

SINGE

2006, 2007 Scott Duensing; 2006, 2007 Jaeger Technologies - <http://www.jaegertech.com>

January 16, 2007

1 Introduction

Welcome to SINGE, the Somewhat Interactive Nostalgic Game Engine for Daphne!

Singe is a Lua-based scripting system built in to Daphne that allows for rapid prototyping of new laserdisc games, or the creation of entirely new games! The language is easy to learn, very powerful, and fast. All the features needed to develop your own game are made available through a simple application programming interface (API).

Confused? Don't be! Read on!

2 Features

Singe provides numerous features to the game developer. Some of the more fun ones are:

- Object-Oriented Programming Language
- Easy-to-Use Interface into Daphne
- Sprites
- TrueType Font Support
- 32-bit Color Space with Transparency
- Multi-channel, Overlapping, Stereo Sound
- Analog and Digital Input Device Support

3 Writing Your Own Games

Although Singe makes writing your own game a pretty simple affair, it's still going to take a lot of work. Game development is hard. Hang in there - you'll survive! The basic steps you need to follow to create your own game are:

1. Create a fun concept for your game.
2. Design your game.
3. Assemble the required assets.
4. Code a prototype.
5. Test your prototype.
6. Complete coding.
7. Polish your game assets.
8. Test your game.
9. Distribute your game.
10. Rule the world!

4 Scripting Overview

A complete tutorial of scripting in Singe is outside the scope of this manual. Singe uses Lua as it's scripting language and extends it with Daphne-specific features. For a great set of tutorials on Lua, vist the Lua User's Wiki at <http://lua-users.org/wiki/TutorialDirectory>.

5 Events

Singe is event-driven. What that means is that your code basically sits and waits until Daphne or Singe needs something from you. At that time, one of your "event handlers" will be called to do whatever is needed. "Event handler" is just a fancy name for a function. (Don't know what a function is? Go back and read more of the Lua tutorials!) The events provided by Singe are described below. Your code must implement every event handler, even if you don't need it. In that case, just declare the function and don't provide any code.

5.1 Special Case - Initial Startup

On startup, all code outside function blocks is executed by Singe before the main game loop begins. Use this to initialize any variables your game needs, load sprites, sounds, and fonts, and generally get things ready for the game to begin running.

5.2 onInputPressed

Key and button events trigger two events when pressed. The first event received is the onInputPressed event. This signals that a key or button has been pressed and not yet released. (See onInputReleased.) Singe will pass in the keycode of the pressed key. Refer to Input under the Singe Framework for valid keycode names.

5.3 onInputReleased

onInputReleased is the second event fired when a key or button is pressed. This event is fired to notify the game that the key or button has been released. Singe will pass in the keycode of the pressed key. Refer to Input under the Singe Framework for valid keycode names.

5.4 onMouseMoved

This event is fired every time the mouse pointer changes locations. Singe passes in the new X and Y coordinates of the mouse as well as the change since the previous onMouseMoved event.

5.5 onOverlayUpdate

Singe fires this event every pass through the main game loop. Do all your game logic in this event. If you alter the overlay, return OVERLAY_UPDATED. If you didn't update the video, return OVERLAY_NOT_UPDATED.

5.6 onShutdown

If the user requests that Daphne be shut down, Singe will call this event before it shuts itself down. You should clean up any resources you have open and perform shutdown operations (such as saving high score tables) here.

5.7 onSoundComplete

The soundPlay API call fires this event when a sound has finished playing. By remembering the handle of a sound you want to loop (such as background music), you can restart it playing once it ends by using this event.

6 Singe Framework

Singe provides a number of functions, constants, events, and other features for interacting with Daphne. You'll use these as the building-blocks of your game. Collectively, they are known as the Singe "Framework".

6.1 Overview

The purpose of the Singe Framework is to provide you with an easy-to-use interface to Daphne to use when scripting your game. The low-level portion of the framework is implemented as part of Singe itself while higher-level features are programmed in Lua. To include the framework in your game so you can use it, you need to place the following command at the top of your main script file:

```
dofile("Singe/Framework.singe")
```

By using the framework, you are (mostly) protected from changes made by the Daphne and Singe teams. If they decide to change how something works, they will do their best to prevent the framework from changing. It is **HIGHLY** recommended that you use the features of the framework rather than going it alone. If the framework provides a feature, use it instead of writing your own!

