



## **Cisco StadiumVision Mobile SDK Guide for Apple iOS and Google Android**

Release 1.3.0  
October 31, 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

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: [www.cisco.com/go/trademarks](http://www.cisco.com/go/trademarks). Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

*Cisco StadiumVision Mobile SDK Guide for Apple iOS and Google Android*  
© 2013 Cisco Systems, Inc. All rights reserved.



## CONTENTS

### **Preface** 5

- About This Guide 5
- About Cisco StadiumVision Mobile 5
- Who Should Use This Guide 6
- Obtaining Source Code 6
- Obtaining Documentation and Submitting a Service Request 6

---

### **CHAPTER 1**

- Cisco StadiumVision Mobile API for Apple iOS** 1-1
  - New Features in Cisco StadiumVision Mobile SDK Release 1.3 1-1
  - Introduction to Cisco StadiumVision Mobile API for Apple iOS 1-2
    - iOS Model View Controller (MVC) Design Pattern 1-2
  - iOS API Prerequisites 1-2
  - Apple iOS SDK Overview 1-3
    - Client Application Integration Overview 1-3
    - Cisco StadiumVision Mobile iOS API Class Overview 1-4
    - Video View Controller Inheritance 1-5
    - Cisco StadiumVision Mobile Application Classes 1-5
    - Cisco StadiumVision Mobile iOS API Summary 1-6
    - Cisco StadiumVision Mobile iOS API 1-7
      - Return Status Object 1-7
      - NS Notification Events 1-18
      - Video Player State Flags 1-19
      - Video Player Background Audio 1-20
      - Video Player Channel Inactive Callback 1-20
      - Receiving Service Up and Down Notifications 1-21
      - In-Venue Detection 1-23
      - Set the SDK Configuration at Run-Time 1-24
  - SDK Workflow 1-25
    - Starting the SDK 1-25
    - Setting the Log Level 1-25
    - Getting the Video Channel List 1-25
    - Presenting the Video Channel List 1-26
    - Playing A Video Channel 1-26
    - Seeking Within the Video Buffer 1-26

- Getting The Data Channel List 1-27
- Observing a Data Channel 1-27
- Getting the SDK Version String 1-27
- Shutting Down the SDK (Optional) 1-28
- Video Player View Controller Customization 1-28
  - Default Cisco Video Player View Controller 1-28
  - Customized Video Player 1-28
  - Cisco Demo Customized Video Player 1-29
- Configuration 1-29
  - Configuration Files 1-29
  - Field of Use Configuration 1-30
  - Wi-Fi Access Point Configuration 1-30
- Integration Checklist 1-30
  - What the SDK Handles 1-32
  - Customer Application Roles 1-32

**CHAPTER 2**

- Cisco StadiumVision Mobile API for Google Android 2-1**
  - New Features in Cisco StadiumVision Mobile Release 1.3 Android SDK 2-1
  - Introduction to Cisco StadiumVision Mobile API for Google Android 2-2
    - Android API Prerequisites 2-2
    - Getting Started With The Android Demo App 2-3
      - Install the tools 2-3
      - Build the app 2-3
      - Customize the app 2-3
    - Android SDK Overview 2-4
      - Cisco StadiumVision Mobile iOS API Class Overview 2-4
      - Android OS Activity Overview 2-5
      - Cisco StadiumVision Mobile Android API Summary 2-7
    - Cisco StadiumVision Mobile Android API 2-8
      - Return Status Object 2-8
      - Video Player Activity API Summary 2-14
    - SDK Workflow 2-18
      - Starting the SDK 2-19
      - Getting the Video Channel List 2-19
      - Presenting the Video Channel List 2-19
      - Playing a Video Channel 2-20
      - Seeking Within the Video Buffer 2-20
      - Setting the Video Dimensions 2-20
      - Fullscreen Video Layout 2-20

Partial-Screen Video Layout	2-21
Getting the Data Channel List	2-21
Observing a Data Channel	2-21
Activity Life-Cycle Notifications	2-22
StadiumVision Mobile Service Up or Down Indicator	2-22
In-Venue Detection	2-24
Set the SDK Configuration at Run-Time	2-25
Get the SDK Configuration	2-26
setConfigWithString API Method	2-26
Get the Available Streamer Servers	2-27
Additional Statistics	2-28
Video Player State Notifications	2-28
Video Player "Channel Inactive" Callback	2-29
Video Player Customization	2-30
Cisco Demo Customized Video Player	2-31
Configuration	2-31
Configuration Files	2-32
WiFi AP Info Configuration (Optional)	2-32
Integration	2-33
Client Application Integration Overview	2-33
Integration Checklist	2-33
Customer Application Roles	2-34
Android Permissions	2-35
SDK Native Libraries	2-35





# Preface

---

Revised: October 31, 2013

**Table 1** Document Revision History

Date	Change Summary
October 31, 2013	Initial version of <i>Cisco StadiumVision Mobile SDK Guide for Apple iOS and Google Android</i> , Release 1.3

## 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: October 31, 2013**

This chapter describes the Cisco StadiumVision Mobile SDK Release 1.3 for Apple iOS, and contains the following sections:

- [New Features in Cisco StadiumVision Mobile SDK Release 1.3, page 1-1](#)
- [Introduction to Cisco StadiumVision Mobile API for Apple iOS, page 1-2](#)
- [iOS API Prerequisites, page 1-2](#)
- [Apple iOS SDK Overview, page 1-3](#)
- [Client Application Integration Overview, page 1-3](#)
- [Cisco StadiumVision Mobile iOS API Class Overview, page 1-4](#)
- [Video View Controller Inheritance, page 1-5](#)
- [Cisco StadiumVision Mobile Application Classes, page 1-5](#)
- [Cisco StadiumVision Mobile iOS API Summary, page 1-6](#)
- [Cisco StadiumVision Mobile iOS API, page 1-7](#)

## New Features in Cisco StadiumVision Mobile SDK Release 1.3

Note the following for release 1.3 of the Cisco StadiumVision Mobile SDK:

- None of the release 1.2 APIs have changed for release 1.3.
- The Cisco StadiumVision Mobile SDK release 1.3 is backwards compatible with release 1.2, and can be imported into your project without any software changes.

### **New Features**

- StadiumVision Mobile Service up and down Indicator and notifications
- In-venue detection and notifications
- Setting the SDK Configuration at Run-Time
- StadiumVision Mobile Streamer detection
- Statistics collection enhancements, including StadiumVision Mobile Reporter upload statistics, multicast channel announcement statistics, and licensing statistics
- Video player State notifications
- Video player Inactive channel callback

# Introduction to Cisco StadiumVision Mobile API for Apple iOS

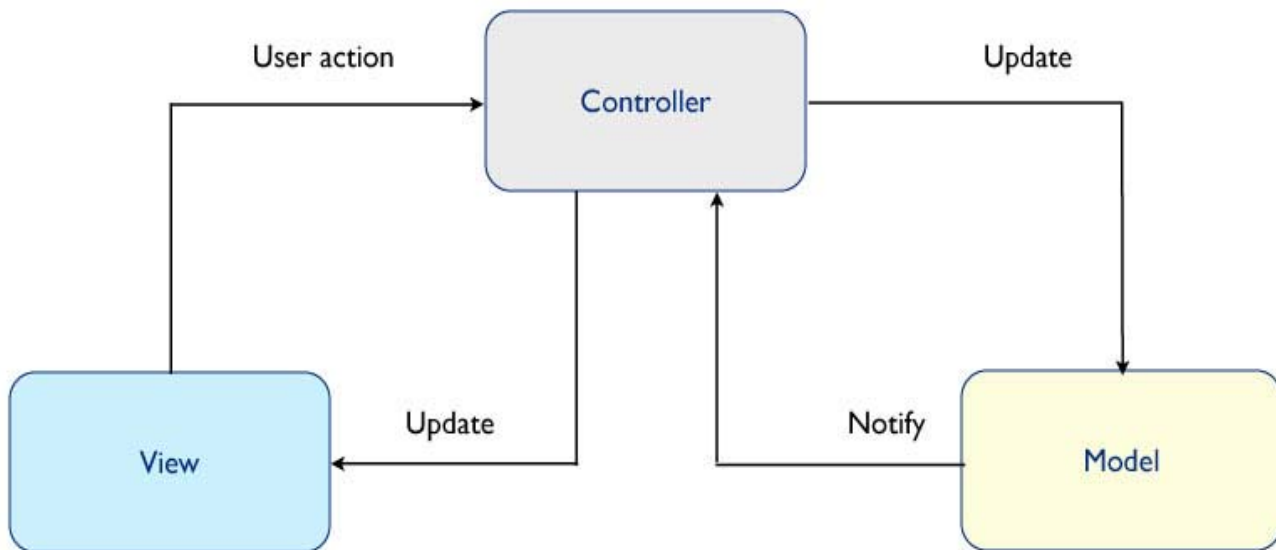
The iOS SDK is provided as a set of static libraries, header files, and an a sample iOS app (with a complete Xcode project). This API uses Objective-C classes and method calls to access the StadiumVision Mobile data distribution and video playback functionality within the StadiumVision Mobile iOS SDK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

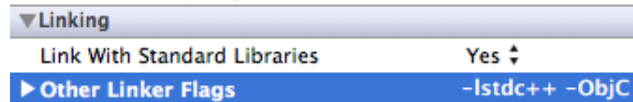
[Table 1-1](#) lists the various iOS SDK build environment requirements.

**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.	<a href="http://www.apple.com">http://www.apple.com</a>
Xcode	4.5.1 or later	Apple development IDE and tool kit.	<a href="http://developer.apple.com/xcode">http://developer.apple.com/xcode</a>

**Note**

Application developers will need to link against the libstdc++ library in their build. They will also need to use the "-ObjC" linker flag to import all of the iOS "categories" from the iOS SDK. Both of the required linker flags can be added in Xcode using Build Settings->Linking->Other Linker Flags->Add. The required Xcode "Other Linker Flags" settings are shown in the image below:



## Apple iOS SDK Overview

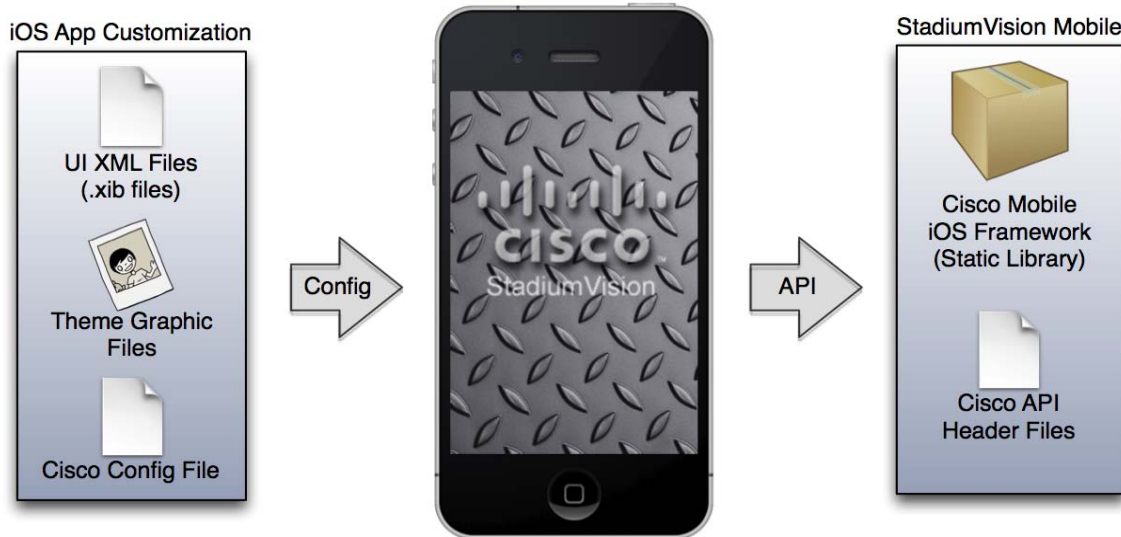
The Cisco StadiumVision Mobile iOS SDK contains the following components:

- 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 modify the sample application, and the right represents how the SDK is packaged.

