Cisco StadiumVision Mobile SDK Guide for Apple iOS and Google Android

Release 1.2.0
March 28, 2013

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Cisco StadiumVision Mobile SDK Guide for Apple iOS and Google Android

CONTENTS

Preface

Revised: March 28, 2013

Table 1 Document Revision History

Date	Change Summary
March 28, 2013	Initial version of *Cisco StadiumVision Mobile SDK Guide for Apple iOS and Google Android*, Release 1.2

About This Guide

This guide describes the Cisco StadiumVision Mobile SDK for third-party developers whose applications will operate with the Cisco StadiumVision Mobile solution. These APIs are a mechanism to insert, retrieve, update, and remove data.

This document covers the Cisco StadiumVision SDKs, which supports both both Apple iOS and Google Android mobile operating systems.

Our implementations of Cisco StadiumVision Mobile SDK, and included sample application may change over time in response to the changing needs of our partner community. We will maintain backward compatibility whenever possible but advise you to expect differences in future releases. A list of changes will be provided for each release to keep API users aware of any necessary code changes that they will need to make.

About Cisco StadiumVision Mobile

Cisco StadiumVision Mobile (SVM) enables reliable and scalable delivery of low-delay video and data streams to WiFi devices at venues. A Venue Operator typically configures and operates SVM, Connected Stadium Wi-Fi and Connected Stadium components. The mobile app developer is responsible for obtaining the SVM SDK from Cisco, working with the Venue Operator on configuration dependencies and integrating the SVM Client.

Who Should Use This Guide

This guide is a technical resource for application developers who build custom user applications that extend Cisco StadiumVision Mobile. You should have an advanced level of understanding of web technology, operation, and terminology and be familiar with Cisco StadiumVision Mobile.

Application developers who use this application programming interface (API) should also have an understanding of the Objective-C language and Apple iOS, and Google Android application development.

Obtaining Source Code

Please contact your Cisco account team to become part of the StadiumVision Mobile SDK partner program.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly What's New in Cisco Product Documentation, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the What's New in Cisco Product Documentation as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS Version 2.0.

CHAPTER 1

Cisco StadiumVision Mobile API for Apple iOS

Revised: March 28, 2013

Introduction to Cisco StadiumVision Mobile API for Apple iOS

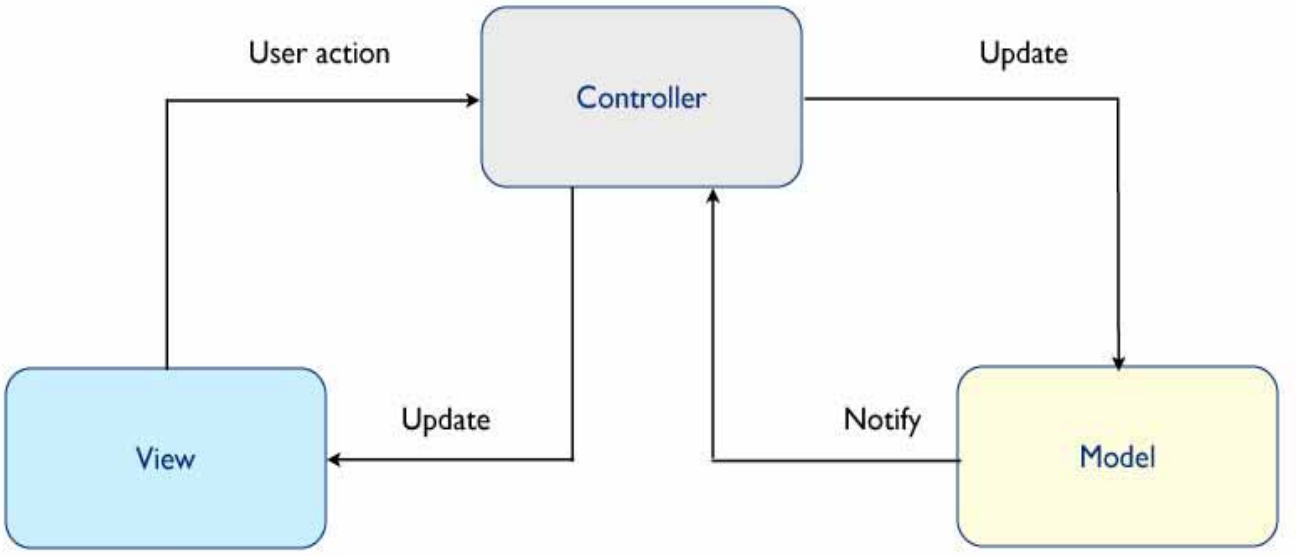
The iOS SDK is provided as a set of static libraries, header files, and an a sample iOS app (with a complete Xcode project). This API uses Objective-C classes and method calls to access the StadiumVision Mobile data distribution and video playback functionality within the StadiumVision Mobile iOS SDK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

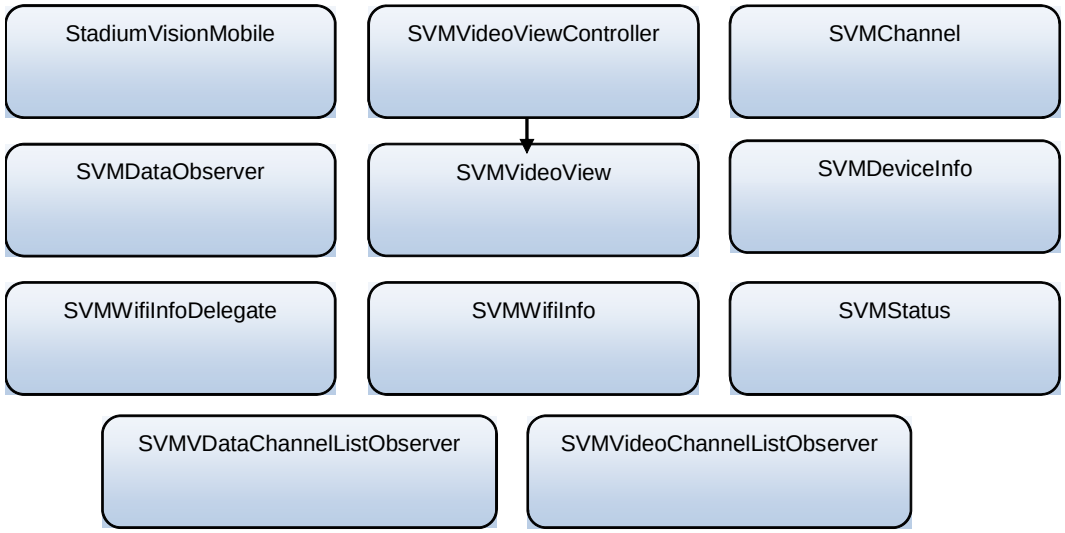
Figure 1-2 illustrates the high-level view of the Cisco StadiumVision iOS API libraries and common framework components. The left side of the graphic represents how to modifythe sample application, and the right reprsents how the SDK is packaged.

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.

