Cisco StadiumVision Mobile SDK Programmer's Guide for Apple iOS, Google Android, and Windows Phone

Release 2.1
June 12, 2015

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

Cisco StadiumVision Mobile SDK Programmer's Guide

CONTENTS

Preface

First Published: May 26, 2015

Revised: June 12, 2015

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 and supplements the API documentation Doxygen build included with the SDK.

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.

Document Revision History

Table 1 Document Revision History

Date	Change Summary
June 12, 2015	• Included a new best practice under Delivering Channel Content for Apple iOS, Google Android, and Windows Phone in the "Cisco StadiumVision Mobile SDK Best Practices" section on page 9. • Added a note in the "Cisco StadiumVision Mobile API for Apple iOS" indicating that Release 2.1 does not include two files ("libvoCTS.a" and "voVidDec.dat") that were previously included. • Added "EVS C-Cast Integration" section on page 28 for Windows Phone. • Changed the title of the guide to: *Cisco StadiumVision Mobile SDK Programmer's Guide*, *Release 2.1*.
June 5, 2015	Revised the "EVS C-Cast Integration" section on page 6.
May 26, 2015	Initial version of *Cisco StadiumVision Mobile SDK Guide*, *Release 2.1*.

About Cisco StadiumVision Mobile

Cisco StadiumVision Mobile (SVM) enables reliable and scalable delivery of low-delay video and data streams to Wi-Fi 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.

Obtaining the SDK

Please contact your Cisco account team to become part of the Cisco 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 Introduction

First Published: May 26, 2015

Revised: June 12, 2015

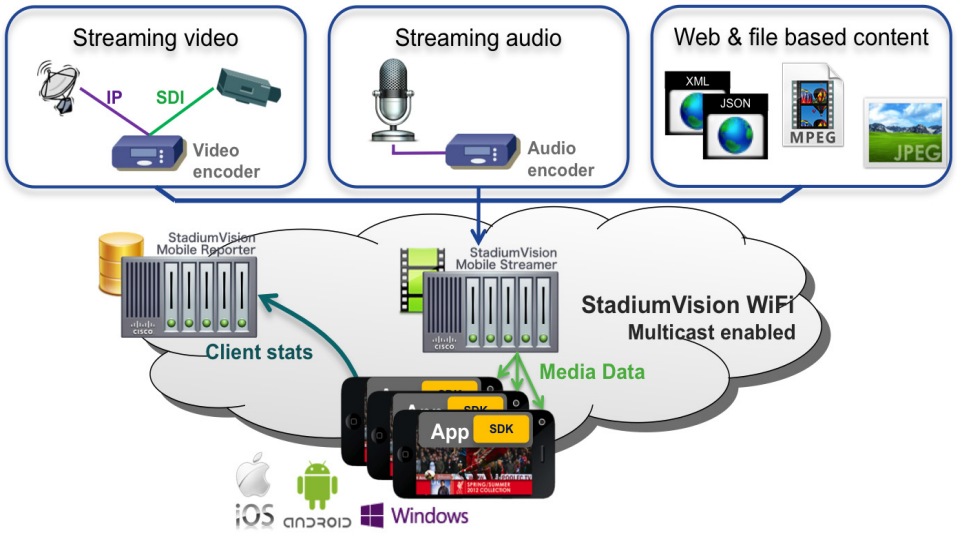
This chapter provides an overview of the Cisco StadiumVision Mobile solution and contains the following sections:

Cisco StadiumVision Mobile Solution Overview

The Cisco StadiumVision Mobile (SVM) solution enables the reliable delivery of low-delay video and data streams to fans' Wi-Fi devices at venues. Figure 1-1 illustrates a high-level view of the Cisco StadiumVision Mobile solution, which has the following attributes:

- Consists of Video Encoder, Streamer and Reporter products
- Requires integration of Cisco Client SDK in the mobile application
- Builds upon Cisco Connected Stadium and Cisco Connected Stadium Wi-Fi solutions

Figure 1-1 Cisco StadiumVision Mobile Architecture



Key Terms and Concepts

The following are key terms and concepts as they apply to the Cisco StadiumVision Mobile solution.

Cisco Demo or Sample App: A standalone mobile application available to a Stadium Operator for testing and evaluating the Cisco StadiumVision Mobile solution.

Repair: In the context of Cisco StadiumVision Mobile, an application-layer mechanism that allows Cisco StadiumVision Mobile Clients to recover lost packets.

Stadium Operator: The entity hosting and configuring the Cisco StadiumVision Mobile solution.

SVM: Cisco StadiumVision Mobile

SVM Reporter: A standalone application used to collect Cisco StadiumVision Mobile Client statistics.

SVM Session: The protocol and associated parameters which define the sender and receiver configuration for the streaming of content.

SVM Session Announcement/Discovery: Methods used by the Cisco StadiumVision Mobile Streamer and SVM Client to allow a mobile device to obtain the list of available sessions and associated session metadata.

SVM Session Triplet Key: A specific combination of **Venue**, **Content Owner**, and **App Developer** used by the SVM Streamer and SVM Client to limit session discovery and content consumption to authorized applications. For additional information, see "Content Access Control–Triplet Key" section on page 1-7. The triplet key components are defined as follows:

- **Venue**: A text string associated with the venue where an Cisco StadiumVision Mobile Streamer is hosted.
- **Content Owner**: A text string associated with an entity that wishes to distribute content over the SVM solution.
- **App Developer**: The text string associated with the Application Developer authorized by a Content Owner to consume the Content Owner's content over the SVM solution.

SVM Streamer: A standalone application used to aggregate and send content to mobile applications with an embedded Cisco StadiumVision Mobile Client.

SVM System: An end-to-end solution for the delivery of digital media content streams consisting of specific products (Video Encoder, Cisco StadiumVision Mobile Streamer, Cisco StadiumVision Mobile Reporter), wireline and wireless infrastructure (Connected Stadium, Connected Stadium Wi-Fi) and mobile apps with an embedded Cisco StadiumVision Mobile Client.

Cisco Stadium Vision Mobile Media Input Types

The Cisco StadiumVision Mobile solution can accept multiple forms of media content input (in-house video feed, IP video feed, and IP data feeds). The media is then routed through the appropriate encoder and into the Streamer. The Cisco StadiumVision Mobile Streamer is a critical component in the Cisco StadiumVision Mobile solution that aggregates video, audio, data, and file content streams and supports the following content types:

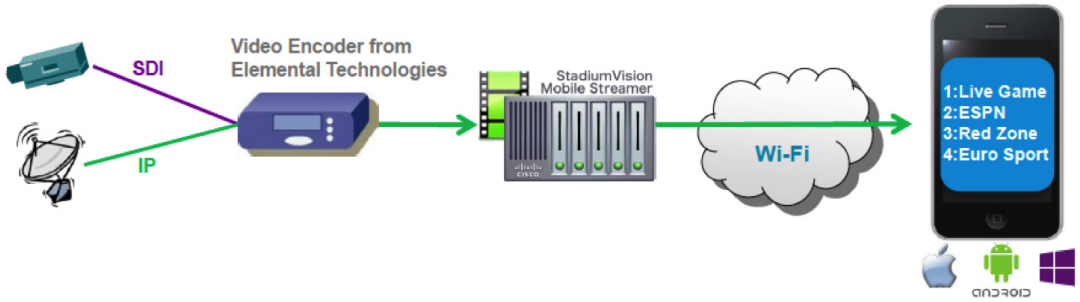
Table 1-1 Channel Types Supported and Use Cases

Channel Type	Maximum Number of Channels	Example Use Cases
Streaming Video	4	• Delivers live real-time venue game feed. • Loops the most recent replays from the live in-venue game. • Provide live out-of-venue game feed taking place at the same time.
Streaming Audio (Available in Android SDK only)	10	• Provides the choice of commentary in multiple languages. • Offers the choice to select between the home team and the away team commentators.
Data	4	• Distributes data through push and pull channels to the client app. **Note** The total number of data channels is 4.
• Data Push		• Triggers all mobile devices to display the same content at the same time, typically used for a sponsored moment of exclusivity.
• Data Pull		• Delivers real-time game statistics overlaid on the video pane.
File	1	• Provides video replays, delivered by EVS C-Cast.

Streaming Video Channels

Figure 1-2 shows the video channel (SDI or IP) inputs in the Cisco StadiumVision Mobile solution. Streaming video can be real-time in-venue game feed, live out of the venue game taking place at the same venue, or loop the most recent replays from the live in-venue game.

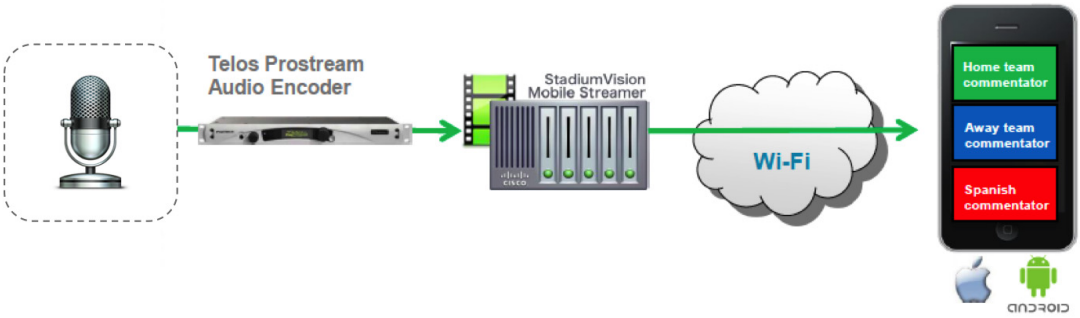
Figure 1-2 Cisco StadiumVision Mobile Streaming Video



Streaming Audio Channels

Figure 1-3 shows how Cisco StadiumVision Mobile allows audio channels to compliment the live game experience. Audio channels consume less bandwidth than video channels, which allows for more room for channels.

Figure 1-3 Cisco StadiumVision Mobile Streaming Audio



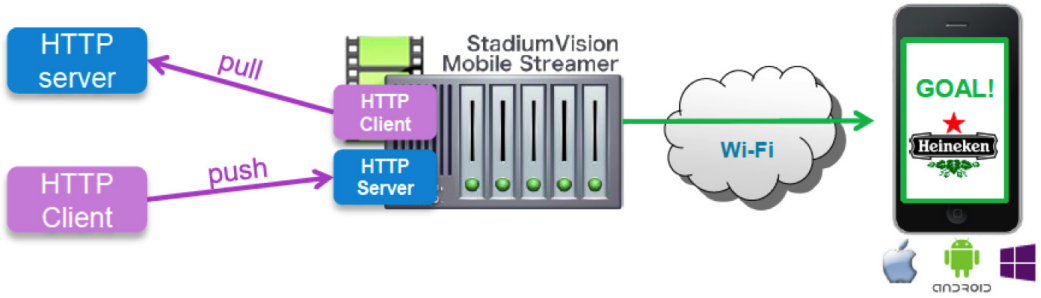
Note Streaming audio is only supported in the Android SDK.

Data Channels

Figure 1-4 shows how Cisco StadiumVision Mobile allows for the data push and pull channels to distribute data of any kind to the client app. The system integrator can decide what types of data is distributed and how it is used by the application. There are two types of data channels:

- Pull: Streamer polls an external web server at regular intervals (default 10 seconds). This channel can be used for data that changes periodically such statistics or thumbnails.
- Push: An external computer pushes data to the Cisco StadiumVision Streamer. This channel can be used for sending on-demand triggers to all mobile devices, for example when a goal is scored.

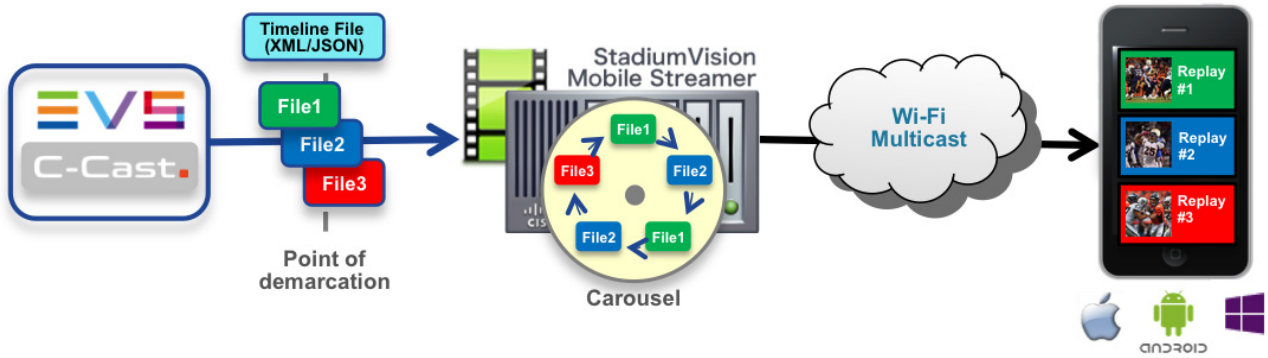
Figure 1-4 Cisco StadiumVision Mobile Data Channels



File Channels

Figure 1-5 shows how Cisco StadiumVision Mobile enables file channels as a way to distribute file-based (video, audio, and data) content to a large number of mobile clients over multicast.

Figure 1-5 Cisco StadiumVision Mobile Files Channels



File Channel Distribution

File channels are often used for replays to mobile devices at a live event where the bottleneck in a stadium is the Wi-Fi network that serves tens of thousands of fans with mobile devices. In order to scale, you can use Cisco StadiumVision Mobile Scalable File Distribution (SFD).

SFD uses multicast over Wi-Fi to scale distribution of the C-Cast video files. Multicast works much like over-the-air broadcast TV where your local TV station sends out a single signal that anyone in the area can receive with an antenna on the roof. From a load perspective it makes no difference to the TV broadcaster if ten subscribers or ten thousand subscribers are watching. Cisco StadiumVision Mobile SFD works in a similar way by sending the files as a single multicast transmission, and any number of mobile devices in the stadium can listen to that signal, receive the file and cache it in local storage for later use.

Generic Ingest

Note Generic ingest is not supported in Cisco StadiumVision Release 2.1.

EVS C-Cast Integration

EVS C-Cast is the platform that Cisco StadiumVision Mobile uses for making replay video clips available to mobile clients over high density Wi-Fi networks. Read more about EVS C-Cast at:

http://www.evs.com/nala/product/c-cast

Note Cisco StadiumVision Mobile is supported with EVS C-Cast version 2.x only. EVS C-Cast version 3.x is not supported.

The traditional way of scaling C-Cast content delivery to a large number of clients is by using a Content Delivery Network (CDN). The CDN caches the content closer to the client, and thus avoids the need for every client to reach back and retrieve the content from the C-Cast Central server. This offloads the C-Cast Central server and reduces the amount of duplicate content that has to traverse the network.

However, the CDN approach does not help in a Wi-Fi environment where the bottleneck is the last 25 meters from an access point to a mobile device. SVM solves this problem by multicasting the C-Cast replays to mobile devices, and therefore avoids sending replays individually to each and every device.

Overview

In a traditional unicast C-Cast deployment (without SVM integration) the client app fetches an XML or JSON formatted C-Cast timeline via HTTP. The timeline describes the sequence of replays (C-Cast calls them events), and the camera angles available for each. Each camera angle consists of a thumbnail graphic and a MP4 video file.

From the perspective of the C-Cast mobile app there is very little difference between the traditional unicast and the Cisco StadiumVision Mobile multicast scenarios. In both cases, the exact same C-Cast timeline provides the app with the info it needs to make replays available to the user. And in both cases, the standard C-Cast media files are used. The only difference between the two scenarios is the transport mechanism used to the deliver the timeline and media files to the mobile devices. And this difference is largely, but not completely, hidden by the Cisco StadiumVision Mobile SDK. The C-Cast timeline is delivered via an SVM data channel and the corresponding media files are delivered via an SVM file channel.

Operation

When an SVM enabled C-Cast app is launched, it starts listening for the timeline on the SVM data channel. By default the timeline is repeated every 10 seconds, so the app may have to wait for up to 10 seconds to receive the latest timeline.

With the timeline in hand, the app extracts the media filenames for each replay and camera angle. The app then queries the SVM SDK to see if the file of interest has been received on the file channel and cached in local storage on the mobile device. If the file is available, the app proceeds to use it.

If a large number of replays and/or camera angles are being distributed, it can take several minutes for all media files to be received by the mobile device. It is therefore entirely possible that the app queries the SVM SDK for a specific file only to find that the file is not yet available. In order to provide a responsive user experience, the app should handle this situation by retrieving the missing file directly from C-Cast via unicast HTTP. The URL for this operation can be constructed from the timeline metadata. Consult the C-Cast API guide for instruction on how to create this URL.

This approach for managing files that are not yet available is referred to as hybrid unicast/multicast. By combining the two we are able to provide a solution that offers scalability and responsiveness.

To obtain the EVS C-Cast API, contact James Stellphlug (j.stellpflug@evs.com) stating that you are developing an app to consume C-Cast clips in a Cisco StadiumVision Mobile venue.

Content Access Control–Triplet Key

An important feature of the Cisco StadiumVision Mobile solution is to limit the consumption of Cisco StadiumVision Mobile encoded content to authorized mobile applications. Consider the following situation:

Content Owner A (e.g., sports team) wishes to use the Cisco StadiumVision Mobile solution to deliver live camera feeds to fans throughout a venue during the team's home games. Content Owner B (e.g., entertainment company) plans to host events at the same venue at a different time and also wishes to deliver live feeds to their fans. The two Content Owners each want to limit content consumption to their chosen and therefore authorized, Application Developer. The reasons for needing to limit content consumption to authorized mobile apps are many. For example, the app might need to be purchased or it may be sponsored by an advertiser. As a result, Cisco StadiumVision Mobile video and data streams configured for Content Owner A's mobile app must not be consumed by Content Owner B's mobile app and vice-versa.

The Cisco StadiumVision Mobile Streamer includes a Triplet (Venue/Content Owner/App Developer) in each announced session. Only mobile apps with the identical Triplet will be able to discover Cisco StadiumVision Mobile sessions and consume the associated content. The Streamer may be configured to support multiple Content Owner and App Developer combinations, though only a single Triplet may be active at any one time.

The Venue, Content Owner, Application Developer (triplet key) settings are critical to enabling content consumption on mobile devices. The Streamer settings must match those used by the App Developer for content to be discovered and consumed by a mobile app. App Developers must be notified of a change in Venue name so that their app may be updated. Conversely, if the App Developer has already deployed the app, App Developers must also be notified if the associated Content Owner/App Developer setting on the Streamer is modified.

Note The Stadium Operator is responsible for correctly configuring the Streamer and working with Content Owner/App Developer to enable content consumption. The Content Owner/App Developer paring must match the values hard coded into the specific SDK for the App Developer contracted for a particular venue.

Additional information regarding the triplet key is available in the *Cisco StadiumVision Mobile Streamer Administration Guide* available on Cisco.com at:

http://www.cisco.com/c/en/us/support/video/stadiumvision/products-maintenance-guides-list.html

Testing Your Cisco StadiumVision Mobile App

The Cisco StadiumVision Mobile SDK includes the ability for developers to test their application without being connected to a Cisco Connected Wi-Fi Network. This capability is provided with a set of files that can emulate the data received over the network. The clean.stream file that comes bundled with the SDK contains just one video channel.

To provide app developers with additional ways to test multiple channels, an additional set of clean.stream files is available for use on Apple iOS and Google Android. This package includes a number of Cisco StadiumVision Mobile stream files for local playback on a mobile device. The stream files enable an SVM app developer to perform some basic testing when access to the Cisco StadiumVision Mobile backend infrastructure is not available.

