CHAPTER 2

Cisco StadiumVision Mobile API for Google Android

Revised: October 31, 2013

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

New Features in Cisco StadiumVision Mobile Release 1.3 Android SDK

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

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

New Features in Release 1.3

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

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

Introduction to Cisco StadiumVision Mobile API for Google Android

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

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

This document contains the following sections:

Android API Prerequisites

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

Table 2-1 Build Environment Requirements

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

Getting Started With The Android Demo App

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

Install the tools

Step 1 Download the Android Developer Tools ADT.

Step 2 Follow these instructions to set up ADT on your computer.

Step 3 Launch the Eclipse application and when prompted select a folder to use as your workspace.

Step 4 Launch the Android SDK Manager from the Window dropdown menu.

Step 5 Open the Android 2.2 (API 8) folder and check the SDK Platform box. Uncheck everything else and then install the selected package.

Build the app

Step 1 Download the StadiumVisionMobileSample-Android-xxxx.tar.bz2 SDK and demo app package.

Step 2 Extract the downloaded package into a directory.

Step 3 Import the demo app project into Eclipse as follows:

a. In Eclipse go to File=>Import

b. Then go to General=>Existing Projects into Workspace, and select Next.

c. Set the Select root directory to the folder where you unpacked the SDK, and then click Finish.

d. Restart Eclipse from File=>Restart.

Step 4 Right click on CiscoStadiumVisionMobile in the left Package Explorer window, and select Android Tools=>Export Signed Application Package.

Step 5 Click Next when the Project Checks window appears.

Step 6 Select Create new keystore, then browse to a folder where you wish to store the key store file. Click Next.

Step 7 Fill in the Key Creation form. There are no right or wrong answers. Click Next.

Step 8 Browse to the folder where you wish to place the apk file, then click Finish.

Step 9 Download the apk file to your Android device by placing it on a web server, emailing it, SD card, USB flash key, etc.

Step 10 Now install the apk on your device.

Customize the app

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

Change the text for the app icon:

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

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

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

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

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

Android SDK Overview

The Cisco StadiumVision Mobile Android SDK contains the following components:

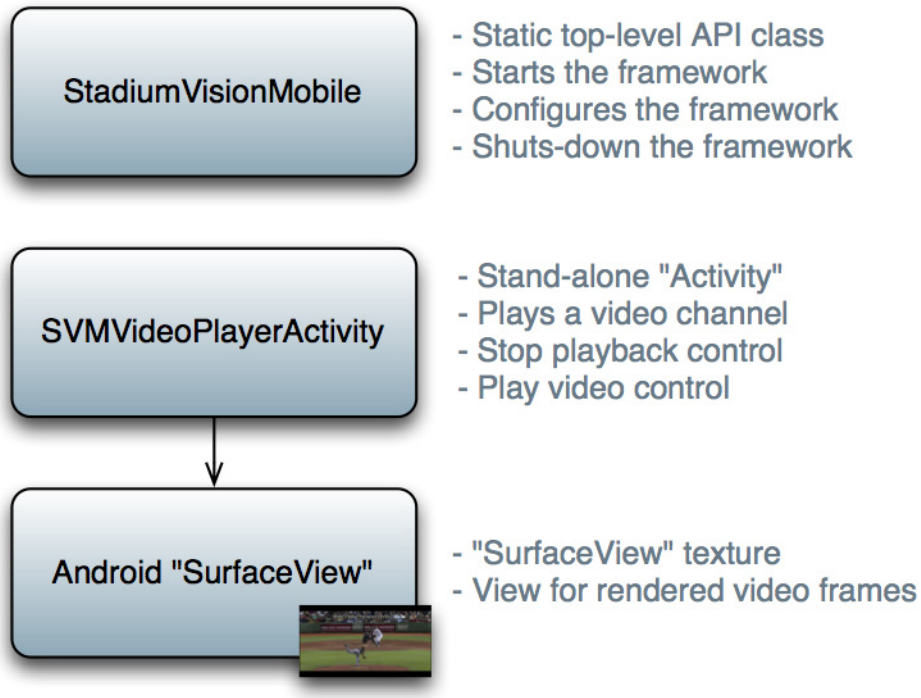
- A set of static libraries, configuration files, player layout XML files, and a sample Android application.
- Customizable video player