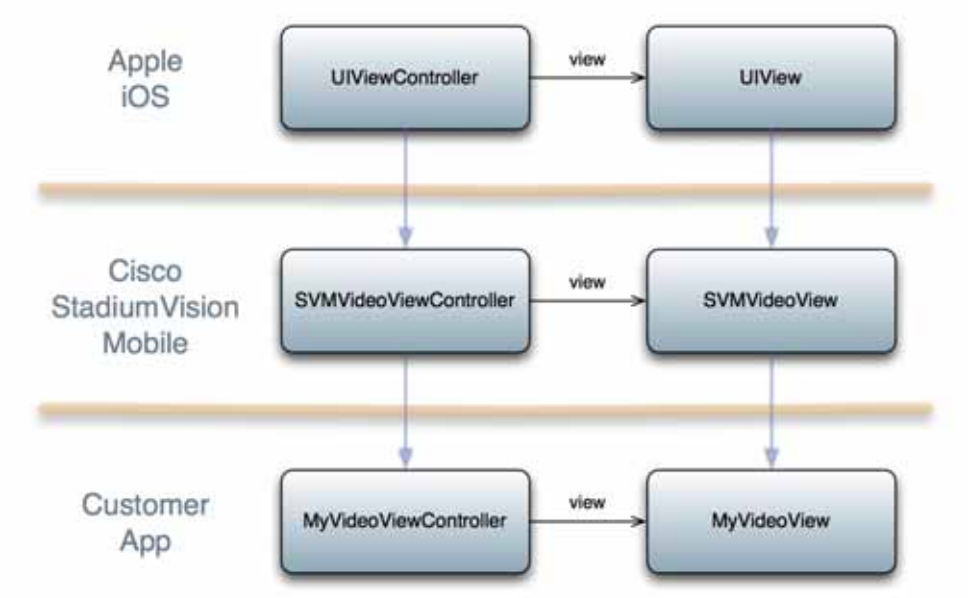
Figure 1-3 Cisco StadiumVision Mobile iOS API Classes



