VideoXpert OpsCenter™ Plugin API

Reference Guide

Document number: C1685M-B

Publication date: 08/20

Table of Contents

Overview 3

Developing Plugins 5

Implementing a Plugin as a Class Library 5

Implementing a Plugin as an Executable 5

Meeting Key Requirements for Plugins 6

Authenticating the Plugin 6

Discovering Plugins 6

Installing a Plugin 6

Storing Temporary Files and Persistent Configuration Data 6

Storing Temporary Files 6

Storing Persistent Configuration Data 7

Configuring Overlay Plugins 7

Signing-On for Plugins 8

Enabling and Using Drawing Overlays 9

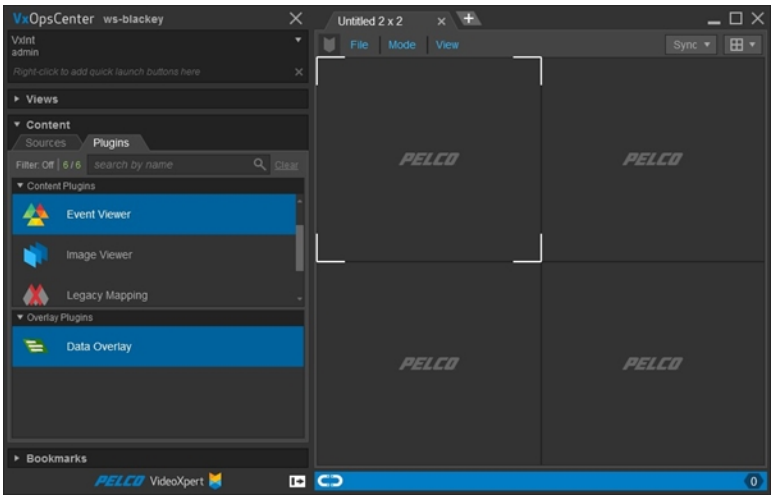
Sequence Diagrams 10

Class and Interface Definitions 13

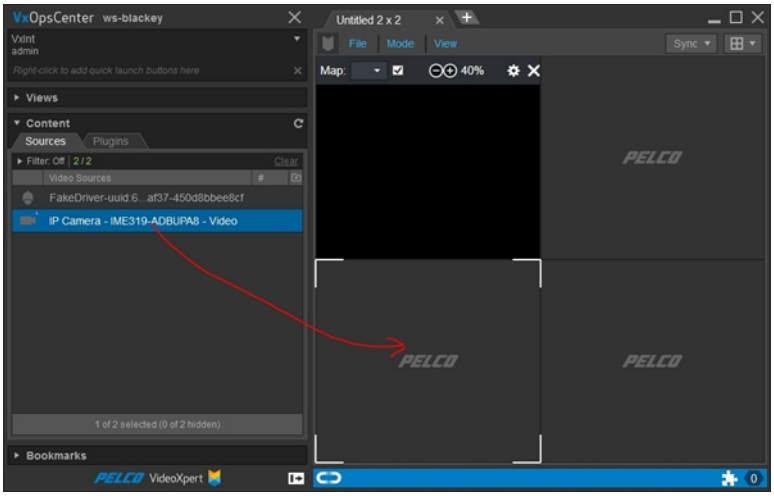
Overview

VxOpsCenter™ supports third-party plugins. This document describes how to create a basic plugin that will appear seamlessly inside a cell within a VxOpsCenter workspace.

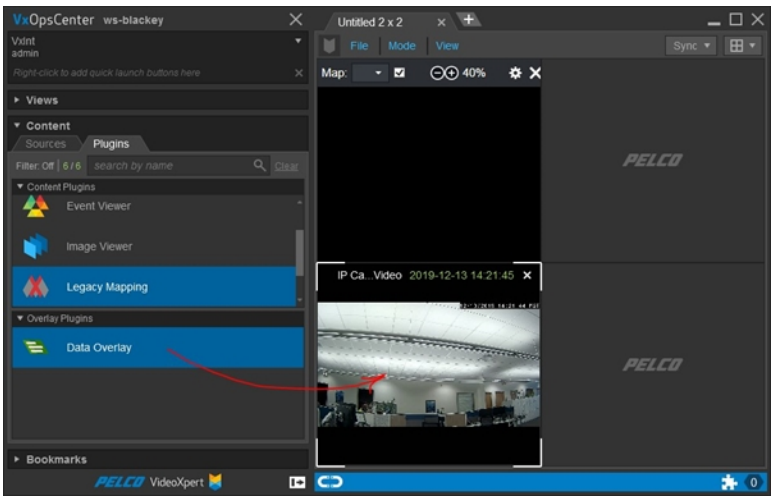
This image shows VxOpsCenter with the prime window on the left. The prime window contains the sources tab listing the available cameras as well as the plugins tab that shows the list of plugins. To the right of the prime window is a 4-up workspace. This workspace contains four cells. Each cell is capable of holding a content plugin, a video stream, or an overlay plugin and a video stream.