You can download the set of files at the same location where you obtained the Cisco StadiumVision Mobile SDK tar.bz2 file or contact your Cisco account team for details as to how to become part of the Cisco StadiumVision Mobile SDK partner program by sending an email to:

svm-sdk@external.cisco.com.

We recommend that you get started using these stream files with the included Cisco StadiumVision Mobile demo app before attempting to use it with the app you are developing. For details consult the read-me files included in the root folder of the SDK package.

All stream files are encoded with the default triplet below. The app listening to these stream files must use this exact triplet in order to receive the streams.

```
"license": { "venueName": "VenueName", "contentOwner": "ContentOwner", "appDeveloper":
"AppDeveloper" }
```

The following stream files are included:

- **video.stream:** This stream file contains 4 video channels, as opposed to the one video channel in the stream file that comes bundled with the SDK. The stream file starts off with one channel. Every 10 seconds another channel appears, until all 4 channels are present. The channels then start dropping off one by one, until there are none left.

  Note The video on channel 1 freezes when the channel disappears from the channel lineup. In order to keep the file size reasonable, only one of the channels includes media.

- **audio.stream:** This stream file contains one audio channel.

  Note Only the Android SDK supports audio channels.

- **ccast.stream:** This stream file contains the required streams to test basic EVS C-Cast functionality. The stream file contains these 3 channels:
  - CcastTimeLineXML: Data channel carrying the C-cast timeline in XML format.
  - CcastTimeLineJSON: Data channel carrying the C-cast timeline in JSON format.
  - CcastTimeMedia: File channel carrying the C-Cast mp4 video and jpg thumbnail files.

Note The iOS Sample app 2.1 does not support file channels. The iOS SDK does however have full support file channels.

Please direct your questions to: svm-sdk@external.cisco.com.

Cisco StadiumVision Mobile SDK Best Practices

Apple iOS

Consider the following best practices when developing and delivering an app for Apple iOS:

Correlating Reporter Data to a Specific Device

- Apple does not permit applications to access any device information that can be used to identify that device or its owner. As a result, the iOS SDK is unable to include the MAC address in the periodic stats that it sends to the Cisco StadiumVision Mobile Reporter. As a substitute for the MAC address, the SDK instead includes a SVM Device UUID (universally unique identifier) that is unique for every device. This UUID allows Reporter data to be correlated with a specific device. In order for the correlation to work, the mobile app must display the UUID somewhere in its menu system (for example on the About or Help tabs).

  The app can retrieve the UUID from the SDK via the code sample below. The getDeviceUUID method is documented in the iOS SVM header file.

```
StadiumVisionMobile *svm = [StadiumVisionMobile sharedInstance];
NSString *deviceUUID = [svm getDeviceUUID];
NSLog(@"Device UUID is %@", deviceUUID);
```

Note The Cisco StadiumVision Mobile Device UUID should not be confused with the Unique Device Identifier (UDID) that is displayed in iTunes.

Google Android

Consider the following best practices when developing and delivering an app for Google Android:

Delivering Channel Content

- Internet Group Management Protocol (IGMP) is a prerequisite for Cisco StadiumVision Mobile multicast to function correctly. Most, but not all, Android devices support IGMP. A user will see an empty channel list in the SVM app if they are using a device that does not support IGMP unless another active SVM client is associated to the same Access Point (AP). As a result, the user would experience sporadic channel support without knowing why. We recommend that all SVM-enabled Android apps perform the IGMP capability check as detailed in the example below. If the IGMP capability check returns false, then the app should warn the user when the user attempts to access any part of the app that relies on the SVM SDK.

```
boolean isCapable = false;
File f = new File("/proc/net/igmp");
     if(f.exists()) {
         isCapable = true;
     }
```

Apple iOS, Google Android, and Windows Phone

Consider the following best practices when developing and delivering an app for Apple iOS, Google Android, and Windows Phone:

Delivering Channel Content

- Start the SDK immediately when the app is launched or when the app detects that it is in venue. Do not wait until the user navigates to the SVM features in the UI. This ensures the best possible user experience so that content is then available to present to the user immediately. This is particularly important when using data and file channels because it can take several minutes for a content rotation to complete.
- When a user selects a channel on their mobile device, there could be a 5-10 second delay before the app starts receiving multicast video if the client has to wait for the IGMP/PIM to set up the multicast tree. The same delay could occur when a device resumes from background or sleep mode. Instead of displaying a black screen, communicate to the user that their request is being processed by showing a transition graphic (for example a spinning wheel image) or text that asks them to please wait while the channel is located.
- A channel will disappear from the lineup if it is stopped on the Streamer or if the Streamer detects a loss of input signal. The app should remove the channel immediately, however if a user is already watching the channel when it disappears, terminate the video rendering and return the user to the channel guide where they can select a new channel.
- If the Cisco StadiumVision Mobile administrator starts the Streamer before the video control room has switched content to the encoder inputs, the user will receive an empty channel listing. To avoid a poor user experience, display a message indicating that there aren't any live channels currently available.

Using the Latest Version of the app

- Prevent or warn a user if they are attempting to access SVM services with an older or incompatible version of the app. Set the app to perform a self-check to see if a newer app version is available. If a new version is detected, the app should:
  - Block or warn the user that their app is out-of-date or may not perform as expected
  - Encourage the user to upgrade

Connecting to Wi-Fi

- If the client looses complete Wi-Fi connectivity, the Operating System sends the app a notification. The app should notify the user that the Wi-Fi service is not available and remind them of the Wi-Fi network (SSID) to connect to.
- If a Streamer service announcement has not been received for 30 seconds, the SDK will notify the user that the Cisco StadiumVision Mobile service is down. This could happen because the user exited the venue and is out of range, if their device roamed to a non SVM SSID, or if they entered an area in the venue without Wi-Fi. When this occurs, notify the user that the SVM service is not available and remind them of the name of the Wi-Fi network (SSID) that they must be connected to in order to receive the SVM service.
- The SDK continuously monitors signal quality as a user moves throughout a venue. As a result, the SDK sends the app a 'service down' or 'service up' notification. These notifications can be used by the app for conditional playback based on network conditions. It is recommended that conditional playback is implemented as a passive informational service (as it should never display prompts that require an active response from the user). Refer to Table 1-2 for app guidelines as to how the app should respond depending on the state of the app when the notification is received.

Table 1-2 App Guidelines for Responding to Notifications

app State	Event or Notification	Recommended app Response
The app is rendering a video.	Service down notification, due to poor quality.	Overlay the active video plane with a transparent graphic that contains the text such as, "Video quality degraded due to poor reception". The app continues the video session without interruption regardless of the poor quality and renders whatever video it receives underneath the text overlay.
The app previously received a service down notification with the poor quality reason code. It is currently rendering degraded video with the "Video quality degraded due to poor reception" overlay.	Service down notification, due to poor quality.	Remove the overlay and resume normal video rendering. Do not restart the video session.
The app is currently not rendering video.	Service down notification, due to poor quality.	Delete all channels from the channel list page and replace them with a message such as "Live video is currently unavailable due to poor reception".
The app is currently not rendering video. The app had previously received a service down notification with the poor quality reason code and is currently displaying the message "Live video unavailable due to poor reception" in the channel guide.	Service up notification, due to quality rebounding.	Remove the "Live video unavailable due to poor reception" message and populate the channel page with the current list of channels available.
The app previously received a service down notification with the poor quality reason code and is currently rendering video with the "Video quality degraded due to poor reception" overlay.	The user now stops the video and returns to the channel page.	Upon returning to the channel page, the user sees the message "Live video unavailable due to poor reception" instead of a list of channels.

CHAPTER 2

Cisco StadiumVision Mobile API for Apple iOS

First Published: May 26, 2015
Revised: June 12, 2015

This chapter describes the Cisco StadiumVision Mobile SDK Release 2.1 for Apple iOS, and contains the following sections:

Introduction to Cisco StadiumVision Mobile SDK for iOS

The Cisco StadiumVision Mobile iOS SDK contains the following components bundled together:

- A set of static libraries, header files
- Sample app (with a complete Xcode project) and SDK video player
- API documentation (Doxygen build)

The 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.

Table 2-1 describes the mobile operating system versions supported by the Cisco StadiumVision Mobile SDK.

Table 2-1 Mobile OS Support

	Apple iOS			
OS	**5.x**	**6.x**	**7.x**	**8.x**
Cisco StadiumVision Mobile SDK Release 2.1	No	No	Yes	Yes
Cisco StadiumVision Mobile SDK Release 2.0	No	Yes	Yes	Yes

For additional information, refer to the *Cisco StadiumVision Mobile Release Notes* available from Cisco.com at:

http://www.cisco.com/c/en/us/support/video/stadiumvision/products-release-notes-list.html

Cisco StadiumVision Mobile and iOS Developer Tools

Table 2-2 lists the various iOS SDK build environment requirements.

Table 2-2 Apple iOS Build Environment Requirements

Tool	Version	Description	URL
Mac OSX	10.10	A Mac is required to build an iOS application which includes the StadiumVision Mobile iOS SDK.	http://www.apple.com
Xcode	6.1	Apple development IDE and tool kit.	http://developer.apple.com/xcode

Requirements

- Download and install the Apple Xcode IDE.
- In order to build and run the project, you must join or be an existing member of the Apple iOS Developer Program. Additional information is available at:

  https://developer.apple.com/programs/ios/
- Latest Cisco StadiumVision Mobile SDK tar.bz2 file, contact your Cisco account team for details as to how to become part of the Cisco StadiumVision Mobile SDK partner program.

Note Beginning February 1, 2015, new iOS apps submitted to the App Store must include 64-bit support and be built with the iOS 8 SDK. Apps that are updated will also need to follow the same requirements beginning on June 1, 2015. It is recommended you use Xcode 6.x to support iOS 8 for new and existing apps.

Download and Unpack the SDK

Step 1 Download **StadiumVisionMobileSample-ios-VERSION-RELEASE.tar.bz2**. If you do not have this file, contact your Cisco account team for details as to how to become part of the Cisco StadiumVision Mobile SDK partner program.

Step 2 Extract the downloaded package into a directory. Table 2-3 lists the extracted content and includes a brief description.

Table 2-3 Cisco StadiumVision Mobile SDK File Content

Contents	Description
clean.stream	Sample stream for the stream sender
Default-568h@2x.png	Default theme graphic
html/	Contains Doxygen API documentation that is accessible by opening the index.html file in a web browser
Makefile	Text file referenced by the make command
Readme.txt	File that contains information to get started
StadiumVisionMobile/	SVM header files and static library
StadiumVisionMobileSample/	Source code to the sample application
StadiumVisionMobileSample.xcodeproj	Xcode project used to build the sample application
StadiumVisionMobileSender/	Stream sender add-on to the API
UnitTests/	Folder for unit tests

Note The Cisco StadiumVision Mobile SDK for iOS Release 2.1 does not include "libvoCTS.a" and "voVidDec.dat." These files are no longer required in Release 2.1.

Note The clean.stream file that comes bundled with the SDK contains just one video channel. To provide app developers with additional ways to test multiple channels, an additional set of clean.stream files is available. For additional information refer to "Testing Your Cisco StadiumVision Mobile App" section on page 1-8.

Step 3 Open the API documentation available in the Doxygen build that is downloaded with the SDK. Navigate to the extracted folder contents, open the **html** folder > double-click **index.html** to launch the documentation in a web browser.

Getting Started with the iOS Sample App

The Cisco StadiumVision Mobile SDK provided to app developers includes the source code for a iOS Sample app. The purpose of the Sample app is to demonstrate what is possible and to enable a new app developer to quickly get a working app up and running.

Note Before creating a new app, review the Cisco StadiumVision Mobile SDK Best Practices, page 11-9.

Compile the Sample App

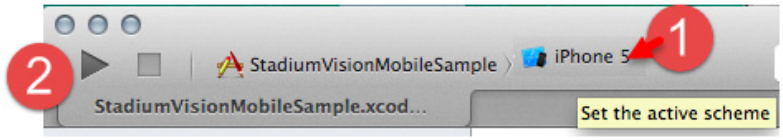
Step 1 Launch **Xcode**.

Step 2 Under **File > Open >** locate and select **StadiumVisionMobileSample.xcodeproj** from the extracted folder contents. Click **Open**.

Step 3 Select the active scheme (iPhone 5 for example) from the iOS Simluator list as shown in Figure 2-1 (1). To run the Sample app from an external device, connect the device to your computer and then select the device from the iOS Simulator list.

Step 4 Click the **Build and then run the current scheme arrow** to build and run the Sample app with the selected scheme as shown in Figure 2-1 (2).

Figure 2-1 Xcode—Set and Run the Active Scheme



Note If the external device you want to test on does not appear in the iOS Simulator list, be sure you've added it to the list of iOS devices in the iOS Developer Program. Cisco StadiumVision Mobile SDK Release 2.1 supports iOS 64-bit, however the SVM SDK for iOS only includes support for the 32-bit simulator and does not provide 64-bit simulator support.

Step 5 If the build was successful, a message appears followed by the Sample app launching in a new iOS Simulator window or on the external device as shown in Figure 2-2.

Figure 2-2 Xcode—Building the Sample App

Step 6 Test the Sample app.

Customize the Sample App

There are many ways to customize the Cisco StadiumVision Mobile Sample app including customizing the Default-568h@2x.png graphic file to include a logo and specific colors.

Cisco Sample app Customized Video Player

The Sample app 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 "StadiumVisionMobileSample" Xcode project folder.

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

Embed the Cisco StadiumVision Mobile SDK in an Existing App

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" static library into the Xcode project.

Note The Cisco StadiumVision Mobile SDK for iOS Release 2.1 does not include "libvoCTS.a" that was previously included. This file is no longer required in Release 2.1.

4. Include at least one objective C++ file in your project. We recommend renaming "main.m" to "main.mm".
5. Set the Xcode Project "Build Settings"

   Add the required linker flag in Xcode using **Build Settings > Linking > Other Linker Flags > Add**. The required Xcode "Other Linker Flags" settings are shown in Figure 2-3.
   - 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.

Figure 2-3 Xcode Other Linker Flags

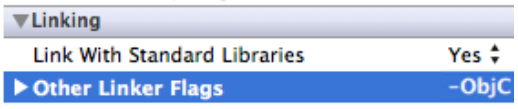
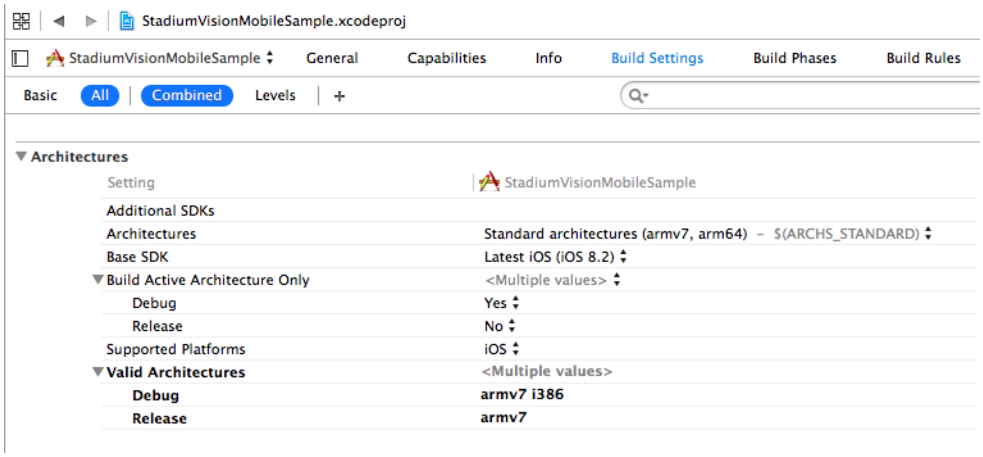


Figure 2-4 shows the Xcode build settings that apply to both the project and target settings.

Figure 2-4 Xcode Build Settings



Note The standard architectures list may or may not include armv7s depending on the Xcode version you are using.

Figure 2-5 shows the settings for generating position dependent and position independent code.

Figure 2-5 Xcode Build Settings—Position Dependent and Independent Code Generation

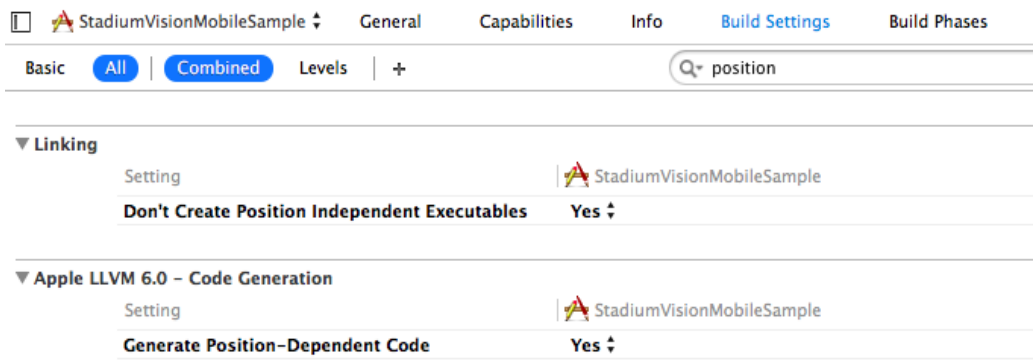
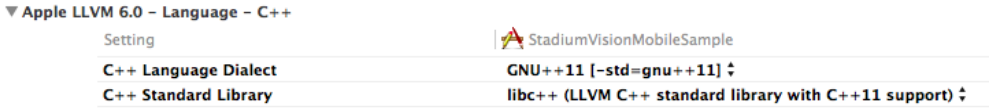


Figure 2-6 shows the Apple LLVM language settings.

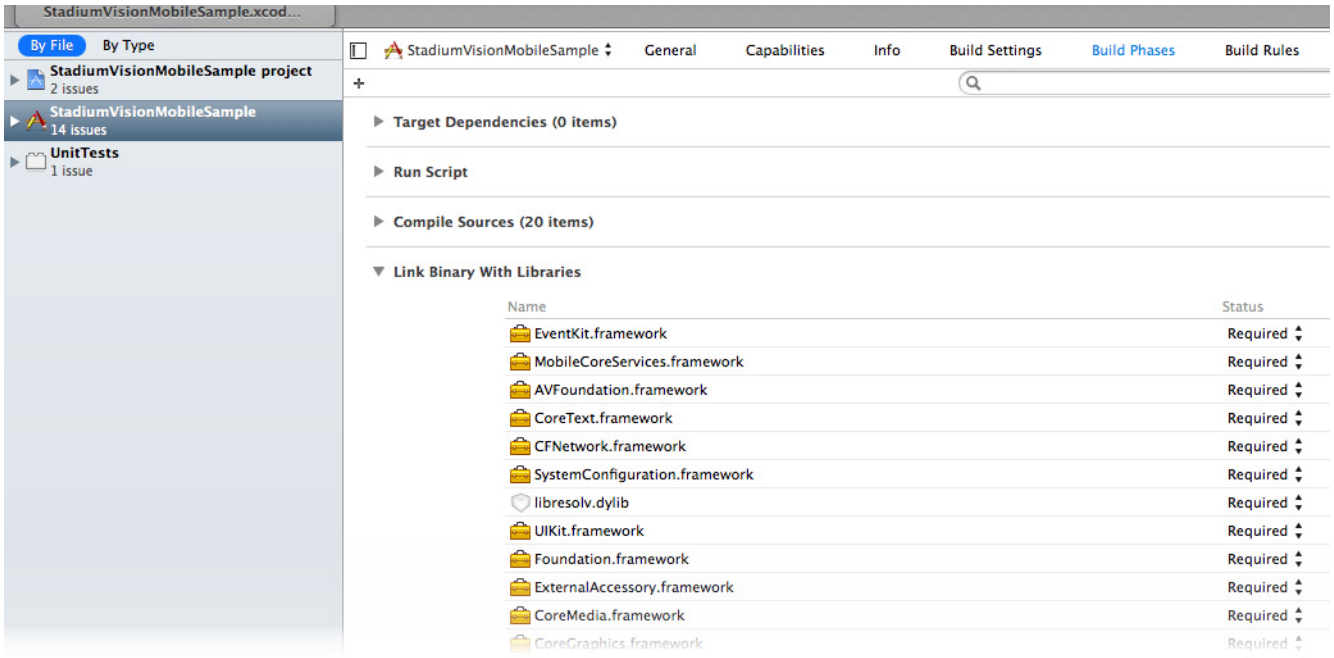
Figure 2-6 Xcode Build Setting—Specify Apple LLVM 6.0 - Language C++



Note If using Xcode version 5 or earlier, set "Apple LLVM 5.1 - Language - C++" > "C++ Standard Library" to "libstdc++ (GNU C++ standard library)". Applications that target iOS 6 and earlier do not need to make this change.