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

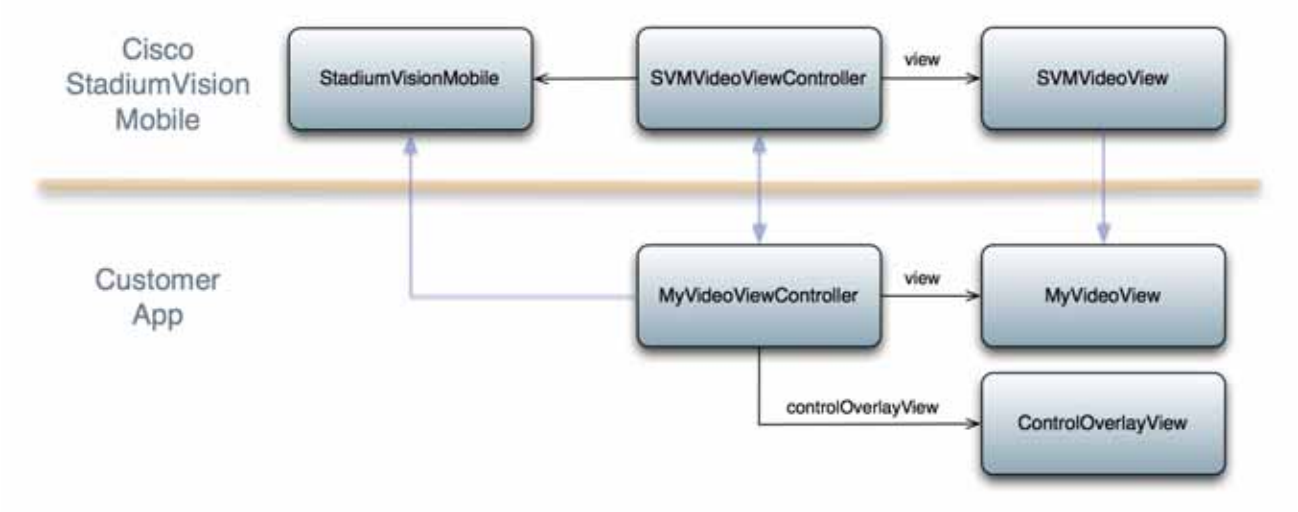


Cisco StadiumVision Mobile Application Classes

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

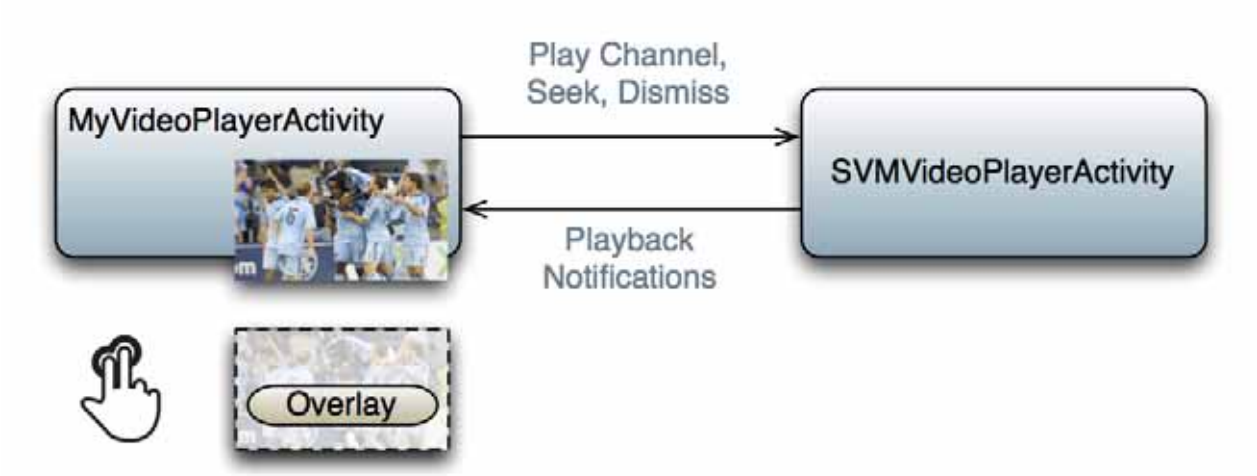
Customized Video Player

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": {
```

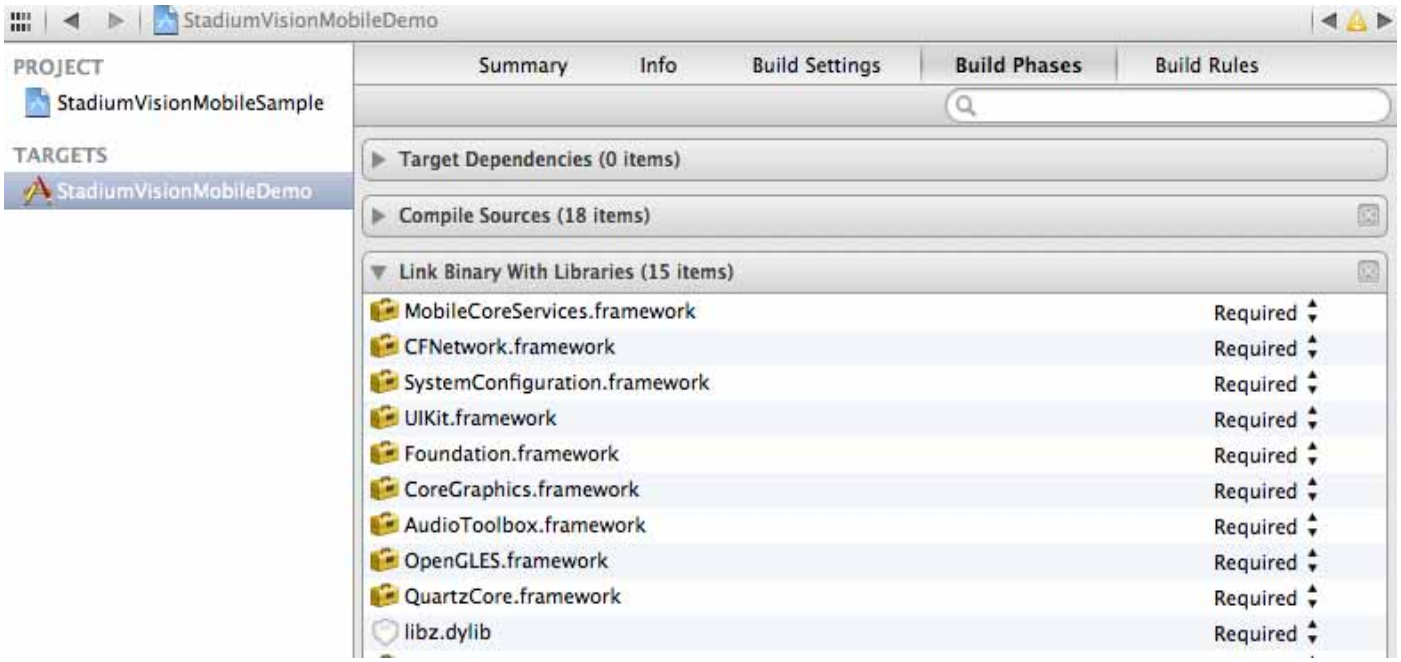
```
        "wifiApInfo": [
            {
                "name": "Press Box Booth 5",
                "bssid": "04:C5:A4:09:55:70"
            }
        ]
    }
}
```

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:

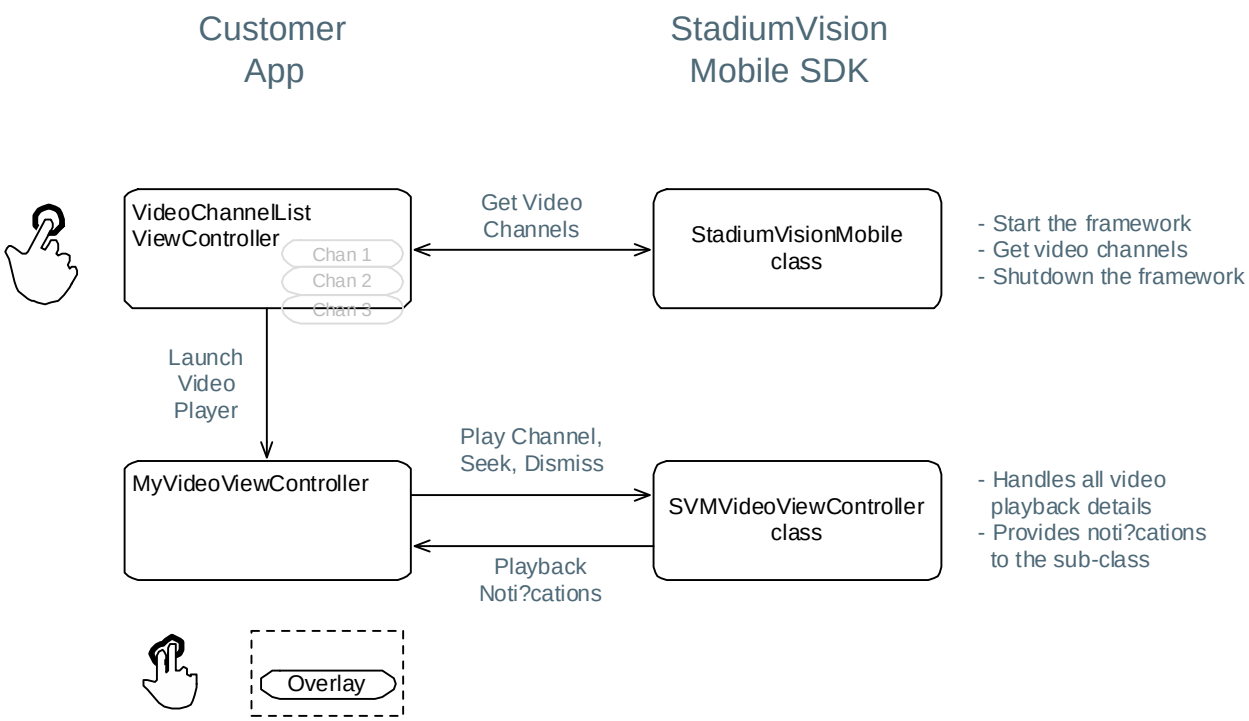
- Dynamic video channel discovery and notification
- Dynamic data channel discovery and notification
- Automatic SDK shutdown / restart in response to WiFi up / down events
- Automatic SDK shutdown / restart in response to iOS life-cycle events
- Management of multicast network data threads
- On-demand 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



CHAPTER 2

Cisco StadiumVision Mobile API for Google Android

March 28, 2013

Introduction to Cisco StadiumVision Mobile API for Google Android

The Cisco StadiumVision Mobile API uses Android and Java classes and method calls to access the StadiumVision Mobile data distribution and video playback functionality within the StadiumVision Mobile Android SDK library.

The Cisco StadiumVision Mobile client application supports Android 2.1 or later.

Android API Prerequisites

Build Environment Requirments

Table 2-1 lists the various Android SDK build environment requirements.

Table 2-1 Build Environment Requirements

Tool	Version	Description	URL
Mac or Windows PC			
Eclipse	3.7.1 (Indigo)	Eclipse "Classic" for Mac OSX (64-bit)	http://www.eclipse.org/downloads/

Android SDK Overview

The Cisco StadiumVision Mobile Android SDK contains the following components:

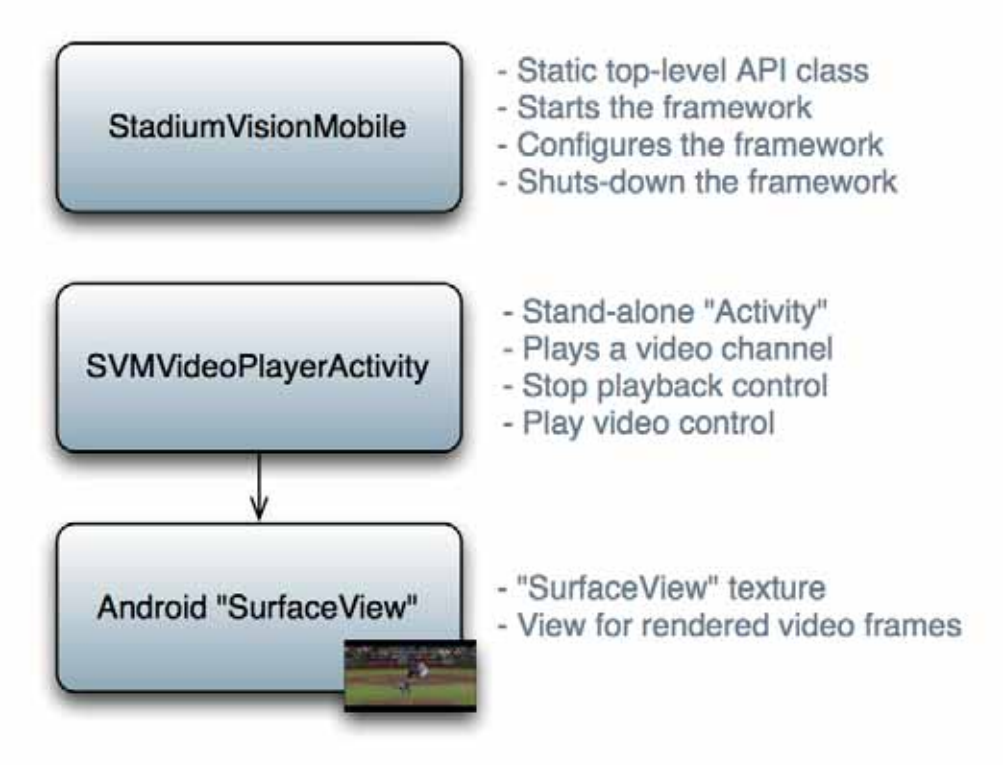
- A set of static libraries, configuration files, player layout XML files, and a sample Android application.
- Customizable video player

Cisco StadiumVision Mobile iOS API Class Overview

Figure 2-1 describes the three main Android API classes used in Cisco StadiumVision Mobile. The top-level StadiumVisionMobile class acts as a custom Android application context. An application context is a structure created within a screen or activity. There is no global state across an Android application.

Each SDK API method is called using the StadiumVisionMobileMETHOD-NAME class. The SVMVideoPlayerActivity class is a customizable stand-alone video player.

Figure 2-1 StadiumVision Mobile Class



Android OS Activity Overview

Figure 2-2 depicts the Android OS with regard to Activities. An Activity represents both the screen layout and controller code. A new Activity is launched by sending an Intent to the Android OS. An intent is a message to Android OS to launch a particular activity. Extra parameters contained in an Intent and are passed to an Activity. The back button is a hard device button used to generically display the previous Activity, and moves back down the Activity stack.

Figure 2-2 Android Activity Overview

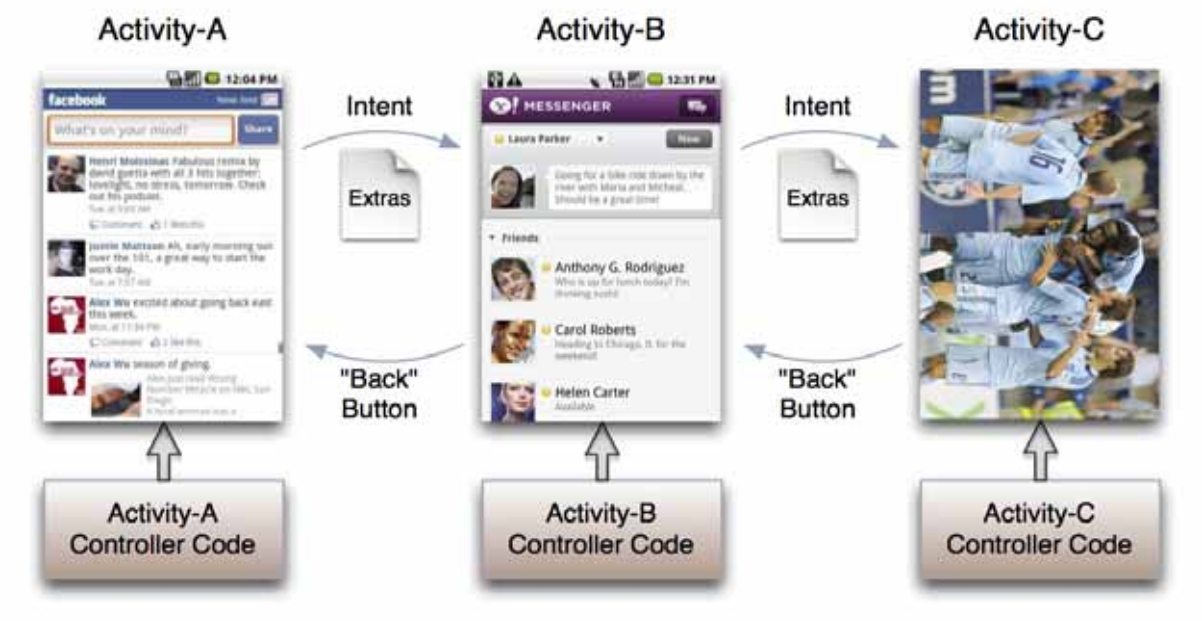
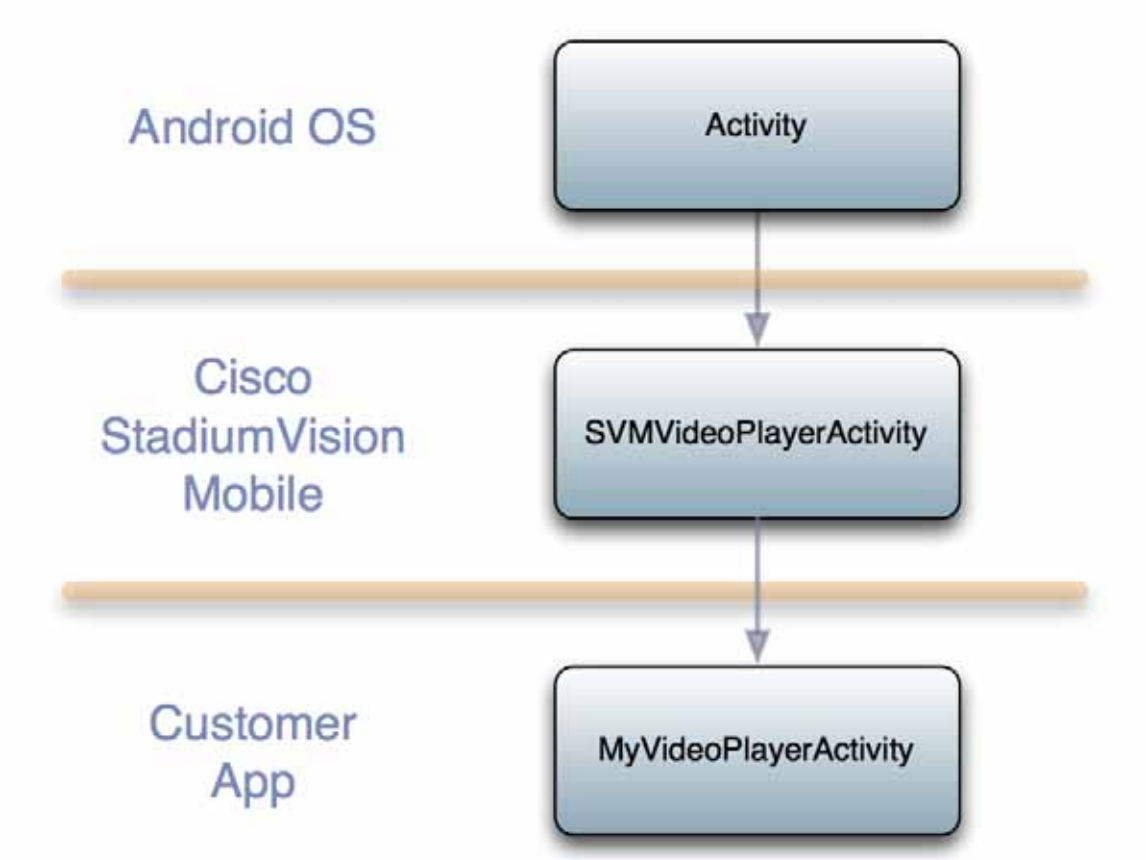


Figure 2-3 depicts the Activity inheritance between the Android OS, Cisco StadiumVision Mobile, and the client application.

Figure 2-3 Android Video Player Activity Inheritance



Cisco StadiumVision Mobile Android API Summary

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

Table 2-2 Cisco StadiumVision Mobile Android API Summary

Return Type	API Method Name	API Method Description
SVMStatus	start	Starts the StadiumVision Mobile SDK
SVMChannel[]	getVideoChannelArray	Get the array of available video channels
ArrayList<SVMChannel>	getVideoChannelArrayList	Get the array list of available video channels
SVMChannel[]	getDataChannelArray	Get the array of available data channels
ArrayList<SVMChannel>	getDataChannelArrayList	Get the array list of available data channels
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

Return Type	API Method Name	API Method Description
HashMap<String,String>	getStats	Gets a HashMap of the current StadiumVision Mobile SDK stats
void	onPause	Forwards each Android Activity's 'onPause' life-cycle notification to the StadiumVision Mobile SDK
void	onResume	Forwards each Android Activity's 'onResume' life-cycle notification to the StadiumVision Mobile SDK
SVMWifiInfo	getWifiInfo	Gets the current WiFi connection info
SVMBatteryInfo	getBatteryInfo	Gets the current battery info for the device
String[]	getLogLevelArray	Gets an array of the available StadiumVision Mobile SDK logging levels
ArrayList<String>	getLogLevelArrayList	Gets an array list of the available StadiumVision Mobile SDK logging levels
