CHAPTER 1

Cisco StadiumVision Mobile API for Apple iOS

Revised: March 28, 2013

Introduction to Cisco StadiumVision Mobile API for Apple iOS

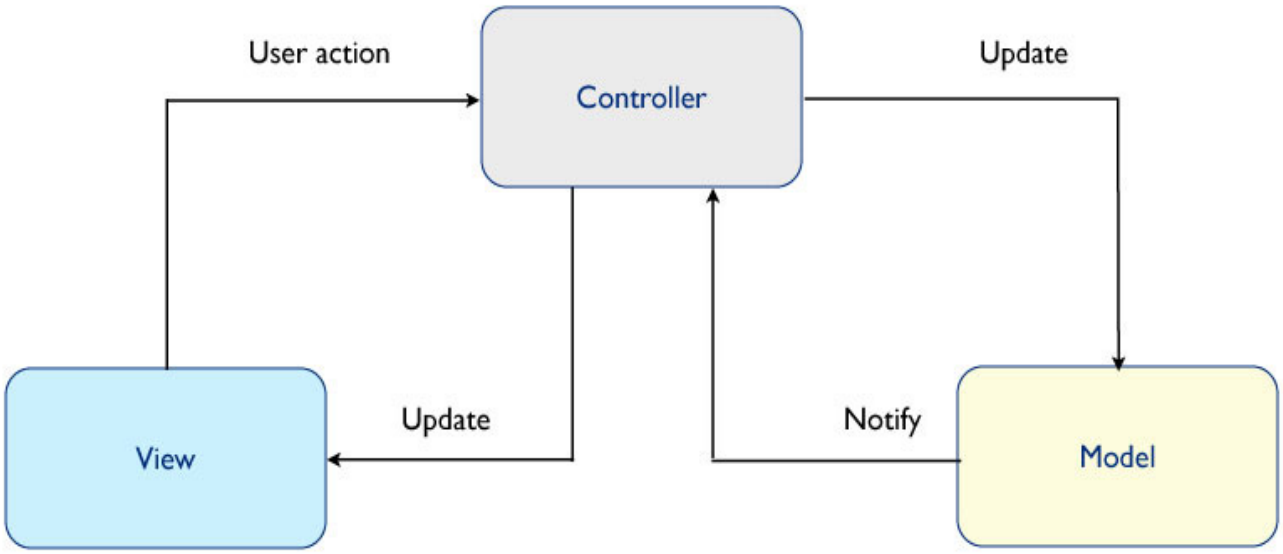
The iOS SDK is provided as a set of static libraries, header files, and an a sample iOS app (with a complete Xcode project). This API uses Objective-C classes and method calls to access the StadiumVision Mobile data distribution and video playback functionality within the StadiumVision Mobile iOS SDK library.

The Cisco StadiumVision Mobile client application supports Apple iOS 5.0 or later.

iOS Model View Controller (MVC) Design Pattern

The Model View Controller (MVC) design pattern separates aspects of an application into three distinct parts and defines how the three communicate. Figure 1-1 illustrates the Apple iOS MVC. As the name implies, the application is divided into three distinct parts: Model, View and Controller. The main purpose for MVC is reusability where you can reuse the same model for different views.

Figure 1-1 MVC Design Pattern



iOS API Prerequisites

Build Environment Requirements

Table 1-1 lists the various iOS SDK build environment requirements.

Table 1-1 Apple iOS Table 2.Build Environment Requirements

Tool	Version	Description	URL
Mac OSX	10.7 or later	A Mac is required to build an iOS application which includes the StadiumVision Mobile iOS SDK.	http://www.apple.com
Xcode	4.5.1 or later	Apple development IDE and tool kit.	http://developer.apple.com/xcode

Note Application developers will need to link against the libstd++ library in their build. This can be done by modifying the Build Settings->Linking->Other Linker Flags-> Add "-lstdc++" in Xcode.

Apple iOS SDK Overview

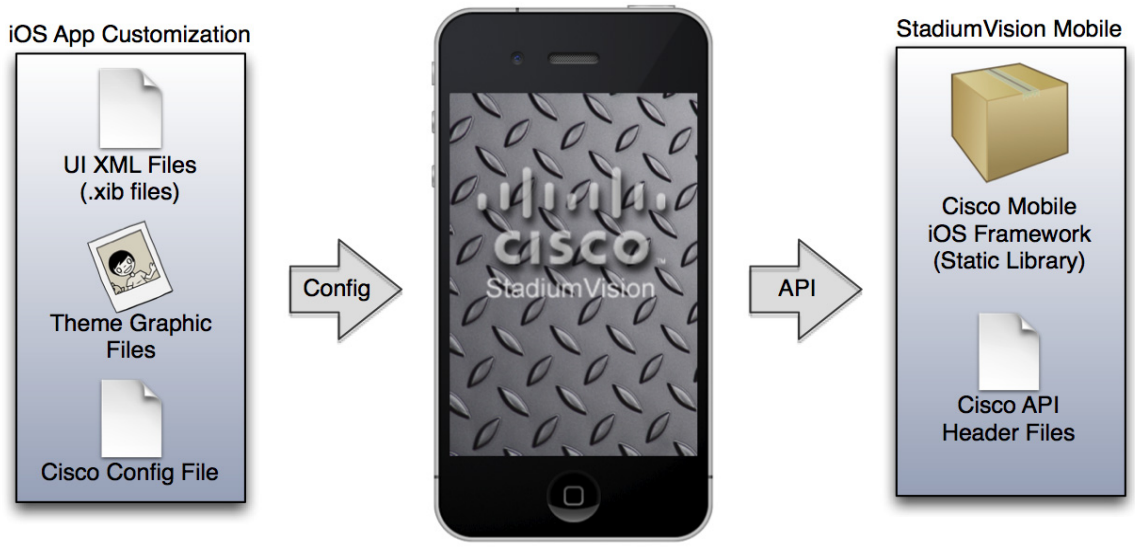
The Cisco StadiumVision Mobile iOS SDK contains the following components:

