Ops Center Client Plug-In API

Version 1.2.2

Revisions

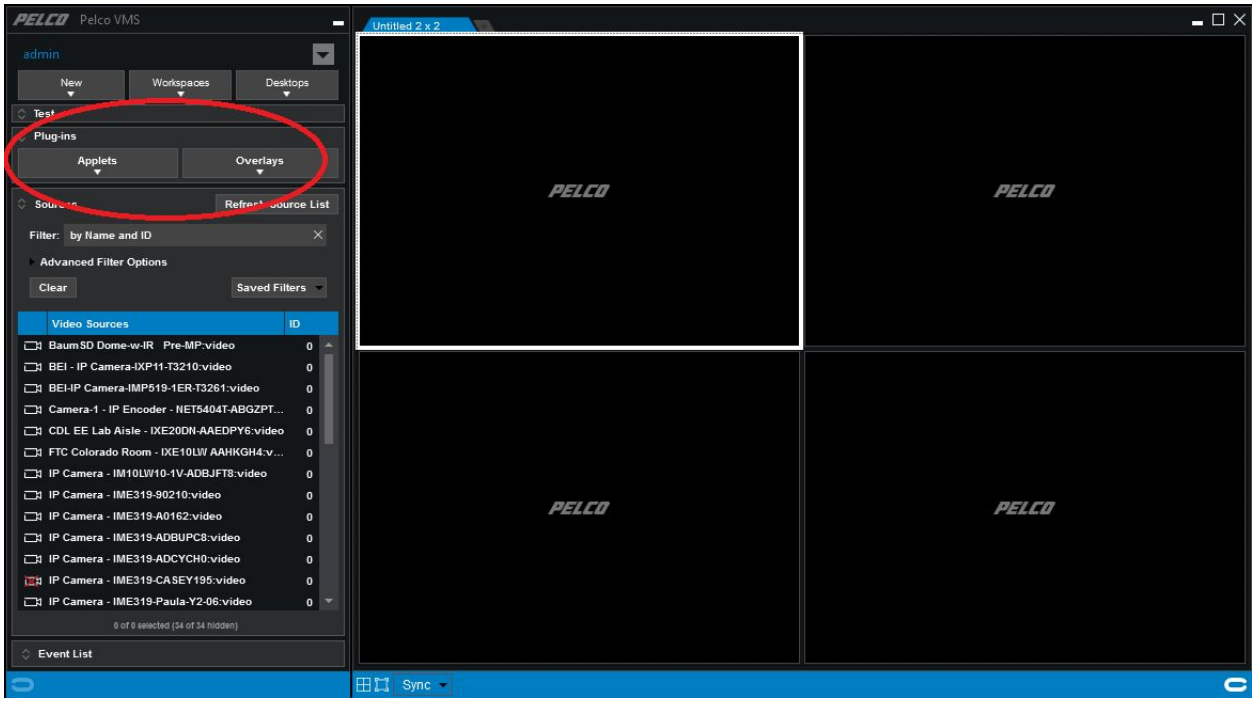
Date Completed	Rev	Description	Author
6/15/2015	1.2.2	• Added ability to drag/drop lists of LaunchContent	GaryK
6/11/2015	1.2.1	• Added RegisterForDragDrop • Added LaunchContent • Added LaunchContentHelpers	GaryK
7/24/2014	1.0.6	• Added IOCCHostSetVideoPosition	GaryK
04/15/2014	1.0.5	• Added some tips for making plug-ins appear more responsive.	Gary K
04/15/2014	1.0.4	• Updated Startup Sequence Diagram.	Gary K
04/15/2014	1.0.3	• Added the plug-in joystick API	Gary K
04/07/2014	1.0.2	• All host calls are available to a plug-in from the moment that it has the IHost interface.	Gary K
04/07/2014	1.0.1	• Updated Sequence Diagrams	Gary K
03/31/2014	1.0	• Broke the interfaces into IPlugin, IOCCPlugin1, IOCCPluginOverlay, IOCCPluginReserved and IHost, IOCCHost1, IOCCHostOverlay, IOCCHostSerenity • Changed On Startup sequence diagram to show that SetPluginState() can be called before CreateControl() is called. • Removed GetPluginInterfaceVersion() and Bits() API. • Added PluginID to IPlugin. • Added “number” to SetDataSource() • Now using DateTime for utcTime everywhere. • Renamed ShutdownComplete() to RequestClose() • Renamed “Above” docking configuration to “InFront” • Overlay’s are now docked only. Not transparent.	Gary K
03/04/2014	0.2.1	• Added sequence diagrams	Gary K
03/03/2014	0.2	• Includes overlay, sign on and plug-in discovery using reflection	Gary K
02/13/2014	0.1	• Initial Revision • (API stable for Mapping Plug-In)	Gary K