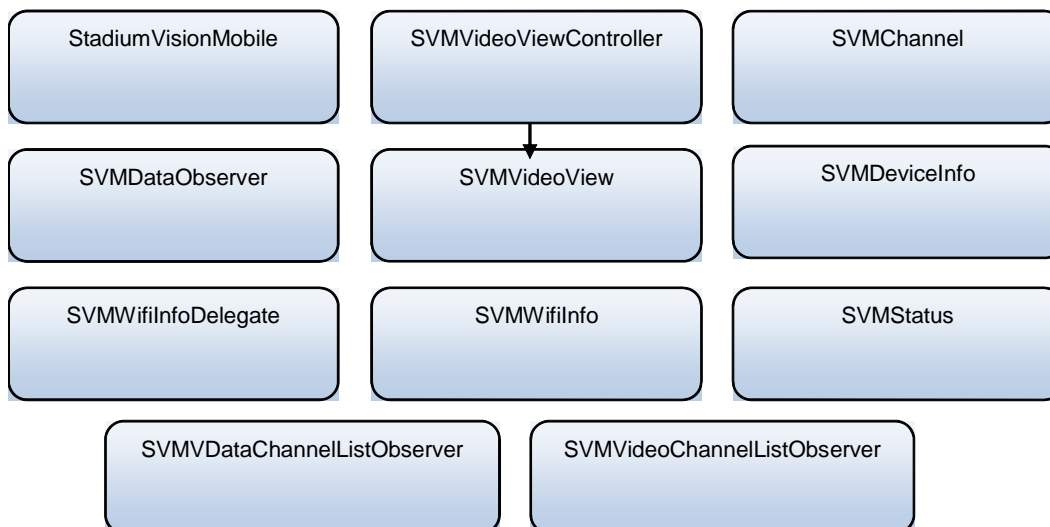
Figure 1-2 Cisco StadiumVision Mobile iOS SDK Components



## 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.

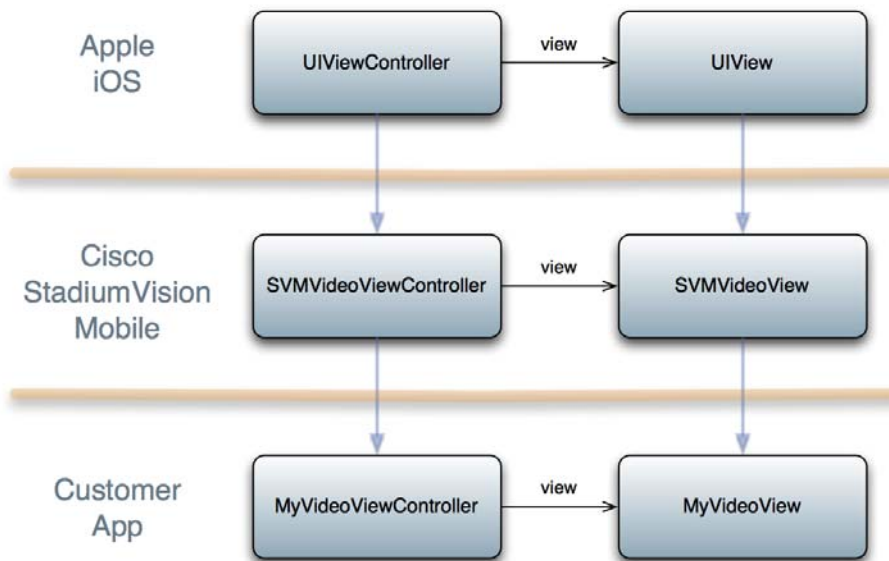
Figure 1-3 Cisco StadiumVision Mobile iOS API Classes



## 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

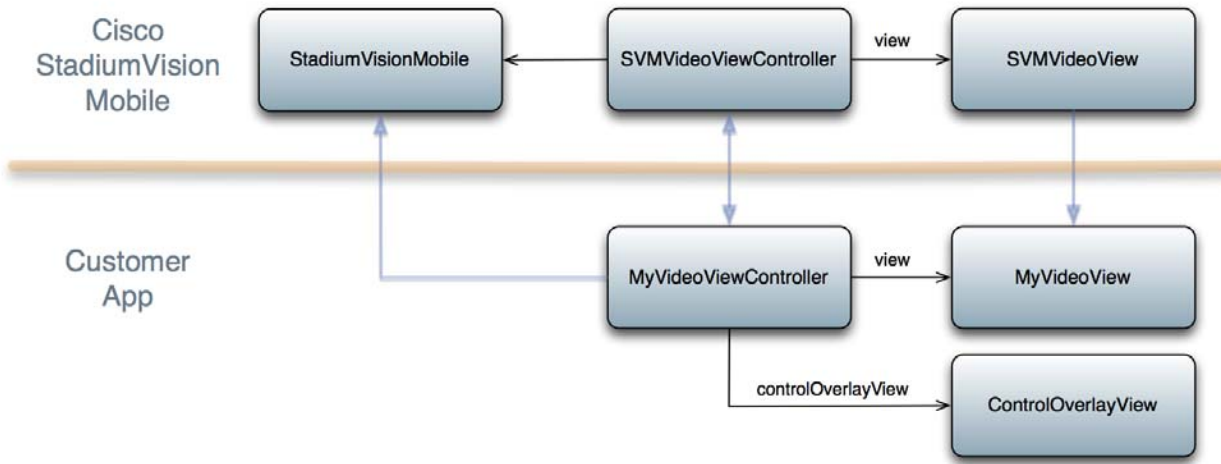


## Cisco StadiumVision Mobile Application Classes

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
NSDictionary*	getConfig	Gets the SDK configuration at run-time
NSArray*	getStreamerArray	Gets an array of detected SVM Streamer servers as 'SVMStreamer' objects
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
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

Return Type	API Method Name	API Method Description
SVMStatus*	addDataChannelObserver:forChannelName:	Registers an observer class to receive all data updates for a particular data channel name
SVMStatus*	removeDataChannelObserver:forChannel:	Unregisters an observer class from receiving any data updates for a particular data channel
SVMStatus*	removeDataChannelObserver:forChannelName:	Unregisters an observer class from receiving any data updates for a particular data channel
SVMStatus*	getVideoChannelListArray	Returns a snapshot array of the currently available video channels.
SVMStatus*	getDataChannelListArray	Returns a snapshot array of the currently available data channels.
NSDictionary	stats	Gets an NSDictionary of current StadiumVision Mobile SDK stats.
SVMStatus*	version	Gets the StadiumVision Mobile version string.
SVMStatus*	setConfig	Sets the SDK configuration at run time.
SVMStatus*	setConfigWithString	Sets the SDK configuration at run time with the config JSON string.

## Cisco StadiumVision Mobile iOS API

The following sections and 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. This section contains the following API calls and tables:

- [SVMStatus class](#)
- [sharedInstance](#)
- [Start](#)
- [addVideoChannelListDelegate](#)
- [setLogLevel](#)
- [removeVideoChannelListDelegate](#)
- [addDataChannelListDelegate](#)
- [removeDataChannelListDelegate](#)
- [addDataChannelListDelegate](#)
- [removeDataChannelListDelegate](#)
- [addDataChannelObserver](#)
- [removeDataChannelObserver](#)
- [setConfig](#)
- [setConfigWithString](#)
- [allowPlaybackWhenViewDisappears](#)

- [getConfig](#)
- [onData](#)
- [Stats](#)
- [Stats API Hash Keys and Descriptions](#)
- [getVideoChannelListArray](#)
- [getDataChannelListArray](#)
- [wifiInfo](#)
- [wifiInfo Object Properties](#)
- [version](#)

Table 1-3 SVMStatus class

<b>Type</b>	BOOL	NSString
<b>Property</b>	isOk	errorString
<b>Description</b>	Boolean indicating whether the API call was successful or not.	If the API call was not successful (isOk == NO), this string describes the error.
<b>Example Usage</b>	<pre>// 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); }</pre>	

Table 1-4 sharedInstance

<b>Method Signature</b>	(StadiumVisionMobile*) sharedInstance
<b>Prerequisites</b>	N/A
<b>Notes</b>	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.
<b>Result</b>	N/A



**Table 1-5**      *Start*

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

**Table 1-6**      *addVideoChannelListDelegate*

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

**Table 1-7**      *setLogLevel*

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

**Table 1-8**      *removeVideoChannelListDelegate*

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

**Table 1-9**      *addDataChannelListDelegate*

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

**Table 1-10** *removeDataChannelListDelegate*

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

**Table 1-11** *addDataChannelListDelegate*

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

**Table 1-12** *removeDataChannelListDelegate*

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

**Table 1-13** *addDataChannelObserver*

<b>Method Signature</b>	(SVMStatus*) addDataChannelObserver: (id<SVMDDataObserver>)delegate forChannel: (SVMChannel*) channel (SVMStatus*) addDataChannelObserver:(id<SVMDDataObserver>)delegate forChannelName: (NSString*) channelName  The following example enables reception of the data announcements:  SVMChannel *selectedChannel1 = [dataChannelList objectAtIndex:0]; [svm addDataChannelObserver:self forChannelName:selectedChannel1.name];
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method registers the given delegate class to receive all data for the given data channel object.
<b>Result</b>	N/A

**Table 1-14** *removeDataChannelObserver*

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

**Table 1-15** *setConfig*

<b>Method Signature</b>	(SVMStatus*) setConfig: (NSDictionary*) runtimeConfigDict ;
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method sets the SDK configuration at run-time.
<b>Result</b>	N/A

**Table 1-16** *setConfigWithString*

<b>Method Signature</b>	(SVMStatus*) setConfigWithString: (NSString*) jsonConfig;
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method sets the SDK configuration at run-time.
<b>Result</b>	N/A

**Table 1-17** *allowPlaybackWhenViewDisappears*

<b>Method Signature</b>	(SVMStatus *) allowPlaybackWhenViewDisappears: (BOOL) isAllowed;
<b>Prerequisites</b>	N/A
<b>Notes</b>	Provides a mode that allows the video player to continue rendering the audio and video channels when the video player view has lost focus.
<b>Result</b>	N/A

**Table 1-18** *getConfig*

<b>Method Signature</b>	(NSDictionary*) getConfig;
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method fetches the SDK configuration.
<b>Result</b>	N/A

Table 1-19 *onData*

<b>Method Signature</b>	<code>(void) onData:(NSData*)data withChannelName:(NSString*)channelName</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method is implemented by the customer app to support the "SVMDDataObserver" 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).
<b>Results</b>	N/A

Table 1-20 *Stats*

<b>Method Signature</b>	<code>(NSDictionary*) stats</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	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).
<b>Result</b>	N/A

**Table 1-21** Stats API Hash Keys and Descriptions

Stats Hash Key	Stats Description
announcement_session_id	The video session announcement ID
announcement_session_title	The session announcement name
announcementsMalformed	Number of malformed channel announcement packets received
announcementsNotAllowed	Number of announcements where the Streamer is not allowed
announcementsReceived	Number of total channel announcements received
channelsAdded	Number of times a channel was added to the channel list
channelsPruned	Number of times a channel was pruned from the channel list
invalidJsonAnnouncements	Number of announcements with an invalid JSON body
licenseMismatchAnnouncements	Number of license key mismatches
listenerIcmpRestarts	Number of announcement listener IGMP restarts
num_dropped_video_frames	The total number of video frames dropped
num_ts_discontinuities	The total number of MPEG2-TS packet discontinuities
protection_windows	The total number of protection windows sent
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)
statsUploadAttempts	Number of Reporter stats upload attempts
statsUploadFailures	Number of Reporter stats upload failures
statsUploadSuccesses	Number of Reporter stats upload successes
total_num_bytes_written	The total number of video bytes played
versionMismatchAnnouncements	Number of announcement version mismatches
window_error	The total number of protection windows with more packets per window than can be supported by Cisco StadiumVision Mobile.
window_no_loss	The total number of protection windows with no dropped 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_recovery_successes	The total number of protection windows with recovered video packets
window_warning	The total number of protection windows with more packets per window than the recommended value

**Table 1-22** *getVideoChannelListArray*

<b>Method Signature</b>	<pre>StadiumVisionMobile *svm = [StadiumVisionMobile sharedInstance]; NSArray *currentChannels = [svm getVideoChannelListArray];</pre>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method returns an array (NSArray*) of the currently available video channels (array of “SVMChannel” objects).
<b>Result</b>	NSArray* of SVMChannel objects

**Table 1-23** *getDataChannelListArray*

<b>Method Signature</b>	<pre>StadiumVisionMobile *svm = [StadiumVisionMobile sharedInstance]; NSArray *currentChannels = [svm getDataChannelListArray];</pre>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method returns an array (NSArray*) of the currently available data channels (array of “SVMChannel” objects)
<b>Result</b>	NSArray* of SVMChannel objects

**Table 1-24** *wifiInfo*

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

Table 1-25 and Table 1-26 contain properties are available within the SVMWifiInfo object.