Cisco StadiumVision Mobile iOS API Class Overview

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

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

Figure 2-1 StadiumVision Mobile Class



Android OS Activity Overview

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

Figure 2-2 Android Activity Overview

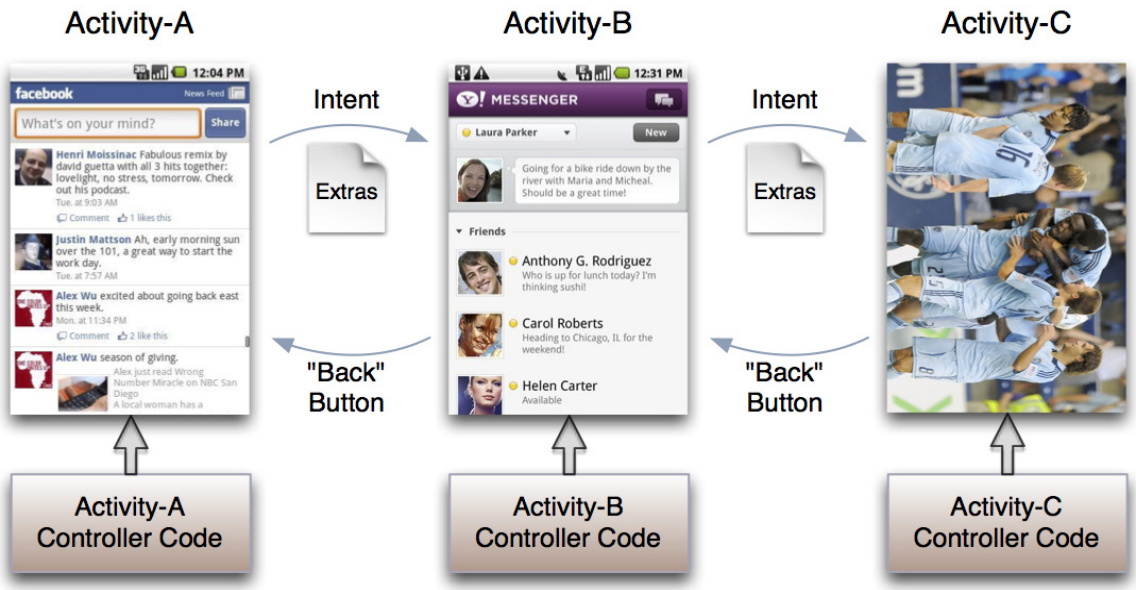
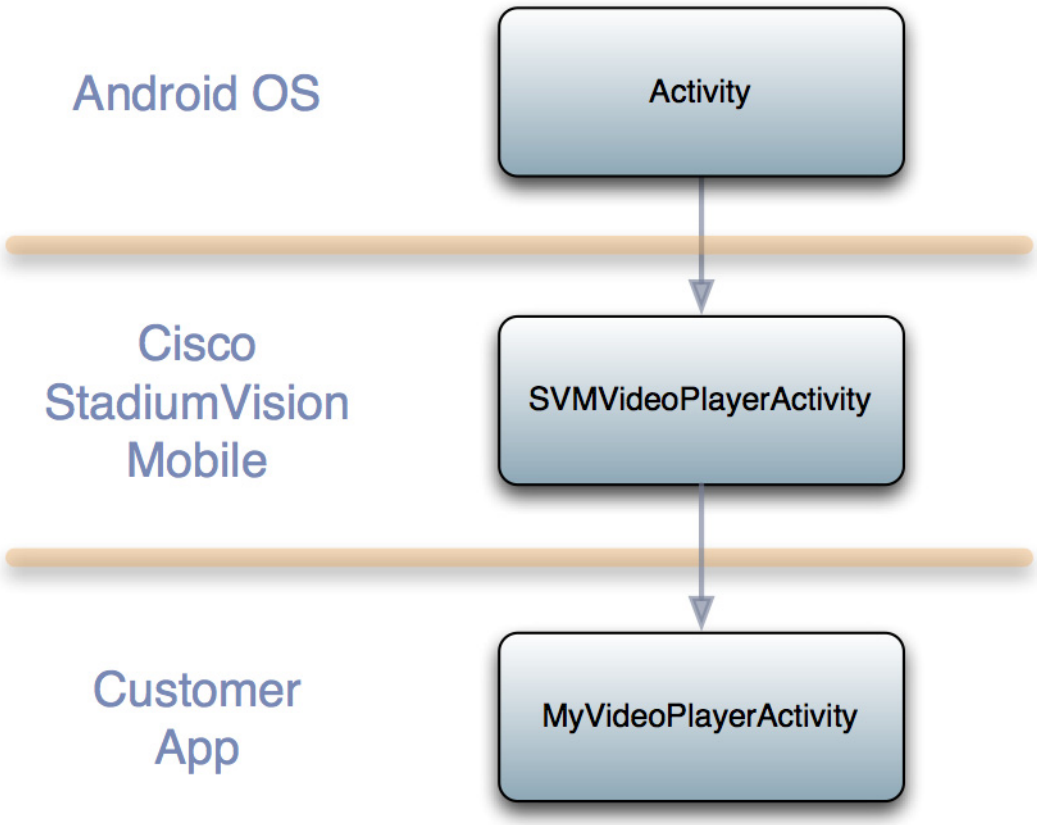