6.2 Constants

Programmers like numbers. Everything is assigned a number. They're impossible to remember, so programmers also like constants. A constant is basically a name given to a value that is easier to remember than the value itself. The following are provided:

6.2.1 Input

The following constants define the values used in the `onInputPressed` and `onInputReleased` events:

- SWITCH_UP
- SWITCH_LEFT
- SWITCH_DOWN
- SWITCH_RIGHT
- SWITCH_START1
- SWITCH_START2
- SWITCH_BUTTON1
- SWITCH_BUTTON2
- SWITCH_BUTTON3
- SWITCH_COIN1
- SWITCH_COIN2
- SWITCH_SKILL1
- SWITCH_SKILL2
- SWITCH_SKILL3
- SWITCH_SERVICE
- SWITCH_TEST
- SWITCH_RESET
- SWITCH_SCREENSHOT
- SWITCH_QUIT
- SWITCH_PAUSE
- SWITCH_CONSOLE

6.2.2 Fonts

Fonts can be rendered in three different quality levels:

- `FONT_QUALITY_SOLID`
- `FONT_QUALITY_SHADED`
- `FONT_QUALITY_BLENDED`

SOLID fonts render the fastest, but look the worst. **SHADED** fonts are medium-quality and render slower than **SOLID**. **BLENDED** fonts are the highest quality, but take the longest to render. (NOTE: **SOLID** fonts do not currently work on PowerPC machines due to a bug in FreeType.)

6.2.3 Sound

When playing a sound, `Singe` returns a handle to the sound, or one of the following values:

- `SOUND_ERROR_INVALID`
- `SOUND_ERROR_NULL`

If you no longer wish to use the handle of a sound, set it to the following value:

- `SOUND_REMOVE_HANDLE`

6.2.4 Video

The `onOverlayUpdate` event expects one of the following values to be returned from it:

- `OVERLAY_UPDATED`
- `OVERLAY_NOT_UPDATED`

7 Functions

`Singe` implements the following collection of functions to allow your program to interact with `Daphne`:

7.1 Color

These functions allow you to set the colors used by other features of the `Singe` engine.

7.1.1 `colorBackground`

Sets the background color to use for future drawing operations. The three parameters are the red, green, and blue values of the color you wish to use. Each value has a range from 0 to 255.

Example:

`colorBackground(0, 0, 0)` – Black

7.1.2 `colorForeground`

Sets the foreground color to use for future drawing operations. The three parameters are the red, green, and blue values of the color you wish to use. Each value has a range from 0 to 255.

Example:

`colorForeground(255, 255, 255)` – White

7.2 Daphne

The Daphne functions provide access to information and features that don't neatly fit into any other category.

7.2.1 daphneGetHeight

Returns the height of the Daphne window (in pixels), or if in full-screen mode, the height of the screen.

Example:
`h = daphneGetHeight()`

7.2.2 daphneGetWidth

Returns the width of the Daphne window (in pixels), or if in full-screen mode, the width of the screen.

Example:
`w = daphneGetWidth()`

7.2.3 daphneScreenshot

Saves a screenshot of the current frame, including the graphics overlay.

Example:
`daphneScreenShot()`

7.3 Debugging

Debugging facilities in Singe are pretty simple at this time. The following are all available:

7.3.1 debugPrint

Prints text to `STDOUT`. On most systems, this is displayed in the window where Daphne was started.

Example:
`debugPrint("The current frame number is " + discGetFrame())`

7.4 Laserdisc

Daphne provides a number of features for controlling either a real or a virtual laserdisc player.

7.4.1 discAudio

Enables or disables audio channels from the laserdisc. The first parameter is the channel (1 or 2) and the second specifies if you wish to turn the audio on or off (true or false).