SVMStatus	setLogLevel	Set the StadiumVision Mobile SDK logging level
String	getLocalIpAddress	Convenience method to get the local device's IP address
String	getDeviceUUID	Gets the unique StadiumVision Mobile identifier for this device
String	getSessionUUID	Gets the unique StadiumVision Mobile identifier for this application session
String	sdkVersion	Property that contains the StadiumVision Mobile SDK version

Cisco StadiumVision Mobile Android API

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

Return Status Object

Each API call returns an 'SVMStatus' object whenever applicable. Table 2-3 lists the SVMStatus object fields.

Table 2-3 SVMStatus Object

Type	BOOL	String
Property	ok	error
Description	Boolean indicating whether the API call was successful or not.	If the API call was not successful (ok =false), this string describes the error.

Example Usage	`// make an api call`
	`SVMStatus status = StadiumVisionMobile.start();`
	`// if an error occurred`
	`if (status.ok == false) {`
	`// display the error description`
	`Log.e(TAG, "Error occurred: " + status.error);`	

Table 2-4 Start

Method Signature	`public static 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 2-5 getVideoChannelArray

Method Signature	`public static SVMChannel[] getVideoChannelArray();`
Prerequisites	N/A
Notes	This method returns a Java array of available video channels as 'SVMChannel' objects.
Result	N/A

Table 2-6 getDataChannelArrayList

Method Signature	`public static ArrayList<SVMChannel> getDataChannelArrayList();`
Prerequisites	N/A
Notes	This method returns a Java ArrayList of available data channels as 'SVMChannel' objects (using Java generics).
Result	N/A

Table 2-7 addDataChannelObserver

Method Signature	`public static ArrayList<SVMChannel> getDataChannelArrayList();`
Prerequisites	N/A
Notes	This method registers the given observer class to receive data for the given 'SVMChannel' data channel object.
Result	N/A

Figure 2-4 onData

Method Signature	`public void onData(String channelName, byte[] data)`
Prerequisites	N/A
Notes	This method is implemented by the customer app and 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 a byte array.
Result	N/A

Table 2-8 removeDataChannelObserver

Method Signature	`public static SVMStatus removeDataChannelObserver(String dataChannelName, ISVMDataObserver observer);`
Prerequisites	N/A
Notes	This method unregisters the given observer class to receive data for the given 'SVMChannel' data channel object.
Result	N/A

Table 2-9 getStats

Method Signature	`public static HashMap<String, String> getStats();`
Prerequisites	N/A
Notes	This method returns the StadiumVision Mobile SDK stats as a hash of name / value pairs.
Result	N/A

Table 2-10 lists the hash keys and stats description for the getStats API.

Table 2-10 getStats API Hash Keys and Stats Description

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
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

Stats Hash Key	Stats Description
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 StadiumVision Mobile.

Table 2-11 onPause

Method Signature	`public static void onPause();`
Prerequisites	N/A
Notes	• This method must be called by each individual client app Activity's "onPause()" method to inform the StadiumVision Mobile SDK of when a client app Activity has stopped. • Forwarding each client app Activity's "onPause()" life-cycle event allows the StadiumVision Mobile SDK to declare the client Android app as "active" and potentially restart all of the internal component managers and threads that use the device's CPU and networking resources.
Result	N/A

Table 2-12 onResume

Method Singature	`public static void onResume();`
Prerequisites	N/A
Notes	• This method must be called by each individual client app Activity's "onResume()" method to inform the StadiumVision Mobile SDK of when a client app Activity has started. • Forwarding each client app Activity's "onResume()" life-cycle event allows the StadiumVision Mobile SDK to declare the client Android app as "inactive" and shutdown all CPU and networking resources used by the StadiumVision Mobile SDK.
Result	N/A

Table 2-13 getWifiInfo

Method Signature	`public static SVMWifiInfo getWifiInfo();`
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

Table 2-14 getBatteryInfo

Method Signature	`public static SVMBatteryInfo getBatteryInfo();`
Prerequisites	N/A
Notes	• This method returns the current device battery information. • This information gets collected in the statistics information that gets uploaded to the Reporter server (if stats collection is enabled).
Result	N/A

Table 2-15 getLogLevelArray

Method Signature	`public static String[] getLogLevelArray();`
Prerequisites	N/A
Notes	This method provides a Java array of available logging levels that can be applied to any component.