6. 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 2-7. A full list of required libraries is listed below Figure 2-7.

Figure 2-7 Adding Frameworks in Xcode



Required iOS Libraries

- EventKit.framework
- MobileCoreServices.framework
- AVFoundation.framework
- CoreText.framework
- CFNetwork.framework
- SystemConfiguration.framework

- libresolv.dylib
- UIKit.framework
- Foundation.framework
- ExternalAccessory.framework
- CoreMedia.framework
- CoreGraphics.framework
- AudioToolbox.framework
- OpenGLES.framework
- QuartzCore.framework
- Security.framework
- MediaPlayer.framework
- libz.dylib
- libStadiumVisionMobile.a
- libStadiumVisionMobileSender.a

Configuration Files

There are two configuration files that must be bundled with any iOS app using the StadiumVision Mobile SDK, as listed in Table 2-4.

Table 2-4 Configuration Files

Configuration File Name	Description
"cisco_svm.cfg"	The Cisco StadiumVision Mobile SDK configuration file that contains the "Field-of-Use" parameters and some optional Wi-Fi network debugging information. The three "field-of-use" properties in the "cisco_svm.cfg" configuration file that need to be configured for each StadiumVision Mobile application are: • Venue Name • Content Owner • App Developer
"vompPlay.cfg"	Video decoder configuration file that contains the tuned decoding parameters. These settings should never be changed. Any changes could result in poor video playback.

Note The Cisco StadiumVision Mobile SDK for iOS does not include "voVidDec.dat" that was previously included. This file is no longer required in Release 2.1.

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:

```
{
    "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 Wi-Fi AP information that will be used by the StadiumVision Mobile SDK for statistics reporting if available. Below is an example Wi-Fi AP info entry in the "cisco_svm.cfg" configuration file:

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

How Cisco StadiumVision Mobile Fits into the iOS Framework

Client Application Integration Overview

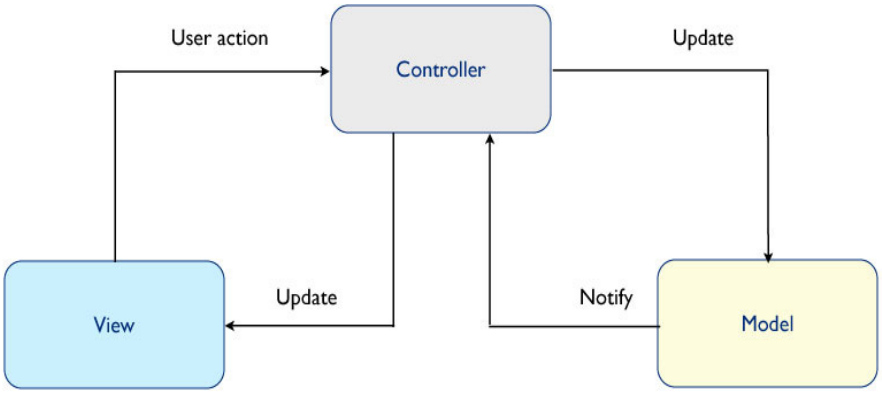
Figure 2-8 illustrates the high-level view of the Cisco StadiumVision iOS API libraries and common framework components. The left side of the graphic represents how to modify the sample application, and the right represents how the SDK is packaged.

Figure 2-8 Cisco StadiumVision Mobile iOS SDK Components

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 2-9 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.

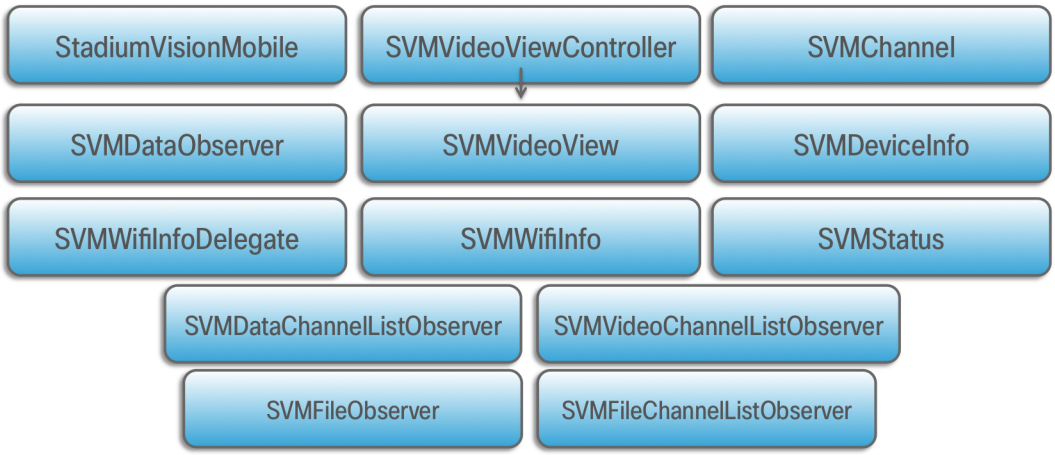
Figure 2-9 MVC Design Pattern



Cisco StadiumVision Mobile iOS API Class Overview

The singleton "StadiumVisionMobile" class provides the top-level API to start, configure, and stop the framework. "SVMVideoViewController" classes are provided to play the video channels and to allow for customization. Figure 2-10 illustrates the Cisco StadiumVision Mobile API classes.

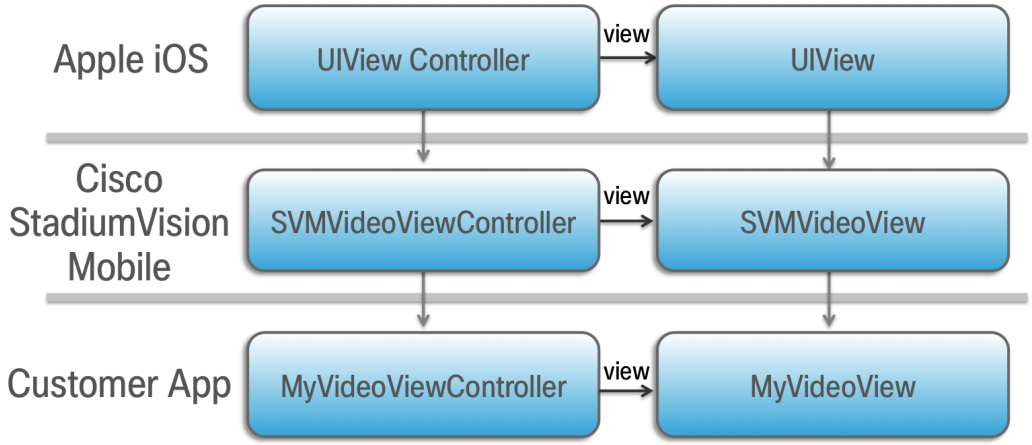
Figure 2-10 Cisco StadiumVision Mobile iOS API Classes



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 2-11 illustrates the UIViewController and UIView classes.

Figure 2-11 Cisco StadiumVision Mobile Video Classes

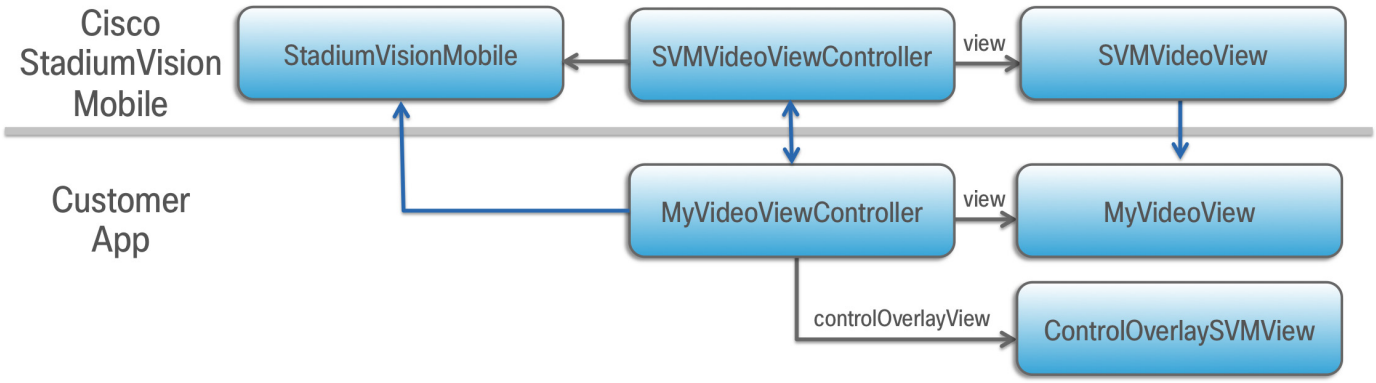


Cisco StadiumVision Mobile Application Classes

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 Cisco StadiumVision Mobile stats.
- Handles gestures to display UI overlays with the MyVideoViewController class.

Figure 2-12 Cisco StadiumVision Mobile Sample Application Classes

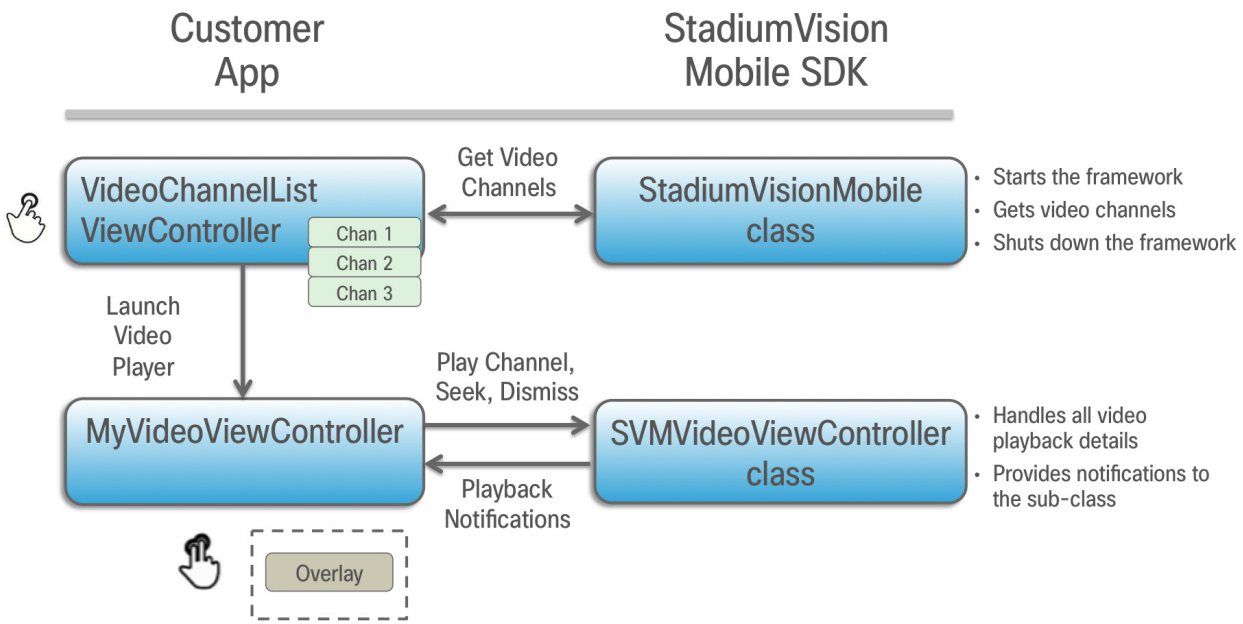


Customer Application Roles

Figure 2-13 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 2-13 Customer Application Responsibilities



Cisco StadiumVision Mobile Methods and Functions for iOS

Cisco StadiumVision Mobile iOS API Summary

Table 2-5 summarizes the iOS API library. Detailed API information is available in documentation Doxygen build that is downloaded with the SDK. Navigate to the extracted folder contents, open the **html** folder > double-click **index.html** to launch the documentation in a web browser.

Table 2-5 Cisco StadiumVision Mobile iOS API Summary

Return Type	API Method Name	API Method Description
BOOL	isConnectedToVenue	Gets whether the device is currently inside or outside of the venue.
NSArray*	getDataChannelListArray	Gets a snapshot array of the currently available data channels.
NSArray*	getFileChannelListArray	Gets a snapshot array of the currently available file channels.
NSArray*	getVideoChannelListArray	Gets a snapshot array of the currently available video channels.
NSArray*	getStreamerArray	Gets an array of detected SVM Streamer servers as 'SVMStreamer' objects
NSDictionary*	getConfig	Gets the SDK configuration at run-time.
NSDictionary*	stats	Gets an NSDictionary of current SVM SDK stats as a dictionary of name/value pairs. **Note** Stats are currently only available for the video channel (not data channels).
NSInteger*	getFileStatusfor Filename:forChannel:	Gets the filesystem filename status for any channel.
NSInteger*	getFileStatusforFilename:forChannel Name:	Gets the filesystem filename status for any channel name.
NSMutableDictionary*	getFileDistributionTable:	Gets the file distribution table details.
NSString*	getFileDistributionLocalFilename:for Channel:	Gets the local filesystem filename for any object given its URI and the file channel.
NSString*	getFileDistributionLocalFilename:for ChannelName:	Gets local filesystem filename for any object given its URI and the file channel name.
NSString*	getDeviceUUID	Gets the device UUID generated by the SVM SDK and is documented in the iOS SVM header file.
NSString*	getAppSessionUUID	Gets the app session UUID that is generated by the SVM SDK. This UUID uniquely identifies each time the SDK is started and is used for consistent statistics collection and reporting.
NSString*	getVideoSessionUUID	Gets the video session UUID.
NSUInteger	getServiceDownReasonsBitmap	Gets the bitmap of reasons why the service state was down.
StadiumVisionMobile*	sharedInstance	Gets a reference to the API singleton class used for all API calls.

Table 2-5 Cisco StadiumVision Mobile iOS API Summary (continued)

Return Type	API Method Name	API Method Description
SVMServiceState	getServiceState	Gets the current SVM service state.
SVMStatus*	addDataChannelListDelegate:	Registers a callback delegate to receive all data channel list updates.
SVMStatus*	addDataChannelObserver:	Registers an observer class to receive data for a particular data channel.
SVMStatus*	addDataChannelObserver:forChannel:	Registers an observer class to receive all data updates for a particular data channel.
SVMStatus*	addDataChannelObserver:forChannel Name:	Registers an observer class to receive all data updates for a particular data channel name.
SVMStatus*	addFileChannelListDelegate:	Registers to callback delegate to receive all file channel list updates.
SVMStatus*	addFileChannelObserver:forChannel:	Registers an observer class to receive all file updates for a particular file channel.
SVMStatus*	addFileChannelObserver:forChannelName:	Registers an observer class to receive all file updates for a particular file channel name.
SVMStatus*	addVideoChannelListDelegate:	Registers a callback delegate to receive all video channel list updates.
SVMStatus*	allowAllStreamers	Allows all Streamers to be processed by the SDK.
SVMStatus*	allowPlaybackWhenViewDisappears	Allows the video player to continue rendering the channels when the video player view has lost focus.
SVMStatus*	allowStreamers:	Allows only specific Streamers in a given array to be processed by the SDK.
SVMStatus*	disableQualityMonitoring	Disables quality monitoring within the SDK.
SVMStatus*	enableQualityMonitoring	Enables quality monitoring within the SDK.
SVMStatus*	initSDK	Initializes the SDK.
SVMStatus*	loadConfigFile	Loads the security configuration data.
SVMStatus*	removeDataChannelListDelegate:	Unregisters the callback delegate from receiving the data channel list updates.
SVMStatus*	removeDataChannelObserver:	Unregisters an observer class from receiving data for a particular data channel.
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 name.
SVMStatus*	removeFileChannelListDelegate:	Unregisters the callback delegate from receiving the file channel list updates.
SVMStatus*	removeFileChannelObserver:forChannel:	Unregisters an observer class from receiving any file updates for a particular file channel.
SVMStatus*	removeFileChannelObserver:forChannel:Name:	Unregisters an observer class from receiving any file updates for a particular file channel name.

Table 2-5 Cisco StadiumVision Mobile iOS API Summary (continued)

Return Type	API Method Name	API Method Description
SVMStatus*	removeVideoChannelListDelegate:	Unregisters the callback delegate from receiving the video channel list updates.
SVMStatus*	setConfig:	Sets the SDK configuration at run time.
SVMStatus*	setConfigWithString:	Sets the SDK configuration at run time with the config JSON string.
SVMStatus*	setLogLevel:	Sets the logging output level of the SDK, with the "DEBUG" level being more verbose than the "INFO" level.
SVMStatus*	setStatsHookDelegate:	Sets the callback stats hook delegate.
SVMStatus*	shutdown	Stops the SVM SDK.
SVMStatus*	start	Starts the SVM SDK and any SVM background threads and component managers.
SVMStatus*	version	Gets the SVM version string.
SVMWifiInfo*	wifiInfo	Returns the current Wi-Fi network connection information. This information gets collected in the statistics information that gets uploaded to the Reporter server.
void	onData	Implemented by the customer app to support the "SVMDataObserver" protocol. This delegate method is used as a callback from the SVM SDK. Each callback from the SDK to the customer app provides a received data message on the given data channel, delivered as a byte array (NSData).

Return Status Object

Each API call returns a SVMStatus object whenever applicable. Table 2-6 lists the SVMStatus object fields.

Table 2-6 SVMStatus class

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

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

Table 2-7 lists the hash keys and description for the Stats API.

Table 2-7 Stats API Hash Keys and Descriptions

Stats Hash Key	Stats Description
announcement_session_id	Video session announcement ID.
announcement_session_title	Session announcement name.
compressedChannelAnnouncementsReceived	The number of compressed channel announcement messages received.
num_channel_announcement_igmp_restarts	The number if IGMP restarts performed on the channel announcement listener.
num_channel_announcement_version_mismatches	The number of channel announcements received from an incompatible Streamer version.
num_channel_announcements_received	Total number of multicast channel announcement messages received.
num_dropped_video_frames	Total number of video frames dropped.
num_license_key_mismatches	The number of channel announcements received where the license keys did not match.
num_ts_discontinuities	Total number of MPEG2-TS packet discontinuities.
protection_windows	Total number of protection windows sent.
session_link_indicator	Health of the Wi-Fi network connection. Ranges from 0 (poor) to 10 (excellent).
session_uptime	Length of time the session has been active (in seconds).
total_num_bytes_written	Total number of video bytes played.
window_error	Total number of protection windows with more packets per window than can be supported by Cisco StadiumVision Mobile.
window_no_loss	Total number of protection windows with no dropped video packets.
window_recovery_failures	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	Total number of protection windows with recovered video packets.
window_warning	Total number of protection windows with more packets per window than the recommended value.

Video Player Activity API Summary

The "SVMVideoVideoController" class can be extended and customized. Table 2-8 lists the SVMVideoPlayerActivity API methods and descriptions. Additional API methods and details are listed in the Doxygen build.

Table 2-8 Video View Controller API Summary

Return Type	API Method Name	API Method Description
SVMStatus*	playLive	Moves the video playback buffer pointer to the head ("live") offset position in the video playback buffer. This convenience method acts as a wrapper for the "seekAbsolute" API method; making "playLive()" equivalent to "seekAbsolute(0)".
SVMStatus*	playVideoChannel:	Starts playback of a particular video channel, changing channels on subsequent calls.
SVMStatus*	rewindForDuration:	Rewinds the video playback buffer pointer relative to the current playback buffer offset position.
SVMStatus*	seekAbsolute:	Moves the video playback buffer pointer relative to the starting "live" video playback buffer offset position. The SVM SDK currently buffers 30 seconds of previously played video data that can be used for playing previously recorded video data. • A positive duration value moves the video play-head away from the latest "live" video data in the video history buffer. • Should a duration be given that is larger than the available size of the video history buffer, then the SVM SDK moves the video play-head to the end of the video history buffer.
SVMStatus*	seekRelative:	Moves the video playback buffer pointer relative to the current video playback buffer offset position. The SVM SDK currently buffers 30 seconds of previously played video data that can be used for playing previously recorded video data. • A positive duration value forwards the video play-head towards the latest "live" video data in the video history buffer. • Should a duration be given (positive or negative) that is larger than the available size of the video history buffer, then the SVM SDK moves the video play-head as far as possible within the video history buffer.
void	setRenderVideoView:	Sets the iOS UI video view where video frames will get rendered.

NS Notification Events

The StadiumVision Mobile SDK broadcasts the following iOS NSNotification events for use by the client application (listed in Table 2-9).

Table 2-9 NSNotification Event Properties

Event Constant	Description
kSVMVideoEventNotification	Constant defining the video event generated by the StadiumVision Mobile API.
kSVMVideoOpenState	Occurs when the video player initially opens the video channel session.

Table 2-9 NSNotification Event Properties (continued)

Event Constant	Description
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.
kSVMVideoPauseState	Occurs when the video player pauses video playback.
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 isOpen;
- BOOL isPlaying;
- BOOL isAppActive;
- BOOL isVisible;
- BOOL isBackgroundPlaybackAllowed;
- BOOL isEventsRegistered;
- BOOL isEventHandlersRegistered

Table 2-10 provides a description of each state flag provided by the StadiumVision Mobile video player ("SVMVideoViewController"):

Table 2-10 Video Player State Flags

State Flag	Description
isAppActive	Boolean flag indicating when the container iOS app is in the foreground.
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").
isEventHandlersRegistered	Boolean flag indicating whether the notification event handlers have been registered.
isEventsRegistered	Boolean flag indicating when an event is registered.
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.
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").

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

Beginning with Release 2.0, 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 channels are available.
- The mobile device’s Wi-Fi interface is disabled.

Poor video quality can occur when the user is receiving a weak Wi-Fi 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 2-11:

Table 2-11 Service Down Reason Notification

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

Note For additional Service Down Notification details, refer to “Cisco StadiumVision Mobile SDK Best Practices” section on page 1-9.

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 2.0, the application can 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] init] autorelease];
NSMutableDictionary *licenseDict = [[[NSMutableDictionary alloc] init] 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];
```