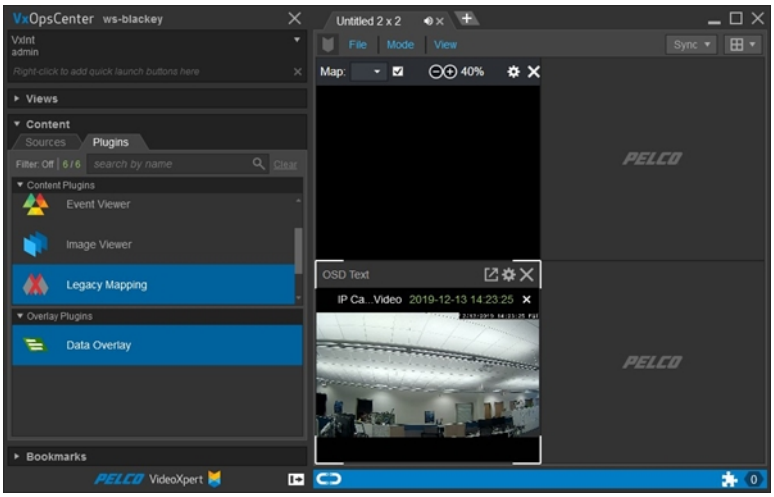
Dragging a plugin out of the content plugins list and dropping it in a cell will cause VxOpsCenter to load that plugin into the cell. Dragging a camera out of the sources list and dropping it in a cell will cause the video from that camera to stream into that cell.



Dragging an overlay plugin into a cell with video causes VxOpsCenter to load the plugin into that cell.



The image below shows VxOpsCenter prime window and a 4-up workspace. The upper left cell is occupied by a content style plugin. The lower left cell is occupied by both a video stream and an overlay style plugin. The right side upper and lower cells are unoccupied.



Developing Plugins

You can implement a plugin as an executable (exe) or a class library (dll). When starting a new project, an executable is recommended because it makes debugging much easier.

Implementing a Plugin as a Class Library

1. Create a class library project.
2. Reference the WPF assemblies PresentationCore, PresentationFramework, System.Xaml and WindowBase.
3. Add a reference to the PluginHostInterfaces assembly. Verify that “copy local” is set to false.
4. Create a WPF user control, such as MainUserControl.
5. Create a class named Plugin that derives from Pelco.Phoenix.PluginHostInterfaces.PluginBase.
6. Place the .dll file into the designated directory of the plugin.
7. Compile the plugin and run the host.

Implementing a Plugin as an Executable

1. Create a WPF application.
2. Create a WPF user control, for example, MainUserControl.
3. Add MainUserControl to the main window of the application.
4. Add a reference to the PluginHostInterfaces assembly. Verify that “copy local” is set to false.
5. Create a class named Plugin that derives from Pelco.Phoenix.PluginHostInterfaces.PluginBase.
6. Place your .exe in the designated directory of the plugin.

A plugin implemented like this can run as a standalone application or within the host. This simplifies debugging plugin code not related to host integration. The class diagram for such a “dual-head” plugin is shown below.

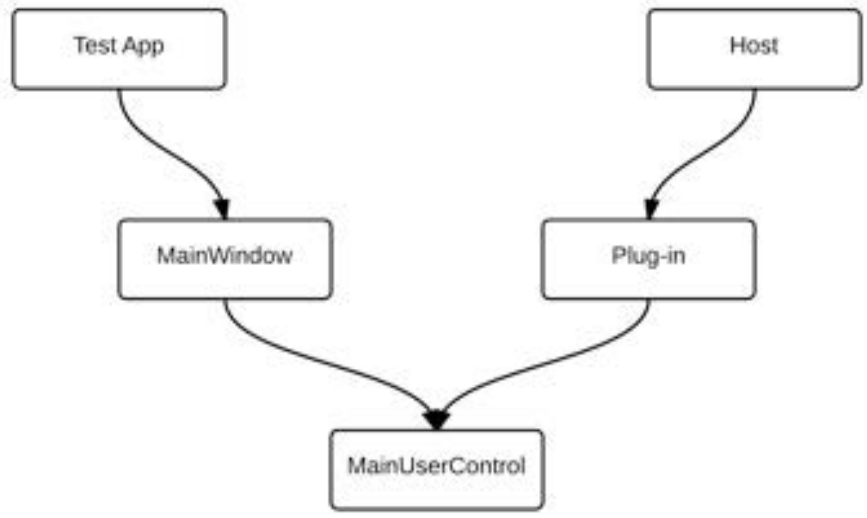


Figure 1: Class diagram for a "dual-head" plugin

Meeting Key Requirements for Plugins

Plugin writers are required to obtain a plugin developer key, issued by Pelco; without it the plugin will not show up in the VxOpsCenter application. Request the key via email to integrations@pelco.com.

Plugin installers must install their software into a new sub-directory below the “Plugins” directory of VxOpsCenter “CommonDataDir” specified in the registry. See the section titled *Discovering Plugins*.

Caution: Plugins without a valid key will not be loaded by VxOpsCenter.

