detailed Description

Class representing a constant input.

4.1.2 Member Function Documentation

4.1.2.1 string doCommand (const string & *command*, const vector< string > & *args*) [virtual]

Execute a command.

Recognised commands:

- **value** *float*
  Set the constant value. Accepts a float argument.

Reimplemented from InputSource.

4.2 FontRenderer Class Reference

This class renders characters on the screen.

`#include <FontRenderer.h>`

Public Member Functions

- ~FontRenderer ()
  Deinitialises object.
- void print (float x, float y, string fmt,...)
  Print text on the screen.
- void initialize (const char *fname, unsigned int h)
  Initialise local font data.

4.2.1 Detailed Description

This class renders characters on the screen. Can render multiline strings with `\n` when calling method FontRenderer::print(). Color of text is set externally by GL.

4.2.2 Member Function Documentation

4.2.2.1 void initialize (const char * *fname*, unsigned int *h*)

Initialise local font data.

Parameters

h Font height
fname Path to TTF file

4.2.2.2 void print (float *x*, float *y*, string *fmt*, ...)

Print text on the screen.

This function will print out text at window coordinates (x,y), using the font initialised with initialize(). The current modelview matrix will also be applied to the text.

Parameters

x Abscissa value

y Ordinate value

fmt Text to print

4.3 InputSource Class Reference

Class representing an input source, a value that changes over time.

`#include <InputSource.h>`

Public Member Functions

- virtual double getCurrent ()=0

  Return the current value. It usually depends on the current frame number.

- virtual string doCommand (const string &command, const vector< string > &args)

  Execute a command.

4.3.1 Detailed Description

Class representing an input source, a value that changes over time. It should be attached to a SignalRenderer.

4.3.2 Member Function Documentation

4.3.2.1 string doCommand (const string & *command*, const vector< string > & *args*) [virtual]

Execute a command.

Recognised commands: none

Reimplemented from ScriptObject.

Reimplemented in ConstantInput, SineWaveInput, SquareWaveInput, and TextFileInput.

4.4 RectRenderer Class Reference

This class renders a rectangle on the screen.

`#include <RectRenderer.h>`

Public Member Functions

- virtual void draw ()

  Virtual function called by Screen on each frame to update the signal.

- virtual string doCommand (const string &command, const vector< string > &args)

  Execute a command.

4.4.1 Detailed Description

This class renders a rectangle on the screen. Brightness depends on the InputSource attached to this object. 0 is interpreted as black, 1 as white.

4.4.2 Member Function Documentation

4.4.2.1 string doCommand (const string & *command*, const vector< string > & *args*) [virtual]

Execute a command.

Recognised commands:

- **attach** *string*

  See VEP::SignalRenderer

- **size** *x y width height*

  Set size and position

- **text** *text*

  Print a text inside the rectangle

- **colors** *r1 g1 b1 r2 g2 b2*

  Fill color for the rectangle (*r*, *g* and *b* are in range [0, 1]). The first triplet is the value used when input=0, the second is used when input=1. Intermediate values are iterpolated.

- **fontcolor** *r g b*

  Global: Set font color (default is black)

- **@colors** *r1 g1 b1 r2 g2 b2*

  Global: Set low-input and high-input colours for all *RectRenderer's*

- **@fontcolor** *r g b*

  Global: Set a font colour for all *RectRenderer's*.

Note

Global parameters are effective only for objects created after they are set. If both global and local parameters are set for a `RectRenderer`, only the local one is taken into account.

Returns

An error message (if any)

Reimplemented from SignalRenderer.

4.5 Screen Class Reference

Class representing the screen.

`#include <Screen.h>`

Classes

- struct RGB
  Struct to hold a colour in float RGB format.

