# RadioKit SDK

## Introduction

Stormy's RadioKit SDK is designed for software developers who want to simplify the iPhone app development process while creating an iPhone radio app.  This SDK is targeted for developers already familiar with Apple's iPhone SDK, but who want to save a large amount of time by using radio streaming features from a well tested library. This will allow a developer to focus more time on creating other portions of their app to make it unique, such as the user interface and other non-radio-related features.

## SDK features

- Time-shifting features with live streams: pause, rewind, and fast forward back to "live"
- MP3 "on demand" file streaming and seeking
- Audio recording to a file
- Support for MP3, AAC, and AAC+ v1 (also called HE AAC)
- pls, m3u, and direct URL support
- Robust error handling including seamless "stream stitching" when data connection is dropped
- Automatically handles other audio interruptions, such as incoming phone call, and will resume playback
- Support for Audio Taps - add hooks for visualizers or performing audio effects on the stream before playback
- Support for parsing AudioVault XML "now playing" information
- Embedded Shoutcast/Icecast metadata parsing if present within the stream

## Licensing

The RadioKit SDK is licensed on a per project basis. One-off costs are $100 per app project.  Volume discounts are available.   You are allowed to resell any apps you develop using this SDK.  How you price such apps is completely up to you.   Using this approach, you can develop radio apps for other clients and you only have to pay a $99 licensing fee for each product developed.  You can charge your clients whatever fee you decide.

If you are interested in volume pricing, please contact Stormy Productions directly via the "Contact Us" link at http://stormyprods.com.

## SDK Documentation

The RadioKit SDK is distributed as a single **libRadioKit.a** library file usable with both simulator and device builds. Also included are **RadioKit.h** and **BufferView.h**, the necessary header files to define the library class methods.

### The RadioKit Class Overview

The RadioKit is composed primarily of one class: **RadioKit**. Under normal usage, you will only need to have one instance of this class in your project. Generally, you will alloc and initialize the class when your app is first launched, and not destroy it until your app is terminated.

After you alloc and init the class, you should immediately authenticate the library based on the license key you have been provided for your project. If you do not have a license key the library will function in demo mode. While in demo mode, the library will only provide radio streaming functionality for 10 minutes on an actual iPhone/iPod device. No time limit is imposed when running the code in the simulator.

The following demonstrates typical initialization of the RadioKit class:

```
radioKit = [[RadioKit alloc] init];
[radioKit authenticateLibraryWithKey1:0x1234 andKey2:0x1234];
```

After you initialize and authenticate the RadioKit, you may start playing an audio stream. This is done via the method *setStreamUrl:isFile:* demonstrated as follows:

```
[radioKit setStreamUrl:@"http://mystream" isFile:NO];
```

Once this method is called, the RadioKit class will automatically attempt to connect to the provided URL (and parse it if it is a pls or m3u file) and begin playing the audio stream. If the connection is ever interrupted via network problems, the RadioKit class will continuously attempt to reconnect without any user intervention.

If you are interested in monitoring the progress of the RadioKit class as it performs its functions in the background, you can register a delegate for the RadioKit class and implement a set of optional protocols. Complete documentation on all the protocols is provided later in this document.

The protocols will allow you to receive notification when the audio stream changes status (such as from connecting to buffering to playing) as well as notification when the meta data for the stream changes (such as new artist / title information).

**RadioKit Class Methods**

- (bool) authenticateLibraryWithKey1: (uint32_t)key1 andKey2: (uint32_t)key2;

Provides the license key information to the library. Returns true/false based on whether the key was valid or not. Will also cause a pop-up alert to be displayed if the authentication failed.

Should be called immediately after alloc / init.

- (id) initWithBufferCount: (unsigned int) bufferCount;

An optional method for initializing the RadioKit object if you want to change the amount of memory allocated for holding the audio packets. The default buffer size is 1200 blocks. Each block is 5K in size, so to get a rough idea of the amount of audio you can multiply 5K by the bufferCount. The default is 6MB (i.e. 1200 x 5K). The exact amount of audio that the buffer can hold will vary depending on the bit-rate and encoding format of the stream being played. Each block must hold complete packets (no partial fragments), so depending on the audio encoding packet size, some memory may be wasted due to the blocksize not being exactly divisible by the current packet size.

