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

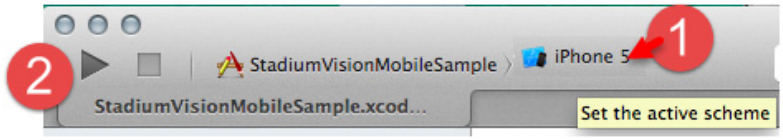
Step 1 Launch **Xcode**.

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

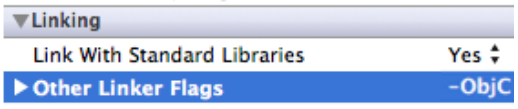
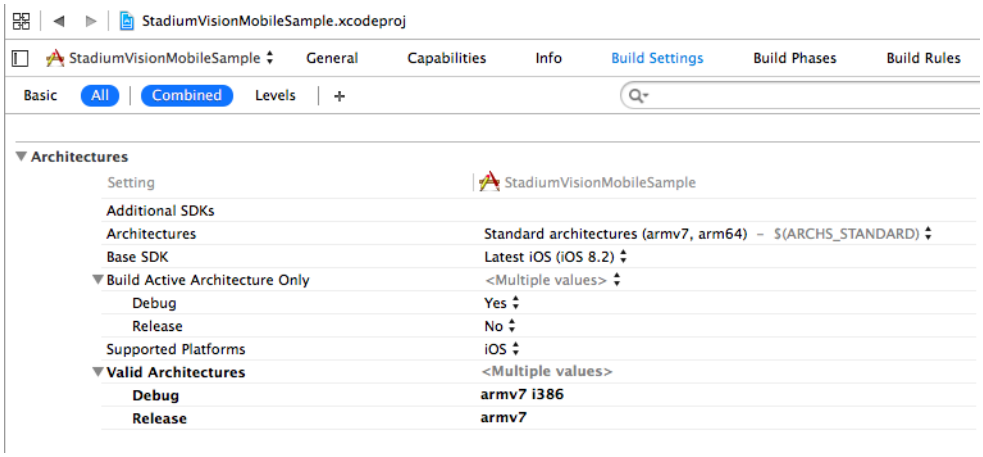


Figure 2-4 shows the Xcode build settings that apply to both the project and target settings.

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

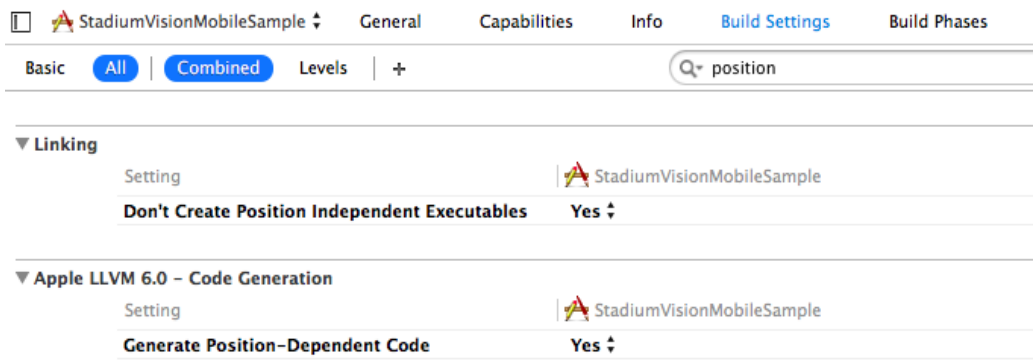
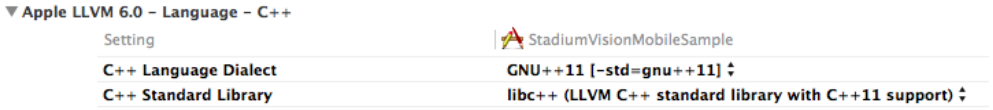


Figure 2-6 shows the Apple LLVM language settings.