Public Member Functions

Various getters and setters

- int getWidth () const
  Get the screen width.
- int getHeight () const
  Get the screen height.
- int getDepth () const
  Get the colour depth.
- int getFrameNumber ()
  Get the current frame number.
- void setFrameNumber (int frame)
  Changes the current frame number.
- void setRateScaling (int scaling)
  Sets the current frame rate downscaling factor See ScriptLoader::load.
- int getRateScaling () const
  Gets the current frame rate downscaling factor.
- void setFontSize (unsigned x)
  Set the font size.
- unsigned getFontSize () const
  Get the font size.
- void setFontPath (const string &path)
  Set TrueType font file path, relative to the current directory.
- string getFontPath () const
  Get TrueType font file path.
- const RGB & getBackgroundColor ()
  Get the current background colour.
- void setBackgroundColor (float r, float g, float b)

Set the background colour.

- int getEndingFrame ()
  Get the ending frame number.
- void setEndingFrame (int endingFrame)
  Set the ending frame number.

Objects

- ScriptObject * getObject (const string &name)
  Get an object by its name.
- void addObject (ScriptObject &object)
  Add an object.
- void reset ()
  Clears the screen and empties the input sources list.

Drawing

- void draw ()
  Draw the next frame.
- void clear ()
  Clears the screen.

Static Public Member Functions

Initialisation and deinitialisation

- static Screen * get ()
  Gets the global object.
- static void create (int width, int height, int depth) throw (ScreenSetupException)
  Creates the global Screen.
- static void destroy ()
  Deinitialises the global Screen object.
- static void setFullScreen (bool b)
  Activates fullscreen mode.

4.5.1 Detailed Description

Class representing the screen.

- It contains its objects and deletes them when it is deinitialised.
- It is a singleton, and it must explicitly be created with create()

- It cannot be created twice.
- If the Screen object is not destroyed explicitly, it is deleted automatically when the program exits (**if `main()` returns and the program does not call `exit()` or `abort()`**).

4.5.2 Member Function Documentation

4.5.2.1 void addObject (ScriptObject & *object*)

Add an object.

Note

Ownership of `object` is taken.

4.5.2.2 static void create (int *width*, int *height*, int *depth*) throw (ScreenSetupException) `[inline, static]`

Creates the global Screen.

Exceptions

ScreenSetupException if an error occurs

4.5.2.3 static void destroy () `[inline, static]`

Deinitialises the global Screen object.

It is needed in order to revert the screen resolution when in fullscreen mode.

4.5.2.4 void draw ()

Draw the next frame.

This is done by calling the SignalRenderer::draw() method on all contained object of type SignalRenderer

4.5.2.5 static Screen* get () `[inline, static]`

Gets the global object.

Returns

The global object, if it has already been created. Otherwise, return `NULL`.

4.5.2.6 int getFrameNumber () [inline]

Get the current frame number.

For synchronisation purposes. Input sources should use this function when InputSource::getCurrent is called.

4.5.2.7 void reset ()

Clears the screen and empties the input sources list.

Frame number is reset to 0. Useful when repeating a test.

4.5.2.8 void setBackgroundColor (float *r*, float *g*, float *b*) [inline]

Set the background colour.

Parameters

r, g, b values in the range [0,1]

4.5.2.9 void setEndingFrame (int *endingFrame*) [inline]

Set the ending frame number.

Note

Value is not used inside this class

4.5.2.10 static void setFullScreen (bool *b*) [inline, static]

Activates fullscreen mode.

Should be called only in tests. Default is `true` and should remain `true`, in order for the vertical synchronisation to work.

4.6 RGB Struct Reference

Struct to hold a colour in float RGB format.

`#include <Screen.h>`

4.6.1 Detailed Description

Struct to hold a colour in float RGB format.

4.7 ScreenSetupException Class Reference

Exception thrown when the screen fails to be initialised.

`#include <Screen.h>`

Public Member Functions

- const char * what () const throw ()
  Get a description for the exception.

4.7.1 Detailed Description

Exception thrown when the screen fails to be initialised.

4.7.2 Member Function Documentation

4.7.2.1 const char* what () const throw () [inline]

Get a description for the exception.

Note

This function is not thread-safe!

4.8 ScriptLoader Class Reference

Class that loads script files.

