60Beat Gamepad SDK

Version 1.1.0

Change History

0.9.0 - Initial Release
0.9.1 - Fixed joystick calibration, added iOS 5.0 support, fixed documentation errors.
0.9.2 - Internal algorithm improvements
0.9.3 - Improved device compatibility, modified demo to call start/stopListening on suspend/resume, updated docs.
0.9.4 - Fix potential crash from unimplemented delegate method, disable processing when joystick not connected.
0.9.5 - Add ability to chain interruption listener, improve cleanup during start/stop, make joystickConnected status more accurate, add allowMixing flag
1.0.0 - Significant updates to API and underlying management of AudioSession
1.1.0 - Adjustments for production hardware, add Control Schemes API

Overview

The 60Beat Gamepad is a gaming controller which connects to iOS devices via the headphone jack. The SDK contains a library which monitors the microphone jack for input from the gamepad and provides decoded status, button presses, and analog stick positions to an app.

Theory of Operation

The SDK utilizes an AudioUnit to receive audio samples from the microphone input, and decode data packets from the gamepad hardware. In order to function, the SDK requires that the app's AudioSession have the category PlayAndRecord.

Since most games already incorporate an AudioSession with some form of sound engine, every effort has been made to streamline integration of the SDK into existing apps. In most situations, the SDK will be enabled by a switch in the app's settings. This switch will affect the "enabled" property of the SDK instance. When the SDK is disabled, it will make no changes to the AudioSession configuration, and should have no effect whatsoever on operation of the game.

When the SDK is enabled, it will first check to see if the AudioSession is in use by determining if the AudioSession has a delegate set. If the AudioSession does not have a delegate, then the SDK sets the AudioSession in PlayAndRecord category, ready to receive input from a gamepad. If the AudioSession did have a delegate, then the following operations are performed when the SDK is enabled and any time the audioRoute changes:

1) If AudioRoute is "Headphone" and the AudioSession Category is not PlayAndRecord, then the category is set to PlayAnd Record for two seconds to check from presence of gamepad. If gamepad is not present, category is restored.
2) If AudioRoute is "Receiver and Microphone", override route to "Speaker and Microphone". This corrects speaker output after disconnection of gamepad.

Integration

Step 1: Include Files

To integrate the SBJoystick SDK into your app, first add the contents of the lib folder (SBJoystick.h, SBControlScheme.h and SBJoystickLib.a) to your project. SBJoystickLib.a is a universal library, so it can be included for both simulated and real targets. If size of your app is a concern, the SBJoystickLib-iphone.a file can be used instead, which contains only code for the device and is approximately half the size of the universal library.

Step 2: Add Frameworks

Add the AVFoundation and AudioToolbox frameworks to the project.

Step 3: Setup Linker

The library requires the C++ linker. If you get a large number of linker errors when trying to build with the library, you most likely need to enable C++ linking. The easiest way to enable it is to change the extension of any *one* objective-c source file in your project from ".m" to ".mm". Typically, the app delegate could be changed to .mm, but it can be any file which does not cause problems in your project. As long as the project contains one or more C++ files, the C++ linker will be enabled.

Step 4: Add a delegate

The SBJoystickDelegate protocol defined in SBJoystick.h contains two methods which should be implemented by your delegate:

```
- (void)joystickStatusChanged:(SBJoystick *)joystick
```

This method is called any time the joystick status is changed (connected or disconnected). The status can be determined at any time by checking the joystickConnected property of the SBJoystick instance.

```
- (void)controlUpdated:(SBJoystick *)joystick;
```

This method is called each time the SDK receives valid data from the joystick. It should be noted that the data may not have changed from the previous sample.

Currently, it is the responsibility of your app to maintain a state history if you need to track button-down and button-up events. A future version of the SDK will include support for registering for button events.

Step 5: Enable monitoring for Gamepad

The gamepad is enabled using the “enabled” property of the SDK:

```
[SBJoystick sharedInstance].enabled = YES;
```

The gamepad SDK should be enabled only *after* your game’s audio engine is set up. The SDK will detect the state of the audio engine and configure itself appropriately.

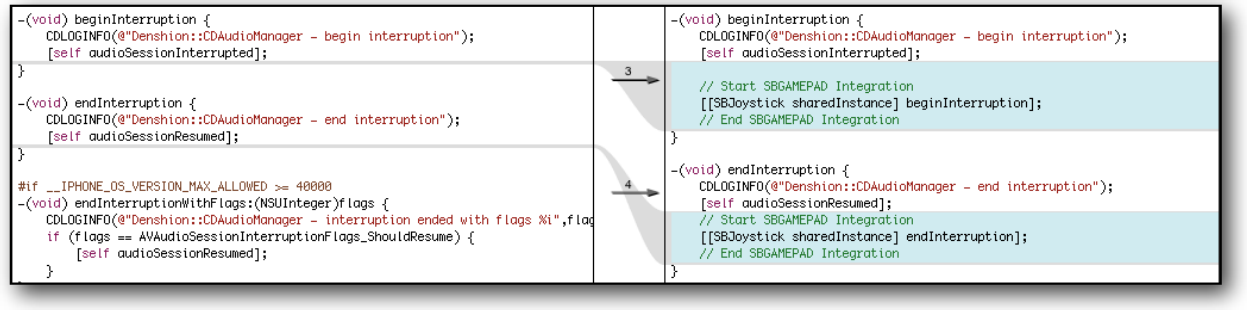
Typically, the enabling of the Gamepad SDK should be done in response to a user setting. The demo app simulates this with a UISwitch at the top left of the screen.