Contents

1.0 Overview

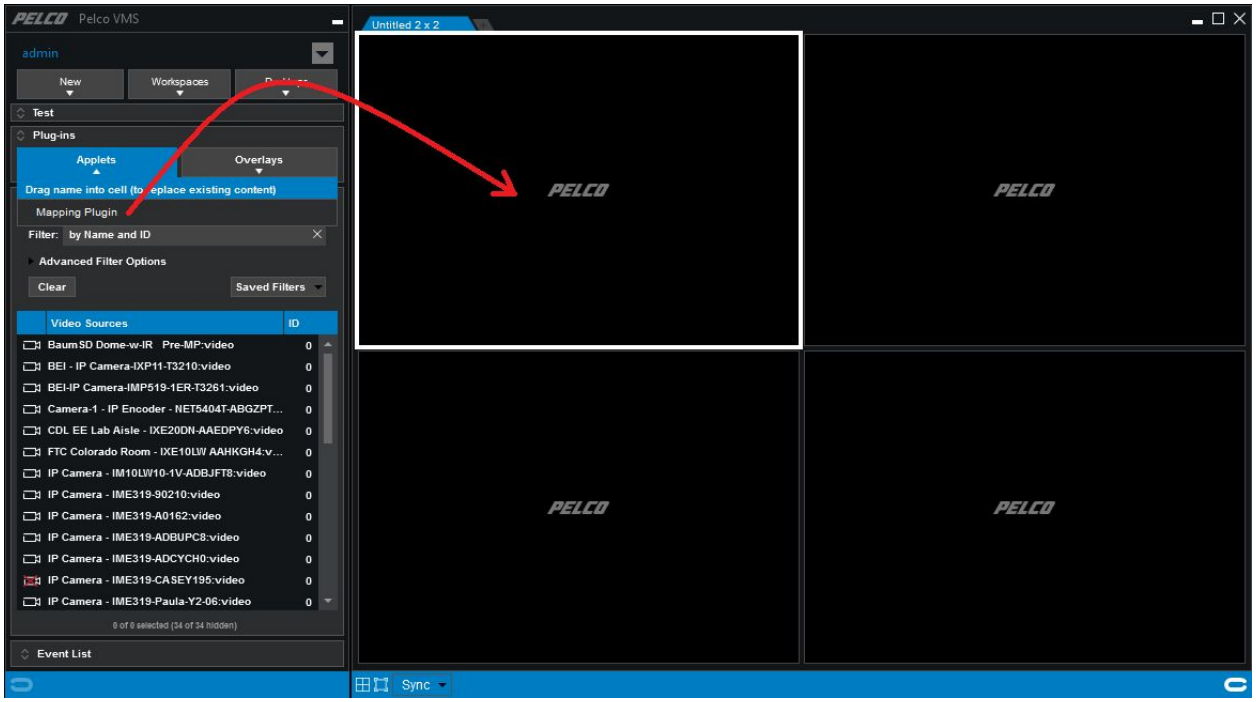
The Ops Center supports 3rd Party Plug-ins. This document describes how to create a basic plug-in that will appear seamlessly inside of a cell within an Ops Center workspace. (See below.)