Figure 2-3 depicts the Activity inheritance between the Android OS, Cisco StadiumVision Mobile, and the client application.

Figure 2-3 Android Video Player Activity Inheritance



Cisco StadiumVision Mobile Android API Summary

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

Table 2-2 Cisco StadiumVision Mobile Android API Summary

Return Type	API Method Name	API Method Description
SVMStatus	start	Starts the StadiumVision Mobile SDK
SVMChannel[]	getVideoChannelArray	Get the array of available video channels
ArrayList<SVMChannel>	getVideoChannelArrayList	Get the array list of available video channels
SVMChannel[]	getDataChannelArray	Get the array of available data channels
ArrayList<SVMChannel>	getDataChannelArrayList	Get the array list of available data channels
SVMStatus	addDataChannelObserver	Registers an observer class to receive data for a particular data channel
SVMStatus	removeDataChannelObserver	Unregisters an observer class from receiving data for a particular data channel

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

Cisco StadiumVision Mobile Android API

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

Return Status Object

Each API call returns an 'SVMStatus' object whenever applicable. Table 2-3 lists the SVMStatus object fields. This section contains the following API calls and tables:

- Start
- getVideoChannelArray
- getDataChannelArrayList
- addDataChannelObserver
- removeDataChannelObserver
- getStats
- getStats API Hash Keys and Stats Description
- onPause
- onResume
- getWifiInfo
- getBatteryInfo

- getLogLevelArray
- getLogLevelArrayList
- setLogLevel
- getLocalIpAddress
- getDeviceUUID
- getAppSessionUUID
- sdkVersion

Table 2-3 SVMStatus Object

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

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

Table 2-4 Start

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

Table 2-5 getVideoChannelArray

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

Table 2-6 getDataChannelArrayList

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

Table 2-7 addDataChannelObserver

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

Figure 2-4 onData

Method Signature	`public void onData(String channelName, byte[] data)`
Prerequisites	N/A
Notes	This method is implemented by the customer app and is used as a callback from the StadiumVision Mobile SDK. Each callback from the SDK to the customer app provides a received data message on the given data channel. The data channel message is delivered as a byte array.
Result	N/A

Table 2-8 removeDataChannelObserver

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

Table 2-9 getStats

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

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

Table 2-10 getStats API Hash Keys and Stats Description

Stats Hash Key	Stats Description
session_link_indicator	The health of the WiFi network connection. Ranges from 0 (poor) to 10 (excellent)
session_uptime	The length of time the session has been active (in seconds)
announcement_session_id	The video session announcement ID
announcement_session_title	The session announcement name
total_num_bytes_written	The total number of video bytes played
num_ts_discontinuities	The total number of MPEG2-TS packet discontinuities
num_dropped_video_frames	The total number of video frames dropped
protection_windows	The total number of protection windows sent
window_no_loss	The total number of protection windows with no dropped video packets
window_recovery_successes	The total number of protection windows with recovered video packets
window_recovery_failures	The total number of protection windows that could not recover dropped packets. Recovery failure occurs when the number of received repair packets is less than the number of dropped video packets

Stats Hash Key	Stats Description
window_warning	The total number of protection windows with more packets per window than the recommended value
window_error	The total number of protection windows with more packets per window than can be supported by StadiumVision Mobile.