- A set of static libraries, header files, and an a sample iOS app (with a complete Xcode project)
- Customizable iOS SDK video player

Client Application Integration Overview

Figure 1-2 illustrates the high-level view of the Cisco StadiumVision iOS API libraries and common framework components. The left side of the graphic represents how to modifythe sample application, and the right reprsents how the SDK is packaged.

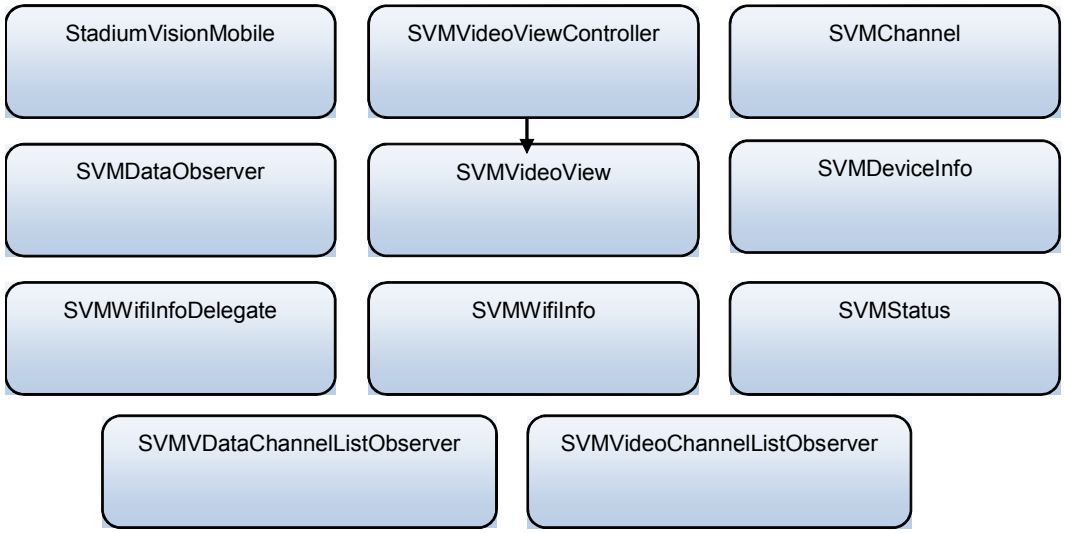
Figure 1-2 Cisco StadiumVision Mobile iOS SDK Components



Cisco StadiumVision Mobile iOS API Class Overview

The singleton "StadiumVisionMobile" class provides the top-level API to start, configure, and stop the framework. Video View Controller classes are provided to play the video channels and allow for customer customization. Figure 1-3 illustrates the StadiumVision Mobile API classes.

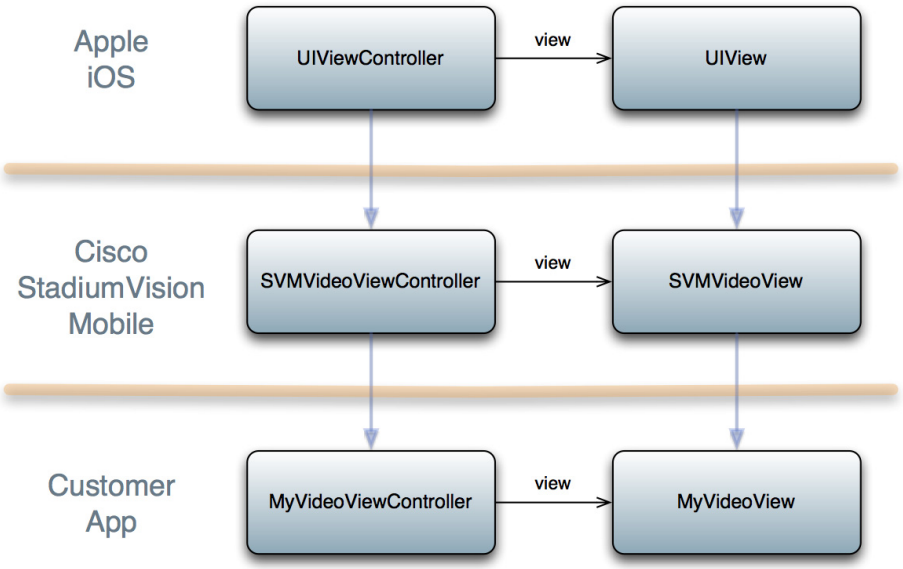
Figure 1-3 Cisco StadiumVision Mobile iOS API Classes



Video View Controller Inheritance

The iOS "UIViewController" and "UIView" classes are used as base classes. The customer application can extend the Cisco StadiumVision Mobile classes. Figure 1-4 illustrates the UIViewController and UIView classes.