You can get a fairly accurate reading of the amount of time the buffer will hold *after* the stream has been started by querying radioKit.`maxBufferUsageInSeconds`.

- (NSString *) version;

Returns the current version number of the library.

- (void) setUserAgent: (NSString *)string;

Set the HTTP User-Agent string to be used when retrieving the pls and m3u files and when connecting to the audio stream.

- (void) setDataTimeout: (NSUInteger)time;

Set the time, in seconds, to wait for data from the streaming server. If no data is received within this time, the connection to the server will be dropped and a new connection will be initiated.

- (void) setPauseTimeout: (NSUInteger)time;

Set the time, in seconds, to wait before stopping audio streaming when paused. Setting a value of 0 means the RadioKit will remain streaming and buffering indefinitely while paused (until the buffer is full). The default value is 300 seconds.

- (void) setMetaEncoding: (NSStringEncoding)encoding;

Set the encoding used by the meta data in the stream. The default is NSUTF8StringEncoding.

- (void) setStreamUrl: (NSString *)url isFile:(bool)isFile;

Set the URL for the audio stream.

*url* can be a pls, m3u or direct link to an audio stream or audio file. *isFile* specifies whether the url references an "on demand" file or an audio stream.

- (void) setStreamUrl: (NSString *)url isFile:(bool)isFile setType(enum SRK_URL_TYPE)urlType;

Set the URL for the audio stream.

*url* can be a pls, m3u or direct link to an audio stream or audio file. *isFile* specifies whether the url references an "on demand" file or an audio stream.

*urlType* specifies what type of URL is being played: SRK_RAW_STREAM, SRK_PLS, or SRK_M3U. This method is only needed if your URL is a pls or m3u file but does not end with .pls or .m3u.

- (void) setStreamURlWithOutRestart: (NSString *)url;

Similar to setStreamUrl above, but will not attempt to start playing the stream.

- (void) playLocalFile: (NSString *) filename;

Play a locally stored audio file. Filename must be the full filesystem path to the audio file.

– (void) setBufferWaitTime: (NSUInteger) waitTime;

Defines the amount of audio playback time (in seconds) to accumulate in the buffer before first starting to play a non-file audio stream. A larger value will provide a more stable listening experience in places with poor connectivity with the trade-off a longer delay before the listener hears any audio. Default value is 10 seconds.

– (void) setBufferWaitTimeForFile: (NSUInteger) waitTime;

Defines the amount of audio playback time (in seconds) to accumulate in the buffer before first starting to play an on demand streamed file. A larger value will provide a more stable listening experience in places with poor connectivity with the trade-off a longer delay before the listener hears any audio. Default value is 3 seconds.

– (void) setThrottleBypassTime: (NSUInteger) waitTime;

Defines the amount of audio playback time (in seconds) to accumulate in the buffer before throttling the data download bandwidth usage so it matches the audio files encoded bitrate. ***This setting only applies to on demand file payback****. Live streaming has a set throttle bypass value of 20 seconds and cannot be altered.*

The default value is 0, which means bandwidth throttling occurs as soon as enough audio data has been received to begin playback.

A larger value will allow more on demand audio to be buffered at the expense of downloading more data in a shorter amount of time.

**IMPORTANT**: per Apple's guidelines, an app which streams audio data over the cellular network should not use more than 5 MB per 5 minutes. Consuming more data may result in having your app rejected during Apple's review process.

– (void) stopStream;

Immediately stop playback of the stream and disconnect from the server.

– (void) pauseStream;

Pause playback of the stream, but continue to receive data from the server and accumulate it in the buffer.

– (void) startStream;

Start/resume playback of the stream. If currently disconnected, connect to the server and begin playback.

– (int) getStreamStatus;

Get the current status of the stream, such as connecting, buffering, playing, and paused. See **RadioKit Class Status Types** below for a list of status values and their meaning.

– (bool) isAudioPlaying;

Returns whether audio is currently playing and an audio queue is in use. This differs from the status values returned via the method *getStreamStatus* in that audio can be playing from the buffer when we do not have a connection. Note: this will also return TRUE when audio playback is in a paused state.

– (void) rewind: (NSUInteger)seconds;

Rewind the audio playback the requested number of seconds. If the number of seconds exceeds the buffer and this is an audio stream, only rewind as many seconds as possible. If it is an audio file being played, perform a HTTP "Range:" request to rewind to an earlier part of the file.