This image shows the Ops Center with the “Prime” window on the left. The Prime window contains the source (camera) list and dropdowns that show the list of plug-ins that are available, as well as some buttons for creating workspaces and desktops. To the right of the “Prime” window is a 4up “Workspace”. This “Workspace” contains 4 “Cells”. Each “Cell” is capable of holding an “Applet” plug-in or a video stream or an “Overlay” plugin AND a video stream.

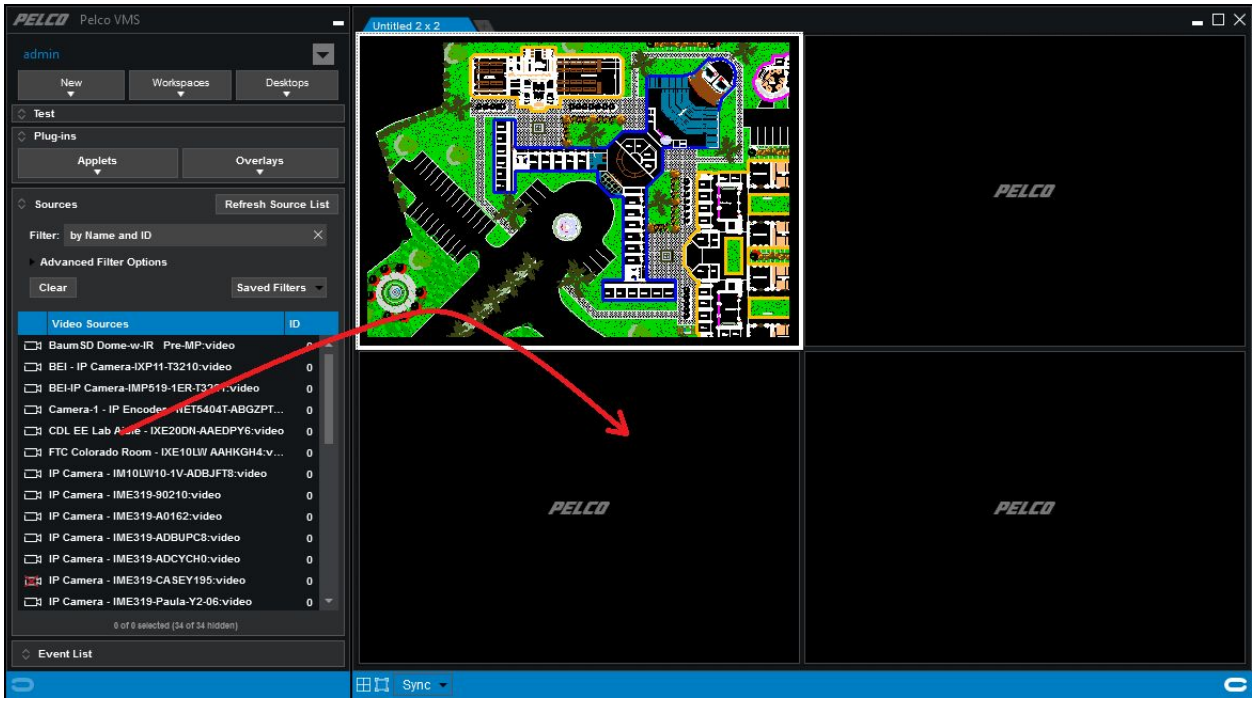


Clicking on the “Applets” dropdown in the “Plug-ins” area will show you a list of “Applet” style plug-ins that are available to run on this Ops Center.