Figure 1-4 Cisco StadiumVision Mobile Video Classes

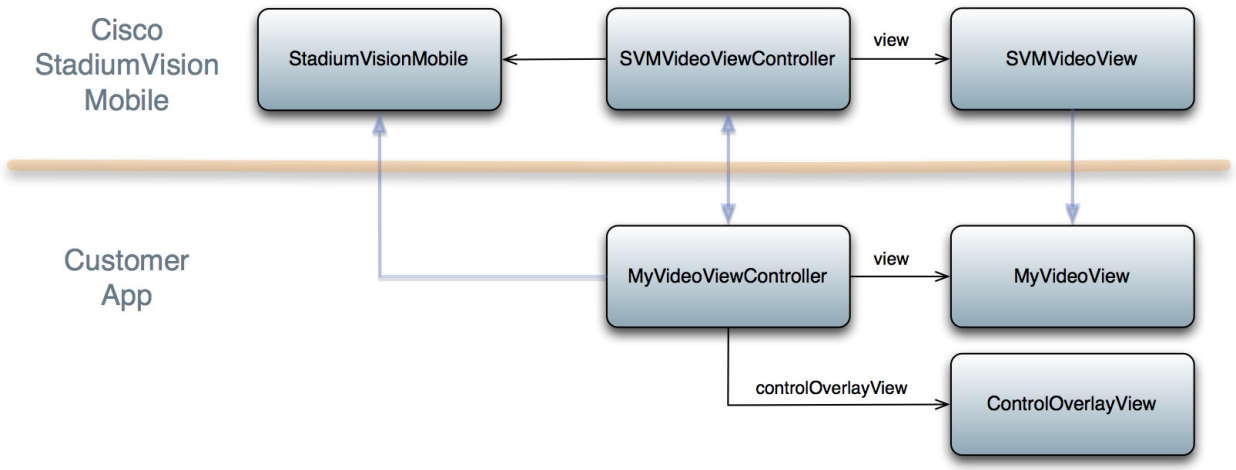


Cisco StadiumVision Mobile Application Classes

The Cisco StadiumVision Mobile application classes:

- Extends and customizes the SVMVideoViewController class
- Adds a UI overlay for controlling video playback (play, stop, close)
- Adds a UI overlay for displaying StadiumVision Mobile stats
- Handles gestures to display UI overlays with the MyVideoViewController class

Figure 1-5 Cisco StadiumVision Mobile Sample Application Classes



Cisco StadiumVision Mobile iOS API Summary

Table 1-2 summarizes the iOS API library. Following the summary are detailed tables for each API call.

Table 1-2 Cisco StadiumVision Mobile iOS API Summary

Return Type	API Method Name	API Method Description
StadiumVisionMobile*	sharedInstance	Gets a reference to the API singleton class used for all API calls
SVMStatus*	start	Starts the StadiumVision Mobile SDK
SVMStatus*	shutdown	Stops the StadiumVision Mobile SDK
SVMStatus*	addVideoChannelListDelegate	Registers a callback delegate to receive all video channel list updates
SVMStatus*	removeVideoChannelListDelegate	Unregisters the callback delegate from receiving the video channel list updates
SVMStatus*	addDataChannelListDelegate	Registers a callback delegate to receive all data channel list updates
SVMStatus*	removeDataChannelListDelegate	Unregisters the callback delegate from receiving the data channel list updates

Return Type	API Method Name	API Method Description
SVMStatus*	addDataChannelObserver	Registers an observer class to receive data for a particular data channel
SVMStatus*	removeDataChannelObserver	Unregisters an observer class from receiving data for a particular data channel
SVMStatus*	addDataChannelObserver:forChannel:	Registers an observer class to receive all data updates for a particular data channel
SVMStatus*	addDataChannelObserver:forChannel Name:	Registers an observer class to receive all data updates for a particular data channel name
SVMStatus*	removeDataChannelObserver:forChan nel:	Unregisters an observer class from receiving any data updates for a particular data channel
SVMStatus*	removeDataChannelObserver:forChan nelName:	Unregisters an observer class from receiving any data updates for a particular data channel
SVMStatus*	getVideoChannelListArray	Returns a snapshot array of the currently avaialable video channels.
SVMStatus*	getDataChannelListArray	Returns a snapshot array of the currently avaialable data channels.
NSDictionary	stats	Gets an NSDictionary of curernt StadiumVision Mobile SDK stats.
SVMStatus*	version	Gets the StadiumVision Mobile version string.

Cisco StadiumVision Mobile iOS API

The following tables describe each API call in more detail, including example usage.

Return Status Object

Each API call returns a SVMStatus object whenever applicable. Table 1-3 lists the SVMStatus object fields.

Table 1-3 SVMStatus class

Type	BOOL	NSString
Property	isOk	errorString
Description	Boolean indicating whether the API call was successful or not.	If the API call was not successful (isOk == NO), this string describes the error.