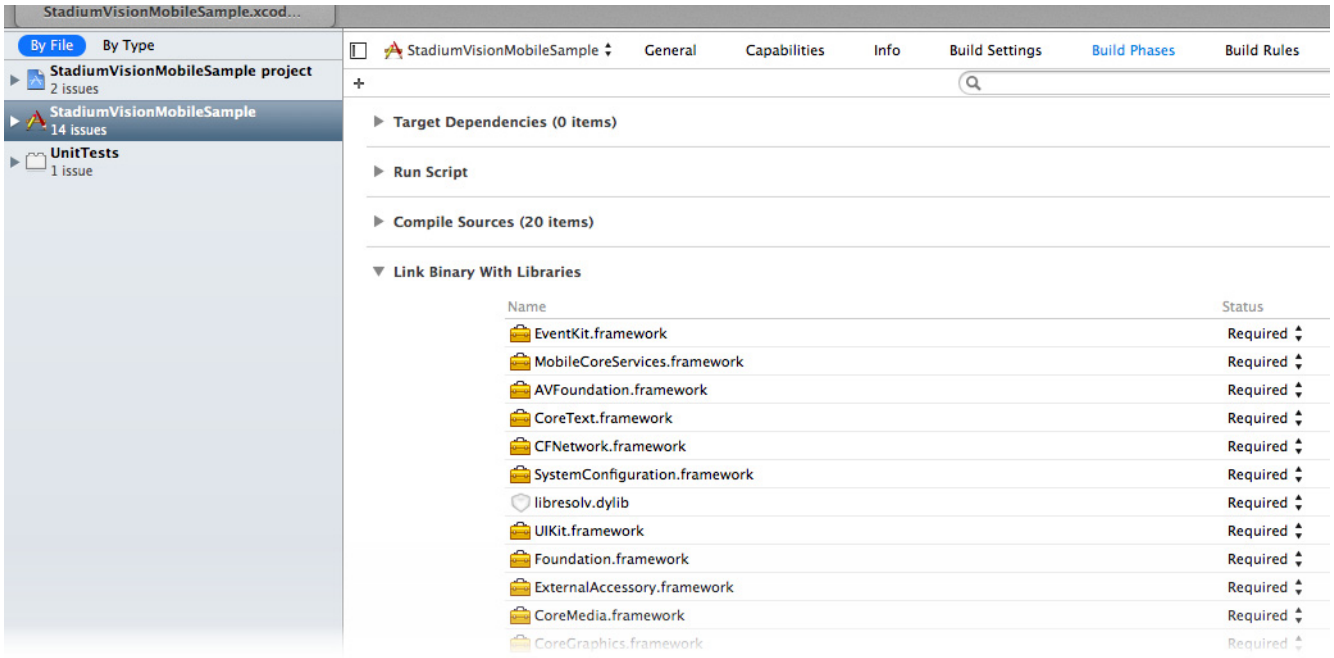
Figure 2-6 Xcode Build Setting—Specify Apple LLVM 6.0 - Language C++



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

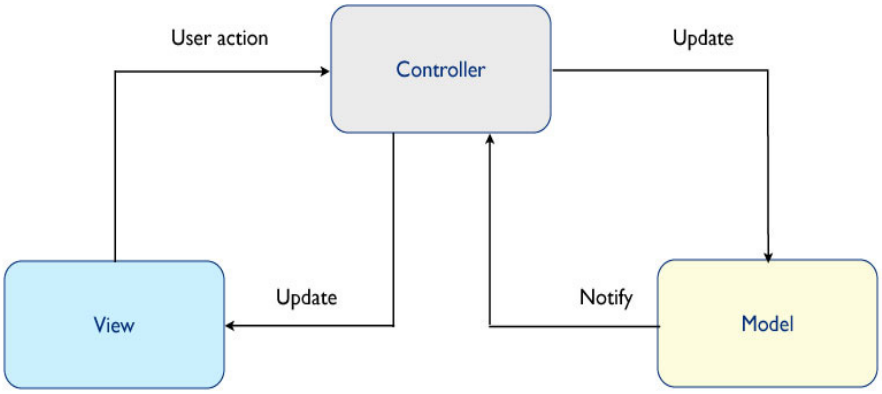
Figure 2-8 illustrates the high-level view of the Cisco StadiumVision iOS API libraries and common framework components. The left side of the graphic represents how to modify the sample application, and the right represents how the SDK is packaged.

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.