Step 6: Chain AudioSession Delegate

If your app sets a delegate for the AVAudioSession (most sound engines do), you will need to locate the following delegate methods and call the corresponding methods on the SBJoystick shared instance:

```
-(void) beginInterruption
-(void) endInterruption
```

Within your these methods in your delegate, you should perform any required processing and then call the corresponding methods on SBJoystick. For example, when integrating with the Cocos-Denshion sound engine, the following changes would be made to CDAudioManager.m:



```
-(void) beginInterruption {
    CDLOGINFO(@"Denshion::CDAudioManager - begin interruption");
    [self audioSessionInterrupted];
}

-(void) endInterruption {
    CDLOGINFO(@"Denshion::CDAudioManager - end interruption");
    [self audioSessionResumed];
}

#if __IPHONE_OS_VERSION_MAX_ALLOWED >= 40000
-(void) endInterruptionWithFlags:(NSUInteger)flags {
    CDLOGINFO(@"Denshion::CDAudioManager - interruption ended with flags %i",flag
    if (flags == AVAudioSessionInterruptionFlags_ShouldResume) {
        [self audioSessionResumed];
    }
```

```
-(void) beginInterruption {
    CDLOGINFO(@"Denshion::CDAudioManager - begin interruption");
    [self audioSessionInterrupted];

    // Start SBGAMEPAD Integration
    [[SBJoystick sharedInstance] beginInterruption];
    // End SBGAMEPAD Integration
}

-(void) endInterruption {
    CDLOGINFO(@"Denshion::CDAudioManager - end interruption");
    [self audioSessionResumed];
    // Start SBGAMEPAD Integration
    [[SBJoystick sharedInstance] endInterruption];
    // End SBGAMEPAD Integration
}
```

If your game does not set a delegate for the AVAudioSession, the SDK will make itself the delegate when enabled is set true.

Step 7: Handle joystick input

The current state of the joystick controls can be determined using the following properties:

```
CGPoint leftJoystickVector;
CGPoint rightJoystickVector;

BOOL button1State;
BOOL button2State;
BOOL button3State;
BOOL button4State;
BOOL buttonUState;
BOOL buttonDState;
BOOL buttonLState;
BOOL buttonRState;

BOOL buttonStartState;
BOOL buttonSelectState;
BOOL buttonModeState;
BOOL buttonL1State;
BOOL buttonL2State;
BOOL buttonR1State;
BOOL buttonR2State;
BOOL buttonRStickState;
BOOL buttonLStickState;
```

The joystick vectors represent the current position of the analog sticks as a vector with range [-1,1]. All other properties report TRUE when the button is pressed, otherwise FALSE.

These states can either be checked from within the controlUpdated: delegate callback, or within your game loop.

The Control Scheme API, below, provides an alternative to reading the button states directly.

Control Schemes

The SDK provides an easy method for mapping gamepad controls into game actions. A set of such mappings is referred to as a control scheme. A control scheme is defined using the SBControlScheme class as follows:

```
SBControlScheme *scheme;
scheme = [[[SBControlScheme alloc] init] autorelease];
scheme.name = @"Default Controls";
[scheme addAssignmentForControl:LEFT_STICK tag:CONTROL_TAG_MOVE label:@"Move"];
[scheme addAssignmentForControl:RIGHT_STICK tag:CONTROL_TAG_LOOK label:@"Look"];
[scheme addAssignmentForControl:BUTTON_1 tag:CONTROL_TAG_FIRE label:@"Fire"];
...
```

Each assignment maps a control on the gamepad to a developer-assigned tag. The label is optionally used when displaying control schemes to the user (see Control Scheme View section, below).

To use a control scheme, simply set the currentControlScheme property of SBJoystick to the scheme you've built:

```
[SBJoystick sharedInstance].currentControlScheme = scheme;
```