Example Usage	`// make an api call`
	`StadiumVisionMobile *svm = [StadiumVisionMobile sharedInstance];`
	`SVMStatus status = svm.start();`
	`// if an error occurred`
	`if (status.isOk == NO) {`
	`// display the error description`
	`NSLog(@"Error occurred: %@" + status.errorString);`	

Table 1-4 sharedInstance

Method Signature	`(StadiumVisionMobile*) sharedInstance`
Prerequisites	N/A
Notes	Class method that returns a reference to the StadiumVision Mobile API singleton class. The returned "StadiumVisionMobile" object reference is used for all subsequent StadiumVision Mobile API calls.
Result	N/A

Table 1-5 Start

Method Signature	`(SVMStatus*)start`
Prerequisites	N/A
Notes	This method starts the StadiumVision Mobile SDK. This will kick-off and start any required StadiumVision Mobile background threads and component managers.
Result	N/A

Table 1-6 addVideoChannelListDelegate

Method Signature	`(SVMStatus*) addVideoChannelListDelegate: (id)delegate`
Prerequisites	N/A
Notes	This method registers the given delegate class to receive all video channel list updates from the StadiumVision Mobile SDK.
Result	N/A

Table 1-7 setLogLevel

Method Signature	`StadiumVisionMobile *svm = [StadiumVisionMobile sharedInstance];` `[svm setLogLevel:SVM_API_LOG_DEBUG]`
Prerequisites	N/A
Notes	Sets the logging output level of the SDK, with the "DEBUG" level being more verbose than the "INFO" level.
Result	SVMStatus*

Table 1-8 removeVideoChannelListDelegate

Method Signature	`(SVMStatus*) addVideoChannelListDelegate: (id)delegate`
Prerequisites	N/A
Notes	This method unregisters the given delegate class from receiving any video channel list updates from the StadiumVision Mobile SDK.
Result	N/A

Table 1-9 addDataChannelListDelegate

Method Signature	`(SVMStatus*) addDataChannelListDelegate: (id)delegate`
Prerequisites	N/A
Notes	This method registers the given delegate class to receive all data channel list updates from the StadiumVision Mobile SDK.
Result	N/A

Table 1-10 removeDataChannelListDelegate

Method Signature	`removeDataChannelListDelegate`
Prerequisites	N/A
Notes	This method unregisters the given delegate class from receiving any data channel list updates from the StadiumVision Mobile SDK.
Example Usage	`(SVMStatus*) removeDataChannelListDelegate: (id)delegate`
Result	N/A

Table 1-11 addDataChannelListDelegate

Method Signature	`(SVMStatus*) addDataChannelListDelegate: (id)delegate`
Prerequisites	N/A
Notes	This method registers the given delegate class to receive all data channel list updates from the StadiumVision Mobile SDK.
Result	N/A

Table 1-12 removeDataChannelListDelegate

Method Signature	`(SVMStatus*) removeDataChannelListDelegate: (id)delegate`
Prerequisites	N/A
Notes	This method unregisters the given delegate class from receiving any data channel list updates from the StadiumVision Mobile SDK.
Result	N/A

Table 1-13 addDataChannelObserver

Method Signature	(SVMStatus*) addDataChannelObserver: (id<SVMDataObserver>)delegate
	`forChannel: (SVMChannel*)channel`
	`(SVMStatus*) addDataChannelObserver:(id<SVMDataObserver>)delegate forChannelName: (NSString*)channelName`
	The following example enables reception of the data announcements:
	`SVMChannel *selectedChannel1 = [dataChannelList objectAtIndex:0];`
	`[svm addDataChannelObserver:self forChannelName:selectedChannel1.name];`

Prerequisites	N/A
Notes	This method registers the given delegate class to receive all data for the given data channel object.
Result	N/A

Table 1-14 removeDataChannelObserver

Method Signature	`(SVMStatus*) removeDataChannelObserver: (id<SVMDataObserver>)delegate forChannel: (SVMChannel*)channel`
Prerequisites	N/A
Notes	This method unregisters the given delegate class from receiving any data for the given data channel name.
Result	N/A

Table 1-15 onData

Method Signature	`(void) onData:(NSData*)data withChannelName:(NSString*)channelName`
Prerequisites	N/A
Notes	This method is implemented by the customer app to support the "SVMDataObserver" protocol. This delegate method is used as a callback from the StadiumVision Mobile SDK. Each callback from the SDK to the customer app provides a received data message on the given data channel. The data channel message is delivered as an array of bytes (NSData).
Results	N/A

Table 1-16 Stats

Method Signature	`(NSDictionary*) stats`
Prerequisites	N/A
Notes	This method returns the StadiumVision Mobile SDK stats as a dictionary of name / value pairs. Stats are currently only available for the video channel (not data channels).
Result	N/A

Table 1-17 Stats API Hash Keys and Descriptions

Stats Hash Key	Stats Description
session_link_indicator	The health of the WiFi network connection. Ranges from 0 (poor) to 10 (excellent).
session_uptime	The length of time the session has been active (in seconds)
announcement_session_id	The video session announcement ID
announcement_session_title	The session announcement name
total_num_bytes_written	The total number of video bytes played
num_ts_discontinuities	The total number of MPEG2-TS packet discontinuities
num_dropped_video_frames	The total number of video frames dropped
protection_windows	The total number of protection windows sent

Stats Hash Key	Stats Description
window_no_loss	The total number of protection windows with no dropped video packets
window_recovery_successes	The total number of protection windows with recovered video packets
window_recovery_failures	The total number of protection windows that could not recover dropped packets. Recovery failure occurs when the number of received repair packets is less than the number of dropped video packets
window_warning	The total number of protection windows with more packets per window than the recommended value
window_error	The total number of protection windows with more packets per window than can be supported by Cisco StadiumVision Mobile.

