CHAPTER 4

Cisco StadiumVision Mobile API for Windows Phone

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

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

Introduction to Cisco StadiumVision Mobile SDK for Windows Phone

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

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

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

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

Note HTML/Javascript is not supported.

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

Table 4-1 Mobile OS Support

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

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

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

Cisco StadiumVision Mobile and Windows Developer Tools

Table 4-2 lists the various build environment requirements.

Table 4-2 Build Environment Requirements

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

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

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

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

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

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

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

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

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

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

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

Download and Unpack the SDK

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

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

Table 4-3 Cisco StadiumVision Mobile SDK File Content

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

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

Getting Started with the Windows Demo App

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

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

Compile the Demo App

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

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

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

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

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

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

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

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

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

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

Customize the Demo App

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

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

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

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

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

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

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

Embed the Cisco StadiumVision Mobile SDK in an Existing App

Integration Checklist

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

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

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

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

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

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

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

Channel ListBox Window Example

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

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

Video Channel Selection Example

The following example illustrates the following actions:

- Plays the video channel selected in the ListBox

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

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

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

Window Life-Cycle Notifications

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

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

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

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

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

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

Configuration

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

Table 4-4 Configuration File

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

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

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

Wi-Fi AP Info Configuration (Optional)

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

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

How Cisco StadiumVision Mobile Fits into a Windows Phone App

Cisco StadiumVision Mobile Class Overview

Figure 4-1 describes the StadiumVision Mobile classes.

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

Figure 4-1 StadiumVision Mobile Class

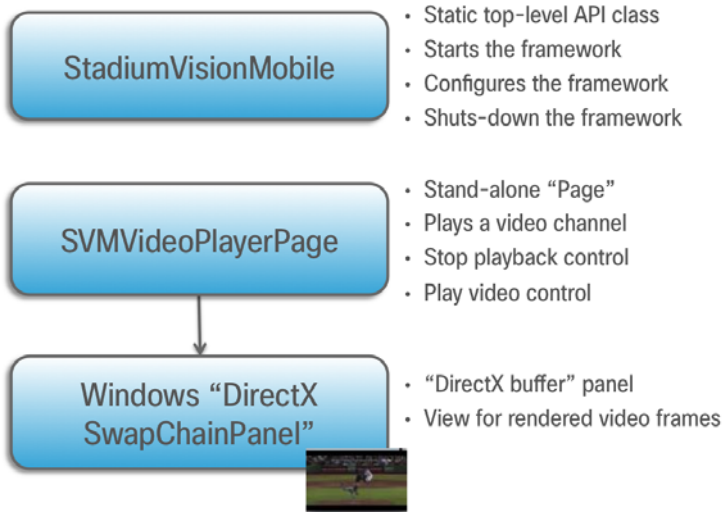


Figure 4-2 depicts the Windows OS with regard to Pages. A Page represents both the screen layout and controller code or Code Behind. A new Page is launched by sending an Event to the Windows OS. An Event is a message to Windows OS to launch a particular page. Extra parameters contained in an Event and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