**Table 1-25** *wifiInfo Object Properties*

<b>Stats Hash Key</b>	<b>Stats Description</b>
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 1-26** *version*

<b>Method Signature</b>	<pre>(NSString*) version</pre>
<b>Prerequisites</b>	N/A

<b>Method Signature</b>	(NSString*) version
<b>Notes</b>	This method returns the Cisco StadiumVision Mobile SDK version string.
<b>Result</b>	N/A

The 'SVMVideoVideoController' class can be extended and customized. The SVMVideoVideoController API methods are listed in [Table 1-27](#). This section contains the following API calls and tables:

- [Video View Controller API Summary](#)
- [setRenderVideoView](#)
- [playVideo Channel](#)
- [getConfig](#)
- [getStreamerArray](#)
- [seekRelative](#)
- [seekAbsolute](#)
- [playLive](#)

**Table 1-27** 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 1-28** setRenderVideoView

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

Table 1-29 *playVideo Channel*

<b>Method Signature</b>	<code>(void)playVideoChannel:(SVMChannel*)channel;</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	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.
<b>Result</b>	N/A

Table 1-30 *getConfig*

<b>Method Signature</b>	<code>(NSDictionary*)getConfig</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method returns the current SDK configuration as an NSDictionary object.
<b>Result</b>	NSDictionary*

Table 1-31 *getStreamerArray*

<b>Method Signature</b>	<code>(NSArray*)getStreamerArray</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method returns an array of Streamer servers detected by the SVM SDK; with each Streamer entry represented as an 'SVMStreamer' object in the array.
<b>Result</b>	NSArray*



Table 1-32 *seekRelative*

<b>Method Signature</b>	(void) seekRelative: (NSInteger)durationMs;
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>• 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.</li> <li>• The StadiumVision Mobile SDK currently buffers 30 seconds of previously played video data that can be used for playing previously recorded video data.</li> <li>• A negative duration value rewinds the video play-head within the video history buffer.</li> <li>• A positive duration value forwards the video play-head towards the latest "live" video data in the video history buffer.</li> <li>• 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.</li> </ul>
<b>Result</b>	N/A

Table 1-33 *seekAbsolute*

<b>Method Signature</b>	(void) seekAbsolute: (NSInteger)durationMs;
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>• 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.</li> <li>• The StadiumVision Mobile SDK currently buffers 30 seconds of previously played video data that can be used for playing previously recorded video data</li> <li>• A positive duration value moves the video play-head away from the latest "live" video data in the video history buffer.</li> <li>• 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.</li> </ul>
<b>Result</b>	N/A

Table 1-34 *playLive*

<b>Method Signature</b>	(void) playLive;
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>This method forwards the video play-head to the starting "live" position at the beginning of the video data buffer.</li> <li>This convenience method acts as a wrapper for the "seekAbsolute" API method; making "playLive()" equivalent to "seekAbsolute(0)".</li> </ul>
<b>Result</b>	N/A

## NS Notification Events

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

Table 1-35 *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
kSVMVideoOpenState	Occurs when the video player initially opens the video channel session
kSVMVideoPlayState	Occurs when the video player starts playing the video channel
kSVMVideoRewindState	Occurs when the video player rewinds (seeks backwards) within the video history buffer
kSVMVideoLiveState	Occurs when the video player moves the play-head to the beginning "live" position
kSVMVideoStopState	Occurs when the video player stop video playback
kSVMVideoCloseState	Occurs when the video player closes the video channel session

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;
    }
}
```

The following example shows how to subscribe to receive the video player broadcast notifications:

```
// subscribe to receive video channel state change notifications
[[NSNotificationCenter defaultCenter] addObserver:self
                                       selector:@selector(onVideoChannelStateChanged:)
                                       name:kSVMVideoPlayerChannelStateChange
                                       object:nil];
```

The following example shows how to parse the video player broadcast notifications for (1) the video channel name and (2) the video channel state:

```
// get the video channel state dictionary from the notification
NSDictionary *stateDict = [notify userInfo];

// get the video channel name
NSString *videoChannelName = [stateDict objectForKey:kSVMVideoPlayerChannelNameKey];

// get the video channel state
NSString *videoChannelState = [stateDict objectForKey:kSVMVideoPlayerChannelStateKey];

// determine the video channel state
if ([videoChannelState isEqualToString:kSVMVideoPlayState] == YES) {
    // video player is now playing
    NSLog(@"### VIDEO PLAYER: PLAYING");
} else if ([videoChannelState isEqualToString:kSVMVideoStopState] == YES) {
    // video player is now stopped
    NSLog(@"### VIDEO PLAYER: STOPPED");
}
```

## Video Player State Flags

The SVM video player class ("SVMVideoViewController") provides a set of state flags that the inherited video player class (ie: "MyVideoViewController") can use to monitor the current video player state:

- BOOL isPlaying;
- BOOL isOpen;
- BOOL isAppActive;
- BOOL isVisible;
- BOOL isBackgroundPlaybackAllowed;

[Table 1-36](#) provides a description of each state flag provided by the StadiumVision Mobile video player ("SVMVideoViewController"):

**Table 1-36** Video Player State Flags

State Flag	Description
isOpen	Boolean flag indicating that the video player has opened a session for video channel playback
isPlaying	Boolean flag indicating when the video player is currently playing a video channel
isBackgroundPlaybackAllowed	Boolean flag indicating if the video player is allowed to continue rendering the audio and video channels when the video player view has lost focus ("allowPlaybackWhenViewDisappears")
isVisible	Boolean flag indicating when the video player view is visible. This is useful when the video player is allowed to continue playing the audio / video channels when the video player has lost focus ("allowPlaybackWhenViewDisappears")
isAppActive	Boolean flag indicating when the container iOS app is in the foreground

## Video Player Background Audio

Starting Cisco StadiumVision Mobile SDK Release 1.3, the SVM video player ("SVMVideoViewController") provides a mode that allows the video player to continue rendering the audio and video channels when the video player view has lost focus. This mode allows the audio to still be played even when the user navigates away from the video player screen (view controller) to a different app screen; causing the video player to be hidden.

The background audio mode is disabled in the "SVMVideoViewController" by default.

The following example shows how to set the "SVMVideoViewController" mode that allows the video player to continue rendering audio and video when the "SVMVideoViewController" loses focus (is not visible):

```
// create the video view controller
self.videoViewController = [[MyVideoViewController alloc] init];

// allow the video player to continue playing when the video view disappears
[self.videoViewController allowPlaybackWhenViewDisappears:YES];
```

## Video Player Channel Inactive Callback

To detect that a currently playing video channel has become invalid (due to Streamer server admin changes), the SVM video player ("SVMVideoViewController") provides a callback to tell the video player sub-class (ie: "MyVideoViewController") that the currently playing channel is no longer valid.

When a channel becomes invalid, playback of the video channel is automatically stopped.

To receive these callbacks, the "onCurrentChannelInvalid" method must be overridden by the 'SVMVideoViewController' sub-class (ie: "MyViewViewController"). The following example shows the method signature and implementation of this overridden callback method:

```
// OVERRIDDEN by the 'SVMVideoViewController' sub-class; indicates that the current
channel is invalid
- (void)onCurrentChannelInvalid
{
    NSLog(@"Current channel is no longer valid: dismissing video view controller");
}
```

```

    // dismiss this modal video view controller
    [self dismissModalViewControllerAnimated:YES];
}

```

## Receiving Service Up and Down Notifications

The Release 1.3 of the Cisco StadiumVision Mobile SDK includes a mechanism to determine if the Cisco StadiumVision Mobile service is available or not. The SDK provides an indicator to the application indicating if the StadiumVision Mobile service is up or down. This indication should be used by the application to indicate to the user whether the StadiumVision Mobile service is available or not. Service is declared 'down' by the SDK when any of the following are true:

- The SVM SDK detects that the video quality is poor
- The SVM SDK detects that no valid, licensed channel are available
- The mobile device's WiFi interface is disabled

Poor video quality can occur when the user is receiving a weak WiFi signal; causing data loss. There are two different ways that the iOS app can get the "Service State" from the SVM SDK:

- Register to receive the "Service Up / Down" notifications
- Fetch the current service state from the SDK on-demand

When the app receives the "Service Down" notification, the SDK will supply a bitmap containing the reasons why the service was declared as 'down' by the SDK. The 'reasons' bitmap is given in [Table 1-37](#):

**Table 1-37**      **Service Down Reason Notification**

Service Down Reason	Constant
Poor video quality networking conditions detected	kSVMServiceDownReasonPoorQuality
WiFi connection is down	kSVMServiceDownReasonWiFiDown
No valid SVM channels have been detected	kSVMServiceDownReasonNoChannels

The following example shows how to register to receive the "Service Up / Down" notifications from the StadiumVision Mobile SDK:

```

#import "StadiumVisionMobile.h"

// subscribe to receive service state up / down change notifications
[[NSNotificationCenter defaultCenter] addObserver:self
                                     selector:@selector(onServiceStateChanged:)
                                     name:kSVMServiceStateChangedNotification
                                     object:nil];

// handle the received service state notifications
- (void)onServiceStateChanged:(NSNotification*)notify
{
    // get the service state dictionary from the notification
    NSDictionary *serviceStateDict = [notify userInfo];

    // get the service state integer value
    NSNumber *serviceStateNumber = [serviceStateDict
    objectForKey:kSVMServiceStateObjectKey];
    NSUInteger serviceState = [serviceStateNumber unsignedIntegerValue];
}

```

```

// if the service state is down
if (serviceState == kSVMServiceStateDown) {
    // service state is down
    NSLog(@"*** SERVICE STATE: DOWN");

    // get the service state down reasons bitmap
    NSNumber *reasonsNumber = [serviceStateDict
objectForKey:kSVMServiceStateChangeReasonsObjectKey];
    NSUInteger reasonsBitmap = [reasonsNumber unsignedIntegerValue];

    // determine the reason(s) why the service state went down
    if (reasonsBitmap & kSVMServiceDownReasonSDKNotRunning) {
        NSLog(@"SERVICE DOWN: SVM SDK was stopped");
    } else if (reasonsBitmap & kSVMServiceDownReasonWiFiDown) {
        NSLog(@"SERVICE DOWN: WiFi connection is down");
    } else if (reasonsBitmap & kSVMServiceDownReasonNoChannels) {
        NSLog(@"SERVICE DOWN: No valid licensed SVM channels available");
    } else if (reasonsBitmap & kSVMServiceDownReasonPoorQuality) {
        NSLog(@"SERVICE DOWN: Poor quality conditions detected");
    }

