CHAPTER 3

Cisco StadiumVision Mobile API for Google Android

First Published: May 26, 2015

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

Introduction to Cisco StadiumVision Mobile SDK for Android

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

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

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

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

Table 3-1 Mobile OS Support

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

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

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

Cisco StadiumVision Mobile and Android Developer Tools

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

Table 3-2 Build Environment Requirements

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

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

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

Requirements

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

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

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

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

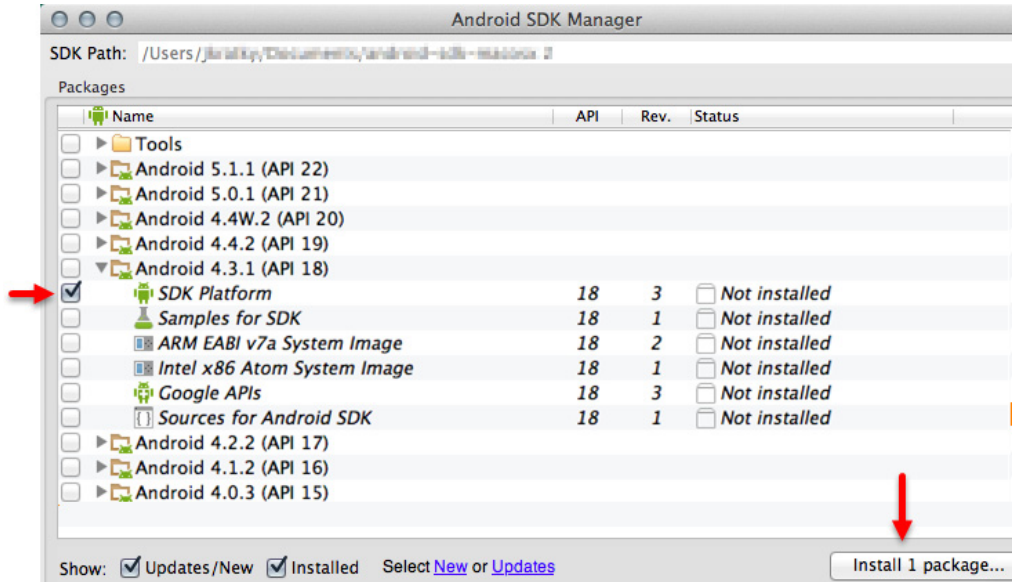
- Download and install the Android Developer Tools (ADT) plug-in available at:

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

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

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

Figure 3-1 Selecting the SDK Platform Box



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

Download and Unpack the SDK

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

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

Table 3-3 Cisco StadiumVision Mobile SDK File Content

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

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

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

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

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

Getting Started with the Android Demo App

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

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

Compile the Demo App

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

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

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

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

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

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

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

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

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

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

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

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

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

Customize the Demo App

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

- Change the text for the app icon:
  In the file "res/values/strings.xml" change "SVM Demo" to "My SVM App"
- Change the name space so that your custom app can be installed side-by-side with the out-of-the-box demo app:
  Edit the file "AndroidManifest.xml"
  - Change "package="com.cisco.sv"" to "package="com.cisco.svm.foo""
  - Change "android:name="com.cisco.svm.app.StadiumVisionMobile"" to "android:name="com.cisco.svm.foo""

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

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

Embed the Cisco StadiumVision Mobile SDK in an Existing App

Integration Checklist

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

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

Android Permissions

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

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

SDK Java Libraries

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

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

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

SDK Native Libraries

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

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

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

Android Project Classpath

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

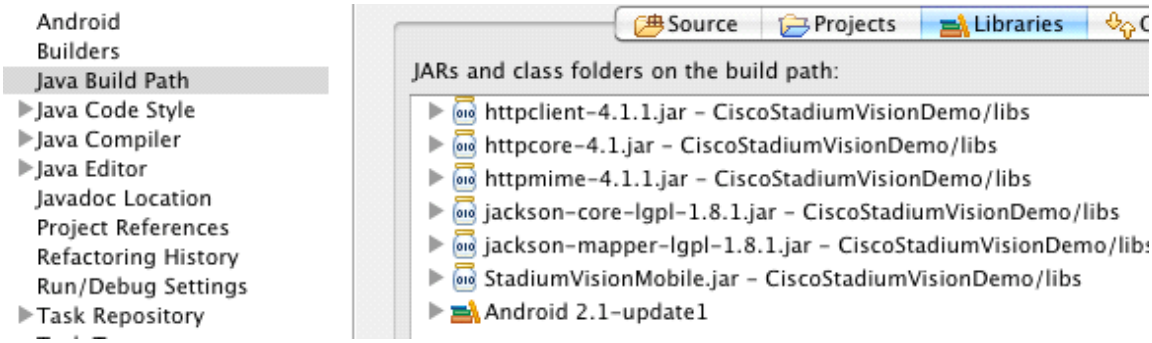
Step 1 Right-click your project in Eclipse.