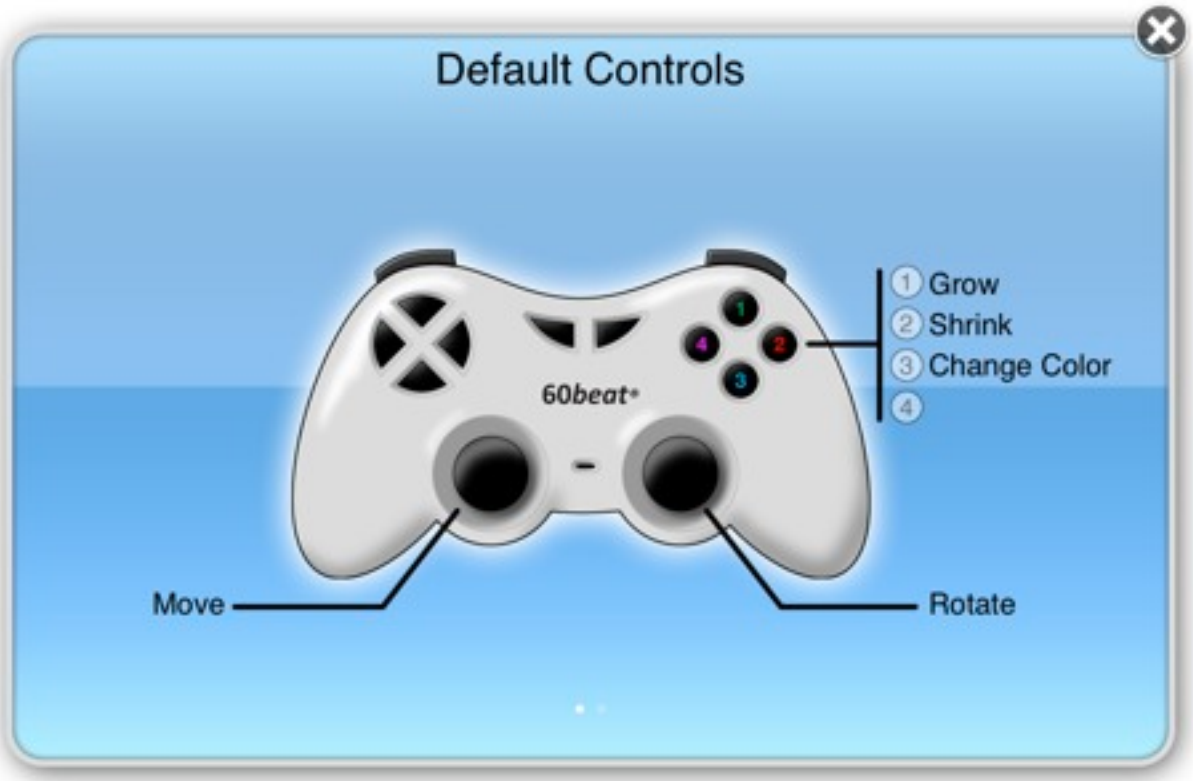
Within your game loop, you can query the state of any button, and the vector of either analog stick as follows:

```
CGPoint moveVector = [[SBJoystick sharedInstance] analogVectorForControlTag:CONTROL_TAG_MOVE];
BOOL fireButtonIsPressed =
              [[SBJoystick sharedInstance] buttonStateForControlTag:CONTROL_TAG_FIRE];
```

The currentControlScheme can be modified at any time, and the control mapping will instantly start responding to the new scheme.

Control Scheme View

The SDK includes a class SBControlSchemeView for displaying and optionally selecting a control scheme. The view looks like this:



The view will display the callouts for any controls that are configured. In order to display the Control Scheme View, you first need to include SBControlSchemeView.h and SBGamepad.bundle in your project. The control scheme view is then displayed as follows:

```
SBControlSchemeView *v = [[SBControlSchemeView alloc] initWithSchemes:schemes];
[v setBackground:@"blue"];
[self.view addSubview:v];
[v release];
```

Note, the control scheme view currently supports landscape orientation only. If one of the schemes passed in to the init function is equal to currentControlScheme, the view will scroll to that scheme on display. When the view is closed, the currently displayed scheme is stored to currentControlScheme, which makes the view act as a scheme selector.

IMPORTANT NOTES

Calibration

Joysticks will be calibrated at the factory. The SDK provides an appropriate deadband and limit checking such that user calibration should not be necessary under normal conditions.

Supported Devices

The Gamepad and SDK has been developed to function with the following devices running iOS version 4.0 or higher:

- iPhone 3GS
- iPhone 4
- iPhone 4S
- iPod Touch 4th Gen
- iPad
- iPad 2

Future Device Compatibility

A future “compact” model of the controller may not incorporate the following buttons: L2, R2, L3, R3 (Push In Left/Right stick). In order to future-proof your game, it is recommended to either provide alternate buttons for any functions assigned to these buttons, or provide one or more control schemes which do not use these buttons.

Troubleshooting

The “loggingEnabled” property of the SDK will enable additional debug messages which may be valuable to understand AudioSession configuration issues. If you experience issues integrating the gamepad into your game, please enable logging, take a trace from the game, and send it to support with a description of the issue.

Known Issues

- If background music is being played by another app (e.g. iPod) when the gamepad SDK is enabled, the background music will typically be paused. Background music can be restarted by double-clicking the home button and pressing play, and will continue to play with or without the gamepad connected.

- While the gamepad is active (AudioSession in PlayAndRecord mode), the device's mute switch will not function. The volume controls will continue to function correctly, although the initial audio level may be different than what was previously set.

- Airplay is not currently supported. Make sure to disable the Gamepad SDK before starting Airplay.