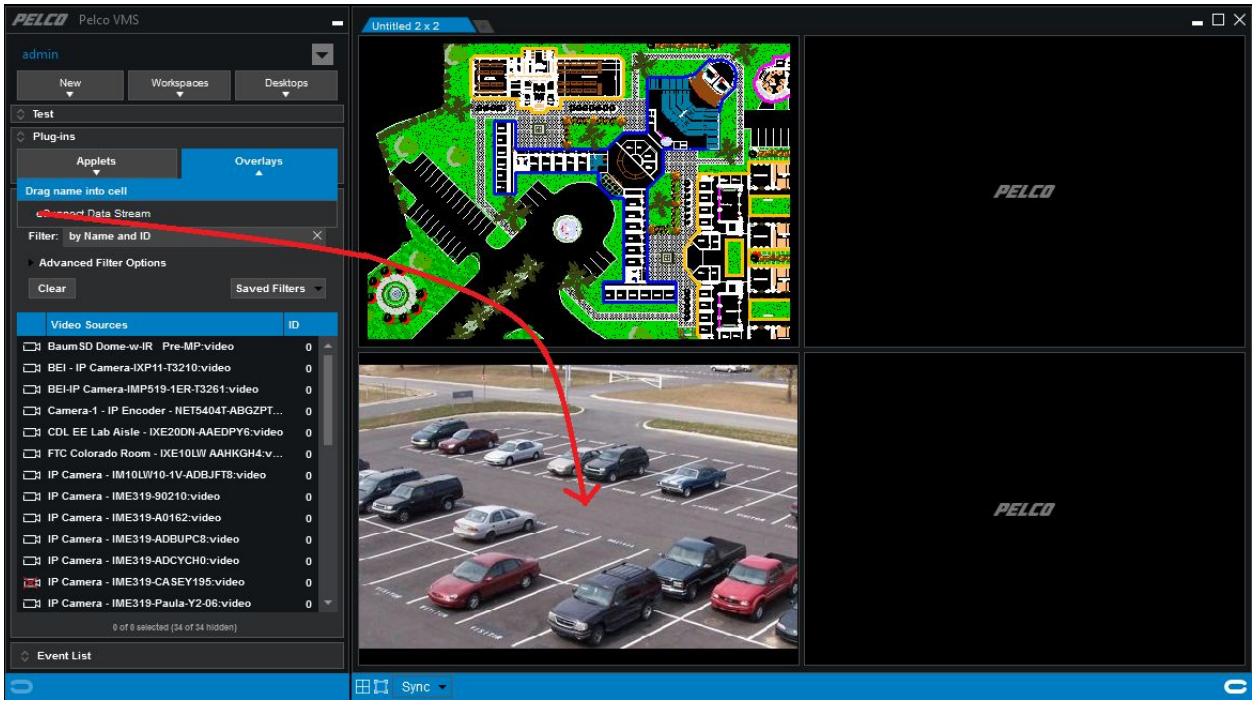
Dragging a plug-in out of the “Applets” list and dropping it in a “Cell” will cause the Ops Center to load that plug-in into the “Cell”.