    // show the service down message
    [self showServiceDownMessage];
} else if (serviceState == kSVMServiceStateUp) {
    // service state is up
    NSLog(@"*** SERVICE STATE: UP");
}
}
}

```

## Getting the Current Service Up or Down State On Demand

The "getServiceState" API method can be used to fetch the current service state from the SDK. The method signature of the "getServiceState" API call is given below:

```

// api call to fetch the current svm 'service state' on-demand
- (SVMServiceState)getServiceState;

```

The following example show how to fetch the current service state from the SVM SDK using the "getServiceState" API call:

```

#import "StadiumVisionMobile.h"

// get the svm api context
StadiumVisionMobile *svm = [StadiumVisionMobile sharedInstance];

// get the current svm service state
SVMServiceState state = [svm getServiceState];

// determine the current service state
if (serviceState == kSVMServiceStateUp) {
    // service state is up
    NSLog(@"*** SERVICE STATE: UP");
} else if (serviceState == kSVMServiceStateDown) {
    // service state is down
    NSLog(@"*** SERVICE STATE: DOWN");
}
}

```

## In-Venue Detection

Cisco StadiumVision Mobile SDK Release 1.3 provides a mechanism to detect whether the mobile device is connected within the SVM-enabled venue or not. There are two different ways that the iOS app can get this "In-Venue Detection" state from the SVM SDK:

1. Register to receive the "In-Venue Detection" notifications
2. Fetch the current "In-Venue" state from the SDK on-demand

### Receiving In-Venue Detection Notifications

The following example shows how to register to receive the "Service Up / Down" notifications from the SVM SDK:

```
// subscribe to receive in-venue connection change notifications
[[NSNotificationCenter defaultCenter] addObserver:self
                                       selector:@selector(onVenueConnectionChanged:)
                                       name:kSVMVenueConnectionUpdateNotification
                                       object:nil];

// handle the venue connection changed event
- (void)onVenueConnectionChanged:(NSNotification*)notify
{
    // get the in-venue detection dictionary from the notification
    NSDictionary *inVenueDetectionDict = [notify userInfo];

    // get the in-venue detection value
    NSNumber *inVenueDetectionNumber = [inVenueDetectionDict
    objectForKey:kSVMVenueConnectionStateObjectKey];
    BOOL isConnectedToVenue = [inVenueDetectionNumber boolValue];

    // log whether we are inside the venue
    NSLog(@"##### Venue Connection Updated: %@", (isConnectedToVenue ? @"INSIDE" :
    @"OUTSIDE"));
}
```

### Get the Current In-Venue State On-Demand

The "isConnectedToVenue" API method can be used to fetch the current in-venue state from the SDK. The method signature of the "isConnectedToVenue" API call is given below:

```
// returns whether the device is connected to the licensed SVM venue or not
- (BOOL)isConnectedToVenue;
```

The following example shows how to fetch the current service state from the SVM SDK using the "getServiceState" API call:

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

// get whether the device is currently connected to the SVM licensed venue
BOOL isConnectedToVenue = [svm isConnectedToVenue];

// log whether the device is currently connected to the SVM licensed venue
NSLog(@"##### Venue Connection State: %@", (isConnectedToVenue ? @"INSIDE" :
    @"OUTSIDE"));
```

## Set the SDK Configuration at Run-Time

Previously, the Cisco StadiumVision Mobile SDK could only be configured by using a JSON-formatted config file ("cisco\_svm.cfg") bundled within the iOS app. Starting with Release 1.3, the application can now set the SDK configuration at run-time through an API method. This allows the application to dynamically configure the SDK. For example, the application can fetch the SDK configuration information from a network connection, and then pass that configuration to the SDK.

Two different methods are provided for setting the SDK configuration at run-time:

- "setConfig"
- "setConfigWithString"

The following example shows how to set the SDK configuration using the "setConfig" API method:

```
#import "StadiumVisionMobile.h"
// get the stadiumvision mobile api instance
StadiumVisionMobile *svmInstance = [StadiumVisionMobile sharedInstance];
// create the config dictionary with the set of licensing keys
NSMutableDictionary *configDict = [[NSMutableDictionary alloc] autorelease];
NSMutableDictionary *licenseDict = [[NSMutableDictionary alloc] autorelease];
[licenseDict setObject:@"MyVenueNameKey" forKey:@"venueName"];
[licenseDict setObject:@"MyContentOwnerKey" forKey:@"contentOwner"];
[licenseDict setObject:@"MyAppDeveloperKey" forKey:@"appDeveloper"];
[configDict setObject:licenseDict forKey:@"license"];
// update the stadiumvision mobile configuration
[svmInstance setConfig:configDict];
```



## SDK Workflow

This section describes the Cisco StadiumVision Mobile SDK workflow, and contains the following sections:

- [Starting the SDK, page 1-25](#)
- [Setting the Log Level, page 1-25](#)
- [Getting the Video Channel List, page 1-25](#)
- [Presenting the Video Channel List, page 1-26](#)
- [Playing A Video Channel, page 1-26](#)
- [Seeking Within the Video Buffer, page 1-26](#)
- [Getting The Data Channel List, page 1-27](#)
- [Observing a Data Channel, page 1-27](#)
- [Getting the SDK Version String, page 1-27](#)
- [Shutting Down the SDK \(Optional\), page 1-28](#)

## 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 = [StadiumVisionMobile 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 = [StadiumVisionMobile 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

Table 1-38 lists the "SVMChannel" video channel objects containing all of the information needed to display the channel list to the user.

**Table 1-38** 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
"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 (NSNotification). 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

This section describes how to customize the video player, and contains the following sections:

- [Default Cisco Video Player View Controller, page 1-28](#)
- [Customized Video Player, page 1-28](#)
- [Cisco Demo Customized Video Player, page 1-29](#)

### 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

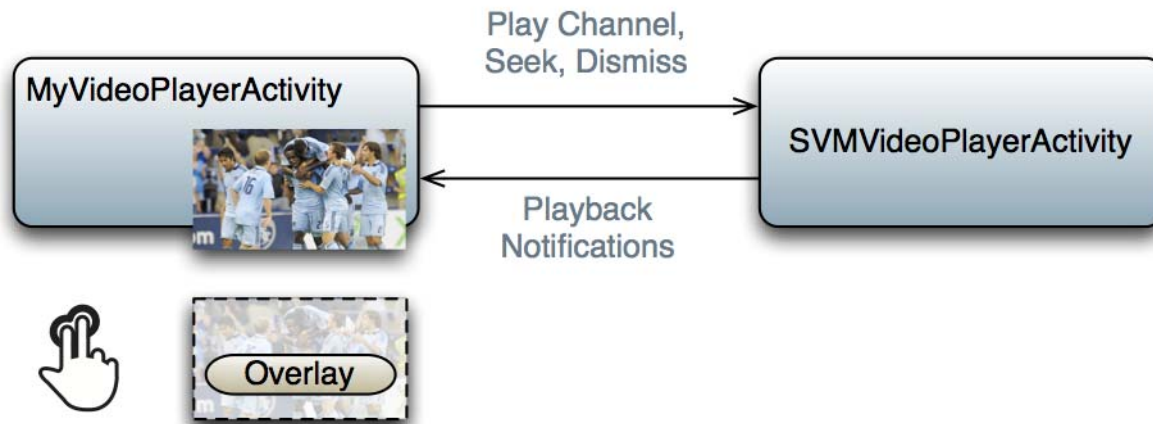
### Customized Video Player

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

This section describes the required configuration files, and contains the following sections:

- [Configuration Files, page 1-29](#)
- [Field of Use Configuration, page 1-30](#)
- [Wi-Fi Access Point Configuration, page 1-30](#)

## 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 1-39 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.
"voVidDec.dat"	Video decoder license file.