`#include <ScriptLoader.h>`

Static Public Member Functions

- static void load () throw (std::exception)
  Load a script.
- static void setFileName (const string &fileName)
  Sets the file name.
- static string getFileName ()
  Get the file name.

4.8.1 Detailed Description

Class that loads script files. A script file is a file with an instruction per line, that specifies what is displayed on the screen and various global options.

See also

Script file format

Note

All the function in this class are static. It has been made a class (instead of being just a module) in order to keep the design clear and strictly object-oriented.

4.8.2 Member Function Documentation

4.8.2.1 static void setFileName (const string & *fileName*) [inline, static]

Sets the file name.

Call it before load(). Useful for repeating tests.

4.9 ScriptLoaderException Class Reference

Exception thrown when loading a script file fails irrecoverably.

`#include <ScriptLoader.h>`

Public Member Functions

- const char * what () const throw ()
  Get the message (not thread-safe).

4.9.1 Detailed Description

Exception thrown when loading a script file fails irrecoverably. However, non-fatal errors are printed but ignored.

4.10 ScriptObject Class Reference

Class representing an object that is created in a configuration script.

`#include <ScriptObject.h>`

Public Member Functions

- ScriptObject (const string &name)
  Creator. Requires a name.
- virtual ~ScriptObject ()
  Virtual destructor.
- string getName () const
  Get the name of the object.

- virtual string doCommand (const string &command, const vector< string > &args)

  Execute an operation.

4.10.1 Detailed Description

Class representing an object that is created in a configuration script. A ScriptObject can be an input source, a signal renderer or an object drawn on the screen.

Every object has a name, given at creation time; it has operations, depending on its type. These are implemented all by the doCommand() function.

4.10.2 Member Function Documentation

4.10.2.1 string doCommand (const string & *command*, const vector< string > & *args*) [virtual]

Execute an operation.

Operations are inherited. This means that if a class does not find the operation specified as a parameter, the call **must** be propagated to the superclass.

Note

The execution always continues if there are no fatal errors.

Returns

An error message (if any), otherwise an empty string.

Exceptions

Any descendant of std::exception can be thrown in case of an unrecoverable error.

Reimplemented in ConstantInput, InputSource, SineWaveInput, SquareWaveInput, TextFileInput, RectRenderer, and SignalRenderer.

4.11 ScriptObjectConstructor Class Reference

Small class that associates a constructor with the class name.

`#include <ScriptObjectConstructor.h>`

Public Member Functions

- virtual ScriptObject * construct (const string &name)=0

  Construct an object.

- string getClassName () const

  Get the name of the class.

- ScriptObjectConstructor (const string &className)

  Create a constructor with a given name.

4.11.1 Detailed Description

Small class that associates a constructor with the class name. It is used to implement a *factory* paradigm. Each class derived from ScriptObject, in order to be usable, **must** define a class derived from ScriptObject-Constructor, which **must** register itself to the ScriptObjectFactory at initialisation time.

Note

Destructor is not virtual! Your subclass can just override the construct method.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 ScriptObjectConstructor (const string & *className*) `[inline]`

Create a constructor with a given name.

Parameters

className The class of the object returned by construct()

4.11.3 Member Function Documentation

4.11.3.1 virtual ScriptObject* construct (const string & *name*) `[pure virtual]`

Construct an object.

Virtual function. It **must** always be overridden.

Parameters

name the name of the object created

Returns

an object whose type is specified in m_className

4.11.3.2 string getClassName () const `[inline]`

Get the name of the class.

Returns

The name of the class of the objects returned by construct()

4.12 ScriptObjectFactory Class Reference

Class implementing the abilty to create an object by its class name.

`#include <ScriptObjectFactory.h>`

Public Member Functions

- void registerClass (ScriptObjectConstructor &constructor)

  Add a class constructor to the list.

- ScriptObject * construct (const std::string &className, const std::string &objName)

  Constructs an object by its class name.

Static Public Member Functions

- static ScriptObjectFactory & get ()

  Returns the global object. It is created on demand.