– (void) fastForward: (NSUInteger)seconds;

Fast forward the audio playback the requested number of seconds. If the number of seconds exceeds the buffer and this is an audio stream, only fast forward as many seconds as possible. If it is an audio file being played, perform a HTTP "Range:" request to fast forward to an later part of the file.

– (BOOL) isFastForwardAllowed: (NSUInteger) seconds;

Can the audio be fast forwarded the number of seconds requested?

– (BOOL) isRewindAllowed: (NSUInteger) seconds;

Can the audio be rewinded the number of seconds requested?

– (AudioQueueRef) getAudioQueue;

Returns a pointer to the underlying AudioQueue object being used to play the audio stream. Useful if you want to modify the attributes of the queue, such as alter the gain.   NOTE: this only returns a relevant point when playing a streaming source.  It is not applicable if playing a local file via the playLocalFile: method.

```
// Buffer statistics
- (NSUInteger) maxBufferSize;
```

Returns the maximum number of buffer blocks usable.

```
- (NSUInteger) currBufferUsage;
```

Return how many buffer blocks are currently in use

```
- (NSUInteger) currBufferPlaying;
```

Return the id of which buffer block is currently playing.

```
- (NSUInteger) currBufferUsageInSeconds;
```

Return the tally of current amount of buffer space being used (in seconds).

```
- (NSUInteger) maxBufferUsageInSeconds;
```

Return the maximum possible tally amount of buffer space (in seconds).  NOTE: While the buffer size is static, the amount of space it represents in seconds will vary depending on the bit rate of the audio being played.

```
- (NSUInteger) timeShift;
```

Returns the number of seconds where are time-shifted in the past with the current audio playback.

```
- (NSUInteger) bufferByteOffset;
```

Returns the byte offset within a file that the current buffer playback represents. Only valid for "on demand" file playback.

```
- (NSUInteger) filePlayTime;
```

Returns the current time offset (in seconds) that we are in playing an "on demand" file. Only valid for file playback, not a live audio stream.

// Ice-cast / Shoutcast info field
- (NSInteger) bitRate;

Returns the bit rate as reported by the shoutcast/icecast headers.

- (char *) streamFormat;

Returns the stream format type (mp3, aac, etc.) as reported by the shoutcast / icecast headers

- (NSString *) genre;

Returns the genre as reported by the shoutcast / icecast headers

// Used to control fetching meta data from XML server (AudioVault XML formating)

- (void) setXMLMetaURL: (NSString *)url;

Set the URL for the AudioVault XML feed file

- (void) setStationTimeZone: (NSString *)timeZone;

Set the time zone value for the station. This should be the standard 3 letter abbreviation for the radio station's time zone, such as @"EST" for Eastern Standard Time. It should be the time zone of the station, not the listener.

- (void) setXmlDelay: (NSTimeInterval) delay;

Set an arbitrary lag time to be added to the XML event timestamps. This is useful if Internet audio stream for a station is lagged by a certain amount of time vs. their terrestrial broadcast.

- (void) beginXMLMeta;

Begin the process periodic fetching of the AudioVault XML feed file. The process handling the fetch will parse the XML and identify based on current clock and timezone when the XML needs to be fetched again for new data.

- (void) endXMLMeta;

Stop the automatic fetching of AudioVault XML meta data.

// Live365 meta data configuration

// Live365 XML parsing

- (void) setLive365MetaURL: (NSString *)url;
  Set the URL for the Live365 meta data feed

- (void) beginLive365Meta;
  Begin the process of fetching Live365 meta data

- (void) endLive365Meta;
  Stop Live365 meta data feed

- (NSString *) titleWithoutArtist;

  Return the title of the current track as parsed from the AudioVault XML file. This is to compensate for the fact that the RadioKit property of currTitle (defined below) returns both the artist and title together as a single string.

- (NSString *) artist;

  Return the artist of the current track as parsed from the AudioVault XML file.

// Audio Metering Info

- (void) enableLevelMetering;

  Enable access to the audio level metering. Should be called **after** the audio stream playback has started. It is recommended to put this function call in the SRKAudioPlayStarted callback if you desire to use it.