Example:
`discAudio(1, false)` – Turn off sound in channel 1

7.4.2 discChangeSpeed

Changes the current playback speed of the laserdisc. Variable speed playback is not supported on all laserdisc players. This function takes two arguments (a numerator and a denominator) which represent a fractional value. Common speeds are: 1X (1/1), 2X (2/1), 3X (3/1), 4X (4/1), 8X (8/1), 1/2X (1/2), 1/3X (1/3), 1/4X (1/4), 1/8X (1/8). Odd speeds such as 7/2 are currently undefined. Not all speeds are available with all players.

Example:
`discChangeSpeed(1, 2)` – Play at 1/2 speed

7.4.3 discGetFrame

Returns the current frame number.

Example:
f = discGetFrame()

7.4.4 discPause

Pauses a currently playing laserdisc.

Example:
discPause()

7.4.5 discPlay

Starts playing the laserdisc from the current frame.

Example:
discPlay()

7.4.6 discSearch

Searches the laserdisc for a specific frame number.

Example:
discSearch(42) – Search to frame 42

7.4.7 discSearchBlanking

Determines if the laserdisc video will be disabled during a search or not. Setting this to true will turn off the video, false leaves the video enabled while the search is underway.

Example:
discSearchBlanking(false) – Do not blank video when searching

7.4.8 discSetFPS

Sets the playback rate of the laserdisc. The Singe Framework defaults to 29.97 frames per second - the same rate as an NTSC DVD.

Example:
discSetFPS(29.97)

7.4.9 discSkipBackward

Skips backwards a specified number of frames.

Example:
discSkipBackward(10) – Skip backwards 10 frames

7.4.10 discSkipBlanking

Determines if the laserdisc video will be disabled during a skip or not. Setting this to true will turn off the video, false leaves the video enabled while the skip is underway.

Example:
discSkipBlanking(true) – Blank the video when skipping frames

7.4.11 discSkipForward

Skips forwardwards a specified number of frames.

Example:
`discSkipForward(10)` – Skip forwards 10 frames

7.4.12 discStepBackward

Steps backwards a single frame.

Example:
`discStepBackward()`

7.4.13 discStepForward

Steps forwards a single frame.

Example:
`discStepForward()`

7.4.14 discStop

Stops a playing laserdisc.

Example:
`discStop()`

7.5 Fonts

Singe supports the use of TrueType fonts through the following functions:

7.5.1 fontLoad

Loads a TrueType font into memory. The font is loaded at the specified point size. If you wish to display the font in a variety of sizes, you will need to load a separate copy of the font at each size you wish to use. Point sizes are roughly equivalent to the number of pixels tall you wish the font to be. This function returns a handle to the loaded font that you will need for other font functions.

Example:
`myTenPointFont = fontLoad("MyFont.ttf", 10)`

7.5.2 fontPrint

Renders text to the display using the currently selected font. The first argument is the horizontal location to begin printing. The second argument is the vertical location. The third argument is the text to render.

Example:
`fontPrint(10, 50, "This is 10 pixels from the left, 50 from the top.")`

7.5.3 fontQuality

Specifies the quality setting to use when rendering the currently selected font. See the above font constants for valid settings.

Example:
`fontQuality(FONT_QUALITY_BLENDED)`

7.5.4 fontSelect

Specifies the font to use for future font operations. Pass the handle of a loaded font to select it.

Example:

```
fontSelect(myTenPointFont)
```

7.5.5 fontToSprite

Font rendering is an "expensive" (slow) operation compared to rendering sprites. For text that is repeated or that is repeatedly rendered on numerous frames, you can render a static block of text to a sprite and then draw it instead. This function takes the text to render as an argument and returns a handle to the newly created sprite.

Example:

```
myTextOnASprite = fontToSprite("This is rendered to a sprite.")
```

7.6 Overlay

The graphics drawn on top of the laserdisc video is referred to as the "overlay". Along with more specific functions for fonts, sprites, etc., Singe provides the following functions for working with the overlay:

7.6.1 overlayClear

Erases the contents of the overlay. Unless specifically cleared, the contents of the overlay remain intact between frames.