Authenticating the Plugin

Each plugin developer will be issued a secret key that has been verifiably created by Pelco. That key will be provided by the plugin to VxOpsCenter at plugin load time. VxOpsCenter will verify that it is a key that has been created by Pelco before allowing the plugin to proceed.

It is understood that this method is not cryptographically secure and that if a malicious plugin were to acquire a valid plugin key no additional measures are being taken to prevent it from starting up in the system. It is generally believed that having plugin level permission to VxOpsCenter is not a security risk. The use of keys is to help Pelco to work more closely with plugin providers and to discourage unauthorized plugins.

Discovering Plugins

Plugins are discovered using .NET reflection.

Installing a Plugin

Plugin installers must install their software into a new sub-directory below the “Plugins” directory of VxOpsCenter “CommonDataDir” specified in the registry.

This “CommonDataDir” key is under HKLM\Software\Pelco\OpsCenter.

- Key=CommonDataDir
- Value=The application data folder
- Type=String

This registry entry is under the WOW6432Node: HKLM\Software\WOW6432\Pelco\OpsCenter

The plugins are under CommonDataDir\Plugins. Each company should create their own sub-folder and folder for each plugin. For example: a new plugin might be placed into “C:\ProgramData\Pelco\OpsCenter\Plugins\ABCompany\LPR\ABCPlugin.exe”.

Storing Temporary Files and Persistent Configuration Data

Storing Temporary Files

- A plugin MIGHT store temporary data in %APPDATA%\CompanyName\Plug-inName.
- A plugin MUST NOT rely on this temporary data being available when it starts up, because it might be started up on a different VxOpsCenter or the temporary data folders might have been deleted.

Caution: Persistent Configuration Data MUST NOT be stored in %APPDATA%\CompanyName\Plug-inName.

Storing Persistent Configuration Data

VxOpsCenter supports saving plugin state as part of saving a workspace. For example: if a 4-up workspace contains a plugin in one of its cells, VxOpsCenter, when appropriate, will call the IOCCPlugin1.GetPluginState() to retrieve the current state of the plugin, and store it as part of the state of the workspace. If the workspace is opened again at some time in the future or on some other VxOpsCenter, the VxOpsCenter will call IOCCPlugin1.SetPluginState() and pass it the state data that it retrieved from the plugin earlier. The plugin is then expected, if possible, to apply the state that it was passed.

Configuring Overlay Plugins

Overlay plugins can be configured by returning true for IsOverlay, otherwise it will be a content plugin. An Overlay plugin is one that occupies the same cell as a video stream. These types of plugins are intended to annotate the video stream in some way.

An overlay plugin can change its configuration at run time by calling IOCCHostOverlay.SetOverlayAnchor ();

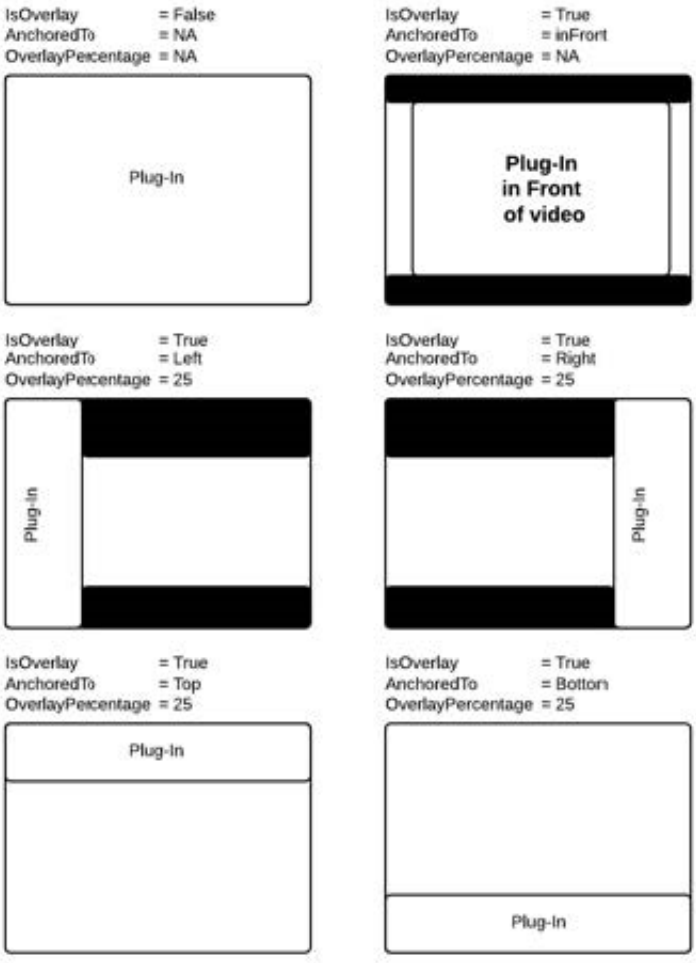


Figure 2: Overlay plugins

Signing-On for Plugins

A plugin that needs to authenticate its users so that it can know what kinds of data that a given user has permission to will follow the steps in the diagram below. These steps minimize the number of times a user has to log in to plugins.

Note: Steps 1 and 2 are VxOpsCenter steps. They are included in the diagram to give the plugin writer some context about when it will receive Login calls.