## 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 three 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"
      }
    ]
  }
}
```

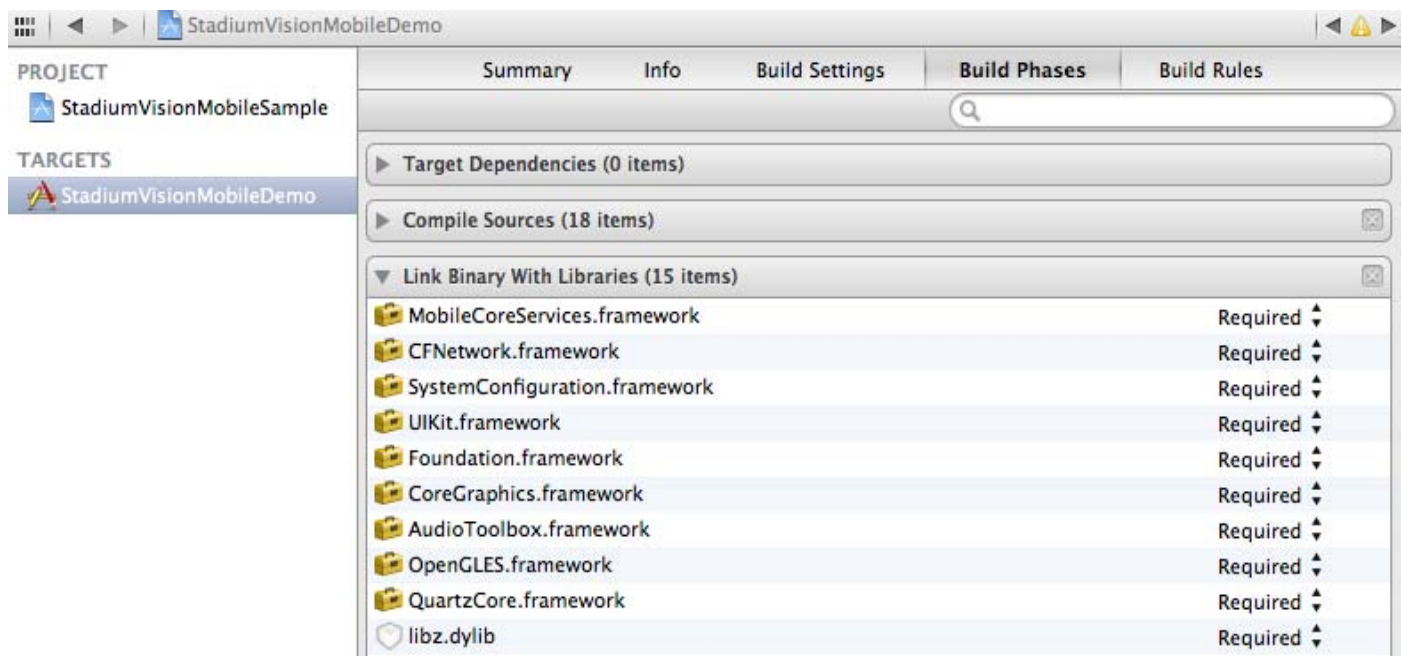
## Integration Checklist

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
- OpenGL.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

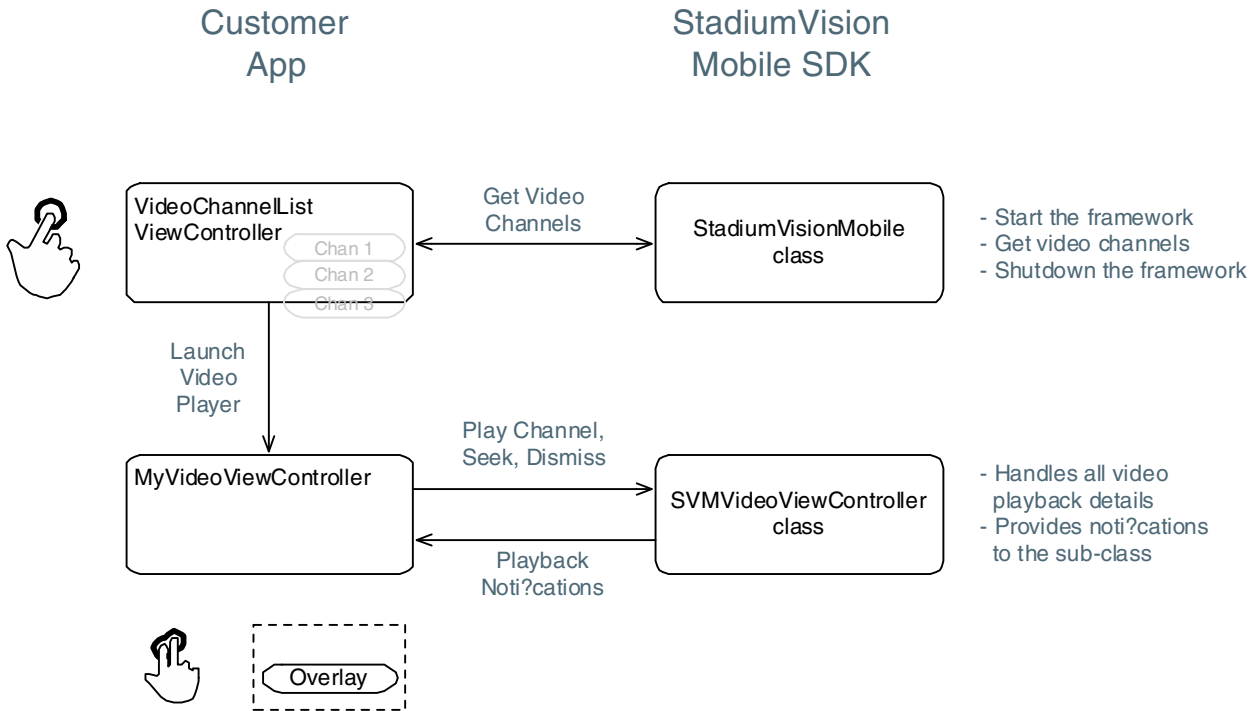
## Customer Application Roles

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

---

**Revised: October 31, 2013**

This chapter describes the Cisco StadiumVision Mobile SDK Release 1.3 for Google Android, and contains the following sections:

- [New Features in Cisco StadiumVision Mobile Release 1.3 Android SDK, page 2-1](#)
- [Introduction to Cisco StadiumVision Mobile API for Google Android, page 2-2](#)
- [Android API Prerequisites, page 2-2](#)
- [Android SDK Overview, page 2-4](#)
- [Cisco StadiumVision Mobile Android API, page 2-8](#)
- [SDK Workflow, page 2-18](#)
- [Video Player Customization, page 2-30](#)
- [Configuration, page 2-31](#)
- [Integration, page 2-33](#)

## New Features in Cisco StadiumVision Mobile Release 1.3 Android SDK

Note the following for release 1.3 of the Cisco StadiumVision Mobile SDK:

- None of the release 1.2 APIs have changed for release 1.3.
- The Cisco StadiumVision Mobile SDK release 1.3 is backwards compatible with release 1.2, and can be imported into your project without any software changes.

### **New Features in Release 1.3**

- StadiumVision Mobile Service up and down indicator and notifications
- In-venue detection and notifications
- Setting the SDK Configuration at Run-Time
- StadiumVision Mobile Streamer detection
- Statistics collection enhancements, including StadiumVision Mobile Reporter upload statistics, multicast channel announcement statistics, and licensing statistics

- Video player State notifications
- Video player background audio
- Video player Inactive channel callback

## 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.

This document contains the following sections:

- [Android API Prerequisites, page 2-2](#)
- [Android SDK Overview, page 2-4](#)
- [Cisco StadiumVision Mobile Android API, page 2-8](#)
- [SDK Workflow, page 2-18](#)
- [Video Player Customization, page 2-30](#)
- [Configuration, page 2-31](#)
- [Integration, page 2-33](#)

## Android API Prerequisites

[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 or later	Eclipse "Classic" for Mac OSX (64-bit)	<a href="http://www.eclipse.org/downloads/">http://www.eclipse.org/downloads/</a>

## Getting Started With The Android Demo App

The Cisco StadiumVision Mobile SDK that we provide to app developers includes the source code for an Android demo app. The purpose of the app is demonstrate what is possible, and to enable a new app developer to quickly get a working app up and running.

### Install the tools

- 
- Step 1** Download the Android Developer Tools ADT.
  - Step 2** Follow these instructions to set up ADT on your computer.
  - Step 3** Launch the Eclipse application and when prompted select a folder to use as your workspace.
  - Step 4** Launch the Android SDK Manager from the Window dropdown menu.
  - Step 5** Open the Android 2.2 (API 8) folder and check the SDK Platform box. Uncheck everything else and then install the selected package.

### Build the app

- 
- Step 1** Download the StadiumVisionMobileSample-Android-xxxx.tar.bz2 SDK and demo app package.
  - Step 2** Extract the downloaded package into a directory.
  - Step 3** Import the demo app project into Eclipse as follows:
    - a. In Eclipse go to File=>Import
    - b. Then go to General=>Existing Projects into Workspace, and select Next.
    - c. Set the Select root directory to the folder where you unpacked the SDK, and then click Finish.
    - d. Restart Eclipse from File=>Restart.
  - Step 4** Right click on CiscoStadiumVisionMobile in the left Package Explorer window, and select Android Tools=>Export Signed Application Package.
  - Step 5** Click Next when the Project Checks window appears.
  - Step 6** Select Create new keystore, then browse to a folder where you wish to store the key store file. Click Next.
  - Step 7** Fill in the Key Creation form. There are no right or wrong answers. Click Next.
  - Step 8** Browse to the folder where you wish to place the apk file, then click Finish.
  - Step 9** Download the apk file to your Android device by placing it on a web server, emailing it, SD card, USB flash key, etc.
  - Step 10** Now install the apk on your device.

### Customize the app

Here are some of the first items you may want to customize in the demo app:

Change the text for the app icon:

- In the the file "res/values/strings.xml" change "SVM Demo" to "My SVM App"

Change the name space so your custom app can be installed side by side with the out of the box demo app:

- Edit the file "AndroidManifest.xml":
  - Change "package="com.cisco.sv"" to "package="com.cisco.svm.foo"
  - Change "android:name="com.cisco.svm.app.StadiumVisionMobile" to "android:name="com.cisco.svm.foo"



---

**Note** The package name must start with "com." (excluding the quotes).

---

- Search and replace com.cisco.sv.R with com.cisco.svm.foo.R in all \*.java files in src/app/demo.

## Android SDK Overview

The Cisco StadiumVision Mobile Android SDK contains the following components:

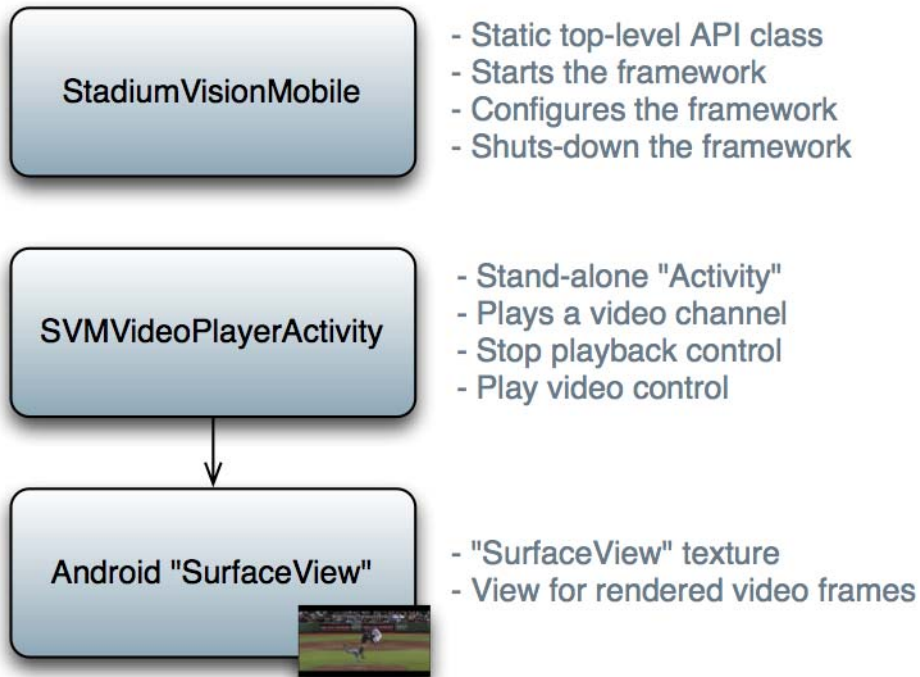
- A set of static libraries, configuration files, player layout XML files, and a sample Android application.
- Customizable video player

## 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 StadiumVisionMobile 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 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

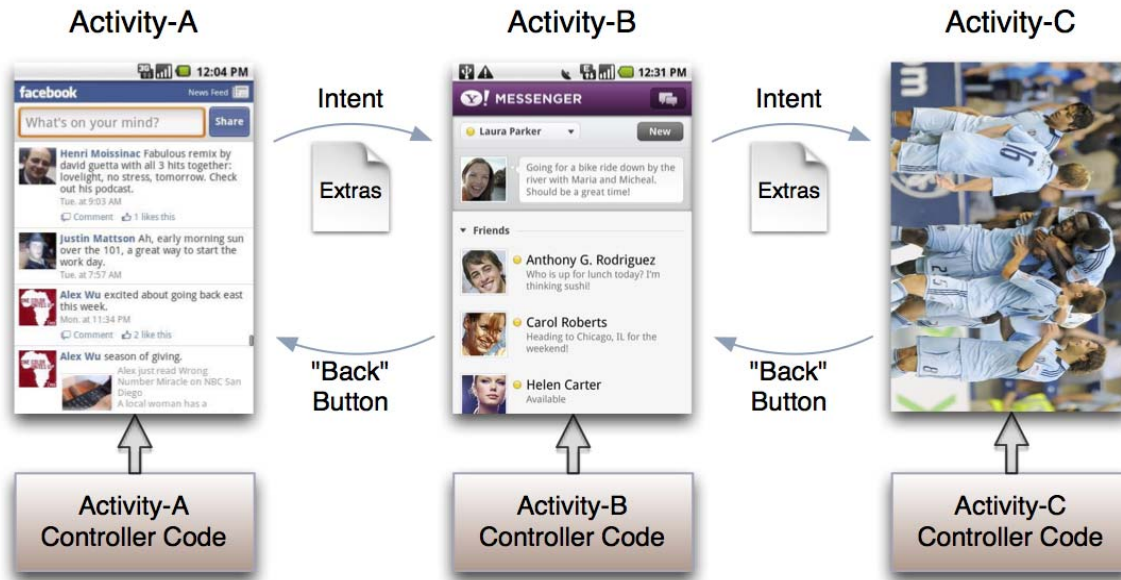
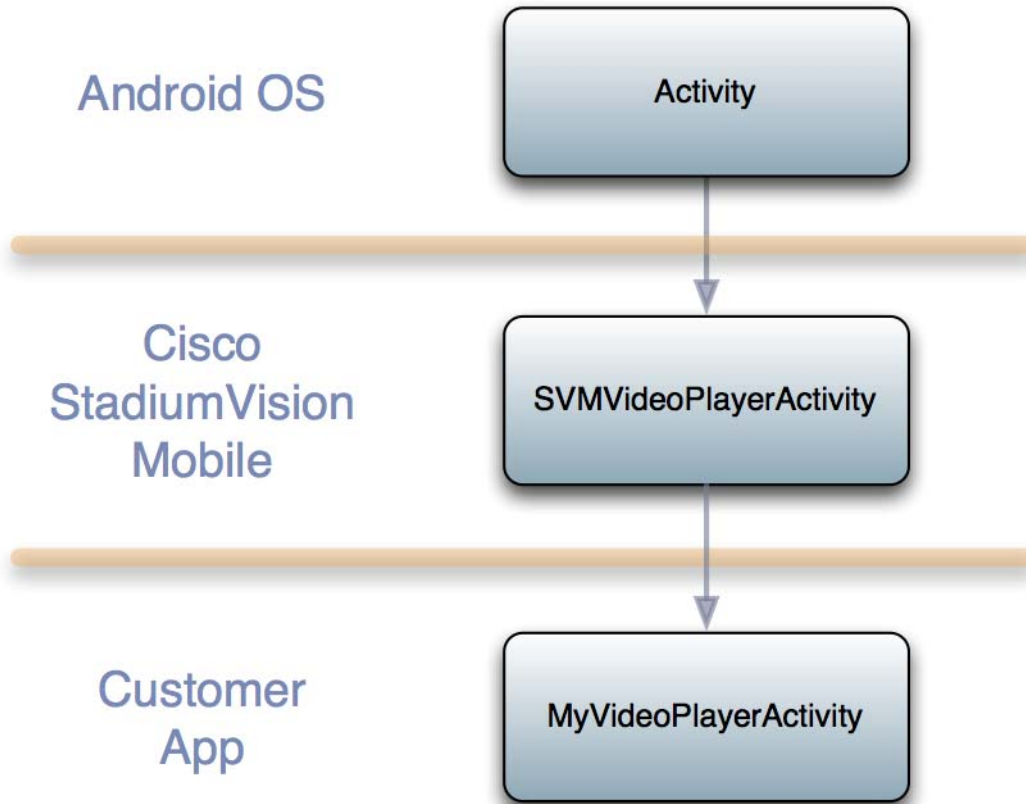


Figure 2-3 depicts the Activity inheritance between the Android OS, Cisco StadiumVision Mobile, and the client application.



**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. This section contains the following API calls and tables:

- [Start](#)
- [getVideoChannelArray](#)
- [getDataChannelArrayList](#)
- [addDataChannelObserver](#)
- [removeDataChannelObserver](#)
- [getStats](#)
- [getStats API Hash Keys and Stats Description](#)
- [onPause](#)
- [onResume](#)
- [getWifiInfo](#)
- [getBatteryInfo](#)

- [getLogLevelArray](#)
- [getLogLevelArrayList](#)
- [setLogLevel](#)
- [getLocalIpAddress](#)
- [getDeviceUUID](#)
- [getAppSessionUUID](#)
- [sdkVersion](#)

**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	<pre>// 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); }</pre>	

**Table 2-4** *Start*

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

**Table 2-6** *getDataChannelArrayList*

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

Table 2-7 *addDataChannelObserver*

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

Figure 2-4 *onData*

<b>Method Signature</b>	<code>public void onData(String channelName, byte[] data)</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	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.
<b>Result</b>	N/A

**Table 2-8** *removeDataChannelObserver*

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

**Table 2-9** *getStats*

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

<b>Stats Hash Key</b>	<b>Stats Description</b>
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

<b>Method Signature</b>	<code>public static void onPause();</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>
<b>Result</b>	N/A

Table 2-12 onResume

<b>Method Signature</b>	<code>public static void onResume();</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>
<b>Result</b>	N/A

Table 2-13 getWifiInfo

<b>Method Signature</b>	<code>public static SVMWifiInfo getWifiInfo();</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>This method returns the current WiFi network connection information.</li> <li>This information gets collected in the statistics information that gets uploaded to the Reporter server.</li> </ul>
<b>Result</b>	N/A

Table 2-14 *getBatteryInfo*

<b>Method Signature</b>	<code>public static SVMBatteryInfo getBatteryInfo();</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>This method returns the current device battery information.</li> <li>This information gets collected in the statistics information that gets uploaded to the Reporter server (if stats collection is enabled).</li> </ul>
<b>Result</b>	N/A

Table 2-15 *getLogLevelArray*

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

Table 2-16 *getLogLevelArrayList*

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

Table 2-17 *setLogLevel*

<b>Method Signature</b>	<code>public static SVMStatus setLogLevel(LogLevel level);</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	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.
<b>Result</b>	N/A

Table 2-18 *getLocalIpAddress*

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

Table 2-19 *getDeviceUUID*

<b>Method Signature</b>	<code>public static String getDeviceUUID();</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>This method returns the device UUID that was generated by the StadiumVision Mobile SDK and saved in the app's shared preferences.</li> <li>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.</li> </ul>
<b>Result</b>	N/A

Table 2-20 *getAppSessionUUID*

<b>Method Signature</b>	<code>public static String getAppSessionUUID();</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>This method returns the app session UUID that is generated by the StadiumVision Mobile SDK.</li> <li>This UUID uniquely identifies each time the StadiumVision Mobile SDK is started and is used for consistent statistics collection and reporting.</li> </ul>
<b>Result</b>	N/A

Table 2-21 *sdkVersion*

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

## Video Player Activity API Summary

The `SVMVideoPlayerActivity` class can be extended and customized. [Table 2-22](#) lists the `SVMVideoPlayerActivity` API methods, and contains the following tables:

- [setVideoSurfaceView](#)
- [seekRelative](#)
- [seekAbsolute](#)
- [rewindForDuration](#)
- [playLive](#)
- [shutdown](#)
- [setConfig](#)



- [setConfigWithString](#)
- [getConfig](#)
- [getStreamerArray](#)
- [getStreamerArrayList](#)

**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*

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

**Table 2-24** *seekRelative*

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

Table 2-25 *seekAbsolute*

<b>Method Signature</b>	<code>public SVMStatus seekAbsolute(int durationMs);</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>• This method moves the video play-head pointer to beginning of stream; relative to the “live” position.</li> <li>• To play most current live video pass in on offset of zero (0 ms).</li> <li>• To play most current live video pass in on offset of zero (0 ms).</li> <li>• 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).</li> </ul>
<b>Result</b>	N/A

Table 2-26 *rewindForDuration*

<b>Method Signature</b>	<code>public SVMStatus rewindForDuration(int durationMs);</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>• This method rewinds the video play-head within the video history buffer for the given amount of time (in milliseconds)</li> <li>• 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</li> <li>• 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)”.</li> </ul>
<b>Result</b>	N/A

Table 2-27 *playLive*

<b>Method Signature</b>	<code>public SVMStatus playLive();</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	<ul style="list-style-type: none"> <li>• This method forwards the video play-head to the starting “live” position at the beginning of the video data buffer</li> <li>• This convenience method acts as a wrapper for the “seekAbsolute” API method; making “playLive()” equivalent to “seekAbsolute(0)”.</li> </ul>
<b>Result</b>	N/A

Table 2-28 *shutdown*

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

Table 2-29 *setConfig*

<b>Method Signature</b>	<code>public static SVMStatus setConfig(JSONObject givenJsonConfig)</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method sets the SVM SDK configuration at run-time using a populated 'JSONObject' object. This method will override any configuration properties set with the 'cisco_svm.cfg' configuration file.
<b>Result</b>	NSDictionary*

Table 2-30 *setConfigWithString*

<b>Method Signature</b>	<code>public static SVMStatus setConfigWithString(String jsonConfigStr)</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method sets the SVM SDK configuration at run-time using a JSON-formatted 'String' object. This method will override any configuration properties set with the 'cisco_svm.cfg' configuration file.
<b>Result</b>	NSDictionary*

Table 2-31 *getConfig*

<b>Method Signature</b>	<code>public static JSONObject getConfig()</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method returns the current SDK configuration as a 'JSONObject' object.
<b>Result</b>	NSDictionary*

Table 2-32 *getStreamerArray*

<b>Method Signature</b>	<code>public static SVMStreamer getStreamerArray()</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method returns an array of Streamer servers detected by the SVM SDK; with each Streamer entry represented as an 'SVMStreamer' object in the array.
<b>Result</b>	NSArray*

Table 2-33 *getStreamerArrayList*

<b>Method Signature</b>	<code>public static ArrayList getStreamerArrayList()</code>
<b>Prerequisites</b>	N/A
<b>Notes</b>	This method returns an 'ArrayList' of Streamer servers detected by the SVM SDK; with each Streamer entry represented as an 'SVMStreamer' object in the array.
<b>Result</b>	NSArray*

## SDK Workflow

This section describes the SDK workflow, and contains the following sections:

- [Starting the SDK, page 2-19](#)
- [Getting the Video Channel List, page 2-19](#)
- [Presenting the Video Channel List, page 2-19](#)
- [Playing a Video Channel, page 2-20](#)
- [Seeking Within the Video Buffer, page 2-20](#)
- [Setting the Video Dimensions, page 2-20](#)
- [Fullscreen Video Layout, page 2-20](#)
- [Partial-Screen Video Layout, page 2-21](#)
- [Getting the Data Channel List, page 2-21](#)
- [Observing a Data Channel, page 2-21](#)
- [Activity Life-Cycle Notifications, page 2-22](#)
- [StadiumVision Mobile Service Up or Down Indicator, page 2-22](#)

- [In-Venue Detection](#), page 2-24
- [Set the SDK Configuration at Run-Time](#), page 2-25
- [Get the SDK Configuration](#), page 2-26
- [setConfigWithString API Method](#), page 2-26
- [Get the Available Streamer Servers](#), page 2-27
- [Additional Statistics](#), page 2-28
- [Video Player State Notifications](#), page 2-28
- [Cisco Demo Customized Video Player](#), page 2-31

## 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 2-34](#).

**Table 2-34** 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 its 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();
}