- (void) getAudioLevels: (Float32 *)levels peakLevels: (Float32 *) peakLevels;

  Retrieves the audio levels. It is required that the levels and peakLevels data pointers passed to this method point to pre-allocated storage holding any array of 2 Float32 values (one for each stereo channel).

## // Audio Recording

```
- (void) startRecording: (NSString *)filename appendToFile: (BOOL)
  append;
```

Start recording incoming audio to the specified file. Due to the nature of recording, this will cause a restart of the audio stream to ensure proper starting headers are included in the file. The supplied filename must include the full filesystem path of where the file will be stored. If the file already exists, it will be deleted first, unless you set appendToFile: YES, in which case the new data will be appended to the end of the file. **NOTE**: you should only append to a file if you are certain the prior contents of the file are the exact same audio type (i.e. bitrate and encoding type). Otherwise, the file may not play back properly.

```
- (void) stopRecording;
```

Stop the current recording and close the file.

```
- (void) startRecording: (NSString *)filename appendToFile: (BOOL)
  append resetStream: (BOOL) resetStream;
```

Start recording incoming audio to the specified file and only reset the stream first if *resetStream* is YES. NOTE: setting *resetStream* to NO should only be used for recording MP3 files. Recording AAC files in this manner will not produce usable recordings.

## // Audio Tap

```
- (void)configureTap:(AudioQueueProcessingTapCallback) callback
          clientData:(void *)clientData
               flags:(UInt32)flags
        outMaxFrames:(UInt32 *)processingMaxFrames
 outProcessingFormat:(AudioStreamBasicDescription *)processingFormat;
```

Call this method after the RadioKit object is created but before any audio stream playing is configured to define an AudioTap. When an audio stream is played, the Audio Tap will automatically be created, and whenever it has data *callback* will be called. If you use this method, you should also define the protocols of *SRKTapCreated* and *SRKTapDisposed* to handle when the tap is first created and when it is disposed. The tap will be disposed and a new one created whenever a new stream is played that has different audio attributes than the previous stream (such as bit-rate or format).

## // Basic Authentication

```
- (void) enableBasicAuthentication: (NSString *) username password:
  (NSString *)password;
```

Call this method if the audio stream requires HTTP 1.1 Basic Authentication with a username and password. This method must be called *before* the stream is defined. If a later stream does not require authentication, then call `disableBasicAuthentication` before defining the new stream.

```
- (void) disableBasicAuthentication;
```

Call this method to disable a previously defined Basic Authentication (username/password) for a stream.

## RadioKit Class Properties

```
@property (nonatomic,retain) id delegate;
```

Set/Get the delegate which implements the RadioKit protocols.

```
@property (nonatomic,readonly) NSString *currTitle;
```

Get the most recent meta data "title" string as reported for the audio stream.

```
@property (nonatomic,readonly) NSString *currUrl;
```

Get the meta data "url" field as reported for the audio stream.

```
@property (nonatomic, readonly) NSUInteger
throttledFileLengthInSeconds;
```

Get the total length of the current file being played (in seconds). Only applies to "on demand" file playback.

```
@property (nonatomic, readonly) bool isFileSeekAllowed;
```

Returns whether the server providing the current audio file supports the "Range:" command. Support for the "Range:" command is necessary in order to perform seeking to any arbitrary position in an "on demand" file.

## RadioKit Class Status Types

SRK_STATUS_STOPPED

RadioKit is currently stopped and no connection exists.

SRK_STATUS_CONNECTING

RadioKit is connecting to the server.

SRK_STATUS_BUFFERING

RadioKit is accumulating data for the audio buffer, but audio is not currently playing.

SRK_STATUS_PLAYING

RadioKit is playing audio from the buffered data.

SRK_STATUS_PAUSED

RadioKit is paused. New audio data is still being buffered and accumulated, but no audio is currently playing.