Table 2-11 onPause

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

Table 2-12 onResume

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

Table 2-13 getWifiInfo

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

Table 2-14 getBatteryInfo

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

Table 2-15 getLogLevelArray

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

Table 2-16 getLogLevelArrayList

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

Table 2-17 setLogLevel

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

Table 2-18 getLocalIpAddress

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

Table 2-19 getDeviceUUID

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

Table 2-20 getAppSessionUUID

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

Table 2-21 sdkVersion

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

Video Player Activity API Summary

The SVMVideoPlayerActivity class can be extended and customized. Table 2-22 lists the SVMVideoPlayerActivity API methods, and contains the following tables:

- setVideoSurfaceView
- seekRelative
- seekAbsolute
- rewindForDuration
- playLive
- shutdown
- setConfig

- setConfigWithString
- getConfig
- getStreamerArray
- getStreamerArrayList

Table 2-22 Video Player Activity API Summary

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

Table 2-23 setVideoSurfaceView

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

Table 2-24 seekRelative

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

Table 2-25 seekAbsolute

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

Table 2-26 rewindForDuration

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

Table 2-27 playLive

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

Table 2-28 shutdown

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

Table 2-29 setConfig

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

Table 2-30 setConfigWithString

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

Table 2-31 getConfig

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

Table 2-32 getStreamerArray

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

Table 2-33 getStreamerArrayList

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

SDK Workflow

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

Starting the SDK

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

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

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

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

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

Getting the Video Channel List

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

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

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

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

Presenting the Video Channel List

Each "SVMChannel" video channel object contains all of the information needed to display the channel list to the user. The SVMChannelObject properties and descriptions are shown in Table 2-34.

Table 2-34 SVMChannel Object Properties

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

Playing a Video Channel

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

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

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

Seeking Within the Video Buffer

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

```
public class MyVideoPlayerActivity extends SVMVideoPlayerActivity {

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

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

```
public class MyVideoPlayerActivity extends SVMVideoPlayerActivity {

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

Setting the Video Dimensions

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

Fullscreen Video Layout

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

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

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

</RelativeLayout>
```

Partial-Screen Video Layout

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

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

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

</RelativeLayout>
```

Getting the Data Channel List

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

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

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

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

Observing a Data Channel

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

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

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

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

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

}
```

Activity Life-Cycle Notifications

The client app needs to notify the StadiumVision Mobile SDK of 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();
```

StadiumVision Mobile Service Up or Down Indicator

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

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

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

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

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

Table 2-35 Service Down Notifications

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

Receiving "Service Up / Down" Notifications

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

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

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

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

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

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

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

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

```
    }
    }
};

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

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

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

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

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

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

In-Venue Detection

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

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

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

Receiving "In-Venue Detection" Notifications

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

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

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

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

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

```
        }
    }
};

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

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

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

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

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

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

Set the SDK Configuration at Run-Time

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

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

- "setConfig"

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

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

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

- "setConfigWithString"

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

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

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

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

Get the SDK Configuration

"getConfig" API Method#

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

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

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

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

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

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

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

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

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

setConfigWithString API Method

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

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

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

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

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

Get the Available Streamer Servers

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

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

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

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

Each "SVMStreamer" object contains the following properties listed in Table 2-36.

Table 2-36 "SVMStreamer" Object Properties

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

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

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

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

Additional Statistics

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

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

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

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

The new stats and their associate dictionary keys and description are given in Table 2-37.

Table 2-37 Stats API Dictionary Keys

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

Video Player State Notifications

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

Table 2-38 Video Player State Notification

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

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

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

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

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

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

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

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

Video Player "Channel Inactive" Callback

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

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

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

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

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

   // shutdown video playback
   shutdown();

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

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

Video Player Customization

This section describes customizing the video player.