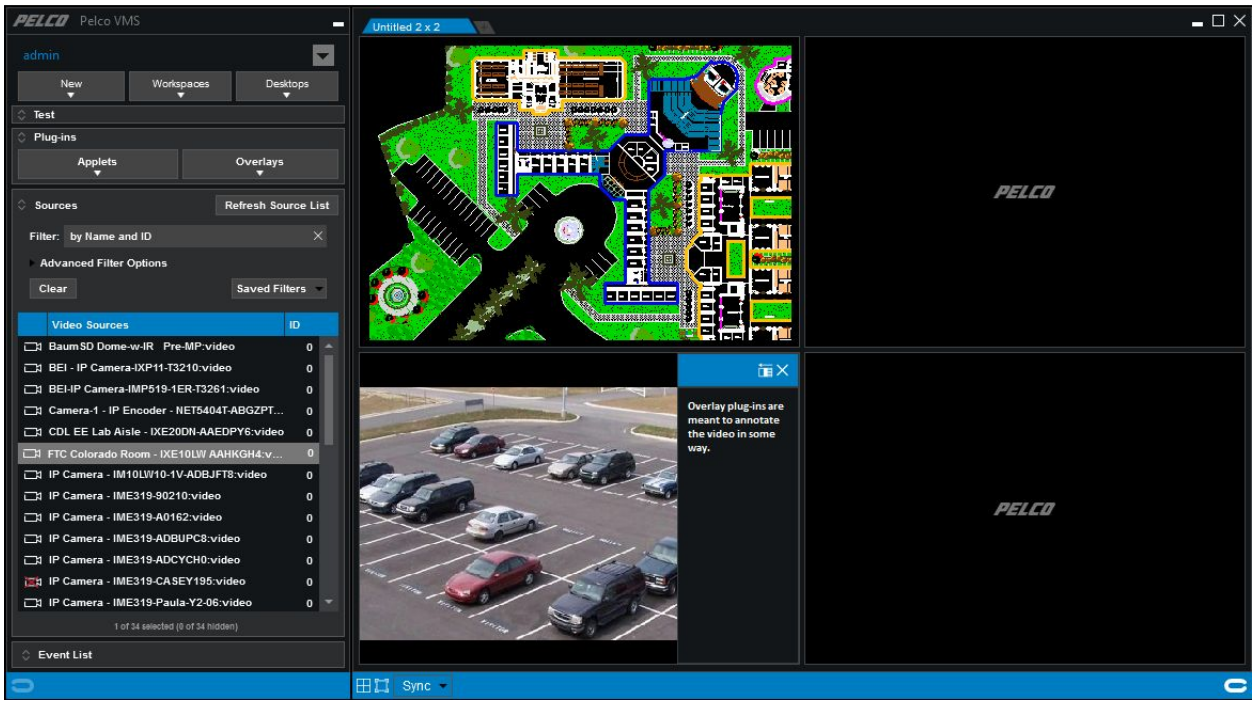
Dragging a camera out of the Source List and dropping it in a “Cell” will cause the video from that camera to stream into that “Cell”.



Clicking on the “Overlays” dropdown will show you a list of the “Overlay” style plug-ins. Dragging an “Overlay” plug-in into a “Cell” with video causes the Ops Center to load the plug-in into that “Cell”.



The image below shows the Ops Center Prime window and a 4up workspace. The upper left “Cell” is occupied by an “Applet” style plug-in. The lower left “Cell” is occupied by both a video stream and an “Overlay” style plug-in. The right side upper and lower cells are unoccupied.



2.0 Developing Plug-Ins

Note: All plugins should be compiled for Any CPU.
Developing Plug-Ins

1. You can implement a plug-in as a class library (DLL) or an executable (EXE). In the DLL scenario, the steps are as follows:
2. Create a new class library project
3. Reference the WPF assemblies PresentationCore, PresentationFramework, System.Xaml and WindowBase.
4. Add a reference to the PluginHostInterfaces assembly. Make sure “copy local” is set to false.
5. Create a new WPF user control, such as MainUserControl.
6. Create a class named Plugin that derives from Pelco.Phoenix.PluginHostInterfaces.PluginBase.
7. Place your .dll file into the designated plug-in’s directory.
8. Compile your plug-in and run the host.