Figure 4-2 Windows Overview

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

Figure 4-3 Windows Video Player Window Inheritance

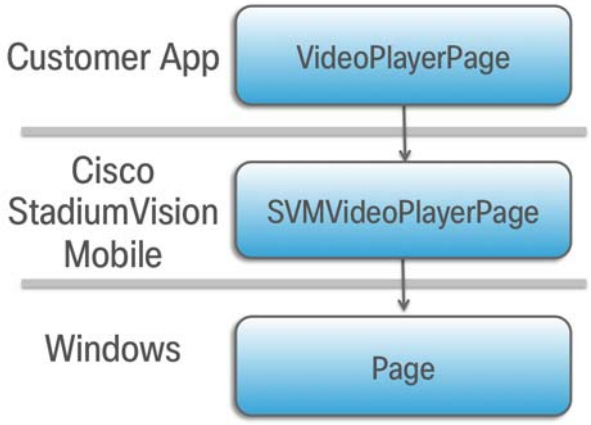
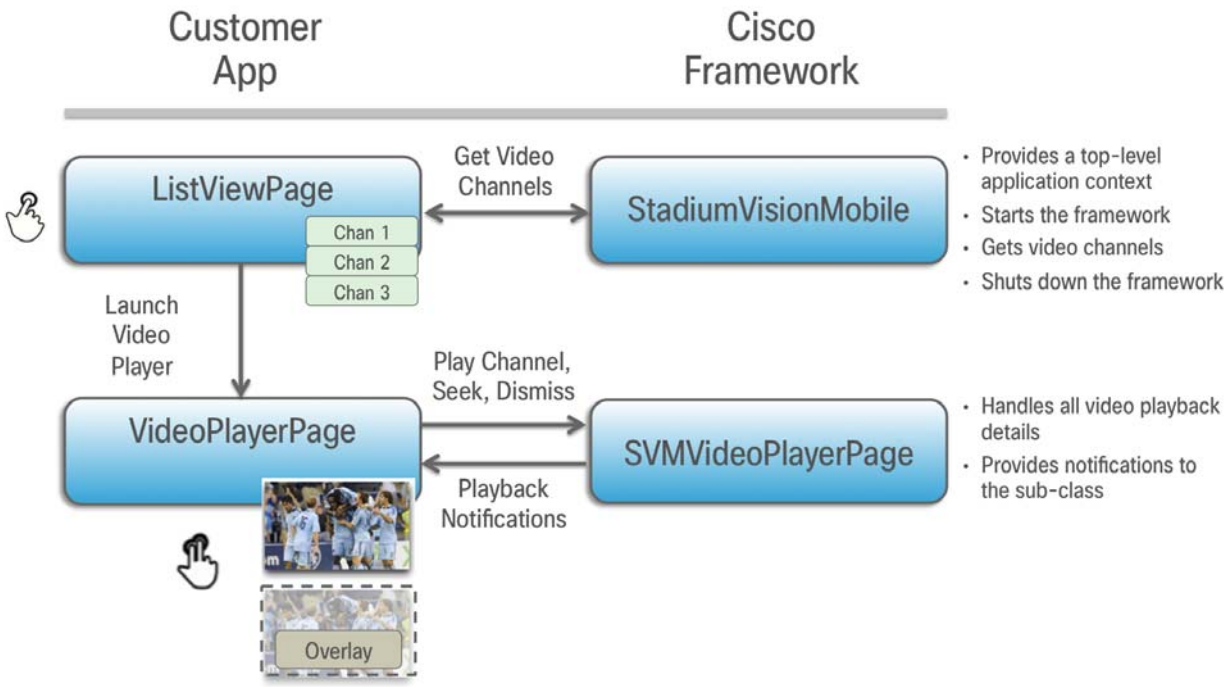


Figure 4-4 Cisco StadiumVision Mobile Integration Overview

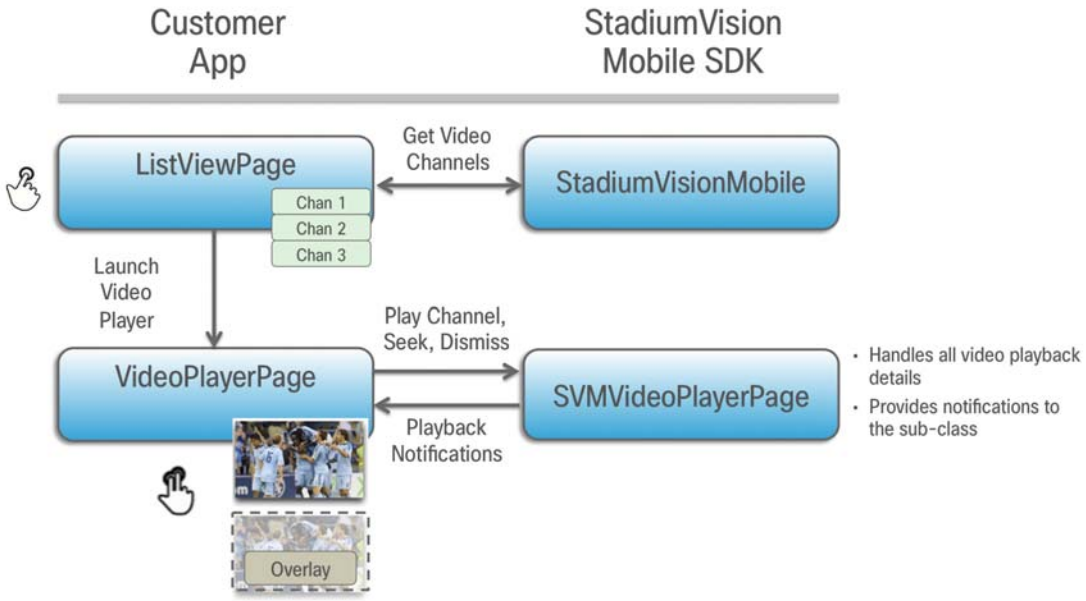


Customer Application Roles

Figure 4-5 illustrates the roles of the customer application. The application must specify:

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

Figure 4-5 Customer Application Responsibilities



Cisco StadiumVision Mobile Methods and Functions for Windows

Cisco StadiumVision Mobile Windows API Summary

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

Table 4-5 Cisco StadiumVision Mobile Windows API Summary

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

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

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

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

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

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

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

Return Status Object

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

Table 4-6 SVMStatus Object

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

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

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

Table 4-7 getStats API Dictionary Keys and Stats Description

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

Video Player Window API Summary

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

Table 4-8 Video Player Window API Summary

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

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

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

Starting the SDK

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

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

Cisco StadiumVision Mobile Service Up or Down Indicator

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

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

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

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

When the app receives the "Service Down" notification, the SDK will supply a bitmap containing the reasons why the service was declared as 'down' by the SDK. The 'reasons' bitmap is given in Table 4-9.