**RadioKit Class Protocols (Optional)**

```
- (void) SRKConnecting;
```

Implement this protocol if you want to be notified when the RadioKit class begins connecting to a server.

```
- (void) SRKIsBuffering;
```

Implement this protocol if you want to be notified when the RadioKit class has connected to the server and is beginning buffering data.

```
- (void) SRKPlayStarted;
```

Implement this protocol if you want to be notified when the RadioKit class has buffered enough data and has begun playback of audio.

```
- (void) SRKPlayStopped;
```

Implement this protocol if you want to be notified when the RadioKit class has stopped playing audio and has disconnected from the server.

```
- (void) SRKPlayPaused;
```

Implement this protocol if you want to be notified when the RadioKit class has paused playback of audio.

```
- (void) SRKNoNetworkFound;
```

Implement this protocol if you want to be notified when the RadioKit class has encountered an error due to no wireless network connectivity.

– (void) SRKMetaChanged;

Implement this protocol if you want to be notified when the RadioKit class has encountered a change in the meta data (artist/title) for the current audio. NOTE: this meta change notification occurs when the audio buffer containing the meta change is queued for playing. If you require notification when the metadata change is detected in the live audio stream, use SRKRealtimeMetaChanged:withUrl:.

– (void) SRKRealtimeMetaChanged: (NSString *)title withUrl: (NSString *) url;

Implement this protocol if you want to be notified when the RadioKit class has encountered a change with the meta data (artist/title) for the live data stream. **title** will contain the metadata title detected in the stream (can be nil). **url** will contain the value of the StreamURL meta tag (can be nil). Depending on the playback buffering, this information might not correlate to what the listener is currently hearing. If you would like to be notified when the buffered audio metadata changes, use SRKMetaChanged.

– (void) SRKBadContent;

Implement this protocol if you want to be notified when the RadioKit class has encountered an error due to a non http: URL being supplied.

– (void) SRKMissingContent;

Implement this protocol if you want to be notified when the RadioKit class encounters a file not found error for the supplied URL, or any other HTTP error > 400.

– (void) SRKHttpError: (CFIndex)errorCode;

Implement this protocol if you want to be notified when the RadioKit class encounters any HTTP Error > 400 and you want to know the error number.

– (void) SRKFileComplete;

Implement this protocol if you want to be notified when the RadioKit class when an "on demand" file has completed playback.

## – (void) SRKURLNotFound;

Implement this protocol if you want to be notified when the RadioKit class encounters an unknown error when attempting to connect to the server.

## – (void) SRKTimeoutExceeded;

Implement this protocol if you want to be notified when the RadioKit class encounters a data timeout while waiting for data from the server. This error may also occur if you attempt to connect to a URL that does not resolve to a valid hostname.

## – (void) SRKQueueExhausted;

Implement this protocol if you want to be notified when the RadioKit class has playback interrupted due to the buffered audio queue becoming exhausted. This could be a sign that the bit rate is too high for the current bandwidth available on the iPhone and could be used to identify when to drop down to a lower bit rate stream.

## – (void) SRKAudioWillBeSuspended;

Implement this protocol if you want to be notified immediately before the RadioKit class suspends playback of audio due to other iPhone OS events, such as an incoming phone call. NOTE: the RadioKit will handle resuming from such interruptions automatically. This protocol is only necessary if your app also needs to be aware such interruption is about to occur.

## – (void) SRKAudioSuspended;

Implement this protocol if you want to be notified when the RadioKit class has playback suspended due to other iPhone OS events, such as an incoming phone call. NOTE: the RadioKit will handle resuming from such interruptions automatically. This protocol is only necessary if your app also needs to be aware such interruption occurred.

## – (void) SRKAudioResumed;

Implement this protocol if you want to be notified when the RadioKit class has resumed playback after an external OS interruption, such as a declined incoming phone call. NOTE: the RadioKit will handle resuming from such interruptions automatically. This protocol is only necessary if your app also needs to be aware such interruption occurred.

- (void) SRKRecordingStopped: (NSException *) exception;

Implement this protocol if you want to be notified when the RadioKit class has stopped recording the audio data due to a file write exception.

- (void) SRKTapCreated;

Implement this protocol if you are using Audio Taps and want to be notified when a new tap is created.

- (void) SRKTapDisposed;

Implement this protocol if you are using Audio Taps and want to be notified when a tap is deleted.

## SDK Version History

2.8.1 (10-12-2013):

Bug fix for potential buffer overflow if audio stream does not contain proper headers.

2.8 (10-7-2013):

Added SRKTimeoutExceeded protocol and corrected missing error notifications when using an invalid URL for .pls or .m3u file.