Result	N/A

Table 2-16 getLogLevelArrayList

Method Signature	`public static ArrayList<String> getLogLevelArrayList();`
Prerequisites	N/A
Notes	This method provides a Java ArrayList of available logging levels that can be applied to any component.
Result	N/A

Table 2-17 setLogLevel

Method Signature	`public static SVMStatus setLogLevel(LogLevel level);`
Prerequisites	N/A
Notes	This method sets the global logging level for the entire StadiumVision Mobile SDK, with all internal components getting their logging level set to the same level.
Result	N/A

Table 2-18 getLocalIpAddress

Method Signature	`public static String getLocalIpAddress();`
Prerequisites	N/A
Notes	This method returns this device's local IP address.
Result	N/A

Table 2-19 getDeviceUUID

Method Signature	`public static String getDeviceUUID();`
Prerequisites	N/A
Notes	• This method returns the device UUID that was generated by the StadiumVision Mobile SDK and saved in the app's shared preferences. • Android does not provide a consistent and reliable device UUID across all of the Android OS versions supported by the StadiumVision Mobile SDK, so a generated device UUID is used instead.
Result	N/A

Table 2-20 getAppSessionUUID

Method Signature	`public static String getAppSessionUUID();`
Prerequisites	N/A
Notes	• This method returns the app session UUID that is generated by the StadiumVision Mobile SDK. • This UUID uniquely identifies each time the StadiumVision Mobile SDK is started and is used for consistent statistics collection and reporting.
Result	N/A

Table 2-21 sdkVersion

Method Signature	`public static String sdkVersion;`
Prerequisites	N/A
Notes	This class property contains StadiumVision Mobile SDK version string.
Result	N/A

Video Player Activity API Summary

The SVMVideoPlayerActivity class can be extended and customized. Table 2-22 lists the SVMVideoPlayerActivity API methods.

Table 2-22 Video Player Activity API Summary

Return Type	API Method Name	API Method Description
SVMStatus	setVideoSurfaceView	Sets the Android UI “SurfaceView” where video frames will get rendered
SVMStatus	playVideoChannel	Starts playback of a particular video channel, changing channels on subsequent calls
SVMStatus	seekRelative	Seeks the playback buffer pointer relative to the current playback buffer offset position
SVMStatus	seekAbsolute	Seeks the playback buffer pointer from the head (“live”) offset position of the video playback buffer
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
SVMStatus	shutdown	Shuts-down and dismisses the video player Activity

Table 2-23 setVideoSurfaceView

Method Signature	`public static String sdkVersion;`
Prerequisites	N/A
Notes	This class property contains StadiumVision Mobile SDK version string.
Example Usage	
Result	N/A

Table 2-24 seekRelative

Method Signature	`public SVMStatus seekRelative(int durationMs);`
Prerequisites	N/A
Notes	This method moves the video play-head pointer forward and backward in time relative to its current position in the video history buffer.
Result	N/A

Table 2-25 seekAbsolute

Method Signature	`public SVMStatus seekAbsolute(int durationMs);`
Prerequisites	N/A
Notes	• This method moves the video play-head pointer to beginning of stream; relative to the "live" position. • To play most current live video pass in on offset of zero (0 ms). • To play most current live video pass in on offset of zero (0 ms). • To play video in the past, a positive duration will be used as an offset for rewinding back in time (relative to the "live" position).
Result	N/A

Table 2-26 rewindForDuration

Method Signature	`public SVMStatus rewindForDuration(int durationMs);`
Prerequisites	N/A
Notes	• This method rewinds the video play-head within the video history buffer for the given amount of time (in milliseconds) • Should a duration be given that is larger than the size of the video history buffer, the StadiumVision Mobile SDK will rewind the video play-head as far as possible • This convenience method acts as a wrapper for the "seekRelative" API method; making the given "durationMs" value negative before calling "seekRelative". For example, "rewindForDuration(20000)" is equivalent to "seekRelative(-20000)".
Result	N/A

Table 2-27 playLive

Method Signature	`public SVMStatus 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

Table 2-28 shutdown

Method Signature	`public SVMStatus shutdown();`
Prerequisites	N/A
Notes	This method stops video playback of the currently playing video channel by stopping the native player, native decoder, and terminating this Android Activity.
Result	N/A

SDK Workflow

Starting the SDK

Start the StadiumVision Mobile SDK from the application's main Android launch Activity, as shown in the following example.