Table 4-9 Service Down Notifications

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

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

Receiving "Service Up/Down" Notifications

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

SVMVideoPlayerPage objVpa;

objVpa = new SVMVideoPlayerPage(currentChannel);

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


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

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

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

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

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

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

```
// get the current svm service state
SVMServiceStateEnum serviceState = objSVM.getServiceState();

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

In-Venue Detection

The StadiumVision Mobile Release 2.1 SDK provides a mechanism to detect whether the mobile device is connected within the SVM-enabled venue or not.

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

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

Receiving "In-Venue Detection" Notifications

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

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

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

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

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

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

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

Set the SDK Configuration at Run-Time

The application can set the SDK configuration at run-time through an API method. This allows the application to dynamically configure the SDK. For example, the application can fetch the SDK configuration information from a network connection and then pass that configuration to the SDK.

Two different ways to set the SDK configuration at run-time:

- "setConfig"

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

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

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

- "setConfigWithString"

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

  ```
  // configure the sdk using a json-formated string containing the configuration
  settings
  ```

```
public static SVMStatus setConfigWithString(String jsonConfigStr)
```

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

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

Get the SDK Configuration

"getConfig" API Method

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

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

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

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

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

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

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

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

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

setConfigWithString API Method

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

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

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

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

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

Get the Available Streamer Servers

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

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

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

Table 4-10 SVMStreamer Object Properties

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

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

```
// get the list of currently available streamer servers
SVMStreamerList  streamerList = StadiumVisionMobile.objSVM.getStreamerList();

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

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

Additional Statistics

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

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

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

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

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

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

Table 4-11 StatsManager Dictionary Keys

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

Video Player State Notifications

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

Table 4-12 Video Player State Notification

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

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

```
// subscribe to channel event

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



         public void unregisterChannelListChanged()
         {


          SVMChannelListener.objSVMChannelListener.channelChangeEvent
-= onChannelStateChanged;
         }



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

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

Video Player "Channel Inactive" Event

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

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

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

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



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

Customizing the Default Video Player

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

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

Figure 4-6 Default Cisco Video Player

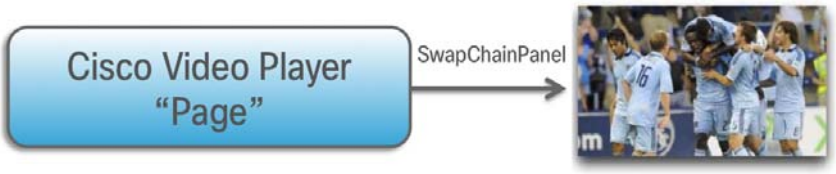
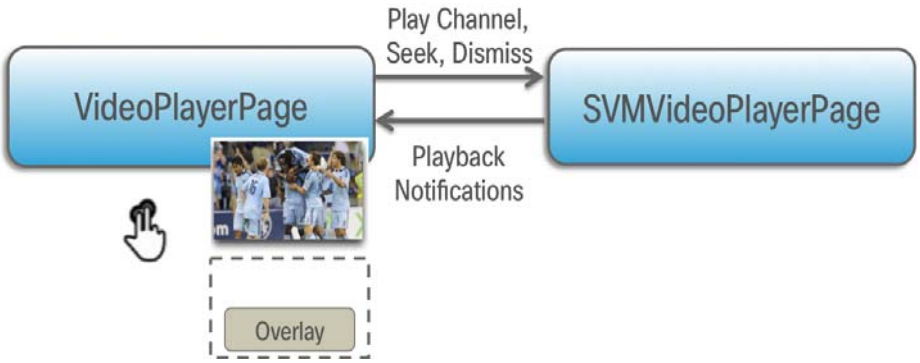


Figure 4-7 SVMVideoPlayerPage API



Cisco Demo Video Player

The Cisco demo video player:

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

The video player's XAML layout file defines:

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

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

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

Video Channels

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

Getting the Video Channel List

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

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

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

Presenting the Video Channel List

Each "SVMChannel" video channel object contains all of the information needed to display the channel list to the user. The SVMChannelObject properties and descriptions are shown in Table 4-13.

Table 4-13 SVMChannel Object Properties

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

Playing a Video Channel

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

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

Seeking Within the Video Buffer

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

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

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

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

Setting the Video Dimensions

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

Data Channels

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

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

Getting the Data Channel List

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

```
using SvmSdk;

// get the list of available data channels
SVMChannel[] channels = StadiumVisionMobile.objSVM.getDataChannelArray();
// display some channel information
Log.d(TAG, "Channel Name = "+ channels[0].name);
Log.d(TAG, "Channel Bandwidth = " + channels[0].bandwidthKbps);
Log.d(TAG, "Channel Body Text = " + channels[0].bodyText);
```

Observing a Data Channel

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

```
using SvmSdk;

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

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

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

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

EVS C-Cast Integration

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

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

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