```

## StadiumVision Mobile Service Up or Down Indicator

Release 1.3 of the Cisco StadiumVision Mobile SDK includes an indicator to the application indicating if the SVM service is up or down. This indication should be used by the application to indicate to the user whether the SVM service is available or not. Service is declared 'down' by the SDK when any of the following are true:

- The SDK detects that the video quality is poor
- The SDK detects that no valid, licensed channel are available
- The mobile device's WiFi interface is disabled

Poor video quality can occur when the user is receiving a weak WiFi signal; causing data loss. There are two different ways that the iOS app can get the "Service State" from the SDK:

- Register to receive the "Service Up / Down" notifications
- Fetch the current service state from the SDK on-demand

When the app receives the "Service Down" notification, the SDK will supply a bitmap containing the reasons why the service was declared as 'down' by the SDK. The 'reasons' bitmap is given in



**Table 2-35 Service Down Notifications**

Service Down Reason	Constant
Poor video quality networking conditions detected	StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_POOR_QUALITY
WiFi connection is down	StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_WIFI_DOWN
No valid SVM channels have been detected	StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_NO_CHANNELS
SDK not running	StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_SDK_NOT_RUNNING

## Receiving "Service Up / Down" Notifications

The following example shows how to register and handle the "Service Up / Down" notifications from the SDK:

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

// define the service state broadcast receiver
private BroadcastReceiver serviceStateReceiver;

// create the service state broadcast receiver
serviceStateReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // get the intent extras
        Bundle bundle = intent.getExtras();

        // get the service state from the bundle
        SVMServiceState serviceState =
        (SVMServiceState)bundle.get(StadiumVisionMobile.SVM_SERVICE_STATE_VALUE_TAG);

        // determine the service state
        if (serviceState == SVMServiceState.SVM_SERVICE_STATE_UP) {
            Log.i(TAG, "### SERVICE STATE: UP");
        } else if (serviceState == SVMServiceState.SVM_SERVICE_STATE_DOWN) {
            Log.i(TAG, "### SERVICE STATE: DOWN");

            // get the service state changed reasons bitmap
            int reasons =
            bundle.getInt(StadiumVisionMobile.SVM_SERVICE_STATE_CHANGED_REASONS_TAG);

            // determine the reasons that the service state changed
            if ((reasons &
            StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_SDK_NOT_RUNNING) != 0) {
                Log.i(TAG, "Reason for Service State Change: SDK NOT RUNNING");
            } else if ((reasons &
            StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_WIFI_DOWN) != 0) {
                Log.i(TAG, "Reason for Service State Change: WIFI DOWN");
            } else if ((reasons &
            StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_NO_CHANNELS) != 0) {
                Log.i(TAG, "Reason for Service State Change: NO CHANNELS AVAILABLE");
            } else if ((reasons &
            StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_POOR_QUALITY) != 0) {
                Log.i(TAG, "Reason for Service State Change: POOR QUALITY");
            }
        }
    }
}
```