Scalable File Distribution

The Cisco StadiumVision Mobile SDK libraries will support file channels that are easily accessible to the mobile client application. Table 2-12 lists the Cisco StadiumVision Mobile scalable file distribution API.

Table 2-12 Scalable File Distribution and Service API Summary

API Return Type	File Service API Method Name	Method Description
NSArray*	getFileChannelListArray	Gets a snapshot array of the currently available file channels.
NSMutableDictionary*	getFileDistributionTable	Gets file distribution table details.
NSString*	getFileDistributionLocalFilename	Gets the local filesystem filename for any object given its URI and the file channel.
NSString*	getFileDistributionLocalFilename:forChannel	Gets the local filesystem filename for any object given its URI and the file channel.
NSString*	getFileDistributionLocalFilename:forChannelName	Gets local filesystem filename for any object given its URI and the file channel name.
SVMStatus*	addFileChannelListDelegate	Registers a callback delegate to receive all file channel list updates.
SVMStatus*	addFileChannelObserver	Registers an observer class to receive data for a particular file channel.
SVMStatus*	addFileChannelObserver:forChannel	Registers an observer class to receive all file updates for a particular file channel.
SVMStatus*	addFileChannelObserver:forChannelName	Registers an observer class to receive all file updates for a particular file channel name.
SVMStatus*	removeFileChannelListDelegate	Unregisters the callback delegate from receiving the file channel list updates.
SVMStatus*	removeFileChannelObserver	Unregisters an observer class from receiving file for a particular file channel.
SVMStatus*	removeFileChannelObserver:forChannel	Unregisters an observer class from receiving any file updates for a particular file channel.
SVMStatus*	removeFileChannelObserver:forChannelName	Unregisters an observer class from receiving any file updates for a particular file channel name.

Data Channels

Table 2-13 lists the Cisco StadiumVision Mobile data channel APIs.

Table 2-13 Data Distribution and Service API Summary

API Return Type	Data Service API Method Name	Method Description
NSArray*	getDataChannelListArray	Gets a snapshot array of the currently available data channels.
SVMStatus*	addDataChannelListDelegate:	Registers a callback delegate to receive all data channel list updates.

Table 2-13 Data Distribution and Service API Summary (continued)

API Return Type	Data Service API Method Name	Method Description
SVMStatus*	addDataChannelObserver:	Registers an observer class to receive data for a particular data channel.
SVMStatus*	addDataChannelObserver:forChannel:	Registers an observer class to receive all data updates for a particular data channel.
SVMStatus*	addDataChannelObserver:forChannelName:	Registers an observer class to receive all data updates for a particular data channel name.
SVMStatus*	removeDataChannelListDelegate:	Unregisters the callback delegate from receiving the data channel list updates.
SVMStatus*	removeDataChannelObserver:	Unregisters an observer class from receiving data for a particular data channel.
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 name.
void	onData	Supports the "SVMDataObserver" protocol when implemented by the customer app. This delegate method is used as a callback from the SVM SDK. Each callback from the SDK to the customer app provides a received data message on the given data channel, delivered as a byte array (NSData).
void	onDataChannelListUpdated	Results in the method being called by our API to notify you of data channel changes.
void	onData:withChannelName:	Results in the method being called by our API to notify you of changes for the given channel.

Adding Cisco StadiumVision Mobile Services to an iOS App—Code Structure and Samples

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 Wi-Fi 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

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

Starting the SDK

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

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

Setting the Log Level

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

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

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

Getting the 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];
```

Displaying the Device UUID

The Cisco StadiumVision Mobile SDK is unable to include the MAC address in the periodic stats that it sends to the Cisco StadiumVision Mobile Reporter because Apple does not permit applications to access any device information that can be used to identify that device or its owner. As a substitute for the MAC address, the SDK instead includes a SVM Device UUID (universally unique identifier) that is unique for every device. The UUID allows Reporter data to be correlated with a specific device. In order for the correlation to work, the mobile app must display the UUID somewhere in its menu system (for example on the About or Help tabs).

The app can retrieve the UUID from the SDK via the code sample below. The getDeviceUUID method is documented in the iOS SVM header file.

```
StadiumVisionMobile *svm = [StadiumVisionMobile sharedInstance];
NSString *deviceUUID = [svm getDeviceUUID];
NSLog(@"Device UUID is %@", deviceUUID);
```

Note The Cisco StadiumVision Mobile Device UUID should not be confused with the Unique Device Identifier (UDID) that is displayed in iTunes.

Shutting Down the SDK (Optional)

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

```
#import "StadiumVisionMobile"

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

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

Video Player View Controller Customization

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

- Default Cisco Video Player View Controller, page 2-28
- Customized Video Player, page 2-29
- Cisco Sample app Customized Video Player, page 2-5

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 Sample app uses a customized video player.

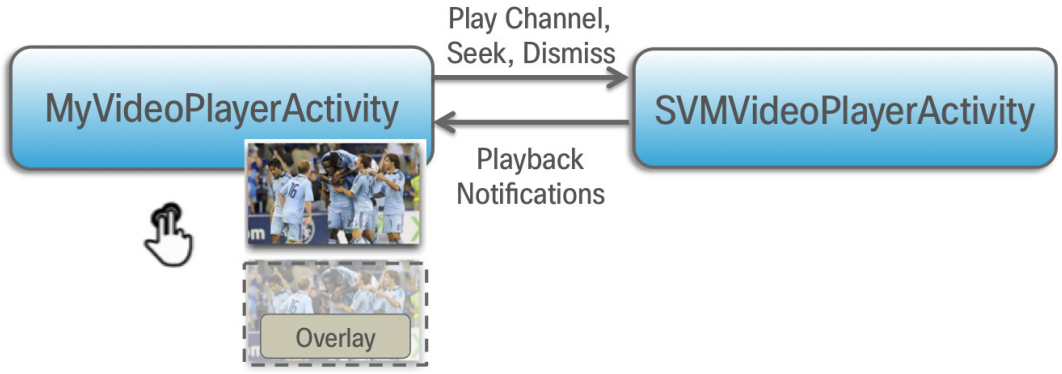
Customized Video Player

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

```
#import "SVMVideoViewController.h";

@interface MyVideoViewController : SVMVideoViewController {
}
```

Figure 2-14 Video Player Customization



Video Channels

This section describes the Cisco StadiumVision Mobile SDK video channels and contains the following sections:

- Presenting the Video Channel List, page 2-29
- Playing a Video Channel, page 2-30
- Getting the Video Channel List, page 2-30
- Seeking Within the Video Buffer, page 2-30
- Video Player View Controller Customization, page 2-28

Presenting the Video Channel List

Table 2-14 lists the "SVMChannel" video channel objects containing all of the information needed to display the channel list to the user.

Table 2-14 SVMChannel Object Properties

SVMChannel Property	Property Description
appDeveloper	Name of the application developer.
bandwidthKbps	Nominal video stream bandwidth (in kbps).
channelText	Complete text description of the video channel.
contentOwner	Name of the content owner.
name	Name of the video channel.

Table 2-14 SVMChannel Object Properties (continued)

SVMChannel Property	Property Description
sessionNum	Session number of the channel.
venueName	Name of the venue.

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];
```

Getting the Video Channel List

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

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

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

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

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 = [StadiumVisionMobile 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 = [StadiumVisionMobile sharedInstance];
// play at the "live" video offset
[svm playLive];
```

Data Channels

This section describes the Cisco StadiumVision Mobile SDK data channels and contains the following sections:

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];}
```

EVS C-Cast Integration

Note Cisco StadiumVision Mobile is supported with EVS C-Cast version 2.x only. EVS C-Cast version 3.x is not supported.

The steps below describe a high level workflow of how an Cisco StadiumVision Mobile powered C-Cast app gains access to the XML timeline and media files.

1. Register a callback to be notified when a file channel becomes available, using **addFileChannelDelegate**
2. Register to receive the channel notification using **[svm addFileChannelObserver:self forChannelName:@"something"]**
3. (Optional) Listen for file channel list updates and potentially register using **(void)onFileChannelListUpdated:(NSMutableDictionary *)fileChannelList {}**
4. Handle the file reception (movies/thumbnails/timeline) using **(void)onFile:(NSData *)file withChannelName:(NSString *)channelName {}**
5. Check if a file channel is already available, using **getFileChannelListArray**
6. If a channel is already available or when a callback notification is received, register a file channel observer, using **addFileChannelObserver**
7. Check if a file with the name ccast-timeline.xml is already available, using **getFileDistributionLocalFilename**
8. If ccast-timeline.xml is not yet available wait for additional files to arrive, using onFile(). Each time onFile() is called do a corresponding check with **getFileDistributionLocalFilename** to see if the new file is ccast-timeline.xml.
9. Once ccast-timeline.xml has been received, parse it using the steps in chapter 5 (How to build the media path) of the C-Cast API spec, then build the media path for all media files. Contact James Stellphlug (j.stellpflug@evs.com) to obtain the C-Cast API documentation.
10. For each file media path, remove the path prefix so that only the filename remains. For example: http://www.mydomain.com/videos/abc/def/ghi/abcdefghijklmnopqrstuvwxyz123456_hls-ipad.m3u8 becomes abcdefghijklmnopqrstuvwxyz123456_hls-ipad.m3u8
11. For each filename cycle through **onFile()** and **getFileDistributionLocalFilename** until all files have been received.
12. Be prepared for ccast-timeline.xml to change at any time. Repeat steps 7-9 whenever it changes.

CHAPTER 3

Cisco StadiumVision Mobile API for Google Android

First Published: May 26, 2015

This chapter describes the Cisco StadiumVision Mobile SDK Release 2.1 for Google Android, and contains the following sections:

Introduction to Cisco StadiumVision Mobile SDK for Android

The Cisco StadiumVision Mobile Android SDK contains the following components bundled together:

- A set of static libraries, header files
- Demo app and SDK video player
- API documentation (Doxygen build)

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.

Table 3-1 describes the mobile operating system versions supported by the Cisco StadiumVision Mobile SDK.

Table 3-1 Mobile OS Support

	Google Android							
OS	**2.3**	**4.0**	**4.1**	**4.2**	**4.3**	**4.4**	**5.0**	**5.1**
Cisco StadiumVision Mobile SDK Release 2.1	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Cisco StadiumVision Mobile SDK Release 2.0	No	Yes	Yes	Yes	Yes	Yes	No	No

For additional information, refer to the *Cisco StadiumVision Mobile Release Notes* available from Cisco.com at:

http://www.cisco.com/c/en/us/support/video/stadiumvision/products-release-notes-list.html

Cisco StadiumVision Mobile and Android Developer Tools

Table 3-2 lists the various Android SDK build environment requirements.

Table 3-2 Build Environment Requirements

Tool	Version	Description	URL
Mac or Windows PC	—	—	—
Eclipse	3.7.2 or greater	Eclipse "Classic" for Mac OSX (64-bit)	https://eclipse.org/downloads/packages/eclipse-classic-372/indigosr2
Android Developer Tools (ADT)		Eclipse plug-in that provides a suite of tools.	https://developer.android.com/sdk/installing/installing-adt.html
Android Stand-alone SDK Tools		Basic tools for app development for use without an Integrated Development Environment (IDE).	https://developer.android.com/sdk/index.html#Other

Note There are many different methods and platforms to use when developing and testing apps for Google Android. Android Studio is an Integrated Development Environment (IDE) that is available, however please note we have not tested using this tool. For additional IDE details and information, go to:

- https://developer.android.com/sdk/index.html

Requirements

- Download and install Eclipse:
  - Eclipse version 3.7.2 is available at:

    https://eclipse.org/downloads/packages/eclipse-classic-372/indigosr2

Note Existing Eclipse installations require the Eclipse JDT plug-in (included in most Eclipse IDE packages) and JDK 6 (JRE alone is not sufficient). For the latest requirements, refer to:

https://developer.android.com/sdk/installing/installing-adt.html

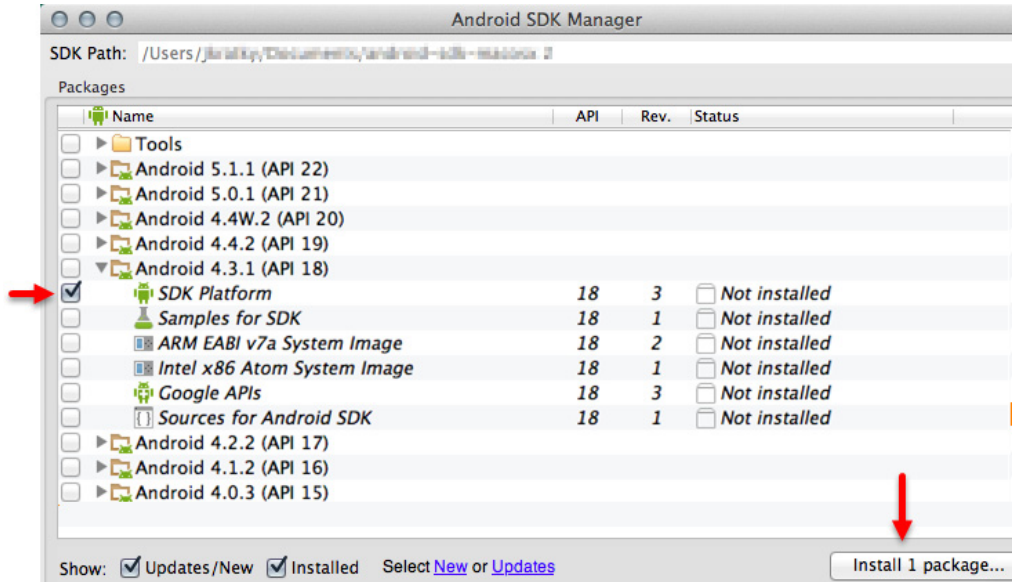
- Download and install the Android Developer Tools (ADT) plug-in available at:

  https://developer.android.com/sdk/installing/installing-adt.html
- Download and unpack/unzip the Android Stand-alone SDK Tools available at:

  https://developer.android.com/sdk/installing/index.html
- Set up the ADT tools plug-in by completing the following:
  - Launch Eclipse and when prompted select a folder to use as your workspace.
  - In Eclipse select **Help > Install New Software**. Click **Add** (top-right corner) and enter "ADT Plugin" in the name field and the following URL for the location: https://dl-ssl.google.com/android/eclipse/. Finish the installation, accept the license agreements, then restart Eclipse.
  - In the "Welcome to Android Development" window that appears, select **Use existing SDKs**. Navigate to and select the location of the Android Stand-alone SDK Tools folder. Click **Next.**
  - From the **Window** drop-down menu, launch the **Android SDK Manager**. Open the applicable **Android** folder and check the **SDK Platform box**. Deselect everything else, then install the selected package as shown in Figure 3-1:

Note Cisco StadiumVision Mobile supports Android 4.0 (API 14) and higher.

Figure 3-1 Selecting the SDK Platform Box



- Latest Cisco StadiumVision Mobile SDK tar.bz2 file, contact your Cisco account team for details as to how to become part of the Cisco StadiumVision Mobile SDK partner program.

Download and Unpack the SDK

Step 1 Download **StadiumVisionMobileSample-Android-VERSION.tar.bz2**. If you do not have this file, contact your Cisco account team for details as to how to become part of the Cisco StadiumVision Mobile SDK partner program.

Step 2 Extract the downloaded package into a directory. Table 3-3 lists the extracted content and includes a brief description.

Table 3-3 Cisco StadiumVision Mobile SDK File Content

Contents	Description
AndroidManifest.xml	File that presents information about your app to the Google Android system.
assets/	Contains files that can be included in the package.
build.xml	File used by ant or Eclipse programs to build an executable.
clean.stream	Sample stream for the stream sender.
html/	Contains Doxygen API documentation that is accessible by opening the index.html file in a web browser.
libs/	Contains library files used by the SDK.
Manifest	Declares app components, file must be located at the root of the project directory.
obj/	Contains temporary files (object or other) used to create the binary package.
proguard-project.txt	File that is automatically generated by Android tools to enable Proguard.

Table 3-3 Cisco StadiumVision Mobile SDK File Content (continued)

Contents	Description
proguard.cfg	File used by the Proguard tool to optimize and objuscate the SDK code.
proguard.cfg.save	File generated by the Proguard program before it obfuscates the SDK code for a new release.
project.properties	Contains information such as build platform target and library dependencies.
README	File that contains information to get started.
res/	Contains drawable objects, animation, layout, string, color, style that the SDK depends on.
src/	Contains the source for SDK components.

Note The clean.stream file that comes bundled with the SDK contains just one video channel. To provide app developers with additional ways to test multiple channels, an additional set of clean.stream files is available. For additional information refer to "Testing Your Cisco StadiumVision Mobile App" section on page 1-8.

Step 3 Open the API documentation available in the Doxygen build that is downloaded with the SDK. Navigate to the extracted folder contents, open the **html** folder > double-click **index.html** to launch the documentation in a web browser.

Getting Started with the Android Demo App

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

Note Before creating a new app, review the Cisco StadiumVision Mobile SDK Best Practices, page 1-9.

Compile the Demo App

Step 1 Import the demo app project into Eclipse as follows:

a. In Eclipse go to **File > Import.**

b. Go to **General > Existing Projects into Workspace**, then select **Next**.

c. Click **Browse** to Select the root directory and navigate to the folder where you unpacked the Cisco StadiumVision Mobile SDK, then click **Finish.**

d. Restart Eclipse from **File > Restart**.

Step 2 Right-click **CiscoStadiumVisionMobile** in the left Package Explorer window, then select **Android Tools > Export Signed Application Package**.

Note If you cannot complete the export due to build errors, check the Android path. Right-click on the Cisco StadiumVisionMobileSample in the left panel, select **Build Path > Configure Build Path**. In the Properties window that appears, select **Android** on the left, then select the check box next to the latest Target Name (for example Android 4.4.2) under Project Build Target. Click **Apply**, then **OK**. Restart Eclipse from **File > Restart**.

Step 3 Click **Next** when the Project Checks window appears.

Step 4 Select **Create new keystore**, then browse to a folder where you wish to store the key store file. Click **Next**.

Step 5 Fill in the Key Creation form (there are no right or wrong answers). Click **Next**.

Step 6 Browse to the folder where you wish to place the apk file, then click **Finish**.

Step 7 Download the **apk file** to your Android device by placing it on a web server, emailing it, SD card, or USB flash key, etc.

Step 8 Install the **apk** on your device.

Customize the Demo 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 file "res/values/strings.xml" change "SVM Demo" to "My SVM App"
- Change the name space so that 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.

Embed the Cisco StadiumVision Mobile SDK in an Existing App

Integration Checklist

The following table outlines the integration steps for embedding Cisco StadiumVision Mobile SDK into an existing app:

Area	Action or Verification
Supported Android OS Version	Set the app's Android version target to v4.0 (API 14) or above.
Android App Permissions	Add the required permissions to "AndroidManifest.xml."
Copy Config Files	Add the config files to the app's "assets" folder.
Copy Libraries	Add the Java and native libraries to the app's "libs" folder.
Set a Video "SurfaceView"	Add a "SurfaceView" to the player Activity's layout XML file.
Life-Cycle Notifications	Forward life-cycle notifications to the StadiumVision Mobile SDK.
Android Project Build Paths	Set the project build path to include the Jar files in "./libs/".

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/libvoAndroidVR_S23_OSMP.so
./libs/armeabi/libvoAudioMCDec_OSMP.so
./libs/armeabi/libvoIOMXDec_jb_OSMP.so
./libs/armeabi/libvoOSSource_OSMP.so
./libs/armeabi/libvompEngn_OSMP.so
```

```
./libs/armeabi/libvoAACDec_OSMP.so
./libs/armeabi/libvoAndroidVR_S40_OSMP.so
./libs/armeabi/libvoH264Dec_OSMP.so
./libs/armeabi/libvoIOMXDec_kk_OSMP.so
./libs/armeabi/libvoPushPDMgr_OSMP.so
./libs/armeabi/libvoAMediaCodec_OSMP.so
./libs/armeabi/libvoAndroidVR_S41_OSMP.so
./libs/armeabi/libvoH264Dec_v7_OSMP.so
./libs/armeabi/libvoLogSys.so
./libs/armeabi/libvoTsParser_OSMP.so
./libs/armeabi/libvoAndroidVR_S16_OSMP.so
./libs/armeabi/libvoAndroidVR_S43_OSMP.so
./libs/armeabi/libvoH265Dec_v7_OSMP.so
./libs/armeabi/libvoMMCCRRS_OSMP.so
./libs/armeabi/libvoVersion_OSMP.so
./libs/armeabi/libvoAndroidVR_S20_OSMP.so
./libs/armeabi/libvoAndroidVR_S50_OSMP.so
./libs/armeabi/libvoIOMXDec_L_OSMP.so
./libs/armeabi/libvoMMCCRRS_v7_OSMP.so
./libs/armeabi/libvoVidDec_OSMP.so
./libs/armeabi/libvoAndroidVR_S22_OSMP.so
./libs/armeabi/libvoAudioFR_OSMP.so
./libs/armeabi/libvoIOMXDec_ics_OSMP.so
./libs/armeabi/libvoOSEng_OSMP.so
./libs/armeabi/libvodl.so
```