```
import com.cisco.svm.app.StadiumVisionMobile;

// app's launch activity 'onCreate' notification
void onCreate() {

   // call the parent method
   super.onCreate();

   // start the StadiumVision Mobile SDK
   StadiumVisionMobile.start();
}
```

Getting the Video Channel List

The StadiumVision Mobile SDK dynamically receives all of the available channels (via WiFi multicast). The client application gets an array of channel objects (SVMChannel[]) through the "getVideoChannelArray" API call, as shown in the following example:

```
import com.cisco.svm.app.StadiumVisionMobile;

// get the list of available video channels
SVMChannel[] channels = StadiumVisionMobile.getVideoChannelArray();

// display some channel information
```

```
Log.d(TAG, "Channel Name = "       + channels[0].name);
Log.d(TAG, "Channel Bandwidth = " + channels[0].bandwidthKbps);
Log.d(TAG, "Channel Body Text = "  + channels[0].bodyText);
```

Presenting the Video Channel List

Each "SVMChannel" video channel object contains all of the information needed to display the channel list to the user. The SVMChannelObject properties and descriptions are shown in Table 2-29.

Table 2-29 SVMChannel Object Properties

"SVMChannel" Property	Property Description
"name"	The name of the video channel
"bandwidthKbps"	The data bandwidth consumed by the video channel (in kbps)
"sessionNum"	The session number of the channel
"channelText"	The complete text description of the video channel

Playing a Video Channel

The following example shows playing a video channel, and performs the following actions:

- Selects a channel from the locally saved channel list
- Starts video playback of the channel by launching the custom video player Activity ("MyVideoPlayer")

Note The "SVMChannel" object is parcelable (instances can be written to and restored from a parcel).

Seeking Within the Video Buffer

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

```
public class MyVideoPlayerActivity extends SVMVideoPlayerActivity {

   // seek backwards 20 seconds in the video buffer
   super.seekRelative(-20000);
}
```

The following example shows jumping back to the top of the video buffer ("live" video playback):

```
public class MyVideoPlayerActivity extends SVMVideoPlayerActivity {

   // seek to the top of the video buffer (0 ms offset)
   super.seekAbsolute(0);
}
```

Setting the Video Dimensions

The video region is rendered within a SurfaceView. The video region is configured using standard Android layout XML files. The video region can be set to full screen or to specific pixel dimensions

Fullscreen Video Layout

The XML layout file below shows how to configure the video ‘SurfaceView’ to fill the entire screen, as shown in the following example:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="@drawable/black">

   <SurfaceView
      android:id="@+id/videoSurfaceView"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_centerInParent="true">
   </SurfaceView>

</RelativeLayout>
```

Partial-Screen Video Layout

The XML layout file below shows how to configure the video ‘SurfaceView’ to specific pixel region, as shown in the following example:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="@drawable/black">

   <SurfaceView
      android:id="@+id/videoSurfaceView"
      android:layout_width=”320px"
      android:layout_height=”240px"
      android:layout_centerInParent="true">
   </SurfaceView>

</RelativeLayout>
```

Getting the Data Channel List

The StadiumVision Mobile SDK dynamically receives all of the available data channels (via WiFi multicast). The client application gets an array of channel objects (SVMChannel[]) through the “getDataChannelArray” API call, as shown in the following example:

```
import com.cisco.svm.app.StadiumVisionMobile;

// get the list of available data channels
SVMChannel[] channels = StadiumVisionMobile.getDataChannelArray();

// display some channel information
Log.d(TAG, “Channel Name = “      + channels[0].name);
Log.d(TAG, “Channel Bandwidth = “ + channels[0].bandwidthKbps);
Log.d(TAG, “Channel Body Text = “  + channels[0].bodyText);
```

Observing a Data Channel

Any data channel can be observed by registering a class to receive callbacks for all data received on that channel. The registered class needs to implement the “ISVMDataObserver” interface, as shown in the following example:

```
import com.cisco.svm.data.ISVMDataObserver;

public class MyDataViewerActivity extends Activity implements ISVMDataObserver {
   ...
}
```

The “onData” method is called to push the received data to the registered class, as shown in the following example:

```
public void onData(String channelName, byte[] data) {
   // display the received data parameters
   Log.d(TAG, "DATA CALLBACK: " +
         "channel name = " + channelName + ", " +
         "data length  = " + data.length);

}
```

Activity Life-Cycle Notifications

The client app needs to notify the StadiumVision Mobile SDK of it’s life-cycle notifications. This allows the StadiumVision Mobile SDK to automatically shutdown and restart as needed. Each client Activity needs to forward its life-cycle notifications, as shown in the following example:

```
import com.cisco.svm.app.StadiumVisionMobile;


void onPause() {

   // notify the cisco sdk of the life-cycle event

   StadiumVisionMobile.onPause();

}


void onResume() {

   // notify the cisco sdk of the life-cycle event

   StadiumVisionMobile.onResume();
```

Video Player Customization

This section describes customizing the video player.

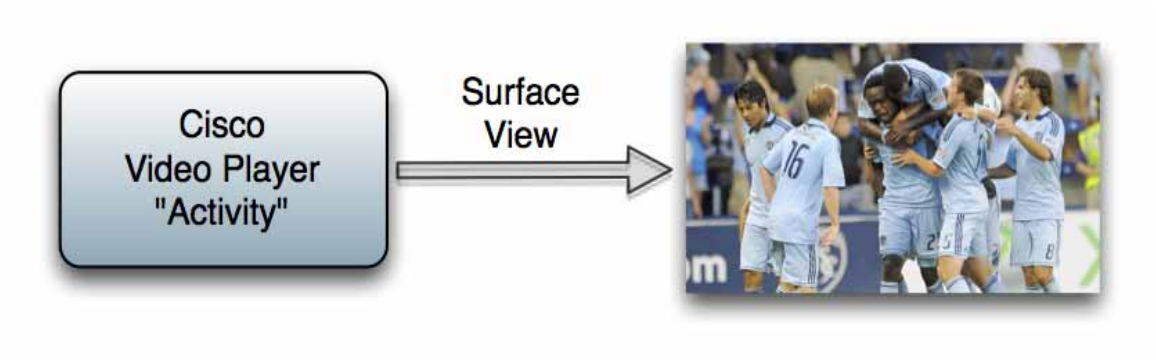
Default Cisco Video Player

The default Cisco video player has the following features:

- Implemented as a separate Android “Activity”
- Supports fullscreen and partial-screen video views

- Renders video frames using an Android “SurfaceView”
- Customizable by extending the “SVMVideoPlayerActivity” class
- Uses a customized video player

Figure 2-5 Default Cisco Video Player



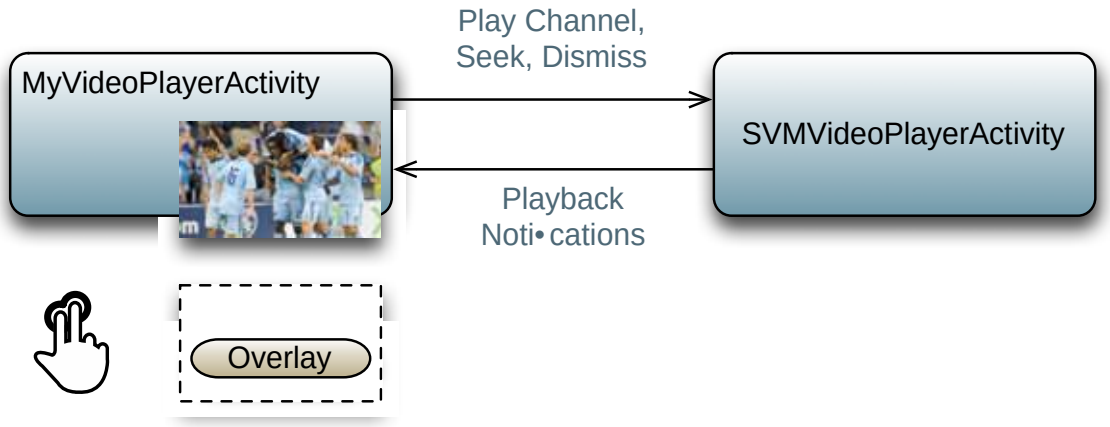
Customized Video Player

The customized video play extends the “SVMVideoPlayerActivity” base class, as shown below:

```
import com.cisco.sv.media.SVMVideoPlayerActivity;

public class MyVideoPlayer extends SVMVideoPlayerActivity {
}
```

Figure 2-6 SVMVideoPlayerActivity API



You need to register the new custom Activity in “AndroidManifest.xml, as shown below:

```
<activity android:label="@string/app_name”
          android:name="com.company.MyVideoPlayer”
          android:screenOrientation="landscape"
          android:configChanges="orientation|keyboardHidden"
          android:theme="@android:style/Theme.NoTitleBar.Fullscreen”>
</activity>
```

Cisco Demo Customized Video Player

The Cisco demo video player:

- Implemented as “MyVideoPlayerActivity”
- Extends the “SVMVideoPlayerActivity” class
- Handles all video overlays and gestures
- Uses standard Android XML layout files (“layout/player.xml”)

The video player’s XML layout file defines:

- The “SurfaceView” video rendering area
- Any transparent video overlays
- Play / Pause / Rewind button graphic files
- Animations used to show / hide the transport controller

Configuration

The following section describes the required configuration.

Configuration Files

There are three configuration files that must be bundled with any Android app using the StadiumVision Mobile SDK.

Table 2-30 Configuration Files

Config 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. The three “field-of-use” properties in the “cisco_svm.cfg” configuration file that need to be configured for each StadiumVision Mobile application are: • Venue Name • Content Owner • App Developer
“vompPlay.cfg”	Video decoder config file that contains the tuned decoding parameters. These settings should never be changed. Any changes could result in poor video or audio playback.
“voVidDec.dat”	Video decoder license file.

An example set of fields in the “cisco_svm.cfg” file is shown below.These fields must match the channel settings in the Cisco “Streaming Server” for the channels to be accessible by the application.

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

WiFi AP Info Configuration (Optional)

The “cisco_svm.cfg” config 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” config file:

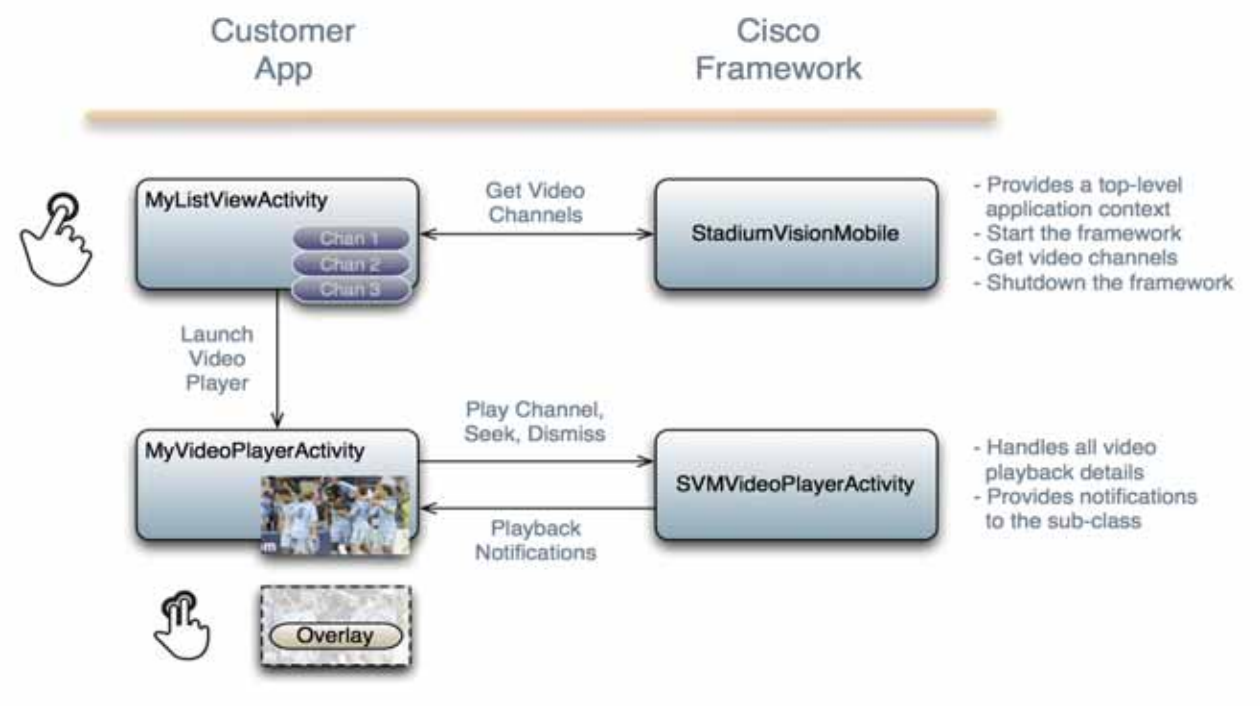
```
{
    "network": {
        "wifiApInfo": [
            {
                "name": "Press Box Booth 5",
                "bssid": "04:C5:A4:09:55:70"
            }
        ]
    }
}
```

Integration

Client Application Integration Overview

This section describes customizing the StadiumVision Mobile application, and contains the following subsections:

Figure 2-7 Cisco StadiumVision Mobile Integration Overview



Integration Checklist

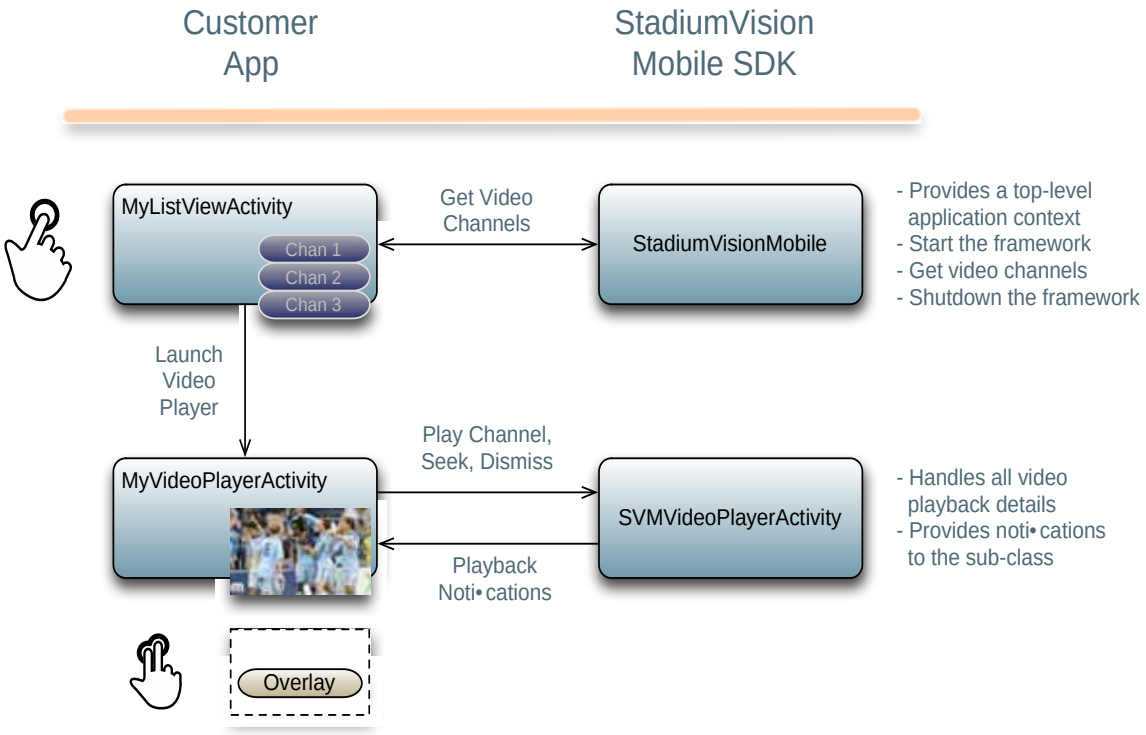
1. Supported Android OS Version
   - Set the app's Android version target to v2.1u1 or above
2. Android App Permissions
   - Add the required permissions to "AndroidManifest.xml"
3. Copy Config Files
   - Add the config files to the app's "assets" folder
4. Copy Libraries
   - Add the Java and native libraries to the app's "libs" folder
5. Set a Video "SurfaceView"
   - Add a "SurfaceView" to the player Activity's layout XML file
6. Life-Cycle Notifications
   - Forward life-cycle notifications to the StadiumVision Mobile SDK
7. Android Project Build Paths
   - Set the project build path to include the Jar files in "./libs/"

Customer Application Roles

Figure 2-8 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 overlay

Figure 2-8 Customer Application Responsibilities



Android Permissions

The following Android permissions are needed by the StadiumVision Mobile SDK. Each permission is set in the "AndroidManifest.xml" file.

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
```

SDK Java Libraries

Each Java JAR library needs to be included in the Android app's "libs" folder, as shown in the following example.