```

    }
  }
};

// register to receive the service state intents
IntentFilter serviceStateIntentFilter = new IntentFilter();
serviceStateIntentFilter.addAction(StadiumVisionMobile.SVM_SERVICE_STATE_CHANGED_INTENT_TAG);
registerReceiver(serviceStateReceiver, serviceStateIntentFilter);

```

## Get the Current "Service Up / Down" State On-Demand

The "getServiceState" API method can be used to fetch the current service state from the SDK. The following example show how to fetch the current service state from the SDK using the "getServiceState" API call:

```

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

// get the current svm service state
SVMServiceState serviceState = StadiumVisionMobile.getServiceState();

// determine the current service state
if (serviceState == SVMServiceState.SVM_SERVICE_STATE_UP) {
    Log.i(TAG, "### SERVICE STATE: UP");
} else if (serviceState == SVMServiceState.SVM_SERVICE_STATE_DOWN) {
    Log.i(TAG, "### SERVICE STATE: DOWN");
}

```

## In-Venue Detection

The Cisco StadiumVision Mobile Release 1.3 SDK provides a mechanism to detect whether the mobile device is connected within the SVM-enabled venue or not.

There are two different ways that the Android app can get this "In-Venue Detection" state from the SDK:

- Register to receive the "In-Venue Detection" notifications
- Fetch the current "In-Venue" state from the SDK on-demand

### Receiving "In-Venue Detection" Notifications

The following example shows how to register and handle the "Service Up / Down" notifications from the SDK:

```

import com.cisco.svm.app.StadiumVisionMobile;

// define the 'in-venue status changed' broadcast receiver
private BroadcastReceiver inVenueReceiver;

// handle the venue connection changed event
venueConnectionReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // get the intent action
        String action = intent.getAction();

        // determine whether the device is inside or outside of the venue
        if (action.equals(StadiumVisionMobile.SVM_VENUE_CONNECTED_INTENT_TAG)) {
            Log.i(TAG, "##### App Received 'VENUE-CONNECTED' Notification");
        } else if (action.equals(StadiumVisionMobile.SVM_VENUE_DISCONNECTED_INTENT_TAG)) {
            Log.i(TAG, "##### App Received 'VENUE-DISCONNECTED' Notification");
        }
    }
}

```

```

    }
}
};

// register to receive the venue connected / disconnected intents
IntentFilter inVenueIntentFilter = new IntentFilter();
inVenueIntentFilter.addAction(StadiumVisionMobile.SVM_VENUE_CONNECTED_INTENT_TAG);
inVenueIntentFilter.addAction(StadiumVisionMobile.SVM_VENUE_DISCONNECTED_INTENT_TAG);
registerReceiver(venueConnectionReceiver, inVenueIntentFilter);

```

### Get the Current "In-Venue" State On-Demand

The "isConnectedToVenue" API method can be used to fetch the current in-venue state from the SDK. The following example shows how to fetch the current service state from the SDK using the "isConnectedToVenue" API call:

```

import com.cisco.svm.app.StadiumVisionMobile;

// get whether the device is currently connected to the SVM licensed venue
boolean isConnectedToVenue = StadiumVisionMobile.isConnectedToVenue();

// log whether the device is currently connected to the SVM licensed venue
Log.i(TAG, "### Connected to the venue: " + (isConnectedToVenue ? "YES" : "NO"));

```

### Set the SDK Configuration at Run-Time

Previously, the Cisco StadiumVision Mobile SDK could only be configured by using a JSON-formatted config file ("cisco\_svm.cfg") bundled within the Android app. Starting with the 1.3 release, the application can now set the SDK configuration at run-time through an API method. This allows the application to dynamically configure the SDK. For example, the application can fetch the SDK configuration information from a network connection, and then pass that configuration to the SDK.

Two different methods are provided for setting the SDK configuration at run-time:

- "setConfig"

The signature of the "setConfig" API method is given below:

```

// configure the sdk using a JSON object containing the configuration settings
public static SVMStatus setConfig(JSONObject givenJsonConfig)

```

// configure the sdk using an nsdictionary containing the configuration settings

- "setConfigWithString"

The signature of the "setConfigWithString" API method is given below:

```

// configure the sdk using a json-formated string containing the configuration
settings
public static SVMStatus setConfigWithString(String jsonConfigStr)

```

The following example shows how to set the SDK configuration using the "setConfigWithString" API method:

```

// create the json config string
String configString =
    @"{
    "    \license\": {
    "        \venueName\": \MyVenueNameKey\",
    "        \contentOwner\": \MyContentOwnerKey\",
    "        \appDeveloper\": \MyAppDeveloperKey\"
    "    }
    }";

```

## Get the SDK Configuration

### "getConfig" API Method#

The signature of the "getConfig" API method is given below:

```
// get the current cisco sdk configuration
public static JSONObject getConfig()
```

The example below fetches the current configuration from the SDK, and then accesses the configuration values in the configuration JSON object:

```
// get the sdk configuration dictionary
JSONObject configObj = StadiumVisionMobile.getConfig();

// get the license dictionary from the config dictionary
JSONObject licenseObj = null;
try {
    licenseObj = configObj.getJSONObject("license");
} catch (JSONException e) {
    e.printStackTrace();
}

// if the license object is valid
if (licenseObj != null) {
    // get the current set of configured license keys
    String venueName = licenseObj.getString("venueName");
    String contentOwner = licenseObj.getString("contentOwner");
    String appDeveloper = licenseObj.getString("appDeveloper");
}
}
```

The following example shows how to set the SDK configuration using the "setConfig" API method:

```
// create the config json object with the set of licensing keys
JSONObject jsonConfig = new JSONObject();
JSONObject licenseConfig = new JSONObject();
try {
    licenseConfig.put("venueName", "MyVenueNameKey");
    licenseConfig.put("contentOwner", "MyContentOwnerKey");
    licenseConfig.put("appDeveloper", "MyAppDeveloperKey");
    jsonConfig.put("license", licenseConfig);
} catch (JSONException e) {
    // log the error
    Log.e(TAG, "Error building the json config object");
    e.printStackTrace();
}

// update the cisco sdk configuration at run-time
StadiumVisionMobile.setConfig(jsonConfig);
```

## setConfigWithString API Method

The signature of the "setConfigWithString" API method is given below:

```
// configure the sdk using a json-formated string containing the configuration settings
public static SVMStatus setConfigWithString(String jsonConfigStr)
```

The following example shows how to set the SDK configuration using the "setConfigWithString" API method:

```
// create the cisco sdk json configuration string
String config =
    "{" +
    "  \"license\": {\" +
    "    \"venueName\": \"MyVenueNameKey\", \" +
    "    \"contentOwner\": \"MyContentOwnerKey\", \" +
    "    \"appDeveloper\": \"MyAppDeveloperKey\"\" +
    "  }\" +
    "}";

// update the cisco sdk configuration at run-time
StadiumVisionMobile.setConfigWithString(config);
```

## Get the Available Streamer Servers

The Android SDK detects the available Streamer servers and provides an API to get the list of servers. A venue will typically only have a single Streamer server. The list is presented as an array of "SVMStreamer" objects.

There are two different methods available that present the "SVMStreamer" objects in either a Java array or ArrayList collection. The signatures for the two API methods are given below:

```
// get the detected streamer servers as a java array of "SVMStreamer" objects
public static SVMStreamer[] getStreamerArray()

// get the detected streamer servers as a java ArrayList of "SVMStreamer" objects
public static ArrayList<SVMStreamer> getStreamerArrayList()
```

Each "SVMStreamer" object contains the following properties listed in [Table 2-36](#).

**Table 2-36** "SVMStreamer" Object Properties

"SVMStreamer" Property	Type	Description
"ipAddress"	String	IP address of the StadiumVision Mobile streamer server
"isAllowed"	boolean	Whether this StadiumVision Mobile Streamer server is allowed by the user of this SDK
"statsPublishIntervalMs"	int	SDK stats HTTP upload interval
"statsSampleIntervalMs"	int	SDK stats sample interval
"statsUploadUrl"	String	StadiumVision Mobile Reporter stats upload http url

The following example shows how to get the list of StadiumVision Mobile Streamer servers detected by the SDK:

```
// get the list of currently available streamer servers
ArrayList<SVMStreamer> streamerList = StadiumVisionMobile.getStreamerArrayList();

// iterate through the list of streamer objects
for (SVMStreamer nextStreamer: streamerList) {
    // get the properties of the next streamer server object
    String ipAddress = nextStreamer.getIpAddress();
    String statsUploadUrl = nextStreamer.getStatsUploadUrl();
    int statsSampleIntervalMs = nextStreamer.getStatsSampleIntervalMs();
    int statsPublishIntervalMs = nextStreamer.getStatsPublishIntervalMs();
    boolean isAllowed = nextStreamer.isAllowed();
}
```

## Additional Statistics

In the Cisco StadiumVision Mobile Release 1.3 SDK, the existing "stats" API call returns the following additional categories of stats information:

- Reporter upload stats
- Multicast channel announcement stats
- Licensing stats

The signature of the existing "getStats" API method is given below:

```
// get the current set of cisco sdk stats as a hashmap
public static HashMap<String, String> getStats()
```

The new stats and their associate dictionary keys and description are given in [Table 2-37](#).

**Table 2-37** Stats API Dictionary Keys

Dictionary Key	Description
"announcementsMalformed"	Number of malformed channel announcement packets received
"announcementsMissed"	Number of total channel announcements missed
"announcementsNotAllowed"	Number of announcements where the Streamer is not allowed
"announcementsReceived"	Number of total channel announcements received
"channelsAdded"	Number of times a channel was added to the channel list
"channelsPruned"	Number of times a channel was pruned from the channel list
"invalidJsonAnnouncements"	Number of announcements with an invalid JSON body
"licenseMismatchAnnouncements"	Number of license key mismatches
"statsUploadFailures"	Number of Reporter stats upload failures
"statsUploadSuccesses"	Number of Reporter stats upload successes
"versionMismatchAnnouncements"	Number of announcement version mismatches
"statsUploadAttempts"	Number of Reporter stats upload attempts

## Video Player State Notifications

The 1.3 SDK generates broadcast Intent notifications for each of the video player state transitions (listed in [Table 2-38](#)). The application can listen to these notifications and take action based on the video player's state transitions.

**Table 2-38** Video Player State Notification

Video Player State Notification	Description
StadiumVisionMobile.SVM_VIDEO_CLOSED_STATE	Occurs when the video player closes the video channel session
StadiumVisionMobile.SVM_VIDEO_DESTROYED_STATE	Occurs when the video player is terminated and destroyed
StadiumVisionMobile.SVM_VIDEO_PAUSED_STATE	Occurs when the video player pauses video playback
StadiumVisionMobile.SVM_VIDEO_PLAYING_STATE	Occurs when the video player starts playing the video channel
StadiumVisionMobile.SVM_VIDEO_RESTARTING_STATE	Occurs when the video player restarts video playback
StadiumVisionMobile.SVM_VIDEO_STOPPED_STATE	Occurs when the video player stops video playback

The following example shows how to subscribe to receive the video player Intent broadcast messages, and then parse the messages for the (1) channel name and (2) video player state:

```
// create the channel state change broadcast receiver
channelStateReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // get the intent action
        String action = intent.getAction();

        // get the intent extras
        Bundle bundle = intent.getExtras();

        // determine the broadcast intent type
        if (action.equals(StadiumVisionMobile.SVM_CHANNEL_STATE_CHANGED_INTENT_TAG)) {
            // get the updated channel name and state info
            String channelName =
                (String)bundle.get(StadiumVisionMobile.SVM_CHANNEL_NAME_VALUE_TAG);
            String channelState =
                (String)bundle.get(StadiumVisionMobile.SVM_CHANNEL_STATE_VALUE_TAG);

            // determine the channel state
            if (channelState.equals(StadiumVisionMobile.SVM_VIDEO_PLAYING_STATE) == true)
            {
                // channel is now playing
            }
        }
    }
};