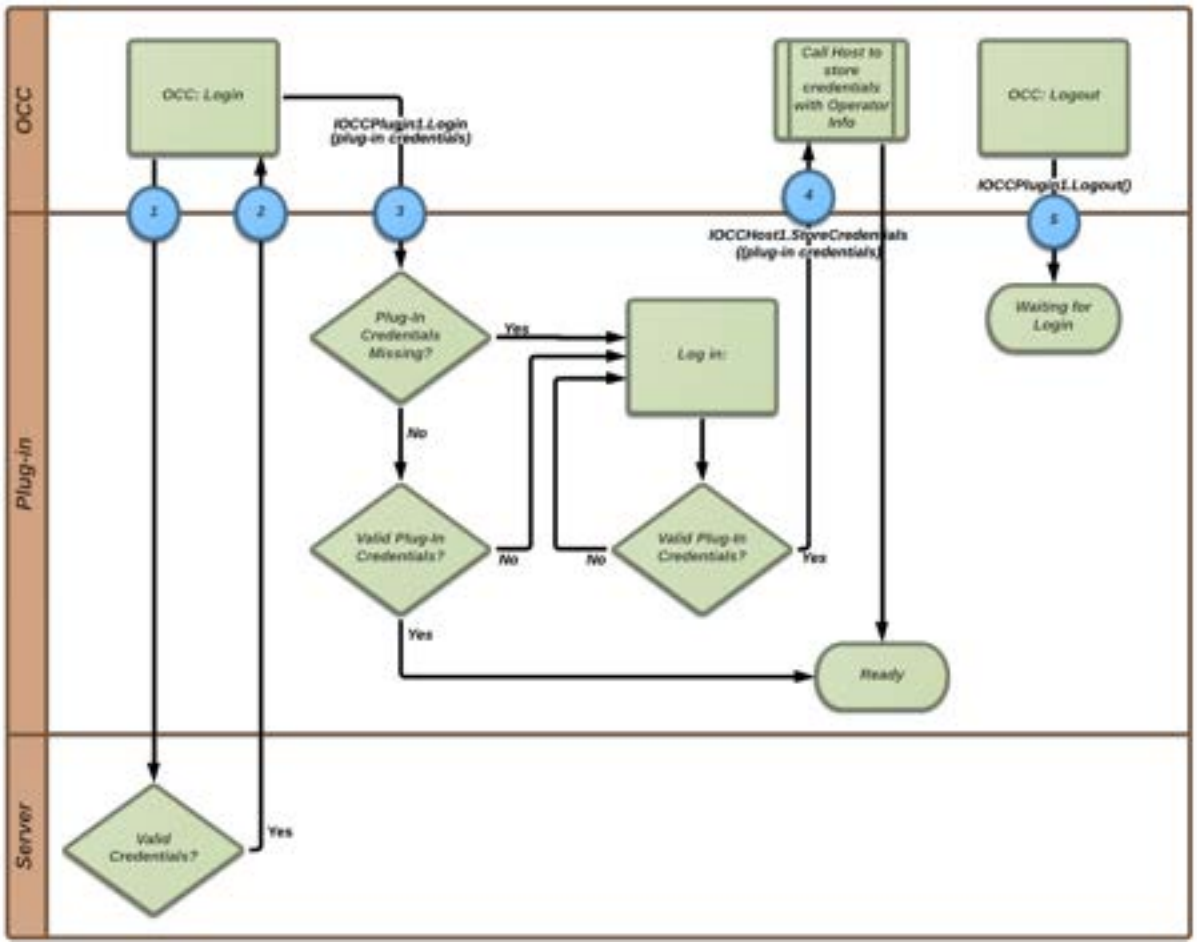


Figure 3: Sign on for plugins

This approach requires that VxOpsCenter store the plugin credentials “with” the operator’s account.

If this is the first time logging in, there will be no plugin credentials. The plugin will request them from the user and then call the host to store them (IOCCHost1.StoreCredentials) with the currently logged in operator’s account.

If this is a subsequent login then the plugin credentials will be provided. It’s up to the plugin to validate the credentials that are passed to it. In some cases the plugin will validate with its own server. If the credentials are valid the plugin is good to go. If the credentials are not valid, then the plugin should report that the user is not authorized for this plugin, and give them an opportunity to log in.

Enabling and Using Drawing Overlays

Use the plugin development kit (PDK) to draw overlays such as shapes and text over video in a VxOpsCenter video cell. For a complete example demonstrating this functionality, see https://github.com/pelcointegrations/VxPDK-OverlayPlugin.

Setup the plugin and allow it to draw arbitrarily over video. To do so:

1. Ensure that the main plugin class implements the IOCCPluginVideoOverlay interface. For an example, see https://github.com/pelcointegrations/VxPDK-OverlayPlugin/blob/master/Plugin/Plugin.cs#L24.
2. Invoke the PDK to call into the plugin and to provide video aspect ratio, rotation, and dptz information. To do so, instantiate IOCCHostVideoOverlay and call the RegisterForVideoViewNotifications method with a “true” argument. For an example, see https://github.com/pelcointegrations/VxPDK-OverlayPlugin/blob/master/Plugin/Services/PluginHost/PluginHostSvc.cs#L28

   Caution: If you do not perform this step, the methods implemented from IOCCPluginVideoOverlay will never be called.