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

Step 3 Select **Add JARs**.

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

Figure 3-2 Adding Java JAR Files in Eclipse



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

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
    <classpathentry kind="src" path="src"/>
    <classpathentry kind="src" path="gen"/>
```

```
    <classpathentry kind="con" path="com.android.ide.eclipse.adt.ANDROID_FRAMEWORK"/>
    <classpathentry kind="lib" path="libs/httpclient-4.1.1.jar"/>
    <classpathentry kind="lib" path="libs/httpcore-4.1.jar"/>
    <classpathentry kind="lib" path="libs/httpmime-4.1.1.jar"/>
    <classpathentry kind="lib" path="libs/jackson-core-lgpl-1.8.1.jar"/>
    <classpathentry kind="lib" path="libs/jackson-mapper-lgpl-1.8.1.jar"/>
    <classpathentry kind="lib" path="libs/StadiumVisionMobile.jar"/>
    <classpathentry kind="output" path="bin"/>
</classpath>
```

App Obfuscation Using ProGuard

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

- Use the latest version of ProGuard (which is version 5.2 as of April, 2015)
- If a crash takes place that you would like Cisco to analyze, please run retrace.jar on the stack trace output with your map file before sending us the un-winded stack trace file.
- Specify our libraries as input jars with "-libraryjars". See the example below and remember to modify the paths as needed:

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

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

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


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

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

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

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

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

-keepclasseswithmembernames class * {
    native <methods>;
```

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

Channel ListView Activity Example

The following example illustrates the following actions:

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

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

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

Configuration Files

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

Table 3-4 Configuration Files

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

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

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

Wi-Fi AP Info Configuration (Optional)

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

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

How Cisco StadiumVision Mobile Fits into the Android Framework

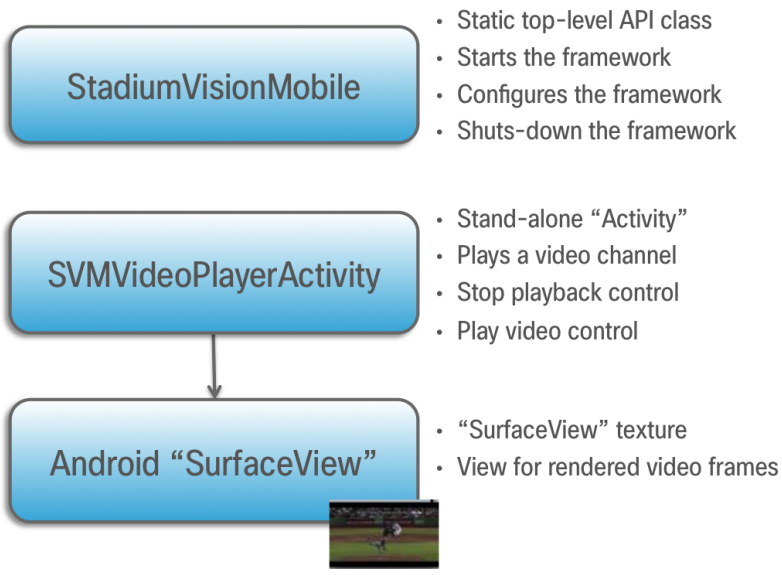
This section describes how SVM fits into the Android Framework, and contains the following sections:

Android API Class Overview

Figure 3-3 describes the three main Android API classes used in Cisco StadiumVision Mobile. The top-level StadiumVisionMobile class acts as a custom Android application context. An application context is a global structure created within the current process. It is tied to the lifetime of the process rather than the current component.