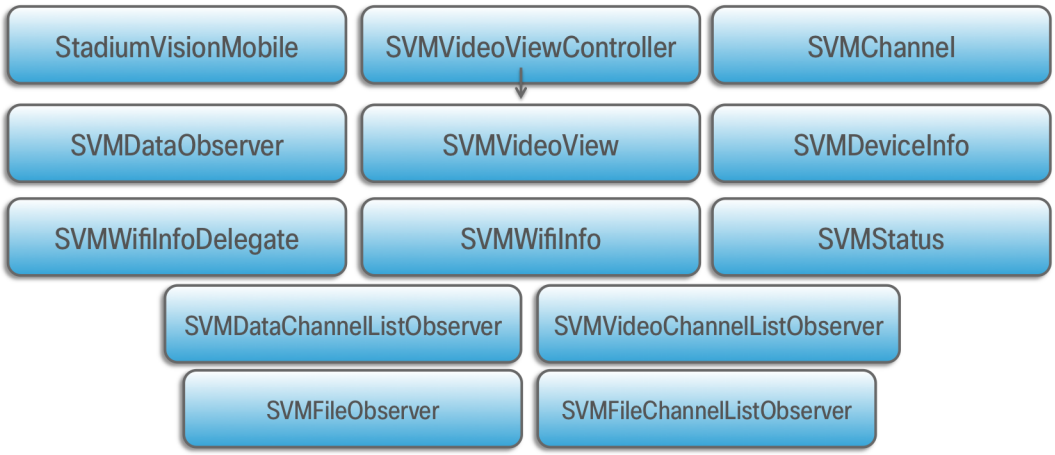
Figure 2-9 MVC Design Pattern



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.

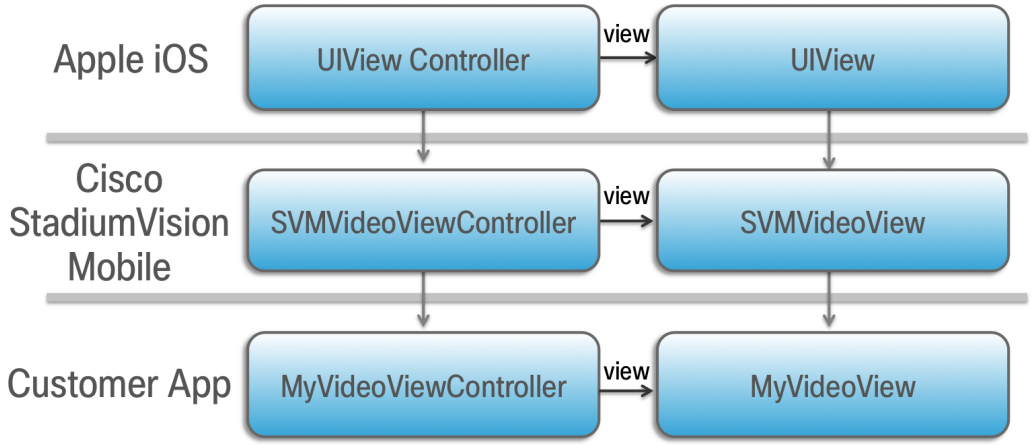
Figure 2-10 Cisco StadiumVision Mobile iOS API Classes