4.12.1 Detailed Description

Class implementing the abilty to create an object by its class name. It is a singleton, but unlike Screen, it is created and destructed automatically.

4.12.2 Member Function Documentation

4.12.2.1 ScriptObject * construct (const std::string & *className*, const std::string & *objName*)

Constructs an object by its class name.

Every object must also have a name.

Parameters

className Name of the class to instantiate

objName Name of the object that will be created

Returns

NULL if `className` is not found

4.13 SignalRenderer Class Reference

Base class for object that display a signal.

`#include <SignalRenderer.h>`

Public Member Functions

- virtual void draw ()=0
  Virtual function called by Screen on each frame to update the signal.
- void setSource (InputSource &source)
  Attach a SignalInput to this object.
- InputSource * getSource ()
  Get the signal input attached to this object.
- virtual string doCommand (const string &command, const vector< string > &args)
  Execute a command.

4.13.1 Detailed Description

Base class for object that display a signal.

4.13.2 Member Function Documentation

4.13.2.1 string doCommand (const string & *command*, const vector< string > & *args*) [virtual]

Execute a command.

Recognised commands:

- **attach** *string* Attach to the input source passed as first and only parameter

Returns

An error message (if any)

Reimplemented from ScriptObject.

Reimplemented in RectRenderer.

4.14 SineWaveInput Class Reference

Class representing a sine wave input.

`#include <SineWaveInput.h>`

Public Member Functions

- virtual double getCurrent ()
  Return the current value. It usually depends on the current frame number.
- virtual string doCommand (const string &command, const vector< string > &args)
  Execute a command.

4.14.1 Detailed Description

Class representing a sine wave input.

Note

Frequencies are measured in frames, not in seconds!

4.14.2 Member Function Documentation

4.14.2.1 string doCommand (const string & *command*, const vector< string > & *args*) [virtual]

Execute a command.

Commands recognised:

- **frequency** *float*
  Frequency in frames
- **phase** *float*
  Initial phase in radians
- **box** *float float*
  Set minimum and maximum (default: 0 and 1)

Reimplemented from InputSource.

4.15 SquareWaveInput Class Reference

Class representing a square wave input.

`#include <SquareWaveInput.h>`

Public Member Functions

- virtual double getCurrent ()
  Return the current value. It usually depends on the current frame number.
- virtual string doCommand (const string &command, const vector< string > &args)
  Execute a command.

4.15.1 Detailed Description

Class representing a square wave input.

Note

Frequencies are measured in frames, not in seconds!

4.15.2 Member Function Documentation

4.15.2.1 string doCommand (const string & *command*, const vector< string > & *args*) [virtual]

Execute a command.

Commands recognised:

- **times** *float float*
  On-time and off-time (in frames, default: 1 1)
- **delay** *float*
  An initial offset (in frames)
- **box** *float float*
  Set minimum and maximum (default: 0 1)

Reimplemented from InputSource.

4.16 TextFileInput Class Reference

This class reads input from a text file.

`#include <TextFileInput.h>`

Public Member Functions

- virtual double getCurrent ()
  Return the current value. It usually depends on the current frame number.
- virtual string doCommand (const string &command, const vector< string > &args)
  Execute a command.
- TextFileInput (const string &name)
  Default constructor. Sets all the options to default.
- virtual ~TextFileInput ()
  Destructor.

4.16.1 Detailed Description

This class reads input from a text file.

Format

File format is analogous to comma or space-separated values. Each line represents a frame. The user can choose which column to use (default is to use the first). Columns are numbered starting with 0.

Note

Frames are wrapped, i.e. when the input sequence ends, values restart from the beginning.

4.16.2 Member Function Documentation

4.16.2.1 string doCommand (const string & *command*, const vector< string > & *args*) [virtual]

Execute a command.

Recognised commands:

- **file** *string*
  Name of the file to open
- **column** *integer*
  Zero-based index of the column to use (default is 0)
- **separator** *string*
  Column separator (default is space)
- **delay** *integer*
  Initial time (row) offset

Returns

An error message (if any).

Reimplemented from InputSource.

Index