Table 1-18 getVideoChannelListArray

Method Signature	`StadiumVisionMobile *svm = [StadiumVisonMobile sharedInstance];` `NSArray *currentChannels = [svm getVideoChannelListArray];`
Prerequisites	N/A
Notes	This method returns an array (NSArray*) of the currently available video channels (array of "SVMChannel" objects).
Result	NSArray* of SVMChannel objects

Table 1-19 getDataChannelListArray

Method Signature	`StadiumVisionMobile *svm = [StadiumVisonMobile sharedInstance];` `NSArray *currentChannels = [svm getDataChannelListArray];`
Prerequisites	N/A
Notes	This method returns an array (NSArray*) of the currently available data channels (array of "SVMChannel" objects)
Result	NSArray* of SVMChannel objects

Table 1-20 wifiInfo

Method Signature	`(SVMWifiInfo*) wifiInfo`
Prerequisites	N/A
Notes	This method returns the current WiFi network connection information. This information gets collected in the statistics information that gets uploaded to the Reporter server.
Result	N/A

The following tables contain properties are available within the SVMWifiInfo object.

Table 1-21 wifiInfo Object Properties

Stats Hash Key	Stats Description
session_link_indicator	The health of the WiFi network connection. Ranges from 0 (poor) to 10 (excellent).
session_uptime	The length of time the session has been active (in seconds)
announcement_session_id	The video session announcement ID

Table 1-22 version

Method Signature	`(NSString*) version`
Prerequisites	N/A
Notes	This method returns the StadiumVision Mobile SDK version string.
Result	N/A

The 'SVMVideoVideoController' class can be extended and customized. The SVMVideoVideoController API methods are listed in Table 1-23.

Table 1-23 Video View Controller API Summary

Return Type	API Method Name	API Method Description
void	setRenderVideoView	Sets the iOS UI video view where video frames will get rendered
SVMStatus	playVideoChannel	Starts playback of a particular video channel, changing channels on subsequent calls
SVMStatus	seekRelative	Moves the video playback buffer pointer relative to the current video playback buffer offset position
SVMStatus	seekAbsolute	Moves the video playback buffer pointer relative to the starting "live" video playback buffer offset position
SVMStatus	rewindForDuration	Rewinds the video playback buffer pointer relative to the current playback buffer offset position
SVMStatus	playLive	Moves the video playback buffer pointer to the head ("live") offset position in the video playback buffer

Table 1-24 Video View API Summary

Return Type	API Method Name	API Method Description
void	setRenderVideoView	Sets the iOS UI video view where video frames will get rendered
SVMStatus	playVideoChannel	Starts playback of a particular video channel, changing channels on subsequent calls
SVMStatus	seekRelative	Moves the video playback buffer pointer relative to the current video playback buffer offset position
SVMStatus	seekAbsolute	Moves the video playback buffer pointer relative to the starting "live" video playback buffer offset position

Return Type	API Method Name	API Method Description
SVMStatus	rewindForDuration	Rewinds the video playback buffer pointer relative to the current playback buffer offset position
SVMStatus	playLive	Moves the video playback buffer pointer to the head ("live") offset position in the video playback buffer

Table 1-25 setRenderVideoView

Method Signature	`(void)setRenderVideoView: (UIView*)aVideoView;`
Prerequisites	N/A
Notes	This method sets the target iOS video view (SVMVideoView) that will be used by the StadiumVision Mobile SDK to render video frames.
Result	N/A

Table 1-26 playVideo Channel

Method Signature	`(void)playVideoChannel:(SVMChannel*)channel;`
Prerequisites	N/A
Notes	This method plays the given video channel object. When subsequently called with a different video channel object, the video view controller will automatically stop the currently playing channel and start playback of the new channel
Result	N/A

Table 1-27 seekRelative

Method Signaure	`(void) seekRelative: (NSInteger)durationMs;`
Prerequisites	N/A

Method Signaure	`(void) seekRelative: (NSInteger)durationMs;`
Notes	• This method moves the video playback buffer pointer within the video history buffer for the given amount of time (in milliseconds) relative to its current position. • The StadiumVision Mobile SDK currently buffers 30 seconds of previously played video data that can be used for playing previously recorded video data. • A negative duration value rewinds the video play-head within the video history buffer. • A positive duration value forwards the video play-head towards the latest "live" video data in the video history buffer. • Should a duration be given (positive or negative) that is larger than the available size of the video history buffer, then the StadiumVision Mobile SDK move the video play-head as far as possible within the video history buffer.
Result	N/A

Table 1-28 seekAbsolute

Method Signature	`(void) seekAbsolute: (NSUInteger)durationMs;`
Prerequisites	N/A
Notes	• This method moves the video playback buffer pointer within the video history buffer for the given amount of time (in milliseconds) relative to the latest "live" video data. • The StadiumVision Mobile SDK currently buffers 30 seconds of previously played video data that can be used for playing previously recorded video data • A positive duration value moves the video play-head away from the latest "live" video data in the video history buffer. • Should a duration be given that is larger than the available size of the video history buffer, then the StadiumVision Mobile SDK move the video play-head to the end of the video history buffer.
Result	N/A

Table 1-29 playLive

Method Signature	`(void) playLive;`
Prerequisites	N/A
Notes	• This method forwards the video play-head to the starting "live" position at the beginning of the video data buffer. • This convenience method acts as a wrapper for the "seekAbsolute" API method; making "playLive()" equivalent to "seekAbsolute(0)".
Result	N/A

NS Notification Events

The StadiumVision Mobile SDK broadcasts the following iOS NSNotification events for use by the client application.