- Cisco StadiumVision Mobile Android SDK
- Apache HTTP Client 4.1

- Jackson JSON 1.8.1

```
./libs/StadiumVisionMobile.jar
./libs/httpclient-4.1.1.jar
./libs/httpcore-4.1.jar
./libs/httpmime-4.1.1.jar
./libs/jackson-core-lgpl-1.8.1.jar
./libs/jackson-mapper-lgpl-1.8.1.jar
```

SDK Native Libraries

Each library needs to be included in the Android app's "libs/armeabi" folder.

```
./libs/armeabi/libsvm-android.a
./libs/armeabi/libvoAACDec.so
./libs/armeabi/libvoAACDec_v7.so
./libs/armeabi/libvoH264Dec.so
./libs/armeabi/libvoH264Dec_v7.so
./libs/armeabi/libvoLiveSrcCTS.so
./libs/armeabi/libvoLiveSrcCTS_v7.so
./libs/armeabi/libvoMMCCRRS.so
./libs/armeabi/libvoMMCCRRS_v7.so
./libs/armeabi/libvoTsParser.so
./libs/armeabi/libvoTsParser_v7.so
./libs/armeabi/libvoVidDec.so
./libs/armeabi/libvojni_svmobile.so
./libs/armeabi/libvojni_vome2_sw_v20.so
./libs/armeabi/libvojni_vome2_sw_v22.so
./libs/armeabi/libvojni_vome2_sw_v23.so
./libs/armeabi/libvojni_vome2_sw_v30.so
./libs/armeabi/libvojni_vome2_sw_v31.so
./libs/armeabi/libvompEngn.so
```

Android Project Classpath

To add Java JAR files to your Eclipse project, use the following steps:

Step 1 Right-click your project in Eclipse

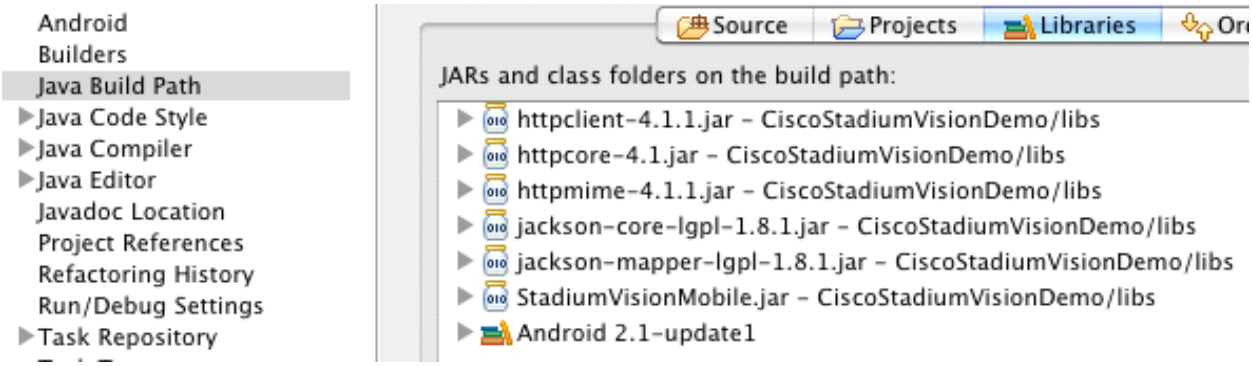
Step 2 Select "Properties"

Step 3 Select "Java Build Properties"

Step 4 Select "Add JARs"

Step 5 Add each of the Java JAR files listed in Adding Java JAR Files in Eclipse14.

Figure 2-9 Adding Java JAR Files in Eclipse



Your “classpath” file should look like the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
    <classpathentry kind="src" path="src"/>
    <classpathentry kind="src" path="gen"/>
    <classpathentry kind="con" path="com.android.ide.eclipse.adt.ANDROID_FRAMEWORK"/>
    <classpathentry kind="lib" path="libs/httpclient-4.1.1.jar"/>
    <classpathentry kind="lib" path="libs/httpcore-4.1.jar"/>
    <classpathentry kind="lib" path="libs/httpmime-4.1.1.jar"/>
    <classpathentry kind="lib" path="libs/jackson-core-lgpl-1.8.1.jar"/>
    <classpathentry kind="lib" path="libs/jackson-mapper-lgpl-1.8.1.jar"/>
    <classpathentry kind="lib" path="libs/StadiumVisionMobile.jar"/>
    <classpathentry kind="output" path="bin"/>
</classpath>
```

App Obfuscation Using ProGuard

If you choose to obfuscate your application with ProGuard, consider the following points:

- Use the latest version of ProGuard (which is version 4.7 as of August, 2012)
- If a crash takes place that you would like Cisco to analyze, please run retrace.jar on the stack trace output with your map file before sending us the un-winded stack trace file.
- Specify our libraries as input jars with “-libraryjars”. See the example below and remember to modify the paths as needed:

```
-libraryjars ./libs/httpclient-4.1.1.jar
-libraryjars ./libs/httpcore-4.1.jar
-libraryjars ./libs/httpmime-4.1.1.jar
-libraryjars ./libs/jackson-core-lgpl-1.8.1.jar
-libraryjars ./libs/jackson-mapper-lgpl-1.8.1.jar
-libraryjars ./libs/StadiumVisionMobile.jar
-libraryjars ./libs/StadiumVisionMobileSender.jar
```

If you extend or implement any of our classes or interfaces please specify that in the config file,, as shown in the following example:

```
-keep public class * extends com.cisco.svm.data.ISVMDataObserver
Specify the following in the configuration file, to work with our JARS, as it prevents the
StadiumVision Mobile JARS from being obfuscated:
-keep public class com.xxxxxx.vome.*
  public protected private *;
}
```

```
       -keep public class com.cisco.** { public protected private *; }

#for the Jackson library
-keepattributes *Annotation*,EnclosingMethod
-keepnames class org.codehaus.jackson.** { *; }
```

If ProGuard complains about “joda.org.time” and you have included the library as well as the configuration options above, you can ignore the warnings with the “–ignorewarnings” flag.

Cisco recommends not obfuscating all the classes that implement or extend the basic Android classes. The following ProGuard configuration is not meant to be a complete configuration, but rather a minimum:

```
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class * extends android.app.backup.BackupAgentHelper
-keep public class * extends android.preference.Preference
-keep public class com.android.vending.licensing.ILicensingService

-keepclasseswithmembernames class * {
    native <methods>;
}
-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet);
}
-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet, int);
}
-keepclassmembers class * extends android.app.Activity {
   public void *(android.view.View);
}
-keepclassmembers enum * {
    public static **[] values();
    public static ** valueOf(java.lang.String);
}
-keep class * implements android.os.Parcelable {
  public static final android.os.Parcelable$Creator *;
}
```

Channel ListView Activity Example

The following example illustrates the following actions:

- Periodically obtains the list of available video channels
- Updatse the Activity’s ListView with the channel list
- Plays the video channel selected in the ListView

```
// set the click listener for the list view
channelListView.setOnItemClickListener(new OnItemClickListener() {
public void onItemClick(AdapterView<?> parentView, View clickedView,
                        int position, long id) {
        // get the selected video channel
        SVMChannel selectedChannel = videoChannels[position];

        Log.d(TAG, "Selected Video Channel = '" + selectedChannel.name);
        // get a reference the StadiumVision Mobile SDK
        StadiumVisionMobile svm = StadiumVisionMobile.getInstance();
        // play the selected video channel with custom video player
        Intent intent = new Intent();
        intent.putExtra("channel", selectedChannel);
```

```
            intent.setClass(MyActivity.this, MyVideoPlayer.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(intent);
        }
    });
```