The default Cisco video player has the following features:

- Implemented as a separate Android “Activity”
- Supports fullscreen and partial-screen video views
- Renders video frames using an Android “SurfaceView”
- Customizable by extending the “SVMVideoPlayerActivity” class
- Uses a customized video player

Figure 2-5 Default Cisco Video Player

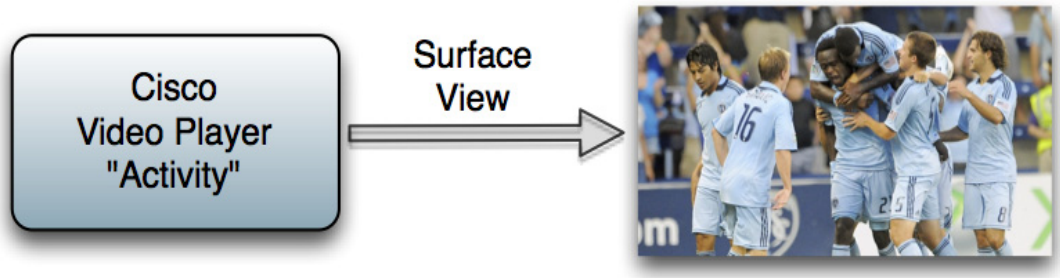
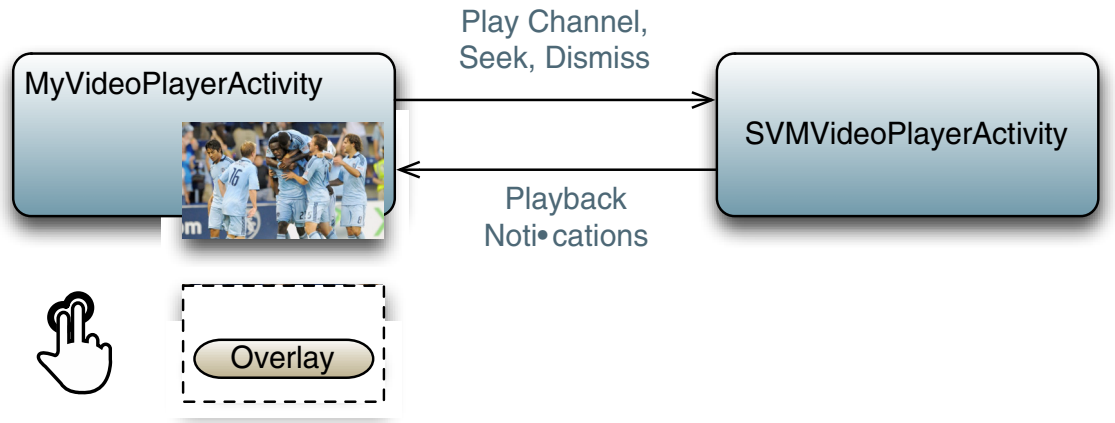


Figure 2-6 SVMVideoPlayerActivity API



Cisco Demo Customized Video Player

The Cisco demo video player:

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

The video player's XML layout file defines:

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

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

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

public class MyVideoPlayer extends SVMVideoPlayerActivity {
}
```

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

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

Configuration

The following section describes the required configuration.

Configuration Files

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

Table 2-39 Configuration Files

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

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

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

WiFi AP Info Configuration (Optional)

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

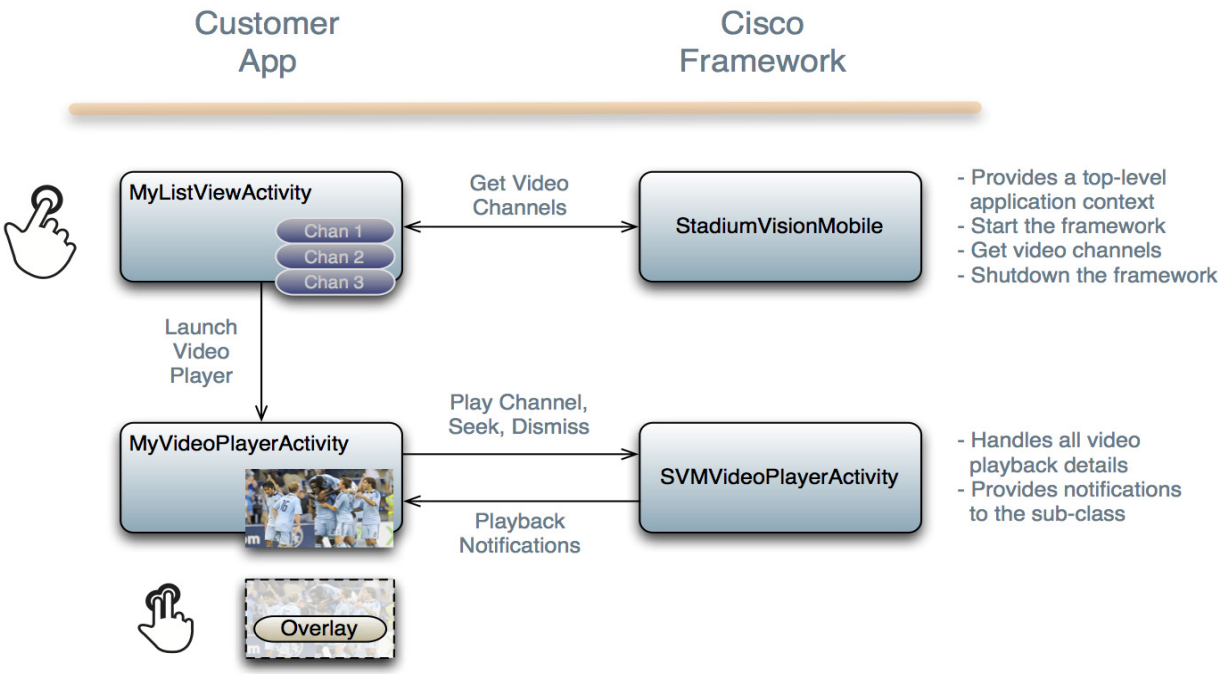
```
{
    "network": {
        "wifiApInfo": [
            {
                "name": "Press Box Booth 5",
                "bssid": "04:C5:A4:09:55:70"
            }
        ]
    }
}
```

Integration

Client Application Integration Overview

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

Figure 2-7 Cisco StadiumVision Mobile Integration Overview



Integration Checklist

1. Supported Android OS Version
   - Set the app's Android version target to v2.1u1 or above
2. Android App Permissions
   - Add the required permissions to "AndroidManifest.xml"
3. Copy Config Files
   - Add the config files to the app's "assets" folder
4. Copy Libraries

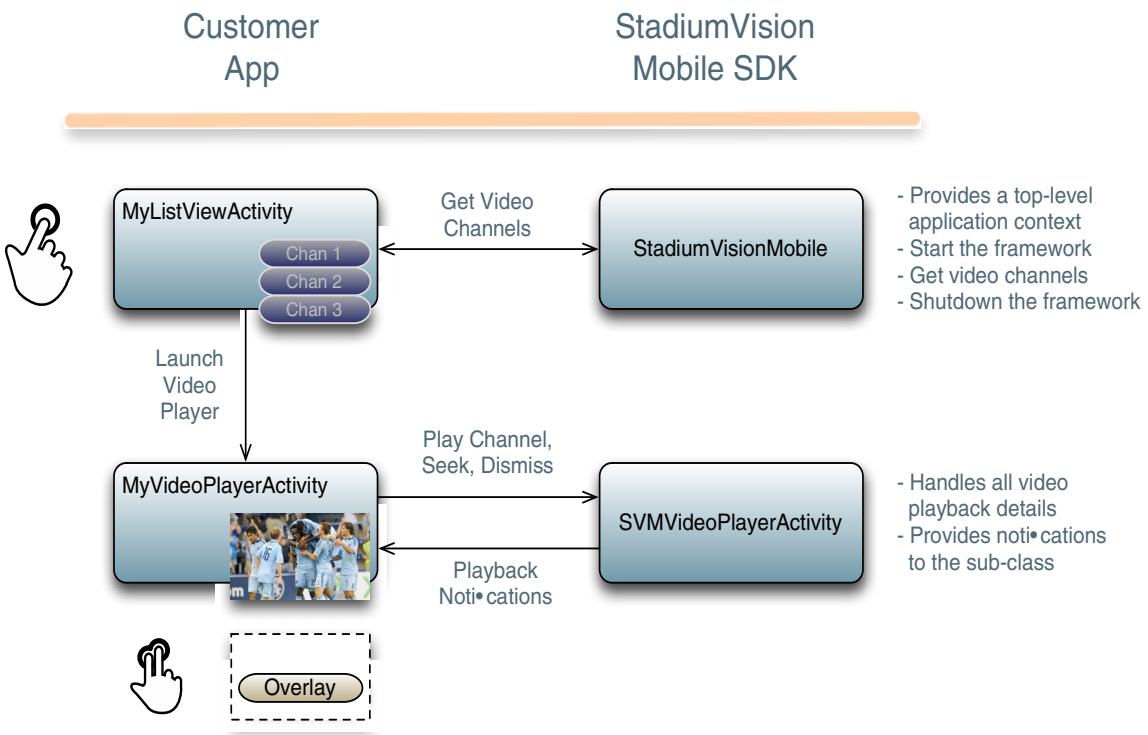
  - Add the Java and native libraries to the app's "libs" folder
5. Set a Video "SurfaceView"
  - Add a "SurfaceView" to the player Activity's layout XML file
6. Life-Cycle Notifications
  - Forward life-cycle notifications to the StadiumVision Mobile SDK
7. Android Project Build Paths
  - Set the project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

Customer Application Roles

Figure 2-8 illustrates the roles of the customer application. The application must specify:

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

Figure 2-8 Customer Application Responsibilities



Android Permissions

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

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

SDK Java Libraries

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

- Cisco StadiumVision Mobile Android SDK
- Apache HTTP Client 4.1
- Jackson JSON 1.8.1

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

SDK Native Libraries

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

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

Android Project Classpath

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

Step 1 Right-click your project in Eclipse

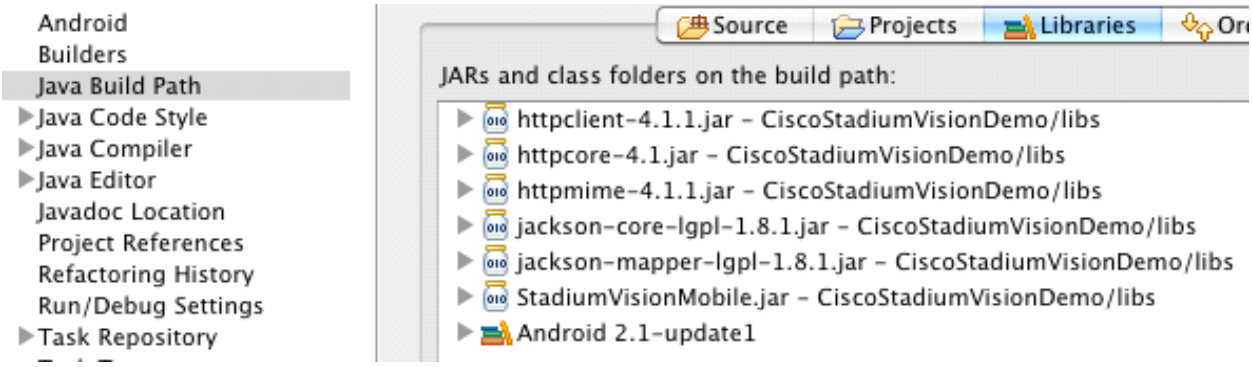
Step 2 Select "Properties"

Step 3 Select "Java Build Properties"

Step 4 Select "Add JARs"

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

Figure 2-9 Adding Java JAR Files in Eclipse



Your "classpath" file should look like the following example:

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

App Obfuscation Using ProGuard

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

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

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

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

```
-keep public class * extends com.cisco.svm.data.ISVMDataObserver
```

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


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

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

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

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

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

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

Channel ListView Activity Example

The following example illustrates the following actions:

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

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

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