Table 1-30 NSNotification Event Properties

Event Constant	Description
kSVMVideoEventNotification	Constant defining the video event generated by the StadiumVision Mobile SDK
kSVMEventTypeVideoBufferingActive	Constant defining the "Video Buffering" type of video event
kSVMEventTypeVideoBufferingInactive	Constant defining the "Video Not Buffering" type of video event

The following source code registers to receive the Cisco video notifications:

```
#include "StadiumVisionMobile.h"
// register to handle the video buffering events
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(onVideoEvent:)
                                             name:kSVMVideoEventNotification
                                           object:nil];
```

The following source code handles the Cisco video notifications:

```
#include "StadiumVisionMobile.h"

// video event notification handler
 (void)onVideoEvent:(NSNotification*)notification {
    // get the passed "SVMEvent" object
    SVMEvent *event = [notification object];

    // determine the video event type
    switch (event.type) {
        case kSVMEventTypeVideoBufferingActive:
            // activate the UI "buffering" indicator
            break;
        case kSVMEventTypeVideoBufferingInactive:
            // deactivate the UI "buffering" indicator
            break;
    }
}
```

SDK Workflow

Starting the SDK

The StadiumVision Mobile SDK needs to be started at the application initialization by calling the "start" API method as in the following example:

```
#import "StadiumVisionMobile.h"
// get a reference to the StadiumVision Mobile API
StadiumVisionMobile *svm = [StadiumVisonMobile sharedInstance];
// start the StadiumVision Mobile SDK
[svm start];
```

Setting the Log Level

Sets the logging output level of the SDK, with the “DEBUG” level being more verbose than the “INFO” level. An example follows:

```
// start method sets logs to INFO by default
StadiumVisionMobile *svm = [StadiumVisionMobile sharedInstance];
[svm start];

// set the desired log level
[svm setLogLevel:SVM_API_LOG_DEBUG];
```

Getting the Video Channel List

The client application registers to receive callback whenever the video channel list is updated, as in the following example:

```
// register to receive video channel list updates
StadiumVisionMobile *svm = [StadiumVisonMobile sharedInstance];
[svm addVideoChannelListDelegate:self];
```

The StadiumVision Mobile SDK will callback the client application with any video channel list updates.

```
#import "StadiumVisionMobile.h"
// implement the "SVMChannelListObserver" protocol
@interface MyViewController : UIViewController <SVMChannelListObserver>
// video channel handler (array of 'SVMChannel' objects)
 -(void)onVideoChannelListUpdated:(NSArray*)channelList;
```

Presenting the Video Channel List

Each "SVMChannel" video channel object contains all of the information needed to display the channel list to the user.

Table 1-31 SVMChannel object properties

"SVMChannel" Property	Property Description
"name”	The name of the video channel
"bandwidthKbps”	The nominal video stream bandwidth (in kbps)
"sessionNum”	The session number of the channel
"channelText”	The complete text description of the video channel
“venueName”	The name of the venue.

"SVMChannel" Property	Property Description
contentOwner	The name of the content owner.
appDeveloper	The name of the application developer.

Playing A Video Channel

The example below demonstrates these actions:

- Selects a channel from the locally saved channel list
- Presents the video view controller modally
- Commands the video view controller to play the selected channel

```
#import "StadiumVisionMobile"

// get the user-selected video channel object
SVMChannel *selectedChannel = [videochannelList objectAtIndex:0];

NSLog(@"Selected Video Channel = %@", selectedChannel.name);

// create the video view controller
MyVideoViewController *myVC = [[MyVideoViewController alloc] init];

// present the modal video view controller
myVC.modalTransitionStyle = UIModalTransitionStyleCrossDissolve;
[self presentModalViewController:myVC animated:YES];

// play the selected video channel
[myVC playVideoChannel:selectedChannel];
```

Seeking Within the Video Buffer

The last 30 seconds of played video is stored in the device RAM. The following example jumps backwards 20 seconds in the video buffer (instant replay).

```
// get a reference to the api object
StadiumVisionMobile *svm = [StadiumVisonMobile sharedInstance];

// rewind 20 seconds
[svm rewindForDuration:-20000];
```

The example below jumps back to the top of the video buffer ("live" video playback):

```
// get a reference to the api object
StadiumVisionMobile *svm = [StadiumVisonMobile sharedInstance];
// play at the "live" video offset
[svm playLive];
```

Getting The Data Channel List

In the following example, the client application registers to receive callback whenever the data channel list is updated.

```
// register to receive data channel list updates
StadiumVisionMobile *svm = [StadiumVisonMobile sharedInstance];
[svm addDataChannelListDelegate:self];
```

In this example, the StadiumVision Mobile SDK will callback the client application with any data channel list updates:

```
#import "StadiumVisionMobile.h"
```

```
// implement the "SVMChannelListObserver" protocol
@interface MyViewController : UIViewController <SVMChannelListObserver>

// data channel handler (array of 'SVMChannel' objects)
 (void)onDataChannelListUpdated:(NSArray*)channelList;
```

Observing a Data Channel

In the following example, the registered class needs to implement the "SVMDataObserver" protocol:

```
#import "SVMDataObserver.h"
@interface DataChannelViewController : UIViewController <SVMDataObserver>
```