Android Project Classpath

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

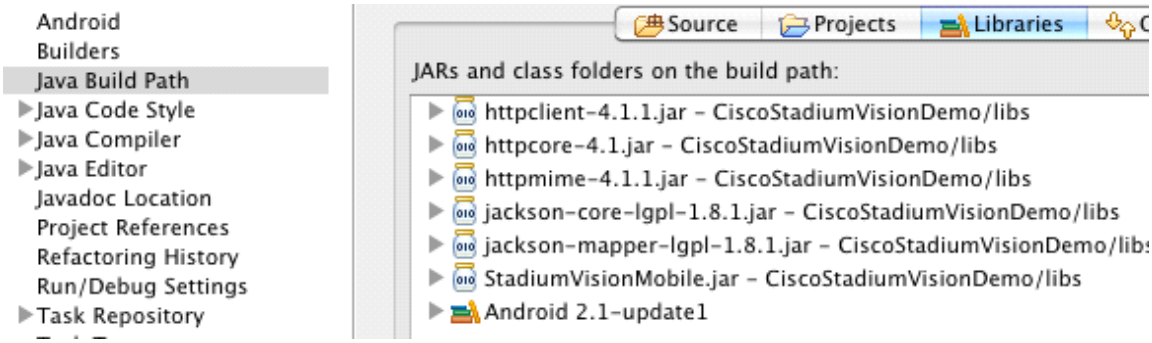
Step 1 Right-click your project in Eclipse.

Step 2 Select **Properties > Java Build Properties**.

Step 3 Select **Add JARs**.

Step 4 Add each of the Java JAR files listed in Adding Java JAR Files in Eclipse14.

Figure 3-2 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 5.2 as of April, 2015)
- 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
- Updates 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);
    }
});
```

Configuration Files

There are three configuration files that must be bundled with any Android app using the StadiumVision Mobile SDK (shown in Table 3-4).

Table 3-4 Configuration Files

Config File Name	Description
"cisco_svm.cfg"	StadiumVision Mobile SDK configuration file that contains the "Field-of-Use" parameters and some optional Wi-Fi network debugging information. The three "field-of-use" properties in the "cisco_svm.cfg" configuration file that need to be configured for each StadiumVision Mobile application are: • Venue Name • Content Owner • App Developer
"vompPlay.cfg"	Video decoder config file that contains the tuned decoding parameters. These settings should never be changed. Any changes could result in poor video or audio playback.
"voVidDec.dat"	Video decoder license file.

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

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

Wi-Fi AP Info Configuration (Optional)

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

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

How Cisco StadiumVision Mobile Fits into the Android Framework

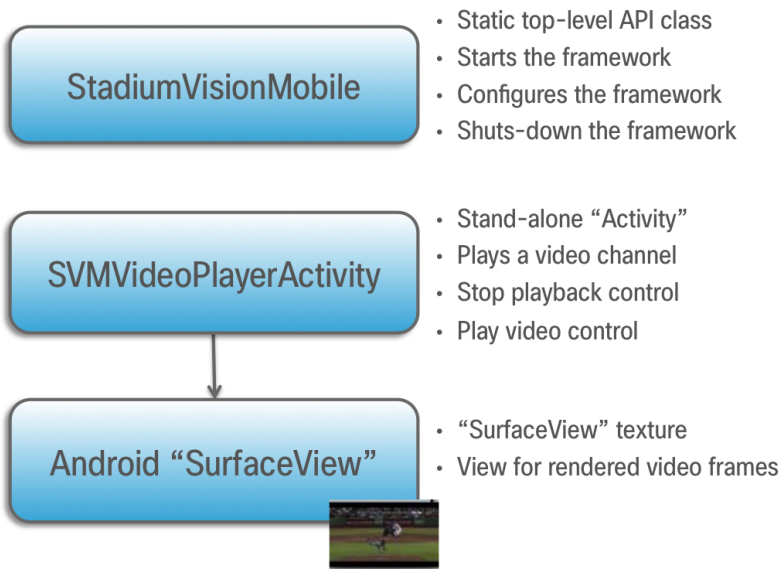
This section describes how SVM fits into the Android Framework, and contains the following sections:

Android API Class Overview

Figure 3-3 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 global structure created within the current process. It is tied to the lifetime of the process rather than the current component.

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

Figure 3-3 StadiumVision Mobile Class



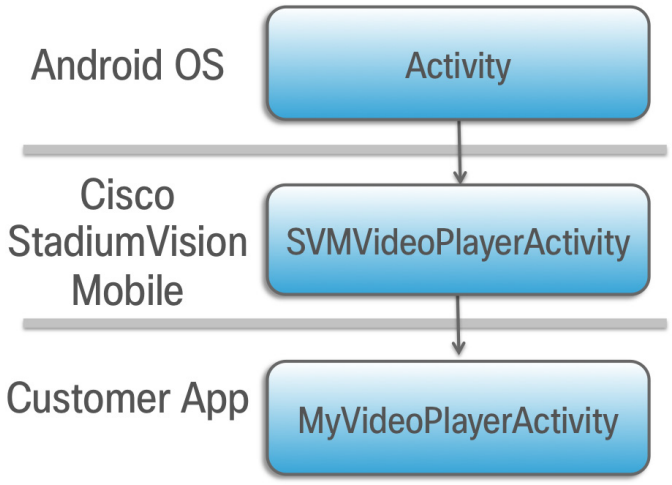
Android OS Activity Overview

Figure 3-4 depicts the Android OS with regard to Activities. An Activity represents both the screen layout and controller code. A new Activity is launched by sending an Intent to the Android OS. An intent is a message to Android OS to launch a particular activity. Extra parameters contained in an Intent and are passed to an Activity. The back button is a hard device button used to generically display the previous Activity, and moves back down the Activity stack.

Figure 3-4 Android Activity Overview

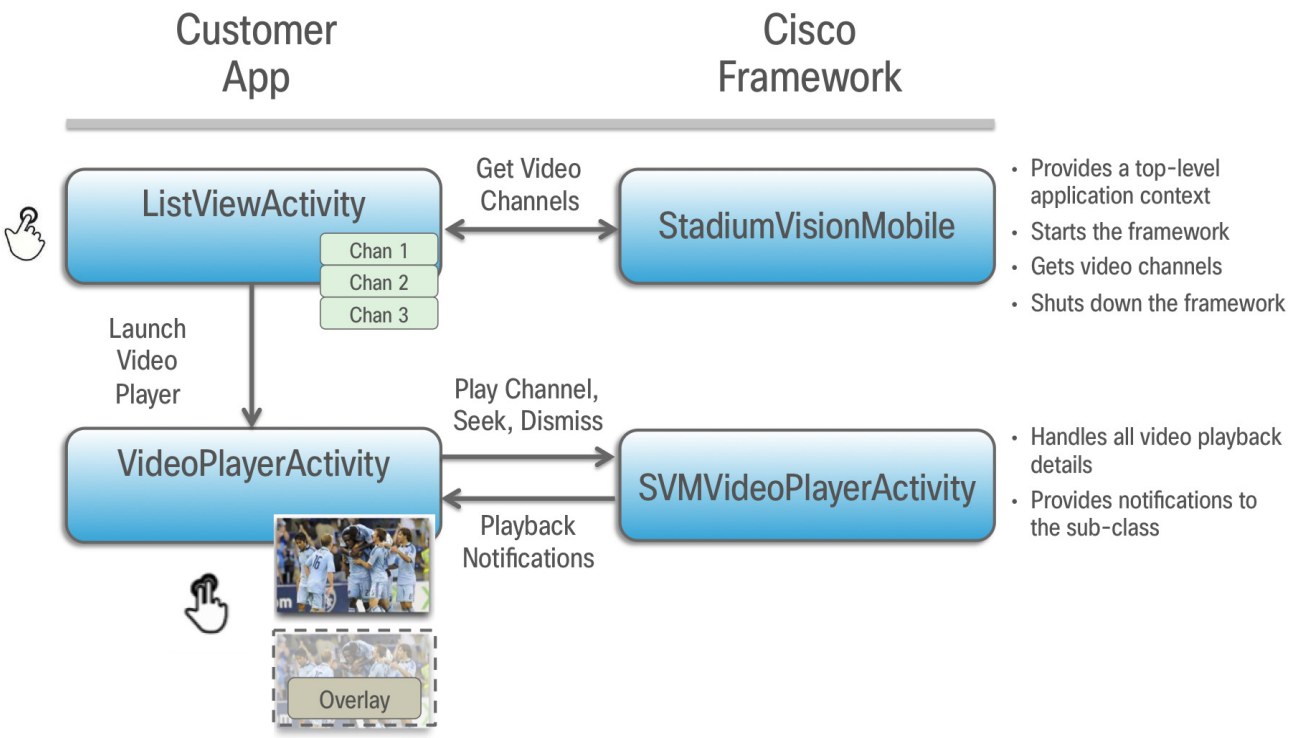
Figure 3-5 depicts the Activity inheritance between the Android OS, Cisco StadiumVision Mobile, and the client application.

Figure 3-5 Android Video Player Activity Inheritance



Client Application Integration Overview

Figure 3-6 Cisco StadiumVision Mobile Integration Overview

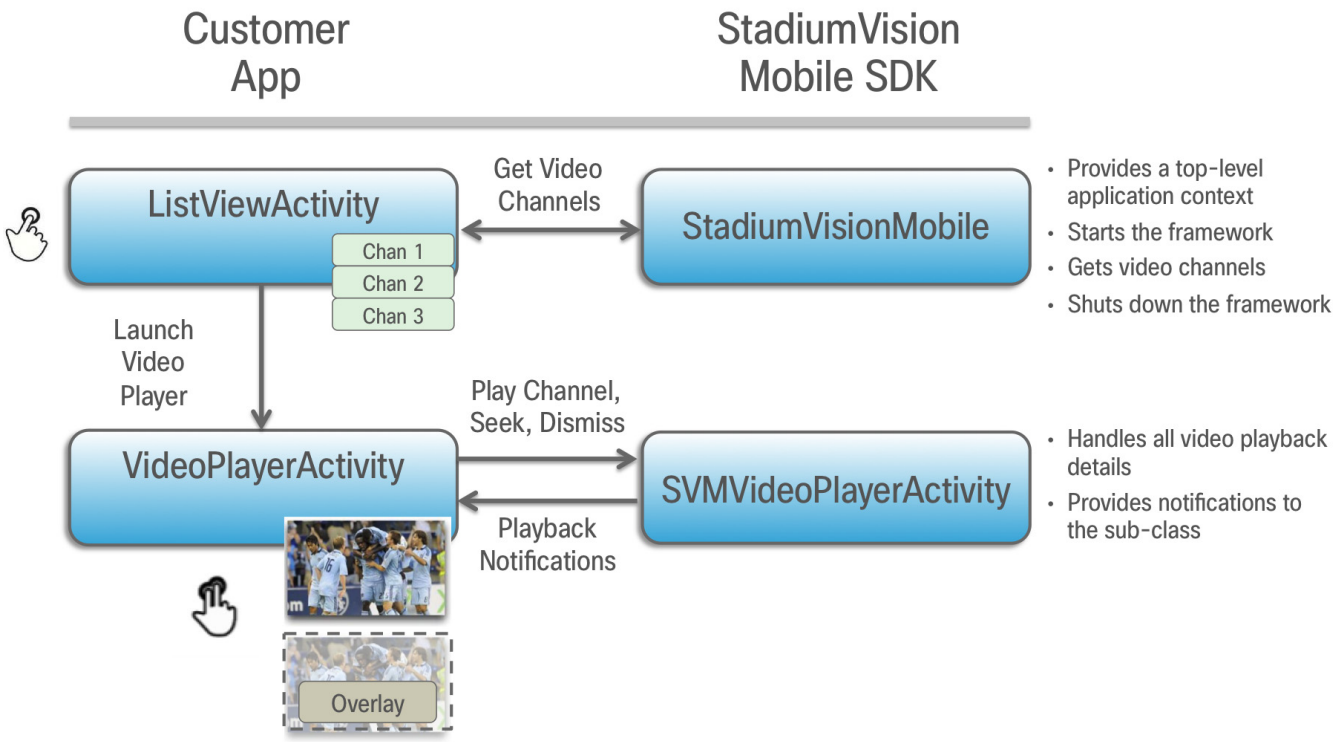


Customer Application Roles

Figure 3-7 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 3-7 Customer Application Responsibilities



Cisco StadiumVision Mobile Methods and Functions for Android

Cisco StadiumVision Mobile Android API Summary

Table 3-5 summarizes the Android API library. Detailed API documentation is available in the Doxygen build that is downloaded with the SDK. Navigate to the **htm1** folder and double-click **index.html** to launch the documentation in a web browser.

Table 3-5 Cisco StadiumVision Mobile Android API Summary

Return Type	API Method Name	API Method Description
ArrayList<String>	getAllowedReporterUrls	Gets a list of the Reporter stats upload URLs associated with Streamer servers (duplicate entries are removed).
ArrayList<String>	getLogComponentArrayList	Gets the array list of available components which can have their component logging level set individually.
ArrayList<String>	getLogLevelArrayList	Gets the array list of available logging levels that can be applied to any component.
ArrayList<SVMChannel>	getAudioChannelArrayList	Gets the array list of available audio channels.
ArrayList<SVMChannel>	getDataChannelArrayList	Gets the array list of available data channels.
ArrayList<SVMChannel>	getFileChannelArrayList	Gets the array list of available file channels.
ArrayList<SVMChannel>	getVideoChannelArrayList	Get the array list of available video channels.

Table 3-5 Cisco StadiumVision Mobile Android API Summary (continued)

Return Type	API Method Name	API Method Description
ArrayList<SVMStreamer>	getStreamerArrayList	Gets the array list of Cisco SVM Streamer servers detected by the SDK.
HashMap<String,Object>	getFileDistributionTable	Gets a HashMap of the current SDK file distribution table.
HashMap<String,String>	getStats	Gets a HashMap of the current SDK stats as a hash name/value.
JSONObject	getConfig	Gets the current SDK configuration as a 'JSONObject' object.
SharedPreferences	getSharedPreferences	Gets the Android SharedPreferences object that can be used to save arbitrary, app-specific preference settings that survives app restarts.
String[]	getLogComponentArray	Gets the array of available components which can have the component logging level set individually.
String[]	getLogLevelArray	Gets the array of the available SVM SDK logging levels that can be applied to any component.
String	getAppSessionUUID	Gets the app session UUID that is generated by the SVM SDK. This UUID uniquely identifies each time the SDK is started and is used for consistent statistics collection and reporting.
String	getDeviceUUID	Gets the device UUID generated by the SVM SDK and is saved in the app's shared preferences. **Note** Android does not provide a consistent and reliable device UUID across all of the Android OS versions supported by the SVM SDK, so a generated device UUID is used instead.
String	getFileDistributionLocalFilename	Gets the local filesystem filename for any object given its URI and the file channel.
String	getLocalIpAddress	Gets the IP address of the local device.
String	getSessionUUID	Gets the unique SVM identifier for the session.
String	getVideoSessionUUID	Gets the unique SVM identifier for the video session.
String	sdkVersion	Property that contains the SVM SDK version string.
SVMBatteryInfo	getBatteryInfo	Gets the current battery info for the device. This information gets collected in the statistics information that is uploaded to the Reporter server (if stats collection is enabled).
SVMChannel[]	getAudioChannelArray	Gets the array of available audio channels.
SVMChannel[]	getDataChannelArray	Gets the array of available data channels.
SVMChannel[]	getFileChannelArray	Gets the array of available file channels.
SVMChannel[]	getVideoChannelArray	Gets the array of available video channels.
SVMChannelManager	getChannelManager	Gets the channel manager.
SVMInventoryManager	getInventoryManager	Gets the internal inventory manager.

Table 3-5 Cisco StadiumVision Mobile Android API Summary (continued)

Return Type	API Method Name	API Method Description
SVMLocation	getCurrentLocation	Gets the current location.
SVMServiceState	getServiceState	Gets the service state.
SVMStatsManagerStats	getStatsManagerStats	Gets the current stats manager information.
SVMStatus	addDataChannelObserver	Registers an observer class to receive data for a particular channel.
SVMStatus	addFileChannelObserver	Registers an observer class to receive data for a particular file channel.
SVMStatus	allowAllStreamers	Allows all Streamers to be processed by the SDK.
SVMStatus	allowStreamers	Allows only the specified Streamers in the list to be processed by the SDK.
SVMStatus	disableQualityMonitoring	Disables quality monitoring within the SDK.
SVMStatus	disableStatsCollection	Disables the SVM SDK from performing statistics collection and thereby disables the uploading of the statistics information to the Reporter server.
SVMStatus	enableQualityMonitoring	Enables quality monitoring within the SDK.
SVMStatus	enableStatsCollection	Enables the SVM SDK from performing statistics collection and uploading to the Reporter server.
SVMStatus	removeDataChannelObserver	Unregisters an observer class from receiving data for a particular data channel.
SVMStatus	removeFileChannelObserver	Unregisters an observer class from receiving data for a particular file channel.
SVMStatus	setConfig	Sets the SVM SDK configuration at run-time using a populated 'JSONObject' object. This method overrides any configuration properties set with the 'cisco_svm.cfg' configuration file.
SVMStatus	setConfigWithString	Sets the SVM SDK configuration at run-time using a JSON-formatted 'String' object. This method overrides any configuration properties set with the 'cisco_svm.cfg' configuration file.
SVMStatus	setLogLevel	Sets the global logging level for the entire SVM SDK, with all internal components getting their logging level set to the same level.
SVMStatus	shutdown	Shuts down the SVM SDK.
SVMStatus	start	Starts the SVM SDK and any required SVM background threads and component managers.
SVMStreamer[]	getStreamerArray	Gets the list of Cisco SVM Streamer servers detected by the SDK.
SVMWifiInfo	getWifiInfo	Gets the current Wi-Fi network connection information. This information gets collected in the statistics information that is uploaded to the Reporter server.

Table 3-5 Cisco StadiumVision Mobile Android API Summary (continued)

Return Type	API Method Name	API Method Description
void	displayMessage	Convenience method displays the given string as an Android "toast" message that overlays anything currently on the device screen.
void	killAppProcess	Kills the entire Android application.
void	onCreate	Calls on application startup since this class extends the Android 'Application' class. **Note** It is required by the customer app to add the 'com.cisco.svm.app.StadiumVisionMobile' class as the global app context. This guarantees that the SVM framework has a valid 'Context' that is not tied to a client application Activity.
void	onData	Implemented by the customer app and is used as a callback from the SVM SDK. Each callback from the SDK to the customer app provides a received data message on the given data channel, delivered as a byte array.
void	onDestroy	Destroys an activity.
void	onPause	Informs the SVM SDK when a client app Activity has stopped. Forwarding each client app Activity's "onPause()" life-cycle event allows the SVM 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.
void	onResume	Informs the SVM SDK when a client app has started. Forwarding each client app Activity's "onResume()" life-cycle event allows the SVM SDK to declare the client Android app as "inactive" and to shutdown all CPU and networking resources used by the SVM SDK.
void	setInactivityTimeoutMs	Sets the inactivity timer timeout threshold used by the StadiumVision Mobile SDK to determine when the client Android app has "stopped".

Return Status Object

Each API call returns an 'SVMStatus' object whenever applicable. Table 3-6 lists the SVMStatus object fields.

Table 3-6 SVMStatus Object

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

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

Table 3-7 lists the hash keys and stats description for the getStats API.

Table 3-7 getStats API Hash Keys and Description

Stats Hash Key	Description
announcement_session_id	Video session announcement ID.
announcement_session_title	Session announcement name.
announcementsMalformed	Number of malformed channel announcements received.
announcementsNotAllowed	Number of received announcements not allowed (source Streamer is not allowed).
announcementsReceived	Number of received channel announcements.
channelsAdded	Number of times that the channel listener added a channel to the channel.
channelsPruned	Number of times that the channel listener pruned a channel from the channel list.
invalidJsonAnnouncements	Number of received announcements with an invalid JSON body.
ipv4Announcements	Number of IPv4 channel announcements received.
ipv6Announcements	Number of IPv6 channel announcements received.
licenseMismatchAnnouncements	Number of received announcements with mismatched license information.
listenerIgmpRestarts	Number of announcement listener IGMP restarts.
num_compressed_announcements	Number of compressed announcements received.
num_dropped_video_frames	Total number of video frames dropped.
num_ts_discontinuities	Total number of MPEG2-TS packet discontinuities.
session_link_indicator	Health of the Wi-Fi network connection. Ranges from 0 (poor) to 10 (excellent).
session_uptime	Length of time the session has been active (in seconds).
total_num_bytes_written	Total number of video bytes played.
protection_windows	Total number of protection windows sent.

Table 3-7 getStats API Hash Keys and Description (continued)

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

Video Player Activity API Summary

The SVMVideoPlayerActivity class can be extended and customized. Table 3-8 lists the SVMVideoPlayerActivity API methods and descriptions.

Table 3-8 Video Player Activity API Summary

Return Type	API Method Name	API Method Description
SVMStatus	playLive	Moves the video playback buffer pointer to the head ("live") offset position in the video playback buffer. This convenience method acts as a wrapper for the "seekAbsolute" API method; making "playLive()" equivalent to "seekAbsolute(0)".
SVMStatus	playVideoChannel	Starts playback of a particular video channel, changing channels on subsequent calls.
SVMStatus	rewindForDuration	Rewinds the video playback buffer pointer relative to the current playback buffer offset position. Should a duration be given that is larger than the size of the video history buffer, the SVM SDK will rewind the video play-head as far as possible. This convenience method acts as a wrapper for the "seekRelative" API method; making the given "durationMs" value negative before calling "seekRelative". For example, "rewindForDuration(20000)" is equivalent to "seekRelative(-20000)".
SVMStatus	seekAbsolute	Seeks the playback buffer pointer from the head ("live") offset position of the video playback buffer. • To play the most current live video pass in on offset of zero (0 ms). • To play video in the past, a positive duration will be used as an offset for rewinding back in time (relative to the "live" position).
SVMStatus	seekRelative	Seeks the playback buffer pointer relative to the current playback buffer offset position.
SVMStatus	setVideoSurfaceView	Sets the Android UI "SurfaceView" where video frames will get rendered.
SVMStatus	shutdown	Stops video playback of the currently playing video channel by stopping the native player, native decoder, and terminating this Android Activity.

Adding Cisco StadiumVision Mobile Services to an Android App—Code Structure and Samples

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

Start 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();
}
```