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

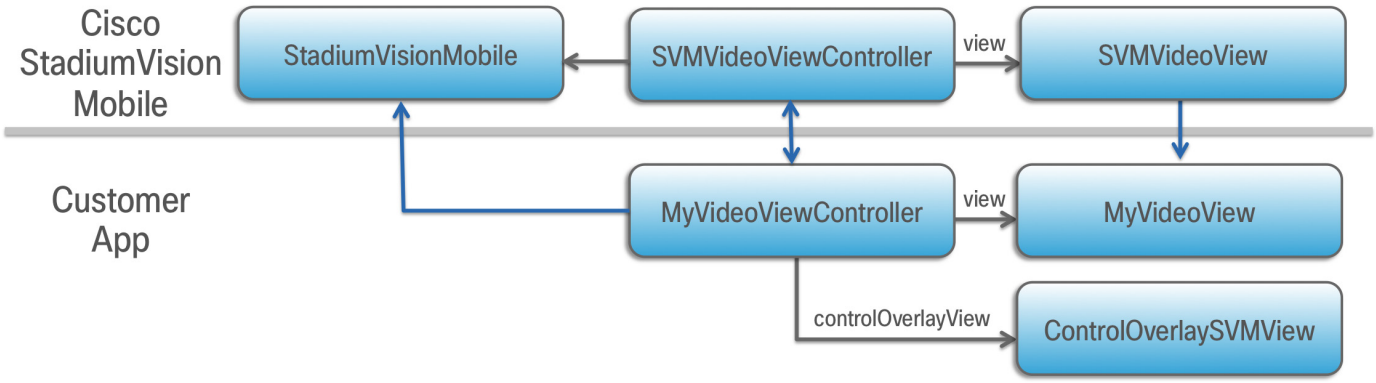


Cisco StadiumVision Mobile Application Classes

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

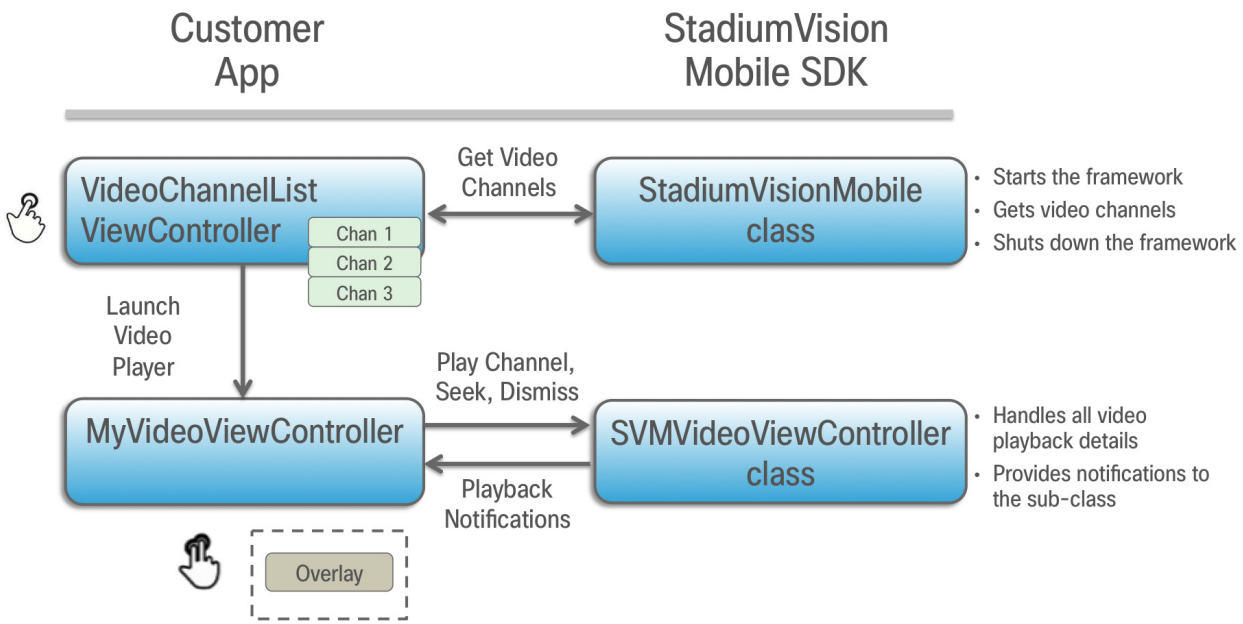


Customer Application Roles

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.

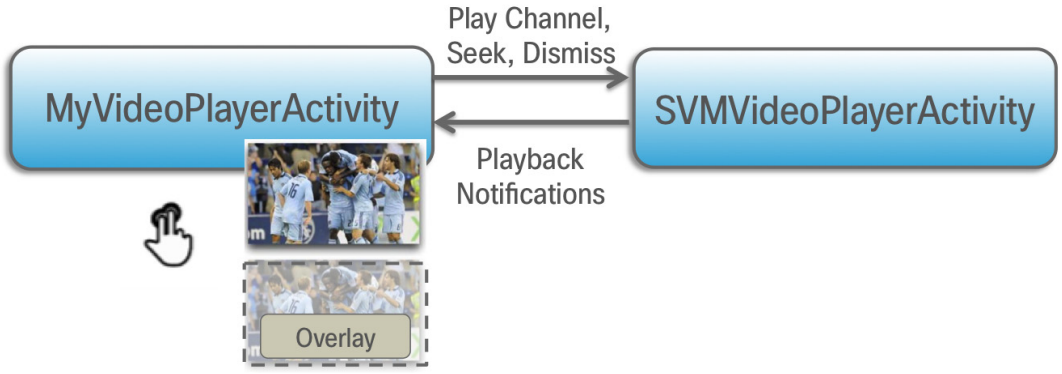
Customized Video Player

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.