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

Figure 3-3 StadiumVision Mobile Class



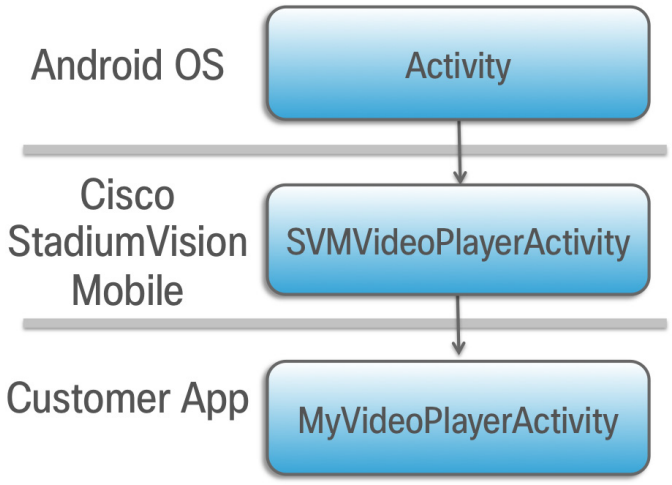
Android OS Activity Overview

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

Figure 3-4 Android Activity Overview

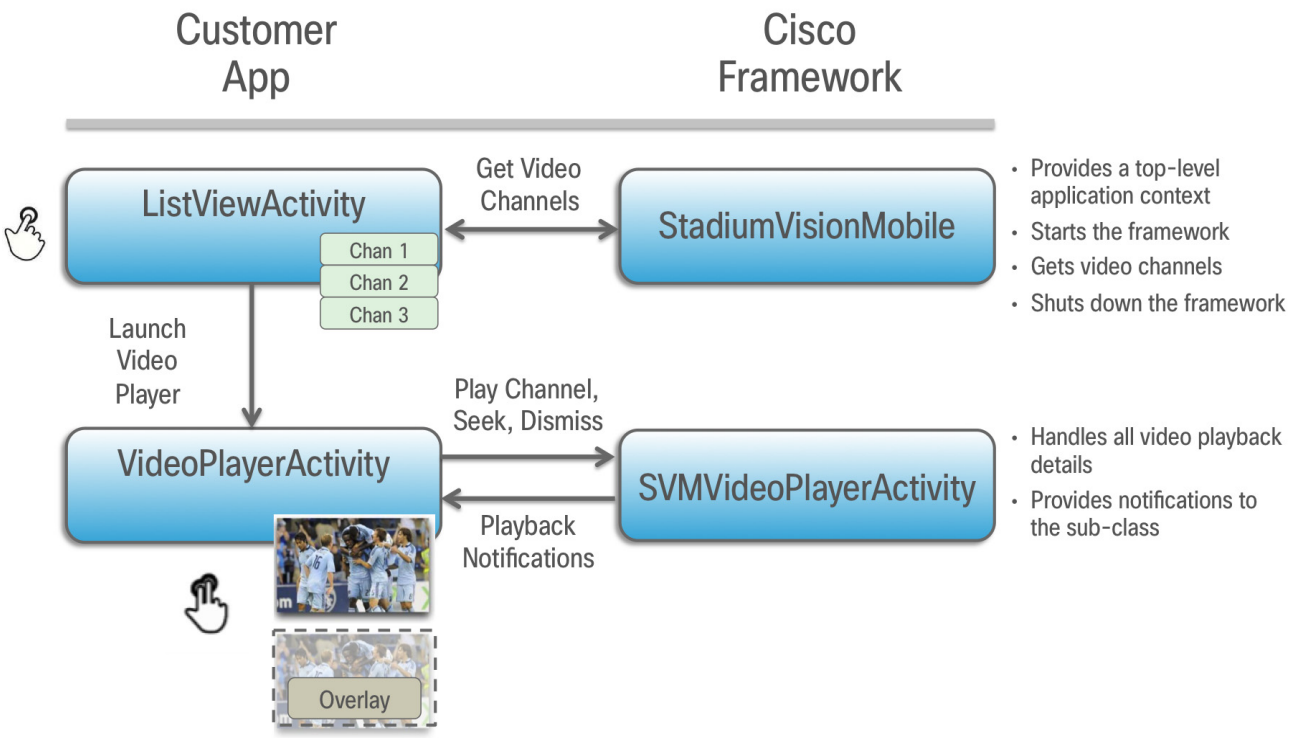
Figure 3-5 depicts the Activity inheritance between the Android OS, Cisco StadiumVision Mobile, and the client application.