Notify Life-Cycle Activity

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

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

void onPause() {
   // notify the cisco sdk of the life-cycle event
   StadiumVisionMobile.onPause();
}

void onResume() {
   // notify the cisco sdk of the life-cycle event
   StadiumVisionMobile.onResume();
```

Indicate StadiumVision Mobile Service: Up or Down

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 Wi-Fi interface is disabled.

Poor video quality can occur when the user is receiving a weak Wi-Fi signal; causing data loss. There are two different ways that the 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 down by the SDK. The reasons bitmap is given in Table 3-9.

Table 3-9 Service Down Notifications

Service Down Reason	Constant
Poor video quality networking conditions detected	StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_POOR_QUALITY
Wi-Fi 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

Note For additional Service Down Notification details, refer to "Cisco StadiumVision Mobile SDK Best Practices" section on page 1-9.

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_TA
G);
registerReceiver(serviceStateReceiver, serviceStateIntentFilter);
```

Getting 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");
}
```

Detect Mobile Device Connection

Beginning in Cisco StadiumVision Mobile Release 2.0, the 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);
```

Getting 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 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\""
    "    }"
    "}";
```

Scalable File Distribution

Table 3-10 lists the Cisco StadiumVision Mobile scalable file distribution API.

Table 3-10 Scalable File Distribution and Service API Summary

API Return Type	File Service API Method Name	Method Description
NSArray*	getFileChannelArrayList	Gets a snapshot array of the currently available file channels.
NSMutableDictionary*	getFileDistributionTable	Gets file distribution table details.
NSString*	getFileDistributionLocalFilename	Get local filesystem filename for any object given its URI and the file channel.
NSString*	getFileDistributionLocalFilename:forChannel	Get local filesystem filename for any object given its URI and the file channel.
NSString*	getFileDistributionLocalFilename:forChannel Name	Get local filesystem filename for any object given its URI and the file channel name.

Table 3-10 Scalable File Distribution and Service API Summary (continued)

API Return Type	File Service API Method Name	Method Description
SVMStatus*	addFileChannelObserver	Registers an observer class to receive data for a particular file channel.
SVMStatus*	addFileChannelObserver:forChannel	Registers an observer class to receive all file updates for a particular file channel.
SVMStatus*	addFileChannelObserver:forChannelName	Registers an observer class to receive all file updates for a particular file channel name.
SVMStatus*	removeFileChannelObserver	Unregisters an observer class from receiving file for a particular file channel.
SVMStatus*	removeFileChannelObserver:forChannel	Unregisters an observer class from receiving any file updates for a particular file channel.
SVMStatus*	removeFileChannelObserver:forChannelName	Unregisters an observer class from receiving any file updates for a particular file channel name.

Data Channels

Table 3-11 lists the Cisco StadiumVision Mobile data channel APIs.

Table 3-11 Data Distribution and Service API Summary

API Return Type	Data Service API Method Name	Method Description
ArrayList<SVMChannel>	getDataChannelArrayList	Gets the array list of available data channels.
SVMChannel[]	getDataChannelArray	Gets the array of available data channels.
SVMStatus	addDataChannelObserver	Registers an observer class to receive data for a particular channel.
SVMStatus	removeDataChannelObserver	Unregisters an observer class from receiving data for a particular data channel.
void	onData	Implemented by the customer app and is used as a callback from the SVM SDK. Each callback from the SDK to the customer app provides a received data message on the given data channel, delivered as a byte array.

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);
```

Set SDK Configuration using 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 3-12.

Table 3-12 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();
}
```

Obtain Additional Statistics

In the Cisco StadiumVision Mobile Release 2.0 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()
```

Note For a detailed table of the hash keys and stats description for the getStats API refer to Table 3-7.

Table 3-13 details the StatsManager dictionary keys and descriptions.

Table 3-13 StatsManager Dictionary Keys

Dictionary Key	Description
statsUploadAttempts	Number of Reporter stats upload attempts.
statsUploadErrors	Number of Reporter stat manager errors other than upload issues (for example, stat generation failures).
statsUploadFailures	Number of Reporter stats upload failures.
statsUploadRejects	Number of Reporter stats delivered but rejected.
statsUploadSuccesses	Number of Reporter stats upload successes.

Receive Video Player State Notifications

The 1.3 SDK generates broadcast Intent notifications for each of the video player state transitions (listed in Table 3-14). The application can listen to these notifications and take action based on the video player's state transitions.

Table 3-14 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);
```

Detect 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();
}
```

Customizing the Default Video Player

This section describes how to customize the default 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.

Figure 3-8 Default Cisco Video Player

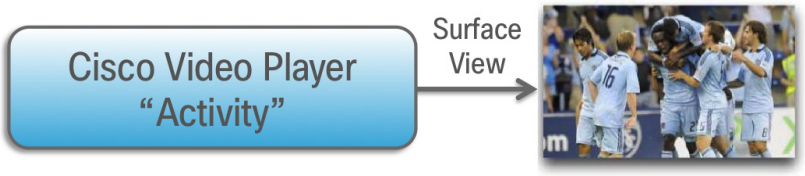


Figure 3-9 SVMVideoPlayerActivity API

Cisco Demo 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>
```

Video Channels

This section describes the Cisco StadiumVision Mobile SDK video channels and contains the following sections:

Getting the Video Channel List

The StadiumVision Mobile SDK dynamically receives all of the available channels (via Wi-Fi 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 3-15.

Table 3-15 SVMChannel Object Properties

SVMChannel Property	Property Description
appDeveloper	Name of the application developer.
bandwidthKbps	Data bandwidth consumed by the channel (in kbps).

Table 3-15 SVMChannel Object Properties (continued)

SVMChannel Property	Property Description
bodyText	Complete text description of the video channel.
channelType	Type of the channel.
contentOwner	Name of the content owner.
name	Name of the channel.
sessionNum	Session number of the channel.
venueName	Name of the venue.

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>
```

Data Channels

This section describes the Cisco StadiumVision Mobile SDK data channels and contains the following sections:

- Getting the Data Channel List, page 3-34
- Observing a Data Channel, page 3-35

Getting the Data Channel List

The StadiumVision Mobile SDK dynamically receives all of the available data channels (via Wi-Fi 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);

}
```

Audio Channels

This section describes the Cisco StadiumVision Mobile SDK audio channels and contains the following sections:

Getting the Audio Channel List

Cisco StadiumVision Mobile supports audio-only channels, in a similar manner as video channels.

Get a reference to the audio manager from the SDK

```
import com.cisco.svm.audio.SVMAudioManager;
    SVMAudioManager audioManager = StadiumVisionMobile.getAudioManager();
```

The application starts the audio channels by invoking

```
    audioManager.startAudioChannel(selectedChannel);
```

Stops them

```
audioManager.stopAudioChannel();
```

Audio channels will continue to play while other activities are active but will terminate when the application enters background unless

```
        // enable background audio
        SVMAudioManager audioManager = StadiumVisionMobile.getAudioManager();
        audioManager.enableBackgroundAudio ();
```

Activities can check to see if audio is playing using isAudioActive ()

```
        if (audioManager.isAudioActive()) {
            // Audio is playing.
        }
```

Available audio channels are discovered the same way that video channels are discovered.

```
        SVMChannel[] channels = StadiumVisionMobile.getAudioChannelArray();