The methods exposed by the IOCCPluginVideoOverlay interface, and descriptions of use, are as follows:

- FrameworkElement CreateVideoOverlay()
  This enables developers to instantiate a WPF user control (drawing surface) and return the reference to the PDK, allowing the system to display the control over the video.
- void OnVideoViewStreamAspectRatio(double aspectRatio)
  This notifies the developer of the video aspect ratio.
- void OnVideoViewDigitalPtzInfo(Rect normalizedDPTZWindow)
  This notifies the developer when a user has digitally zoomed on the video; the developer can then scale the overlay accordingly. This method also provides a normalizedDPTZWindow parameter with the values in percentage points, allowing the developer to calculate the area that is currently magnified.
- void OnVideoViewWindow(Rect normalizedVideoWindow, double rotation)
  This notifies the developer when a user has rotated the video, which allows the developer to also rotate the overlay surface accordingly. This method also provides a normalizedVideoWindow parameter with the values in percentage points, to account for pillar and letterbox spacing.

Sequence Diagrams

The following sequence diagrams show what methods are called and in what order.

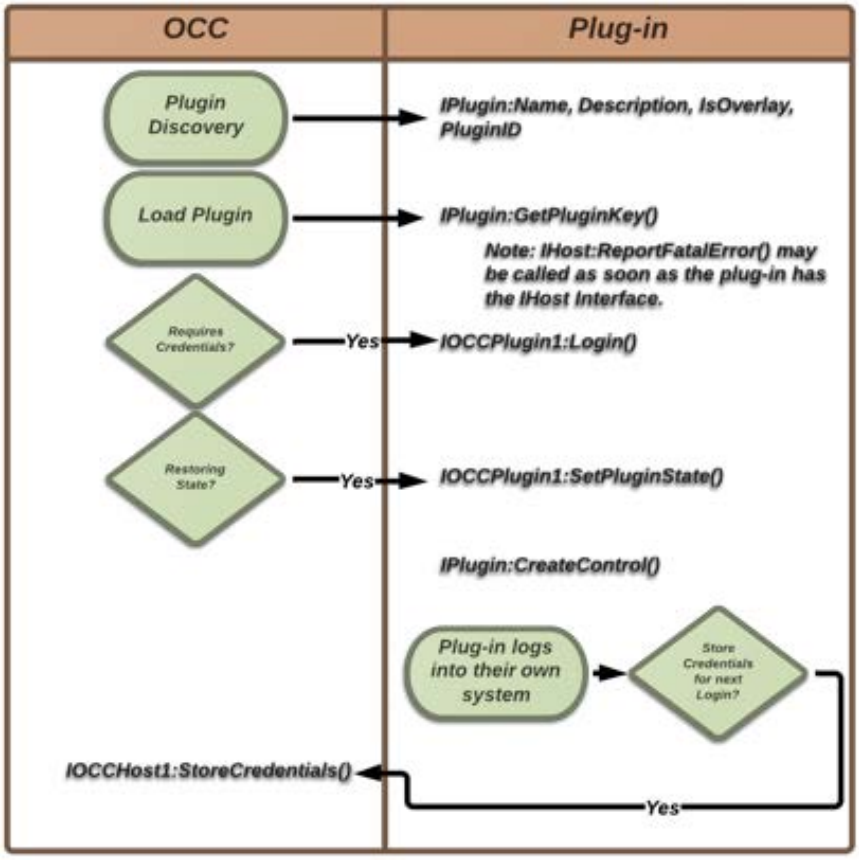


Figure 4: Startup sequence

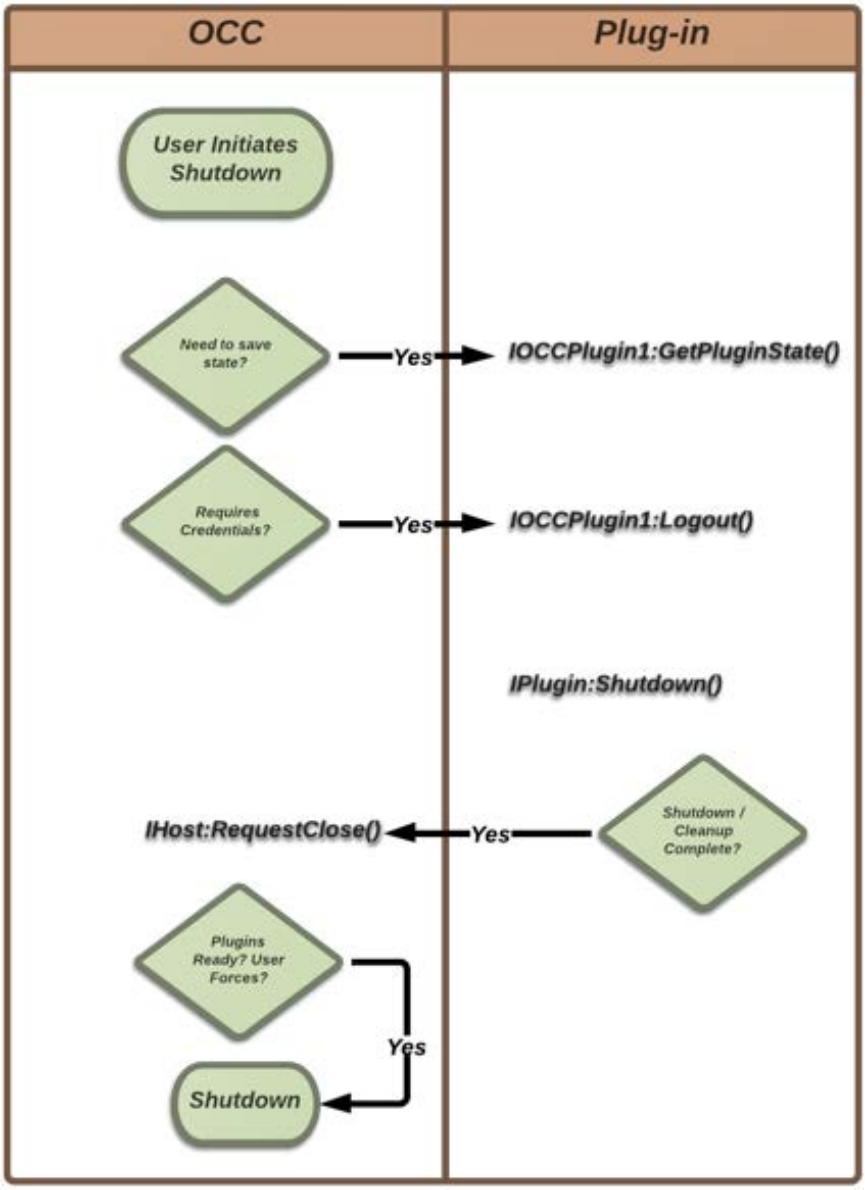


Figure 5: Shutdown sequence

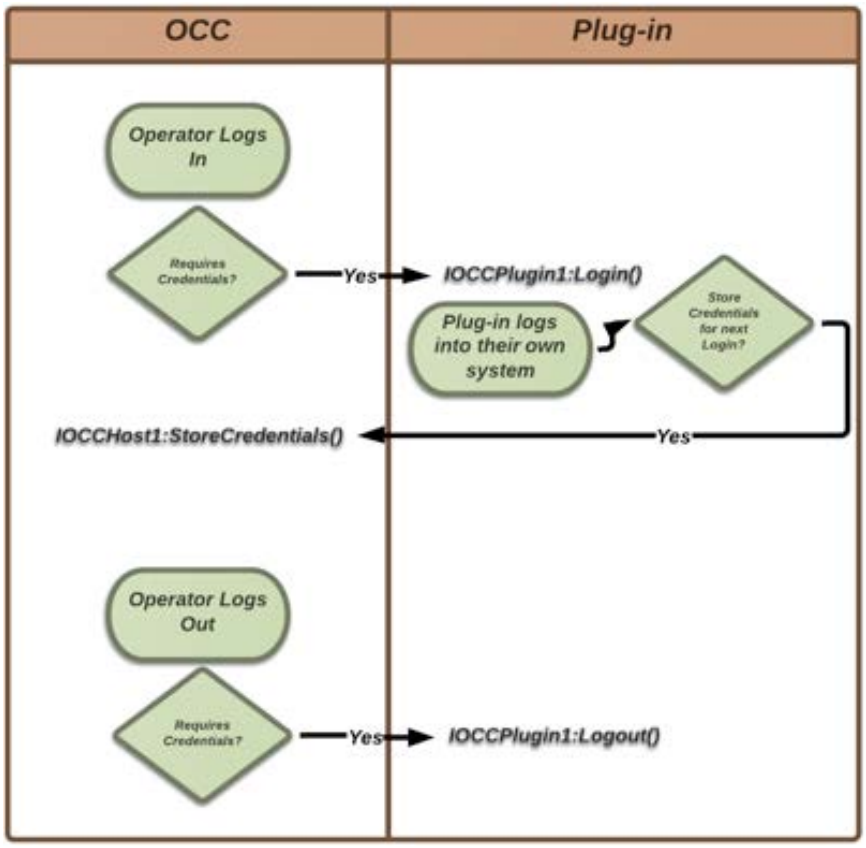


Figure 6: Sequence to Login/Logout of VxOpsCenter Client

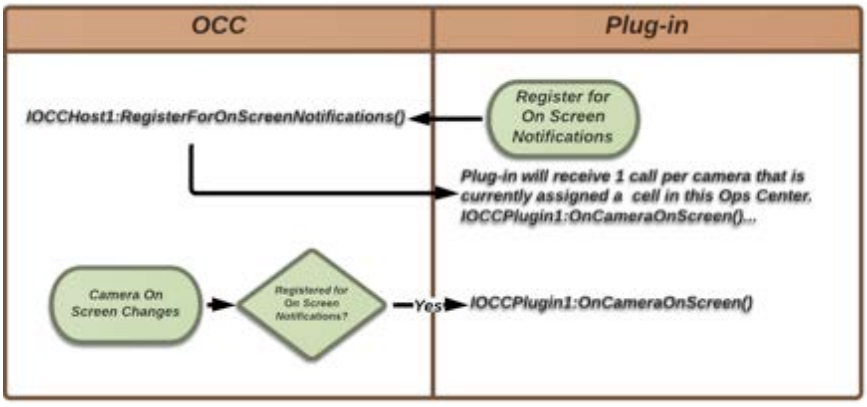


Figure 7: Sequence to register for camera onscreen notifications

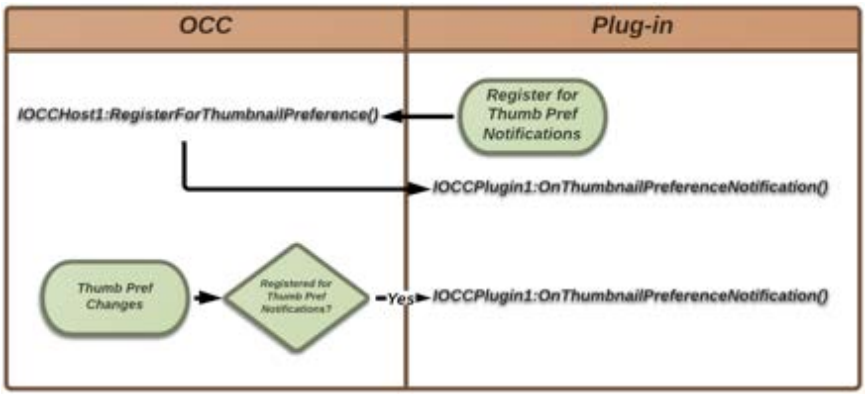


Figure 8: Sequence to register for thumbnail preferences for notifications

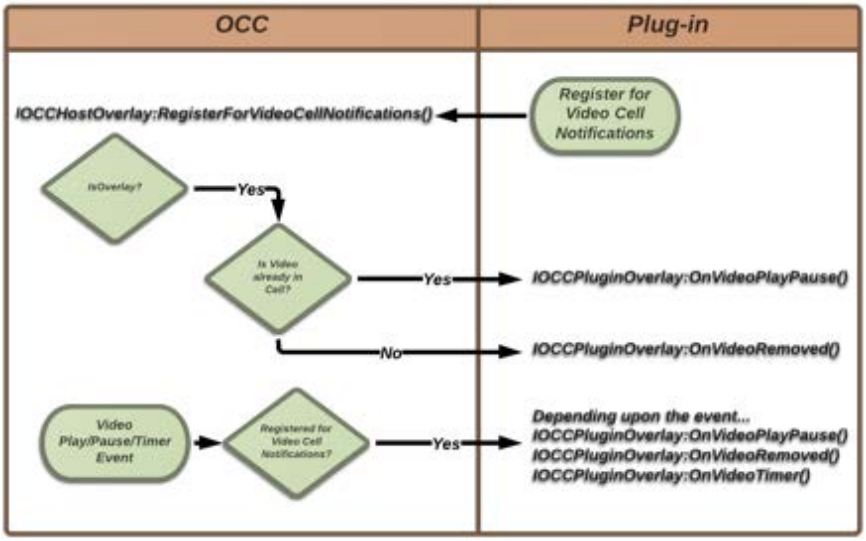


Figure 9: Sequence to register for video cell notifications

Class and Interface Definitions

To keep your plugin performing optimally:

- Be prepared to have the host call into your plugin as soon as your plugin constructor returns. It will send you information that you will need to set up or restore the state of your plugin before it actually becomes visible to the user. To discover what plugin APIs can happen before the host calls CreateControl() on your plugin, see the section titled *Sequence Diagrams*.
- Spend as little time as possible responding to IPlugin or IOCCPlugin calls. Although you are in a separate process, the UI is still serialized and long response times might make VxOpsCenter appear to be frozen.
  For example: DO NOT do synchronous server-side communication inside an IPlugin or IOCCPlugin call. If the server is slow to respond, your plugin and possibly the entire VxOpsCenter will appear to be frozen. Do the minimum and return.
- A race condition with plugins happens when a plugin spends a too much time in CreateControl(). When the plugin appears to be unresponsive (because its taking a long time to load), the user might give up on the plugin and close it. This can cause a hard shutdown of the plugin when it is trying to start up.

The easiest way to avoid this is to respond quickly to CreateControl() without completely loading everything. Either put up a “Loading…” progress UI or build the UI asynchronously as you get the information you need. In either case the user will see that the plugin is not dead, it is just not done loading yet.

The following interfaces can be instantiated by the plugin to call operations on VxOpsCenter.