Figure 3-5 Android Video Player Activity Inheritance



Client Application Integration Overview

Figure 3-6 Cisco StadiumVision Mobile Integration Overview

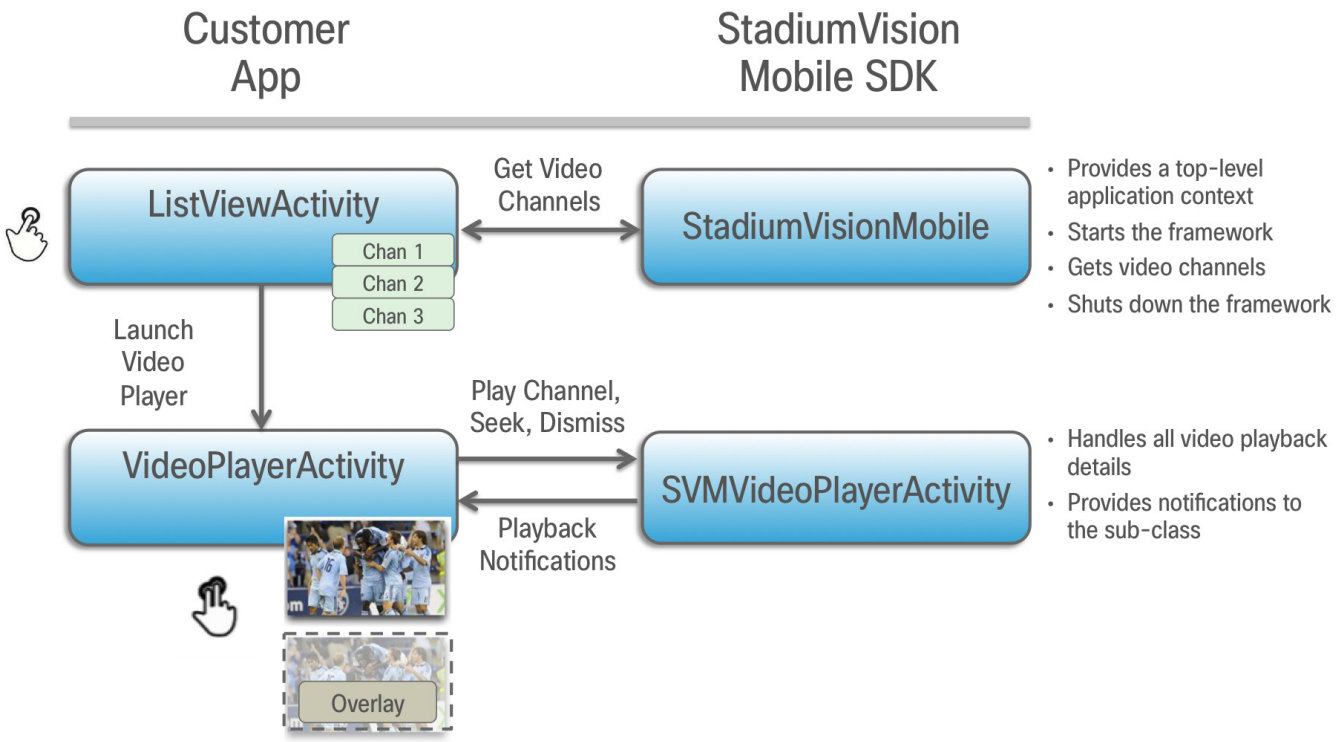


Customer Application Roles

Figure 3-7 illustrates the roles of the customer application. The application must specify:

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

Figure 3-7 Customer Application Responsibilities



Cisco StadiumVision Mobile Methods and Functions for Android

Cisco StadiumVision Mobile Android API Summary

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

Table 3-5 Cisco StadiumVision Mobile Android API Summary

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

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

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

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

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

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

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

Return Status Object

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

Table 3-6 SVMStatus Object

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

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

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

Table 3-7 getStats API Hash Keys and Description

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

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

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

Video Player Activity API Summary

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

Table 3-8 Video Player Activity API Summary

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

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

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

Start the SDK

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

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

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

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

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

Notify Life-Cycle Activity

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

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

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

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

Indicate StadiumVision Mobile Service: Up or Down

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

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

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

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

When the app receives the "Service Down" notification, the SDK will supply a bitmap containing the reasons why the service was declared down by the SDK. The reasons bitmap is given in Table 3-9.