2.7 (9-17-2013):

Added setMetaEncoding: method to allow specifying the string encoding used in the meta data of the stream.

2.6.2 (9-14-2013):

Removed debug message logging that was enabled.

2.6.1 (9-10-2013):

Minor bug fix related to setting of the stream properties.

2.6:

Added new optional initWithBufferCount: method to allow specifying a different sized buffer to hold the audio data.

2.5:

Added new optional protocol: SRKHttpError:(CFIndex)errorCode Implement this if you want to be notified of an HTTP connection error when starting the stream (such as error 404 or 401).

2.4.1:

Restored accidentally removed support for SRKRealtimeMetaChanged:withUrl:

2.4:

Added support for streams requiring HTTP 1.1 Basic Authentication. (see enableBasicAuthentication: method)

2.3.8 & 2.3.9:
Bug fix for recording when resetStream is set to NO.

2.3.7:
Bug fix for not properly detecting an invalid M3U or PLS URL.  It will now call the `SRKURLNotFound` protocol method if this error is detected.

2.3.6:
Bug fix for fast forward /rewind within a fixed length file that is contained in a set of files listed in an M3U.

2.3.3:
Added new method for forcing the URL type when setting the stream URL. Only necessary if the URL points to a pls or m3u file that does not end with .pls or .m3u.

2.3.2:
Bug fix for delay which could occur when switching audio streams.
Added method to perform stream recording without reseting the stream. (Only useful for MP3 streams.)

2.3.1:
A call to startRecording will force a reset of the audio stream if it is already playing. This is done to ensure full starting headers are stored in the file (which is necessary for AAC audio types).

2.3:
Added support for Audio Taps

2.2: Bug fix for handling Live365 meta data

2.1:
Bug fixes: added fix for improper bit-rate detection of MPEG v2.5 files and also fixed crash that could occur if an app using RadioKit is

in the background and is interrupted by another audio event (such as a phone call) for longer than 10 minutes.

2.0:

Added support for recording audio (**startRecording**:, stopRecording, and SRKRecordingStopped:).

1.25.1:

iOS 5 compatibility fix for audio not resuming after an answered phone call is ended when NOT using headphones.

1.25:

iOS 5 compatibility fix for crash after declined phone call.

1.24:

iOS 5 compatibility fix.

1.23:

Better handling of On-Demand audio files shorter than the buffer wait time.

1.22:

Added new method: getAudioQueue

1.21:

Added new methods for supporting Live365 meta data feed

1.20.0:

Added new method:   setThrottleBypassTime:

1.19:

Added new method:   setPauseTimeout:

1.18:

Yet another bug fix related to On-Demand file playback. Corrected problem of buffer wrap-around when hitting the end of a long (30 minutes or more) low-bitrate (32K) file.

1.17:

Another bug fix related to On-Demand file playback. Corrected occasional false restart of playback when end of file was reached.

1.16:

Bug fixes related to On-Demand file play time calculations.

If the file contains an ID3 tag, the total time duration of the file is not calculated properly.  Also, fixed rare case of pausing an On-Demand file for a long time causing a miscalculation of the elapsed play time.

1.15:

Improved handling of AACv2.  Now properly drops down to AACv1 support if the device (such as iPhone 3G) does not support AACv2.

1.14:

Added setBufferWaitTimeForFile: method.

Minor bug fix.

1.13:

Fix to handle audio resume when interrupted while playing in the background.

1.12:

Added support for HE AACv2

Added support for playing local audio files via new "playLocalFile" method .

1.11:

Added support for audio level metering

Added support for out-of-band AudioCast metadata handling (used by some Icecast streams and Live365).

1.10:

Bug fix for audio interruption handling (such as incoming call) when audio was already in a paused state. Audio will no longer automatically resume playing if it was in a prior paused state. Also, if the interruption came from the iPod feature, resuming from a paused state now works properly.

Added support for SRKRealtimeMetaChanged:withUrl:

1.9:

Added support for SRKAudioWillBeSuspended protocol

Changed timing for callback for SRKAudioPlayStarted when resuming from a paused state – the callback gets called **after the pause state has been cleared.**

1.8:

Corrected callback behavior if a stream connection is lost and regained while audio playback is paused. Now properly calls SRKPlayPaused rather than SRKPlayStarted.