// create the intent filter
IntentFilter channelStateReceiverIntentFilter = new IntentFilter();
channelStateReceiverIntentFilter.addAction(StadiumVisionMobile.SVM_CHANNEL_STATE_CHANGED_I
NTENT_TAG);

// register the intent filter
context.registerReceiver(channelStateReceiver, channelStateReceiverIntentFilter);
```

## Video Player "Channel Inactive" Callback

To detect that a currently playing video channel has become invalid (due to Streamer server admin changes), the SVM video player ("SVMVideoPlayerActivity") provides a callback to tell the video player sub-class (ie: "MyVideoPlayerActivity") that the currently playing channel is no longer valid.

When a channel becomes invalid, playback of the video channel is automatically stopped.

To receive these callbacks, the "onCurrentChannelInvalid" method must be overridden by the 'SVMVideoPlayerActivity' sub-class (ie: "MyVideoPlayerActivity"). The following example shows the method signature and implementation of this overridden callback method:

```
@Override
protected void onCurrentChannelInvalid() {
    // call the parent method
    super.onCurrentChannelInvalid();

    /*
     * This "MyVideoPlayerActivity" implements the following app-specific
     * behavior when receiving the 'onCurrentChannelInvalid' callback
     * from the Cisco SVM SDK
     *
     * 1) Stop video player
     * 2) Display a toast message describing why video playback was stopped
     * 3) Dismiss the video player Activity
     */

    // shutdown video playback
    shutdown();

    // display a notification that the channel is no longer valid
    Toast.makeText(this, "\nChannel is no longer valid and the video player has been
stopped\n", Toast.LENGTH_LONG).show();

    // exit this video player activity now
    thisActivity.finish();
}
```

## Video Player Customization

This section describes customizing the 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*

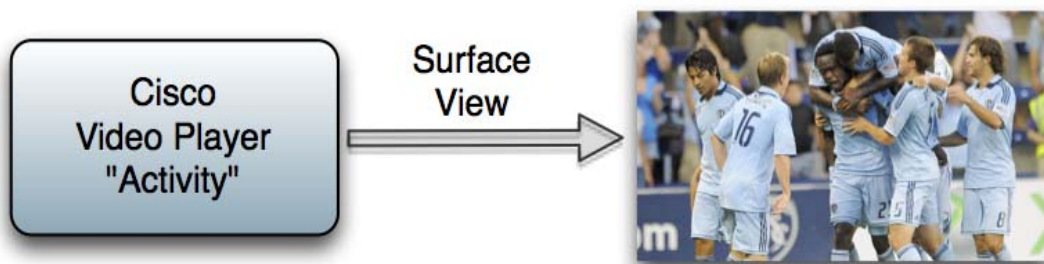
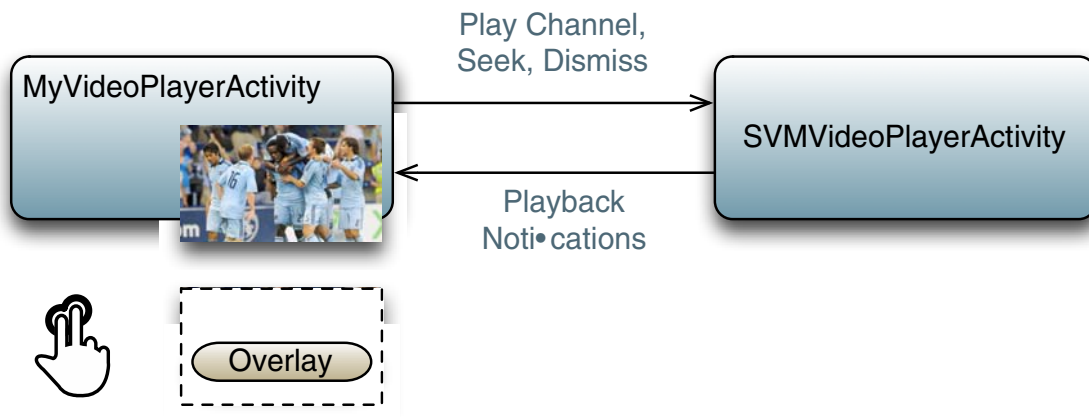




Figure 2-6 SVMVideoPlayerActivity API



## 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

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

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

public class MyVideoPlayer extends SVMVideoPlayerActivity {
}

```

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>

```

## 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 2-39** 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: <ul style="list-style-type: none"> <li>• Venue Name</li> <li>• Content Owner</li> <li>• App Developer</li> </ul>
“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"
      }
    ]
  }
}
```

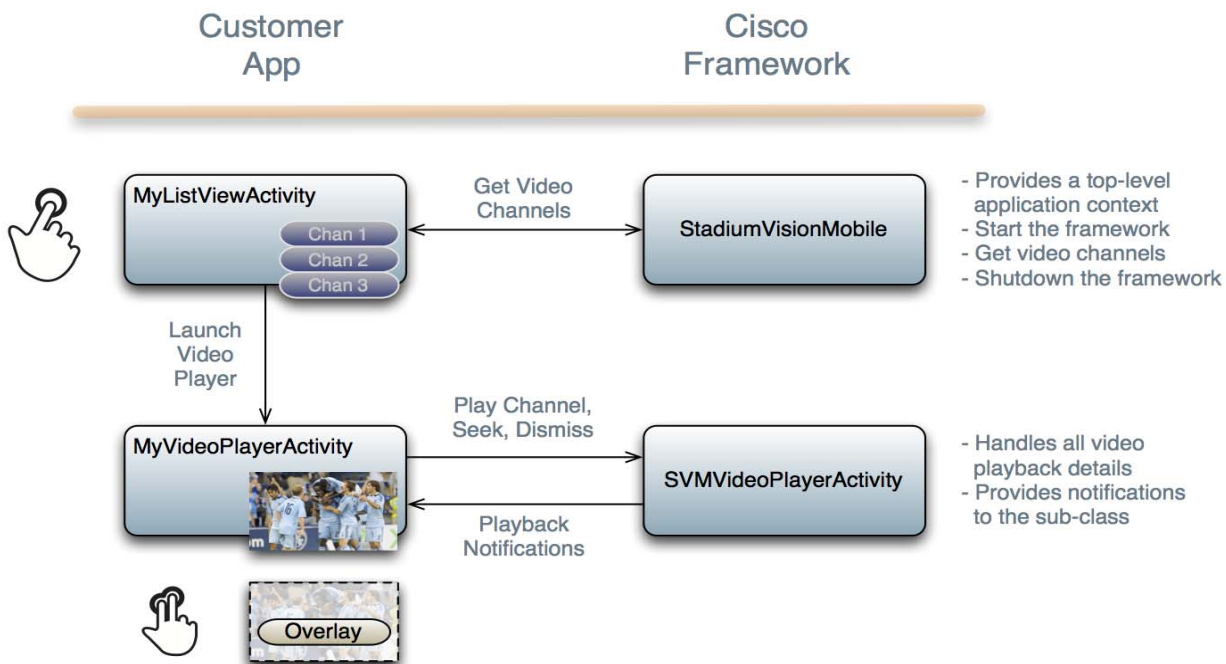
# Integration

## Client Application Integration Overview

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

- [Integration Checklist, page 2-33](#)
- [Customer Application Roles, page 2-34](#)
- [Android Permissions, page 2-35](#)
- [SDK Native Libraries, page 2-35](#)

**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

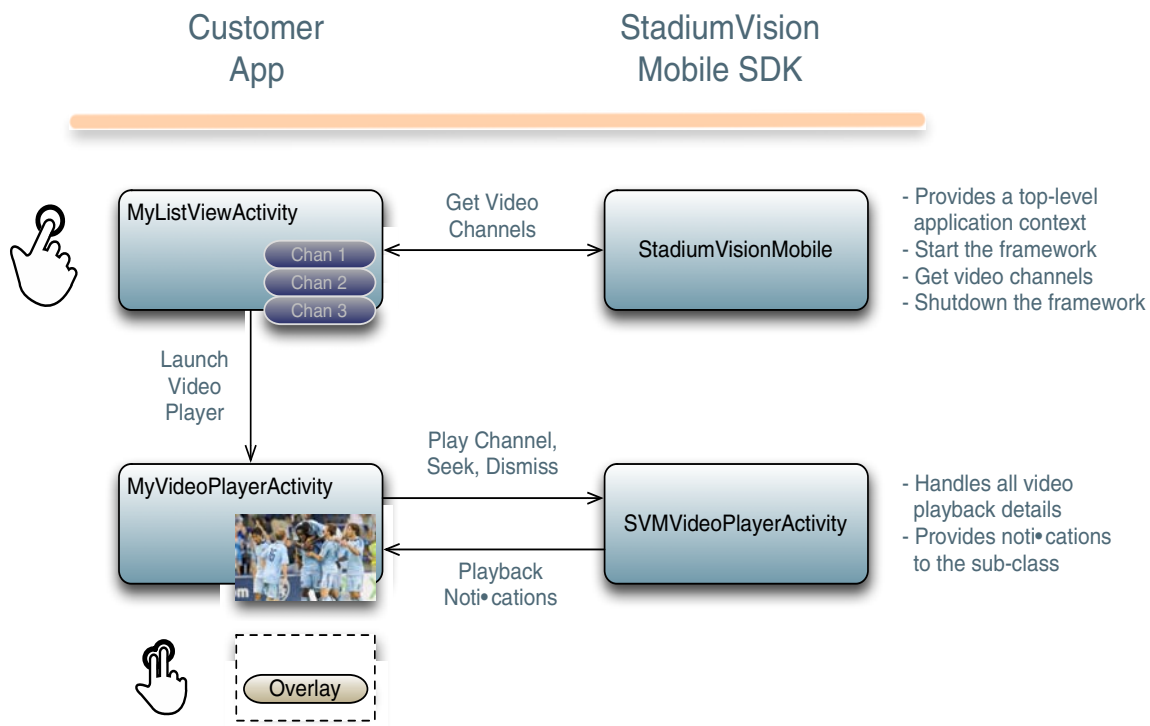
- Add the Java and native libraries to the app's "libs" folder
- 5. Set a Video "SurfaceView"
  - Add a "SurfaceView" to the player Activity's layout XML file
- 6. Life-Cycle Notifications
  - Forward life-cycle notifications to the StadiumVision Mobile SDK
- 7. Android Project Build Paths
  - Set the 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
```

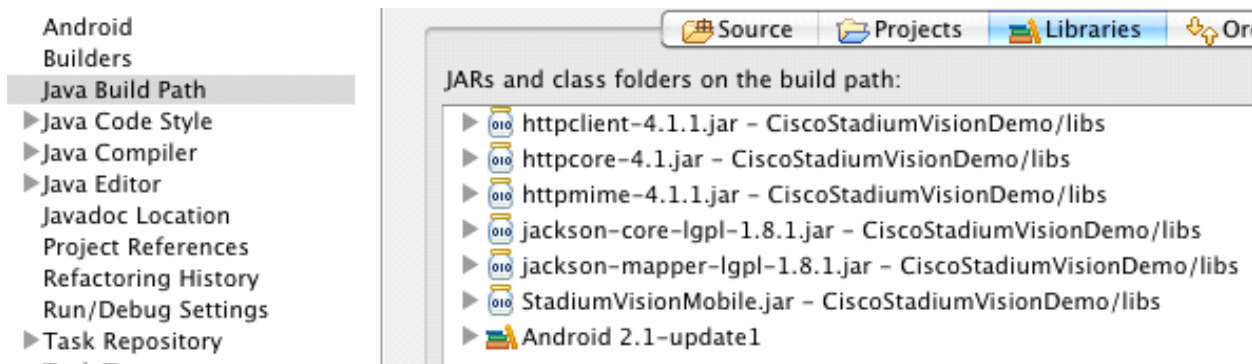
### Android Project Classpath

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
- Update 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);  
    }  
});
```