}
```

EVS C-Cast Integration

Note Cisco StadiumVision Mobile is supported with EVS C-Cast version 2.x only. EVS C-Cast version 3.x is not supported.

The steps below describe a high level workflow of how a Cisco StadiumVision Mobile powered C-Cast app gains access to the XML timeline and media files.

1. Register a BroadcastReceiver to be notified when a file channel becomes available using **public Intent registerReceiver (BroadcastReceiver receiver, IntentFilter filter)**
2. Register to receive the channel notification using **public static com.cisco.svm.app.SVMStatus addFileChannelObserver (com.cisco.svm.channel.SVMChannel fileChannel, com.cisco.svm.file.ISVMFileObserver observer)**
3. Handle the file reception (movies/thumbnails /timeline) using **public void onFile (String channelName, String fileName,Integer fileState)**
4. Check to see if a file channel is already available, using **getFileChannelListArray.**
5. If a channel is already available, or when a callback notification is received, register a file channel observer, using **addFileChannelObserver**
6. Check to see if a file named ccast-timeline.xml is already available, using **getFileDistributionLocalFilename**
7. If the ccast-timeline.xml is not yet available, wait for additional files to arrive using **onFile()**. Each time onFile() is called, do a corresponding check with **getFileDistributionLocalFilename** to see if the new file is ccast-timeline.xml.
8. Once the ccast-timeline.xml file has been received, parse it using the steps in chapter 5 (How to build the media path) of the C-Cast API spec, and then build the media path for all media files.
9. For each file media path, remove the path prefix so that only the filename remains. For example: http://www.mydomain.com/videos/abc/def/ghi/abcdefghijklmnopqrstuvwxyz123456_hls-ipad.m3u8 becomes abcdefghijklmnopqrstuvwxyz123456_hls-ipad.m3u8
10. For each filename, cycle through **onFile()** and **getFileDistributionLocalFilename** until all files have been received.
11. Be prepared for the ccast-timeline.xml file to change at any time and repeat steps 6-8 whenever it changes.

CHAPTER 4

Cisco StadiumVision Mobile API for Windows Phone

First Published: May 26, 2015
Revised: June 12, 2015

This module describes the Cisco StadiumVision Mobile SDK Release 2.1 for Windows Phone and contains the following sections:

Introduction to Cisco StadiumVision Mobile SDK for Windows Phone

The Cisco StadiumVision Mobile Windows SDK contains the following components bundled together:

- .NET components, configuration files, player and layout XML files
- Windows Demo app with SDK video player
- API documentation (Doxygen build)

Note Cisco StadiumVision Mobile client application is designed for Windows Phones 8.1 and later, it is also supported on ARM processor-powered devices. It is not supported on Windows Phones 8.0 and earlier, all tablets, and x86 phone platforms. This means that the Windows Phone Emulator in Visual Studio is not supported because the emulator operates in x86 mode.

The API uses .NET classes on Windows to access the Cisco StadiumVision Mobile data distribution and video playback functionality within the Cisco StadiumVision Mobile Windows SDK library. DirectX is used to display video in a SwapChainPanel XAML element. Due to the .NET interface, the Cisco StadiumVision Mobile API can be called by C#/XAML client applications.

Note HTML/Javascript is not supported.

Table 4-1 describes the mobile operating system versions supported by the Cisco StadiumVision Mobile SDK.

Table 4-1 Mobile OS Support

	Windows Phone	
OS	**8**	**8.1**
Cisco StadiumVision Mobile SDK Release 2.1	No	Yes

For additional information, refer to the *Cisco StadiumVision Mobile Release Notes* available from Cisco.com at:

http://www.cisco.com/c/en/us/support/video/stadiumvision/products-release-notes-list.html

Cisco StadiumVision Mobile and Windows Developer Tools

Table 4-2 lists the various build environment requirements.

Table 4-2 Build Environment Requirements

Tool	Version	Description	URL
Mac or Windows PC	Windows 8.1	USB support is required if testing a device.	—
Visual Studio	2013 Update 4 or later	Development tools: Visual Studio 2013, Professional or Express.	http://msdn.microsoft.com/en-us/library/dd831853.aspx

Complete the following steps below in the same order to enable a successful setup:

Step 1 Run Microsoft Windows 8.1, check for and install any additional security patch updates.

Note Windows version 8.1 is required if using a Mac. We recommend using BootCamp, however there are multiple ways to emulate a Windows environment (such as virtual machine window or VMWare Fusion) that haven't been tested.

Step 2 Sign in to or create a Microsoft account at:

https://signup.live.com/signup.aspx?lic=1

Step 3 Install Visual Studio 2013 Update 4 (or later) Professional or Express.

Step 4 Plug in your Windows Phone to your workstation using a USB cable.

Step 5 Register your open/unlocked device for development at:

https://msdn.microsoft.com/en-us/library/windows/apps/dn614128.aspx

Step 6 Obtain the latest **StadiumVisionMobileSample-Windows Phone 8-xxxx.zip** file, contact your Cisco account team for details as to how to become part of the Cisco StadiumVision Mobile SDK partner program.

Download and Unpack the SDK

Step 1 Download **StadiumVisionMobileSample-Windows Phone 8-xxxx.zip**. If you do not have this file, contact your Cisco account team for details as to how to become part of the Cisco StadiumVision Mobile SDK partner program.

Step 2 Extract the downloaded package into a directory. Table 4-3 lists the extracted content and includes a brief description.

Table 4-3 Cisco StadiumVision Mobile SDK File Content

Contents	Description
SV Mobile for WP81/	Contains binaries and Doxygen documentation.
CiscoSvmDemo/	Contains the SVM header files, static library, and demo app source code.

Step 3 Open the API documentation available in the Doxygen build that is downloaded with the SDK. Navigate to the extracted folder contents, open the **SV Mobile for WP81** folder **> StadiumVisionMobile > Doxygen > html**. Double-click **index.html** to launch the documentation in a web browser.

Getting Started with the Windows Demo App

The Cisco StadiumVision Mobile SDK provided to app developers includes the source code for a Windows Demo app. The purpose of the Demo app is to demonstrate what is possible and to enable a new app developer to quickly get a working app up and running.

Note Before creating a new app, review the Cisco StadiumVision Mobile SDK Best Practices, page 1-9.

Compile the Demo App

Step 1 Launch **Visual Studio** to import the Demo app.

Step 2 Under **File > Open > Project/Solution**, locate and select **CiscoSvmDemo.sln** from the extracted folder contents, click **Open**. You can also launch Visual Studio by double-clicking the **CiscoSvmDemo.sln** file.

Step 3 Select the applicable device from the center of the icon bar located near the top of the Visual Studio window. If the bar does not show the Device selection, change it to the ARM selection in the **Build | Configuration Manager**. The Device selection will then be visible.

Note It is not possible to operate the Demo app using the Windows Phone emulator, however it is possible to operate the Demo app on a device because the emulator requires x86 support which is not currently available with the Cisco StadiumVision Mobile SDK.

Although you can manually select a different target, in order for the change to work you must make the change in the **Build | Configuration Manager**.

- A selection of ARM will change the target to the Device.
- A selection of Win32 will change the target to one of the emulator versions.

Even if you change the target drop-down from the center of the icon bar, Visual Studio will still build for the last platform selected in the Configuration Manager. In order to prevent compatibility issues, you must make the change in the Configuration Manager and not just the target drop-down.

Step 4 Click the **Build | Rebuild** menu bar selection. The build output can be seen in the window at the bottom of the screen made visible by the **View | Output** menu bar selection.

Step 5 Click the device selected above, start the build and run.

Note A device must be registered for development on a Microsoft account and be open (unlocked) in order to function.

Customize the Demo App

There are many ways to customize the Cisco StadiumVision Mobile Windows demo app including the following:

Step 1 Create a copy of the **CiscoSvmDemo** folder.

Step 2 Open the copied **CiscoSvmDemo.sln** file from the CiscoSvmDemo folder.

Step 3 Right-click the **View Designer link** from the VideoPage.xaml entry (located under the SVM Demo project in Solutions Explorer) to open the XAML designer.

Step 4 Use the XAML Designer or Blend to make changes as appropriate for the name of the application, additional buttons, and so on. The SwapChainPanel element can contain sizing information and also can be placed in other elements, such as Grid.

Note Results are not always predictable and some experimentation is required as the SwapChainPanel element does not give expected results if placed in certain elements such as ViewBox.

Step 5 After changes are made to the XAML file, build, and then run the changed file as described in Compile the Demo App, page 4-4.

Embed the Cisco StadiumVision Mobile SDK in an Existing App

Integration Checklist

To embed the Cisco StadiumVision Mobile SDK into an existing app, follow the integration list below:

1. Supported Windows OS and Visual Studio Versions
   - You must be running Windows 8.1 or later with all of the current updates.
2. Windows Project Template
   - Select one of the project templates from **Store Apps | Windows Phone Apps**.
3. Windows App Permissions
   - Add any required permissions to "Package.appxmanifest" using the UI display.
4. Add References
   - Add a reference to Microsoft Visual C++ 2013 Runtime Package for Windows Phone from the Windows 8.1 Extensions category.
   - Add a reference to the **StadiumVisionMobile** by browsing to "…\SV Mobile for WP81\StadiumVisionMobile\bin\ARM\Debug\StadiumVisionMobile.dll".

     The correct mode (Debug or Release) StadiumVisionMobile library needs to be added as a reference or an immediate crash will occur at runtime. You can either add the library reference manually when switching modes, or modify the CiscoSvmDemo.csproj file by having it select the proper mode for the reference. Use $(Configuration) rather than Debug or Release in the lines:

```
<ItemGroup>
   <Reference Include="StadiumVisionMobile, Version=1.0.0.0, Culture=neutral,
processorArchitecture=ARM">
      <SpecificVersion>False</SpecificVersion>
      <HintPath>..\SV Mobile for
WP81\StadiumVisionMobile\bin\ARM\$(Configuration)\StadiumVisionMobile.dll</HintPat
h>
    </Reference>
  </ItemGroup>
```

5. Set a Video "SwapChainPanel"
   - Add a "SwapChainPanel" to the player Window's layout XAML file. SwapChainPanel is not in the Designer Toolbox, but it is derived from the "Grid" component and is manually entered into the XAML text. The SwapChainPanel element can contain sizing information and also can be placed in other elements, such as Grid.

Note Results are not always predictable and some experimentation is required as the SwapChainPanel element does not give expected results if placed in certain elements such as ViewBox.

6. NuGet Packages
   - The app requires reference to the NuGet package "Newtonsoft.Json" version 6.0.8 or later for the platform "wp81".
7. Life-Cycle Notifications
   - Forward life-cycle notifications to the Cisco StadiumVision Mobile SDK, which is similar to how it's done the CiscoSvmDemo app sample.

Channel ListBox Window Example

The function "doInBackground" in the CiscoSvmDemo program "VideoPage.xaml.cs" illustrates the following actions:

- Periodically obtains the list of available video channels
- Update the Window's ListBox with the channel list

Video Channel Selection Example

The following example illustrates the following actions:

- Plays the video channel selected in the ListBox

```
        private void VideoFilesList_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            if (channels != null)
            {
                // get the selected channel
                SVMChannel selectedChannel =
channels[((ListBox)sender).SelectedIndex];

                Log.d(TAG, "Selected Video Channel = '" +
selectedChannel.name +
                    "', bandwidthKbps = " +
selectedChannel.bandwidthKbps +
                    "', timestampMs = " +
selectedChannel.timestampMs);

                // play the selected channel by launching the
"VideoPlayerPage"
                String parms =
String.Format("channel={0}&scaling={1}", selectedChannel.name,
StadiumVisionMobile.objSVM.VideoScalingModeAspectFit);
                Frame.Navigate(typeof(VideoPlayerPage), parms);
            }
```

Window 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 shut down and restart as needed. Each client Window needs to forward its life-cycle notifications, as shown in the following example:

```
public void onResume()
{
    // notify the Cisco StadiumVision Mobile framework of the life-cycle event
    StadiumVisionMobile.objSVM.onResume();

    objBkgThread = ThreadPool.RunAsync(new WorkItemHandler(doInBackground));

    // Loop until worker thread activates.
    while (objBkgThread.Status != AsyncStatus.Started) ;
}

public void onPause()
{
    // terminate the channel update background thread
    RequestStop();

    // notify the Cisco StadiumVision Mobile framework of the life-cycle event
    StadiumVisionMobile.objSVM.onPause();
}
```

Configuration

There is one configuration file that must be bundled with any Windows app using the StadiumVision Mobile SDK (shown in Table 4-4).

Table 4-4 Configuration File

Config File Name	Description
"cisco_svm.cfg"	Cisco StadiumVision Mobile SDK configuration file that contains the "Field-of-Use" parameters and some optional Wi-Fi network debugging information. The three "field-of-use" properties in the "cisco_svm.cfg" configuration file that need to be configured for each StadiumVision Mobile application are: • Venue Name • Content Owner • App Developer

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"
    }
}
```

Wi-Fi AP Info Configuration (Optional)

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

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

How Cisco StadiumVision Mobile Fits into a Windows Phone App

Cisco StadiumVision Mobile Class Overview

Figure 4-1 describes the StadiumVision Mobile classes.

The SVMVideoPlayerPage class is a class within StadiumVisionMobile rather than an inherited class for the app's video page. A connection from the Windows Phone app to the SVMVideoPlayerPage class includes a reference to the "Windows DirectX SwapChainPanel" when the StadiumVisionMobile class is instantiated.

Figure 4-1 StadiumVision Mobile Class

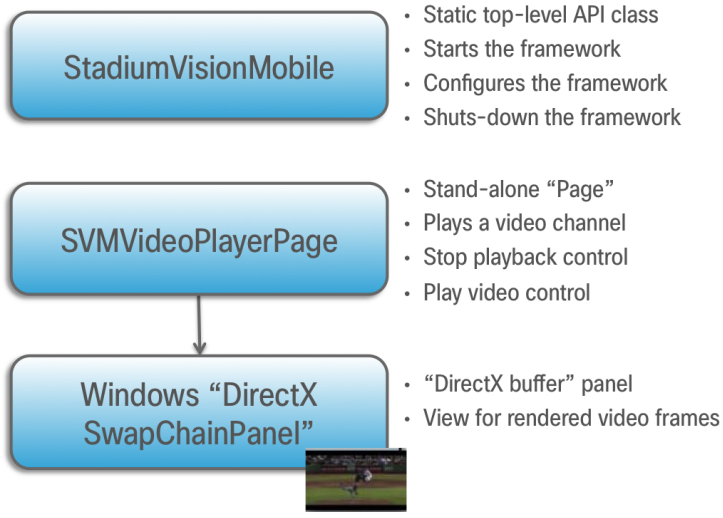


Figure 4-2 depicts the Windows OS with regard to Pages. A Page represents both the screen layout and controller code or Code Behind. A new Page is launched by sending an Event to the Windows OS. An Event is a message to Windows OS to launch a particular page. Extra parameters contained in an Event and are passed to a Page. The back button is typically a hard (sometimes soft) device button used to generically display the previous Page, and moves back down the Page stack.

Figure 4-2 Windows Overview

Figure 4-3 depicts the Window inheritance between the Windows OS, Cisco StadiumVision Mobile, and the Customer App.

Figure 4-3 Windows Video Player Window Inheritance

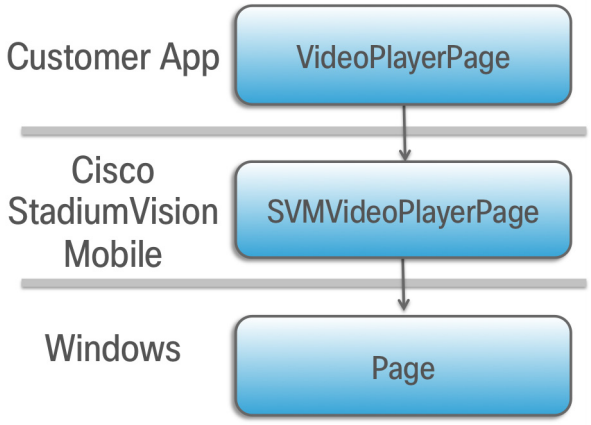
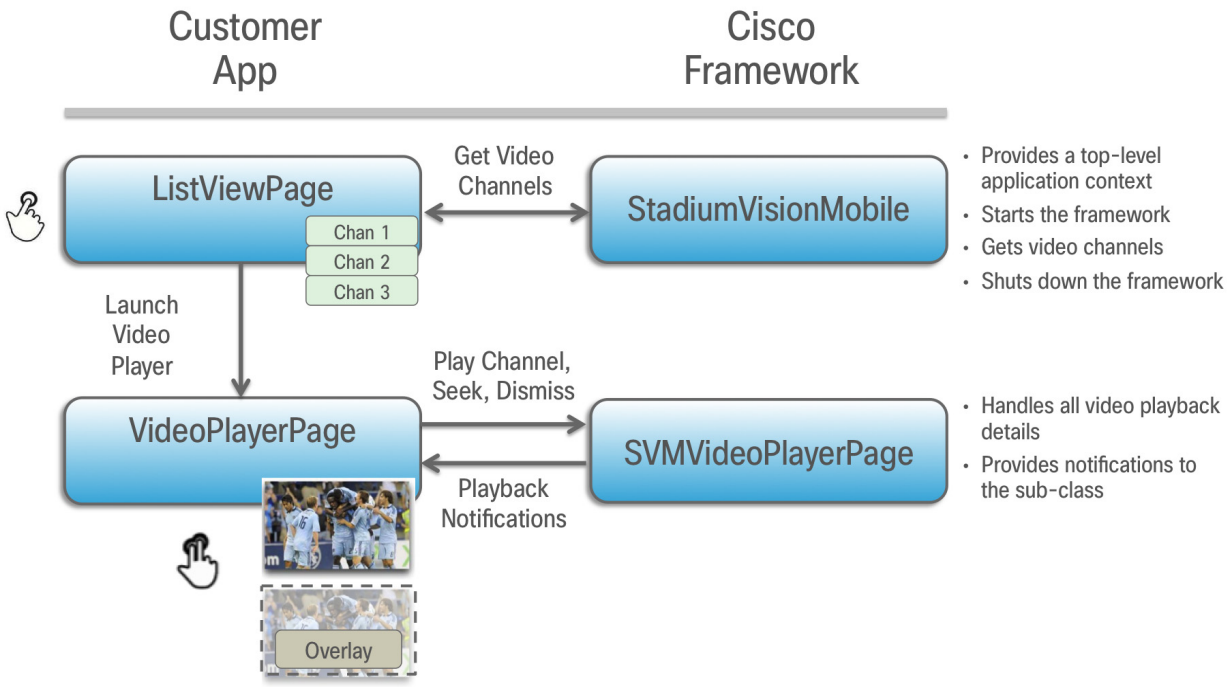


Figure 4-4 Cisco StadiumVision Mobile Integration Overview

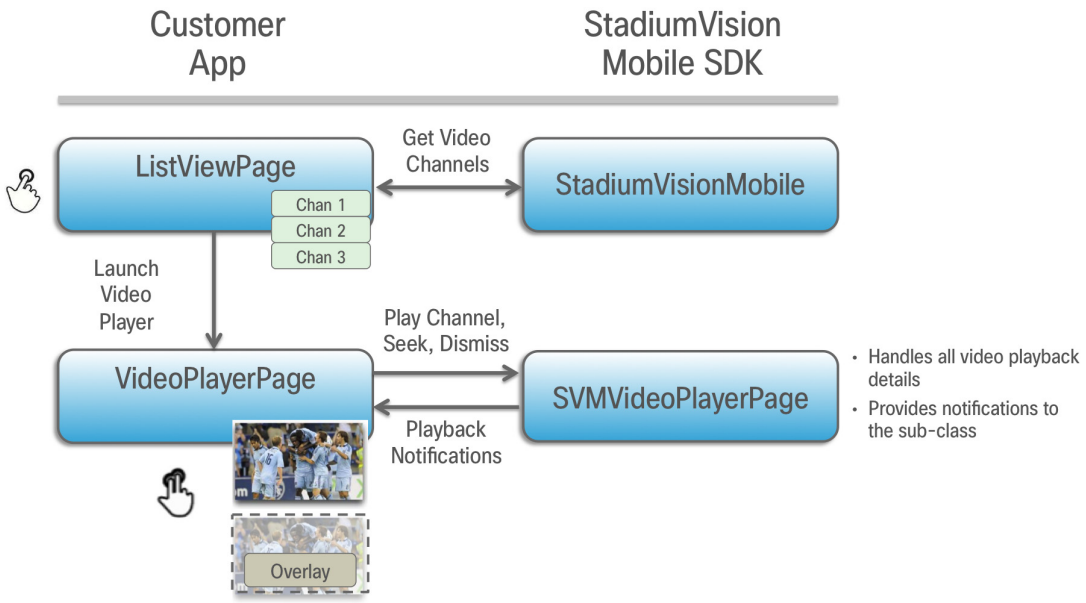


Customer Application Roles

Figure 4-5 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 4-5 Customer Application Responsibilities



Cisco StadiumVision Mobile Methods and Functions for Windows

Cisco StadiumVision Mobile Windows API Summary

Table 4-5 summarizes the Windows API library. Detailed API information is available in documentation Doxygen build that is downloaded with the SDK. Navigate to the SV **Mobile for WP81** folder > **StadiumVisionMobile > Doxygen > html**. Double-click **index.html** to launch the documentation in a web browser.

Table 4-5 Cisco StadiumVision Mobile Windows API Summary

Return Type	API Method Name	API Method Description
ApplicationDataContainer	getSharedPreferences	Gets the SharedPreferences object that can be used to save arbitrary, app-specific preference settings that survives app restarts.
async Task	start	Starts the SVM SDK and any required SVM background threads and component managers.
async Task <Dictionary <String, Object>>	getFileDistributionTable	Gets the current SDK file distribution table.
Dictionary<String, String>	getStats	Gets a C# Dictionary of current SVM SDK stats/counter values.
JObject	getConfig	Gets the current SDK configuration as a 'JObject' object.
List<String>	getAllowedReporterUrls	Gets a list of the Reporter stats upload URLs associated with Streamer servers (duplicate entries are removed).
List<String>	getLogComponentList	Gets a C# list of available components which can have their component logging level set individually.

Table 4-5 Cisco StadiumVision Mobile Windows API Summary (continued)

Return Type	API Method Name	API Method Description
List<String>	getLogLevelList	Gets a C# list of available logging levels that can be applied to any component.
String	getAppSessionUUID	Gets the app session UUID that is generated by SVM SDK. This UUID uniquely identifies each time the SVM SDK is started and is used for consistent statistics collection and reporting.
String	getDeviceUUID	Gets the device UUID that was generated by the SVMSDK and saved in the app's shared preferences. **Note** Windows does not provide a consistent and reliable device UUID across all of the Windows Phone OS versions supported by the SVM SDK, therefore a generated device UUID is used instead.
String	getFileDistributionLocalFilename	Gets the local filesystem filename for any object given its URI and the file channel.
String	getLocalIpAddress	Gets the IP address of the local device.
String	getVideoSessionUUID	Gets the video session UUID that is generated by the SVM SDK. This UUID uniquely identifies the current active video channel and is used for consistent statistics collection and reporting.
String	sdkVersion	Property that contains the SVM SDK version
String[]	getLogComponentArray	Gets a C# array of available components which can have their component logging level set individually.
String[]	getLogLevelArray	Gets a C# array of available logging levels that can be applied to any component.
SVMBatteryInfo	getBatteryInfo	Gets the current battery info for the device. This information gets collected in the statistics information that is uploaded to the Reporter server (if stats collection is enabled).
SVMChannel[]	getDataChannelArray	Gets a C# array of available data channels.
SVMChannel[]	getFileChannelArray	Gets a C# array of available file channels.
SVMChannel[]	getVideoChannelArray	Gets a C# array of available video channels.
SVMChannelList	getDataChannelList	Gets a C# list of available data channels.
SVMChannelList	getFileChannelList	Gets a C# list of available file channels.
SVMChannelList	getVideoChannelList	Gets a C# list of available video channels.
SVMChannelManager	getChannelManager	Gets the channel manager.
SVMInventoryManager	getInventoryManager	Gets the internal inventory manager.
SVMLocation	getCurrentLocation	Gets the current location.
SVMServiceStateEnum	getServiceState	Gets the service state.
SVMStreamer[]	getStreamerArray	Returns an array of Streamer servers detected by the SVM SDK; with each Streamer entry represented as an 'SVMStreamer' object in the array.