Table 3-9 Service Down Notifications

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

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

Receiving "Service Up/Down" Notifications

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

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

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

// create the service state broadcast receiver
serviceStateReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
```

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

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

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

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

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

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

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

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

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

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

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

Detect Mobile Device Connection

Beginning in Cisco StadiumVision Mobile Release 2.0, the SDK provides a mechanism to detect whether the mobile device is connected within the SVM-enabled venue or not.

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

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

Receiving "In-Venue Detection" Notifications

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

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

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

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

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

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

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

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

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

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

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

Set the SDK Configuration at Run-Time

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

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

- "setConfig"

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

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

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

- "setConfigWithString"

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

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

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

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

Scalable File Distribution

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

Table 3-10 Scalable File Distribution and Service API Summary

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

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

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

Data Channels

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

Table 3-11 Data Distribution and Service API Summary

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

Get the SDK Configuration

"getConfig" API Method

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

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

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

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

// get the license dictionary from the config dictionary
```

```
JSONObject licenseObj = null;
try {
   licenseObj = configObj.getJSONObject("license");
} catch (JSONException e) {
   e.printStackTrace();
}

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

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

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

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

Set SDK Configuration using setConfigWithString API Method

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

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

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

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

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

Get the Available Streamer Servers

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

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

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

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

Each "SVMStreamer" object contains the following properties listed in Table 3-12.

Table 3-12 SVMStreamer Object Properties

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

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

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

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

Obtain Additional Statistics

In the Cisco StadiumVision Mobile Release 2.0 SDK, the existing "stats" API call returns the following additional categories of stats information:

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

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

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

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

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

Table 3-13 StatsManager Dictionary Keys

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

Receive Video Player State Notifications

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

Table 3-14 Video Player State Notification

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

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

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

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

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

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

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

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

Detect Video Player "Channel Inactive" Callback

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

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

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

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

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

   // shutdown video playback
   shutdown();

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

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

Customizing the Default Video Player

This section describes how to customize the default video player. The default Cisco video player has the following features:

- Implemented as a separate Android "Activity."
- Supports fullscreen and partial-screen video views.
- Renders video frames using an Android "SurfaceView."
- Customizable by extending the "SVMVideoPlayerActivity" class.

Figure 3-8 Default Cisco Video Player

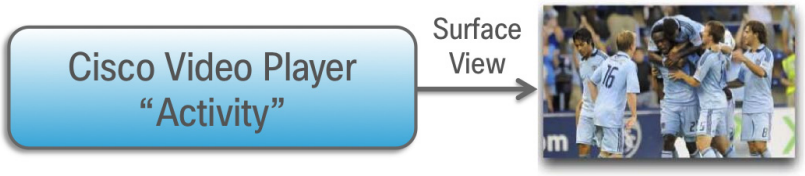


Figure 3-9 SVMVideoPlayerActivity API

Cisco Demo Video Player

The Cisco demo video player:

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

The video player's XML layout file defines:

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

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

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

public class MyVideoPlayer extends SVMVideoPlayerActivity {
}
```

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

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

Video Channels

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

Getting the Video Channel List

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

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

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

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

Presenting the Video Channel List

Each "SVMChannel" video channel object contains all of the information needed to display the channel list to the user. The SVMChannelObject properties and descriptions are shown in Table 3-15.

Table 3-15 SVMChannel Object Properties

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

Table 3-15 SVMChannel Object Properties (continued)

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

Playing a Video Channel

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

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

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

Seeking Within the Video Buffer

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

```
public class MyVideoPlayerActivity extends SVMVideoPlayerActivity {

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

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

```
public class MyVideoPlayerActivity extends SVMVideoPlayerActivity {

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

Setting the Video Dimensions

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

Fullscreen Video Layout

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

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
```

```
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="@drawable/black">

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

</RelativeLayout>
```

Partial-Screen Video Layout

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

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

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

</RelativeLayout>
```

Data Channels

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

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

Getting the Data Channel List

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

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

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

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

Observing a Data Channel

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

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

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

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

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

}
```

Audio Channels

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

Getting the Audio Channel List

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

Get a reference to the audio manager from the SDK

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

The application starts the audio channels by invoking

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

Stops them

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

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

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

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

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

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

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

}
```

EVS C-Cast Integration

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

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

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