Example:

```
overlayClear()
```

7.6.2 overlayGetHeight

Returns the height (in pixels) of the overlay. The height of the overlay is not the same as the height of the laserdisc video or the height of the Daphne display.

Example:

```
h = overlayGetHeight()
```

7.6.3 overlayGetWidth

Returns the width (in pixels) of the overlay. The width of the overlay is not the same as the width of the laserdisc video or the width of the Daphne display.

Example:

```
w = overlayGetWidth()
```

7.6.4 overlayPrint

Displays "bitmapped" text on the overlay. This text is rendered with a font built in to Daphne. Unlike TrueType fonts, the coordinates for the bitmap fonts are represented by characters, not pixels. The first argument is the column location. The second is the row. The third argument is the text to display.

Example:

```
overlayPrint(5, 10, "This is 5 characters from the left and 10 from the top.")
```

7.7 Sound

In addition to laserdisc audio, Daphne has the ability to play back multiple, overlapping, digital sound effects. Sound effects must be 44khz, 16 bit stereo samples in WAV format.

7.7.1 soundLoad

Loads a sound file into memory. The argument passed is the name of the sound file to load. This function returns a handle to the loaded sound for use in other sound functions.

Example:

```
myNeatSound = soundLoad("neat.wav")
```

7.7.2 soundPlay

Begins playback of the specified sound. This function returns one of the sound constants listed above on failure, or the playback handle of sound. This handle is different from the handle returned when you load a sound. Use the playback handle along with the onSoundCompleted event to know when your sound has finished playing. If you no longer need the handle, set it to `SOUND_REMOVE_HANDLE` to remove it.

Example:

```
handle = soundPlay(myNeatSound)
```

7.8 Sprites

Sprites are rectangular graphical objects that contain areas of transparency. Sprites render quickly and can be loaded from a variety of file formats.

7.8.1 spriteDraw

Draws a loaded sprite at the specified location. The first argument is the horizontal location of the upper-left corner of the sprite (in pixels). The second argument is the vertical location of the upper-left corner. The third argument is the handle of the loaded sprite that you wish to draw.

Example:

```
spriteDraw(50, 25, mySprite) – Draw mySprite 50 pixels from the left and 25 from the top
```

7.8.2 spriteGetHeight

Returns the height (in pixels) of the requested sprite. The argument passed is the handle of a loaded sprite.

Example:

```
h = spriteGetHeight(mySprite)
```

7.8.3 spriteGetWidth

Returns the width (in pixels) of the requested sprite. The argument passed is the handle of a loaded sprite.

Example:

```
w = spriteGetWidth(mySprite)
```

7.8.4 spriteLoad

Loads a sprite into memory and returns a handle to it for use in other sprite functions. Sprites should be 32-bit ("true-color") images with an 8-bit alpha channel. The PNG (Portable Network Graphic) format is highly recommended, although other formats may be loaded (although transparency information may be lost).

Example:

```
mySprite = spriteLoad("guybrush.png")
```

7.9 Virtual Laser Disc Player

Along with physical laserdisc players, Daphne supports a "virtual" laserdisc player based on MPEG2 video and OGG audio. The majority of Singe games will likely use the virtual player. Controlling the virtual laserdisc player is the same as controlling a real player, with the following added features:

7.9.1 vldpGetHeight

Returns the height (in pixels) of the laserdisc video. The height of the video is not the same as the height of the overlay or the height of the Daphne display.

Example:

```
h = vldpGetHeight()
```

7.9.2 vldpGetPixel

Returns the red, green, and blue values of the specified location of the virtual laserdisc video. The two arguments are the horizontal and vertical location (in pixels) you wish to read. These coordinates are in "overlay space" and should be specified using coordinates based on the size of the overlay, not the size of the video. Note that this function has three return values instead of the usual single value.

Example:

```
r, g, b = vldpGetPixel(50, 100) – Return the Red, Blue, and Green values of the pixel 50 from the left and 100 from the top
```

7.9.3 vldpGetWidth

Returns the width (in pixels) of the laserdisc video. The width of the video is not the same as the width of the overlay or the width of the Daphne display.

Example:

```
w = vldpGetWidth()
```