In this example, the "onData:withChannelName" method is called to push the received data to the registered class:

```
-(void)onData:(NSData*)data withChannelName:(NSString *)channelName {
    // convert the data bytes into a string
    NSString *dataStr = [[NSString alloc] initWithBytes:[data bytes]
                                                 length:[data length]
                                               encoding:NSUTF8StringEncoding];

    // display the data bytes and associated channel name
    NSLog(@"ChannelListViewController: onData callback: "
          "channelName = %@, data = %@", channelName, dataStr);

    [dataStr release];}
```

Getting the SDK Version String

The example below gets the StadiumVision Mobile SDK Version string:

```
#import "StadiumVisionMobile"

// get a reference to the api object
StadiumVisionMobile *svm = [StadiumVisonMobile sharedInstance];
// get the sdk version string
NSString *sdkVersion = [svm version];
```

Shutting Down the SDK (Optional)

The StadiumVision Mobile SDK automatically shuts-down and restarts based upon the iOS life-cycle notifications (NSNotifications). The client iOS application does not need to explicitly stop and restart the StadiumVision Mobile SDK. This 'shutdown' API is provided in case a customer use-case requires an explicit SDK shutdown.

```
#import "StadiumVisionMobile"

// get a reference to the api object
StadiumVisionMobile *svm = [StadiumVisonMobile sharedInstance];

// shutdown the StadiumVision Mobile SDK
[svm shutdown];
```

Video Player View Controller Customization

Default Cisco Video Player View Controller

The default Cisco video player has the following features:

- Implemented as a separate iOS "UIViewController"
- Support for fullscreen and partial-screen video views
- Video frames rendered using an iOS "UIView" and OpenGL layer (CAEAGLLayer)
- Customizable by extending the "SVMVideoViewController" class
- The Cisco demo app uses a customized video player

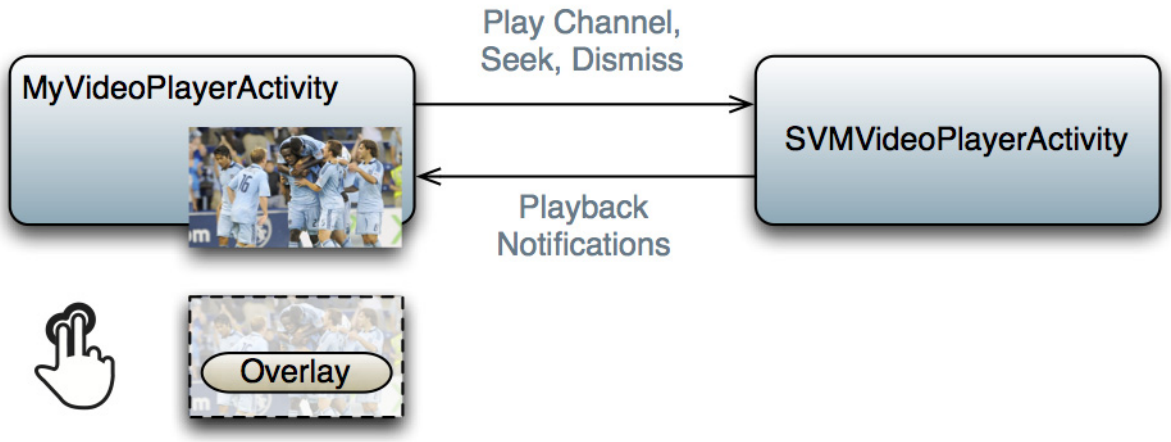
Customized Video Player

To customize the video player, extend the "SVMVideoViewController" base class as in the following example:

```
#import "SVMVideoViewController.h";

@interface MyVideoViewController : SVMVideoViewController {
}
```

Figure 1-6 Video Player Customization



Cisco Demo Customized Video Player

The demo customized video player has the following properties:

- Implemented as "MyVideoViewController"
- Extends the "SVMVideoViewController" class
- Handles all video overlays and gestures
- Single-tap gesture and "Back", "Rewind" / "Live" overlay buttons

- Two-finger double-tap gesture and stats overlay
- Uses the "MyVideoViewController~iphone.xib" to layout the screen
- Located in the "Customer App / App UI Resources / UI XML Files" Xcode project folder

The video view shown in Interface Builder is connected to the "videoView" property and is of class type "MyVideoView".

Configuration

Configuration Files

There are three configuration files that must be bundled with any iOS app using the StadiumVision Mobile SDK, as listed in the following table:

Table 1-32 Configuration Files

Configuration File Name	Description
"cisco_svm.cfg”	StadiumVision Mobile SDK configuration file that contains the "Field-of-Use" parameters and some optional WiFi network debugging information
"vompPlay.cfg”	Video decoder configuration file that contains the tuned decoding parameters. These settings should never be changed. Any changes could result in poor video or audio playback.

Field of Use Configuration

There are three "field-of-use" (also known as the triplet key) properties in the "cisco_svm.cfg" configuration file that need to be configured for each StadiumVision Mobile application: These fields must match the channel settings in the Cisco StadiumVision Mobile Streamer for the channels to be accessible by the application.