- IOCCHost1
  - void InitiateStreams(List<string> cameraList);
  - void RegisterForThumbnailPreference(bool send);
  - void RegisterForOnScreenNotifications(bool send);
  - void StoreCredentials(string credentials);
- IOCCHostGeneral
  - void LaunchContent(LaunchContent content);
  - void LaunchContentList(List<LaunchContent> contents);
  - void RegisterForThumbnailPreference(bool send);
  - void RegisterForOnScreenNotifications(bool send);
  - void StoreCredentials(string credentials);
- IOCCHostJoystick
  - void RegisterForJoystickNotifications(bool send);
  - void SetJoystickLimits(int xMin, int xMax, int yMin, int yMax, int zMin, int zMax, int uMin, int uMax);
  - void EscJoystick();
- IOCCHostLookupDatasourceIDs
  - List<string> LookupDatasourceIDs(string cameraNumber);
- IOCCHostOverlay
  - void RegisterForVideoCellNotifications(bool send);
  - void SetOverlayAnchor(AnchorTypes anchoredTo, double percentage, int min, int max);
- IOCCHostOverlayDrawingService
  - void Draw(Overlay overlay);
  - void Update(Overlay overlay);
  - void RemoveOverlay(string overlayId);
  - void ClearOverlays();
- IOCCHostOverlayEx
  - void RegisterForVideoCellNotifications(bool send, TimeSpan? onVideoTimerInterval = null);
- IOCCHostPlaybackController
  - void Play();
  - void Pause();

  - void Seek(DateTime utcTime);
  - void AdjustSpeed(PlaybackSpeed speed);
  - void JumpToLive();
  - void SetOnScreenDataSource(string datasourceId, DateTime? utcStartTime);
  - void RegisterForVideoPlaybackNotifications(bool shouldSend, bool provideMetadataAssociations = false);
- IOCCHostRegisterForDragDrop
  - void RegisterForDragDrop(bool send);
- IOCCHostSerenity
  - string GetAuthToken();
  - string GetBaseURI();
  - List<SystemInfo> GetConnectedSystems();
- IOCCHostSetDatasource - Interface deprecated, please use IOCCHostPlaybackController instead
  - void SetDatasource(string datasourceId, DateTime? utcTime);
- IOCCHostSetVideoPosition - Interface deprecated, please use IOCCHostPlaybackController instead.
  - void SetVideoPosition(DateTime utcTime);
- IOCCHostVideoOverlay
  - void RegisterForVideoViewNotifications(bool send);

The following interfaces can be implemented by the plugin to get invoked by VxOpsCenter and to provided information.

- PluginBase - Required to be implemented by the plugin (implements IPlugin)
  - public abstract FrameworkElement CreateControl();
  - public virtual object GetService(Type serviceType);
  - public virtual void Dispose();
  - public abstract string GetPluginKey();
  - public abstract string Name { get; }
  - public abstract string Description { get; }
  - public abstract string Version { get; }
  - public abstract bool IsOverlay { get; }
  - public abstract string PluginID { get; }
  - public abstract void Shutdown();
- IPlugin - Implemented by PluginBase
  - FrameworkElement CreateControl();
  - string GetPluginKey();
  - string Name { get; }

  - string Description { get; }
  - string Version { get; }
  - bool IsOverlay { get; }
  - string PluginID { get; }
  - void Shutdown();
- IOCCPlugin1
  - void OnThumbnailPreferenceNotification(bool show);
  - void OnCameraOnScreeen(string cameraId, bool onScreen);
  - string GetPluginState();
  - void SetPluginState(string pluginState);
  - bool RequiresCredentials { get; }
  - void Login(string credentials);
  - void Logout();
- IOCCPluginGeneral
  - void OnThumbnailPreferenceNotification(bool show);
  - void OnCameraOnScreen(string cameraId, bool onScreen);
  - string GetPluginState();
  - void SetPluginState(string pluginState);
  - bool RequiresCredentials { get; }
  - void Login(string credentials);
  - void Logout();
- IOCCPluginJoystick
  - void OnJoystickNotification(int X, int Y, int Z, int U);
- IOCCPluginOverlay
  - void OnVideoPlayPause(string dataSourceId, string number, bool live, bool playing, DateTime utcTime);
  - void OnVideoTimer(DateTime utcTime);
  - void OnVideoRemoved();
- IOCCPluginPlaybackNotifications
  - void OnVideoRemoved();
  - void OnNewVideoSourcePlaying(PlayingState state);
  - void OnVideoPaused();
  - void OnPlayUpdate(bool isLive, DateTime? anchorTime, DateTime? initiationTime, double speed);
- IOCCPluginVideoOverlay
  - FrameworkElement CreateVideoOverlay();
  - void OnVideoViewStreamAspectRatio(double aspectRatio);

- void OnVideoViewDigitalPtzInfo(Rect normalizedDPTZWindow);
- void OnVideoViewWindow(Rect normalizedVideoWindow, double rotation);

VideoXpert OpsCenter™ Plugin API Reference Guide

Pelco, Inc.
625 W. Alluvial Ave., Fresno, California 93711 United States
(800) 289-9100 Tel
(800) 289-9150 Fax
+1 (559) 292-1981 International Tel
+1 (559) 348-1120 International Fax
www.pelco.com