Table 4-5 Cisco StadiumVision Mobile Windows API Summary (continued)

Return Type	API Method Name	API Method Description
SVMStreamerList	getStreamerList	Gets the list of Cisco SVM Streamer servers detected by the SDK.
SVMStatsManagerStats	getStatsManagerStats	Gets the current stats manager information.
SVMStatus	addFileChannelObserver	Registers an observer class to receive data for a particular file channel.
SVMStatus	addDataChannelObserver	Registers an observer class to receive data for a particular data channel.
SVMStatus	allowAllStreamers	Allows all Streamers to be processed by the SDK.
SVMStatus	allowStreamers	Allows only the specified Streamers in the list to be processed by the SDK.
SVMStatus	disableQualityMonitoring	Disables quality monitoring within the SDK.
SVMStatus	disableStatsCollection	Disables the SVM SDK from performing statistics collection and thereby disables the uploading of the statistics information to the Reporter server.
SVMStatus	enableQualityMonitoring	Enables quality monitoring within the SDK.
SVMStatus	enableStatsCollection	Enables the SVM SDK from performing statistics collection and uploading to the Reporter server.
SVMStatus	removeDataChannelObserver	Unregisters an observer class from receiving data for a particular data channel name.
SVMStatus	removeFileChannelObserver	Unregisters an observer class from receiving data for a particular file channel.
SVMStatus	setLogLevel	Sets the global logging level for the entire SVM SDK, with all internal components getting their logging level set to the same level.
SVMStatus	setConfig	Sets the SVM SDK configuration at run-time using a populated 'JObject' object. This method will override any configuration properties set with the 'cisco_svm.cfg' configuration file.
SVMStatus	setConfigWithString	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.
SVMStatus	shutdown	Shuts down the SVM SDK.
SVMWifiInfo	getWifiInfo	Gets the current Wi-Fi connection information. This information gets collected in the statistics information that gets uploaded to the Reporter server (if stats collection is enabled).
void	displayMessage	Displays the given string as a Windows Phone "toast" message that overlays anything currently on the device screen.
void	killAppProcess	Kills the entire Windows Phone application.
void	onCreate	Calls on application startup.

Table 4-5 Cisco StadiumVision Mobile Windows API Summary (continued)

Return Type	API Method Name	API Method Description
void	onData	Implements the customer app and is used as a callback from the SVM SDK. Each callback from the SDK to the customer app provides a received data message on the given data channel, delivered as a byte array.
void	onDestroy	Destroys an activity.
void	onPause	Forwards each client app Window's "onPause" life-cycle event allows the SVM SDK to declare the client Windows app as "active" and potentially restart all of the internal component managers and threads that use the device's CPU and networking resources. This method must be called by each individual client app Window's "onPause" method to inform the SVM SDK of when a client app Window has stopped.
void	onResume	Forwards each Windows Page 'onResume' life-cycle notification to the SVM SDK to declare the client Windows app as "inactive" and shutdown all CPU and networking resources used by the SVM SDK. This method must be called by each individual client app Window's "onResume()" method to inform the SVM SDK of when a client app Window has started.
void	setInactivityTimeoutMs	Sets the inactivity timer timeout threshold used by the SVM SDK to determine when the client Windows app has "stopped".

Return Status Object

Each API call returns an 'SVMStatus' object whenever applicable. Table 4-6 lists the SVMStatus object fields.

Table 4-6 SVMStatus Object

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

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

Table 4-7 lists the dictionary keys and stats description for the getStats API.

Table 4-7 getStats API Dictionary Keys and Stats Description

Stats Dictionary Key	Stats Description
announcementsMalformed	Number of malformed channel announcements received.
announcementsNotAllowed	Number of received announcements not allowed (source Streamer is not allowed).
announcementsReceived	Number of received channel announcements.
announcement_session_id	Video session announcement ID.
announcement_session_title	Session announcement name.
channelsAdded	Number of times that the channel listener added a channel to the channel.
channelsPruned	Number of times that the channel listener pruned a channel from the channel list.
invalidJsonAnnouncements	Number of received announcements with an invalid JSON body.
ipv4Announcements	Number of IPv4 channel announcements received.
ipv6Announcements	Number of IPv6 channel announcements received.
licenseMismatchAnnouncements	Number of received announcements with mismatched license information.
listenerIgmpRestarts	Number of announcement listener IGMP restarts.
num_compressed_announcements	Number of compressed announcements received.
num_dropped_video_frames	Total number of video frames dropped.
num_ts_discontinuities	Total number of MPEG2-TS packet discontinuities.
protection_windows	Total number of protection windows sent.
session_link_indicator	Health of the Wi-Fi network connection. Ranges from 0 (poor) to 10 (excellent).
session_uptime	Length of time the session has been active (in seconds).
total_num_bytes_written	Total number of video bytes played.
window_error	Total number of protection windows with more packets per window than can be supported by SVM.
window_no_loss	Total number of protection windows with no dropped video packets.
window_recovery_failures	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	Total number of protection windows with recovered video packets.
window_warning	Total number of protection windows with more packets per window than the recommended value.
versionMismatchAnnouncements	Number of received announcements with a mismatched version number.

Video Player Window API Summary

The SVMVideoPlayerPage class can be extended and customized. Table 4-8 lists the SVMVideoPlayerPage API methods and descriptions.

Table 4-8 Video Player Window API Summary

Return Type	API Method Name	API Method Description
SVMStatus	playLive	Moves the video playback buffer pointer to the head ("live") offset position in the video playback buffer. This convenience method acts as a wrapper for the "seekAbsolute" API method; making "playLive()" equivalent to "seekAbsolute(0)".
SVMStatus	playVideoChannel	Starts playback of a particular video channel, changing channels on subsequent calls.
SVMStatus	rewindForDuration	Rewinds the video playback buffer pointer relative to the current playback buffer offset position. Should a duration e given that is larger than the size of the video history buffer, the SVM SDK will rewind the video play-head as far as possible. This convenience method acts as a wrapper for the "seekRelative" API method; making the given "durationMs" value negative before calling "seekRelative". For example, "rewindForDuration(20000)" is equivalent to "seekRelative(-20000)".
SVMStatus	seekAbsolute	Seeks the playback buffer pointer from the head ("live") offset position of the video playback buffer • To play the most current live video pass in on offset of zero (0 ms). • To play a video from the past, a positive duration will be used as an offset for rewinding back in time (relative to the "live" position).
SVMStatus	seekRelative	Seeks the playback buffer pointer relative to the current playback buffer offset position.
SVMStatus	setVideoSurfaceView	Sets the Windows UI "SwapChainPanel" where video frames will get rendered.
SVMStatus	shutdown	Stops video playback of the currently playing video channel by stopping the native player, native decoder, and terminating the video player window.

Adding Cisco StadiumVision Mobile Services to a Windows App—Code Structure and Samples

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

Starting the SDK

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

```
static public StadiumVisionMobile objSVM = new StadiumVisionMobile();  // create exactly
once
```

Cisco StadiumVision Mobile Service Up or Down Indicator

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 Wi-Fi interface is disabled.

Poor video quality can occur when the user is receiving a weak Wi-Fi signal; causing data loss. There are two different ways that the Windows 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 4-9.

Table 4-9 Service Down Notifications

Service Down Reason	Constant
Poor video quality networking conditions detected	StadiumVisionMobile.SVM_SERVICE_STATE_DOWN_REASON_POOR_QUALITY
Wi-Fi 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

Note For additional Service Down Notification details, refer to "Cisco StadiumVision Mobile SDK Best Practices" section on page 1-9.

Receiving "Service Up/Down" Notifications

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

SVMVideoPlayerPage objVpa;

objVpa = new SVMVideoPlayerPage(currentChannel);

```
            // register to receive events from SVM
            objVpa.onServiceUp += onServiceUp;
            objVpa.onServiceDown += onServiceDown;


        /// <summary>
        /// Called to notify of service up
        /// </summary>
        private void onServiceUp()
        {
            Log.i(TAG, "CLIENT: SERVICE UP");

            // Create a toast notification that the svm service is up
            PopToast("SVM Service is UP");
        }

        /// <summary>
        /// Called to notify of service down
        /// </summary>
        private void onServiceDown()
        {
            Log.i(TAG, "CLIENT: SERVICE DOWN");

            // Create a toast notification that the svm service is down
            PopToast("SVM Service is DOWN");
            /*
             * EXIT THIS PAGE NOW!
             */
            Application.Current.Exit();
        }
```

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

```
// get the current svm service state
SVMServiceStateEnum serviceState = objSVM.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 StadiumVision Mobile Release 2.1 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 Windows 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 "In Venue Up/Down" notifications from the SDK:

```
// register to receive events from SVM
objSVMChannelManager.venueEvent += onVenueChange;

        /// <summary>
        /// Called to notify of venue change
        /// </summary>
        private void onVenueChange (object sender, VenueEventArgs e)
        {
if (e.venueState == StadiumVisionMobile.objSVM.SVM_VENUE_CONNECTED_EVENT_TAG)
{
            Log.i(TAG, "CLIENT: VENUE CONNECTED");
} else {
            Log.i(TAG, "CLIENT: VENUE DISCONNECTED");
}
```

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:

```
// get whether the device is currently connected to the SVM licensed venue
boolean isConnectedToVenue = StadiumVisionMobile.objSVM.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

The application can 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 ways to set 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(JObject 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 JObject 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
JObject configObj = StadiumVisionMobile.getConfig();

// get the license dictionary from the config dictionary
JObject licenseObj = null;
try {
    licenseObj = configObj.getJObject("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
JObject jsonConfig = new JObject();
JObject licenseConfig = new
JObject(); 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 Windows 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.

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

Each "SVMStreamer" object contains the following properties listed in Table 4-10.

Table 4-10 SVMStreamer Object Properties

SVM Streamer 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
SVMStreamerList  streamerList = StadiumVisionMobile.objSVM.getStreamerList();

// iterate through the list of streamer objects
```

```
foreach (SVMStreamer nextStreamer in
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

Beginning with the Cisco StadiumVision Mobile Release 2.0 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 dictionarymap
public Dictionary<String, String> getStats()
```

Note For a detailed table of the hash keys and stats description for the getStats API refer to Table 4-7.

Table 4-11 details the StatsManager dictionary keys and descriptions.

Table 4-11 StatsManager Dictionary Keys

Dictionary Key	Description
statsUploadAttempts	Number of Reporter stats upload attempts.
statsUploadErrors	Number of Reporter stat manager errors other than upload issues (for example, stat generation failures).
statsUploadFailures	Number of Reporter stats upload failures.
statsUploadRejects	Number of Reporter stats delivered but rejected.
statsUploadSuccesses	Number of Reporter stats upload successes.

Video Player State Notifications

The SDK generates event notifications for each of the video player state transitions (listed in Table 4-12. The application can listen to these notifications and take action based on the video player's state transitions.

Table 4-12 Video Player State Notification

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

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

```
// subscribe to channel event

    public void registerChannelListChanged()
         {
             SVMChannelListener.objSVMChannelListener.channelChangeEvent
+= onChannelStateChanged;
         }



         public void unregisterChannelListChanged()
         {


          SVMChannelListener.objSVMChannelListener.channelChangeEvent
-= onChannelStateChanged;
         }



    private void onChannelStateChanged(object sender,
StadiumVisionMobile.ChannelChangeEventArgs e)
{
// e.channelName & e.channelState are two string arguments reported by
the event

// Video Player State Notification is contained in string e.channelState
}
```

Video Player "Channel Inactive" Event

To detect that a currently playing video channel has become invalid (due to Streamer server admin changes), the SVM video player ("SVMVideoPlayerPage") provides an event to tell the video player sub-class (ie: "VideoPlayerPage") 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 'SVMVideoPlayerPage' sub-class (ie: "MyVideoPlayerPage"). The following example shows the method signature and implementation of this event method:

```
SVMVideoPlayerPage objVpa;
objVpa = new SVMVideoPlayerPage(currentChannel);
// register to receive events from SVM
objVpa.onCurrentChannelInvalid += onCurrentChannelInvalid;



/// <summary>
/// Called to notify of channel invalid (inactive)
/// </summary>
private void onCurrentChannelInvalid()
{
}
```

Customizing the Default Video Player

This section describes customizing the video player. The default Cisco video player has the following features:

- Implements as a separate Windows "Page."
- Supports fullscreen video views or partial screen views inside the "SwapChainPanel" XAML control.
- Renders video frames using Window "SwapChainPanel."
- Uses a customized video player.

Figure 4-6 Default Cisco Video Player

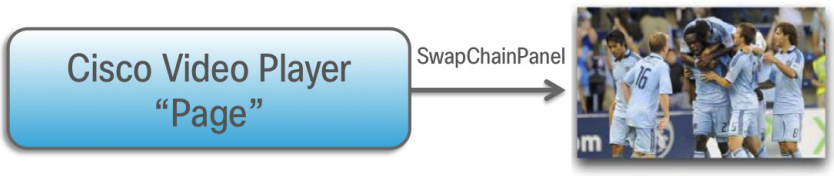
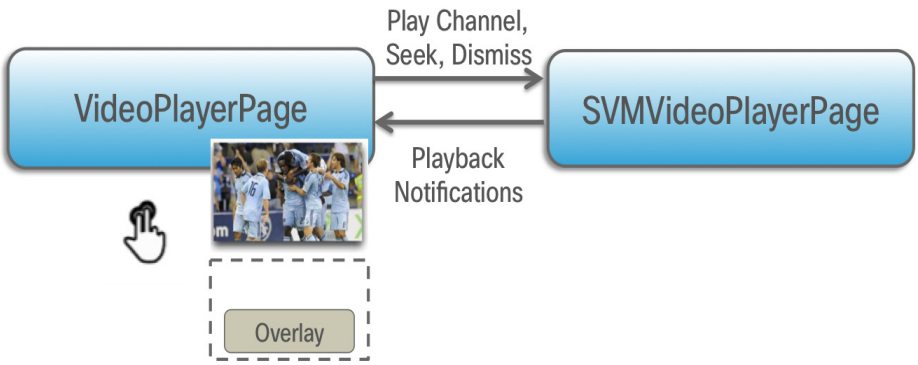


Figure 4-7 SVMVideoPlayerPage API



Cisco Demo Video Player

The Cisco demo video player:

- Implemented as "VideoPlayerPage."
- Extends the "SVMVideoPlayerPage" class.
- Handles all video overlays and gestures.
- Uses standard Windows XAML layout files.

The video player's XAML layout file defines:

- The "SwapChainPanel" video rendering area.
- Any transparent video overlays.
- Play/Pause/Rewind button graphic files.
- Animations used to show/hide the transport controller (splash screen).

The customized video play extends the "SVMVideoPlayerPage" base class, as shown below:

```
   SVMVideoPlayerPage objVpa;
objVpa = new SVMVideoPlayerPage(currentChannel);
```

Video Channels

This section describes the Cisco StadiumVision Mobile SDK video channels and contains the following sections:

Getting the Video Channel List

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

```
using SvmSdk;
// 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 4-13.

Table 4-13 SVMChannel Object Properties

SVMChannel Property	Property Description
appDeveloper	Name of the application developer.
bandwidthKbps	Data bandwidth consumed by the channel (in kbps).
bodyText	Complete text description of the video channel.
channelType	Type of the channel.
contentOwner	Name of the content owner.
name	Name of the channel.
sessionNum	Session number of the channel.
venueName	Name of the venue.

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 page ("VideoPlayer").

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):

```
// rewind video playback 20 seconds
objVpa.seekRelative(-20000);
```

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

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

Setting the Video Dimensions

The video region is rendered within a SwapChainPanel. The video region is configured using standard Windows layout XAML techniques.

Data Channels

This section describes the Cisco StadiumVision Mobile SDK data channels and contains the following sections:

- Getting the Data Channel List, page 4-27
- Observing a Data Channel, page 4-27

Getting the Data Channel List

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

```
using SvmSdk;

// get the list of available data channels
SVMChannel[] channels = StadiumVisionMobile.objSVM.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 an event to receive callbacks for all data received on that channel. The registered event needs to implement the "ISVMDataObserver" interface, as shown in the following example:

```
using SvmSdk;

    // register to receive data from the given data channel
    StadiumVisionMobile.objSVM.nativeAPI.dataAvailableEvent += onDataHandler;
```

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

```
/**
  * This method is the implementation of the ISVMDataObserver interface.
  * The latest received data for the given 'channelName' is forwarded to
  * this method from a Windows event class, so that this method can
  * safely update the WP8 UI
  */

private void onDataHandler(object sender, NativePlayer.DataAvailableEventArgs e)
{
    // UI update
    onData(e.channelName, e.dataBytes);
}
public async void onData(String channelName, byte[] data)
{
    // publish the incremental update in UI thread
    await
CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, ()
=>
            {
                // log the received data parameters
                Log.d(TAG, "DATA CALLBACK: channelName = " + channelName + ", data length
= " + data.Length);
            });
        }
```

EVS C-Cast Integration

Note Cisco StadiumVision Mobile is supported with EVS C-Cast version 2.x only. EVS C-Cast version 3.x is not supported.

The steps below describe a high level workflow of how a Cisco StadiumVision Mobile powered C-Cast app gains access to the XML timeline and media files. Variations are possible, for instance the file list can be polled every few seconds via **StadiumVisionMobile.objSVM.getFileChannelArray** instead of getting an event as described in Step 2 below.

1. Register an event to be notified when a C-Cast file channel becomes available. For example: **StadiumVisionMobile.objSVM.nativeAPI.fileAvailableEvent += onFileHandler;**
2. Register an event to be notified when the media data file becomes available. For example: **StadiumVisionMobile.objSVM.nativeAPI.dataAvailableEvent += onDataHandler;**
3. Handle the file reception (movies/thumbnails/timeline).
4. Check to see if a file channel is already available using **StadiumVisionMobile.objSVM.getFileChannelArray**
5. If a channel is already available or when a callback notification is received, register a file channel event handler.
6. Wait for the timeline to arrive via multicast on the data channel. At the same time, other files may arrive on the file channel.
   - Each time a new file arrives, perform a corresponding check to see if the new file is in the timeline.
   - If the timeline is not yet available, wait for additional files to arrive.
7. Once the timeline data has been received, parse it using the steps in chapter 5 (How to build the media path) of the C-Cast API spec, and then build the media path for all media files. Contact James Stellphlug (j.stellpflug@evs.com) to obtain the C-Cast API documentation.
8. For each file media path, remove the path prefix so that only the filename remains. For example: http://www.mydomain.com/videos/abc/def/ghi/abcdefghijklmnopqrstuvwxyz123456_hls-ipad.m3u8 becomes abcdefghijklmnopqrstuvwxyz123456_hls-ipad.m3u8
9. For each filename, cycle through until all files have been received.
10. Be prepared for the ccast-timeline.xml file to change at any time and repeat steps 6-8 whenever it changes.