- Venue Name
- Content Owner
- App Developer

An example set of fields in the "cisco_svm.cfg" file is shown below:

```
{
    "license": {
        "venueName": "Stadium-A",
        "contentOwner": "Multi-Tenant Team-B",
        "appDeveloper": "Vendor-C"
    }
}
```

Wi-Fi Access Point Configuration

The "cisco_svm.cfg" configuration file can optionally include an array of WiFi AP information that will be used by the StadiumVision Mobile SDK for statistics reporting if available. Below is an example WiFi AP info entry in the "cisco_svm.cfg" configuration file:

```
{
    "network": {
```

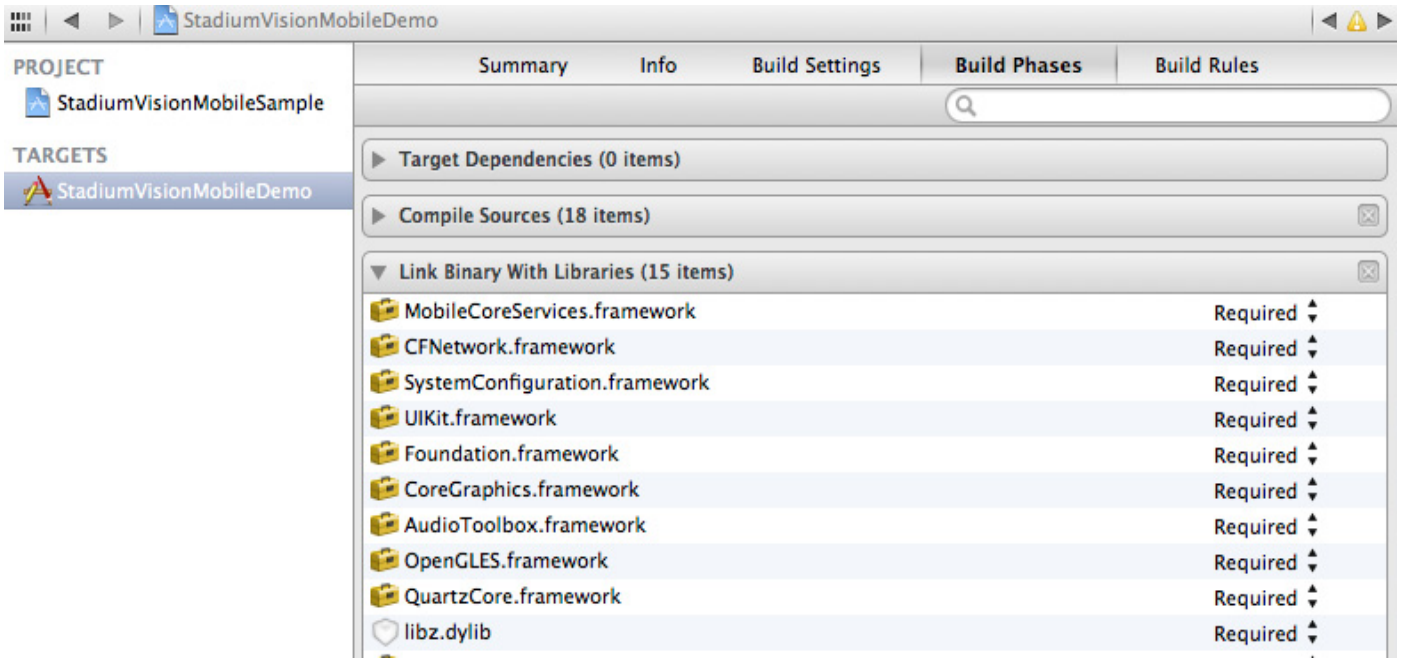
```
        "wifiApInfo": [
            {
                "name": "Press Box Booth 5",
                "bssid": "04:C5:A4:09:55:70"
            }
        ]
    }
}
```

CIntegration Checklist

The following list outlines integration steps for using the Cisco StadiumVision Mobile SDK.

1. Supported iOS version
   - Set the app's iOS version target set to iOS v4.0 or above
2. Copy configuration files
   - Copy the "cisco_svm.cfg" and vompPlay.cfg" config files, and the "voVidDec.dat" license file into the Xcode project.
3. Copy libraries
   - Copy the "libStadiumVisionMobile.a" and "libvoCTS.a" static libraries into the Xcode project.
4. Set the Xcode Project "Build Settings"
   - Add the "-ObjC" flag to the "Other Linker Flags" build setting. This ensures all Objective-C categories are loaded from the StadiumVision Mobile static library.
   - Add the "-lstdc++" flag to the "Other Linker Flags" build setting. This ensures that the C++ video decoder library is properly linked to the final app build.
5. Include Required iOS Libraries by adding frameworks in the target build phases pane of the Xcode project, under "Link Binary With Libraries" section, as shown in Figure 1-7.

Figure 1-7 Adding frameworks in Xcode



Required iOS Libraries

- UIKit.framework
- Foundation.framework
- CoreGraphics.framework
- AudioToolbox.framework
- OpenGLES.framework
- QuartzCore.framework
- CFNetwork.framework
- SystemConfiguration.framework
- MobileCoreServices.framework
- libz.dylib

What the SDK Handles

The StadiumVision Mobile SDK automatically handles the following events:

- Dynamic video channel discovery and notification
- Dynamic data channel discovery and notification
- Automatic SDK shutdown / restart in response to WiFi up / down events
- Automatic SDK shutdown / restart in response to iOS life-cycle events
- Management of multicast network data threads
- On-demand management of video / audio decoding threads
- Automatic statistics reporting to the StadiumVision Mobile Reporter server

Customer Application Roles

Figure 1-8 illustrates the roles of the customer application. The application must specify:

- Getting the list of video channels
- Displaying the list of video channels
- Handling user gestures for selecting video channels
- Adding video overlays and layouts
- Handling user gestures to control video overlays

Figure 1-8 Customer Application Responsibilities