A minimal plug-in class looks like this:

```
1.  public class Plugin : PluginBase
2.  {
3.     public override FrameworkElement CreateControl()
4.     {
5.        return new MainUserControl();
6.     }
7.     public override string GetPluginKey()
8.     {
9.        return  “plugin key”;
10.    }
11.    public override string Name
12.    {
13.       get { return "My Plug-in’s Name"; }
14.    }
15.    public override string Description
16.    {
17.       get { return "Description of plug-in."; }
18.    }
19.    public override string Description
20.    {
21.       get { return "Description of plug-in"; }
22.    }
23.    public override string Version
24.    {
25.       get { return "1.1a.0"; }
26.    }
```

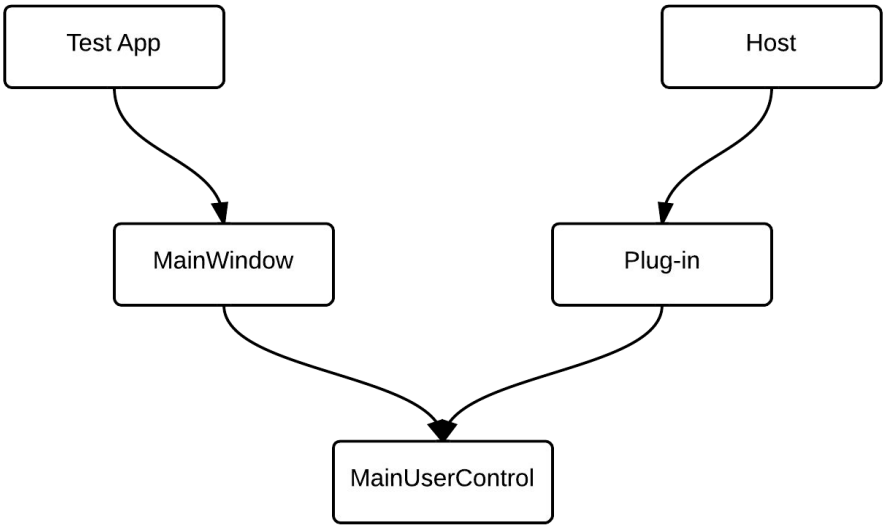
```
    public override bool IsOverlay
    {
        get { return false; }
    }
    public override bool RequiresCredentials
    {
        get { return false; }
    }
    public override string PluginID
    {
        get { return "1"; }
    }
}
```

Alternatively, a plug-in can be implemented as an executable. In this case, the steps are:

1. Create a WPF application.
2. Create a WPF user control, for example, MainUserControl
3. Add MainUserControl to the application's main window.
4. Add a reference to the PluginHostInterfaces assembly. Make sure "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 plug-in's directory
    a. Your plug-in class would look exactly like the preceding example, and your main window XAML should contain nothing but a reference to MainUserControl.

```
<Window x:Class="MyPlugin.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="clr-namespace:MyProject"
  Title="My Plugin" Height="600" Width = "750">
  <Grid>
    <local:MainUserControl />
  </Grid>
</Window>
```

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



Host Services and Plug-In Services

Ops Center Plug-Ins will be required to implement IPlugIn and they will be able to call into the Ops Center using the corresponding IHost Interface. (There are a number of other interfaces that are optional to use as well.)

3.0 Plug-In Key Requirements

Plug-In Writers are required to obtain a Plug-in Developer Key, issued by Pelco.

Plug-in installers must install the .dll or .exe that contains the plug-in class that implements IPlugin into a directory below the “Plugins” directory. Plug-ins may also install other files into that location as well. (See also: 4.0 Plug-In Discovery)

The Ops Center will provide a way for a user to check for available plug-ins. This will cause the ops center to display all the available plug-ins and allow the user to select the one(s) they wish to use.

Once the user selects the plug-in they wish to enable, the Ops Center will launch the plug-in and get the plug-in’s key. If the key is valid the Ops Center will host the plug-in. If it is not valid, the Ops Center will shut down the plug-in and show a warning message to the user. The Ops Center will also provide a way for the user to permanently disable the plug-in.

The important thing here is that Plug-ins without a valid Key will NOT be launched by the Ops Center.

Authentication Steps:
Each Plug-in developer will be issued a secret key that has been verifiably created by Pelco. That key will be provided by the plug-in to the Ops Center at plug-in startup time. The Ops Center will verify that it is a key that has been created by Pelco before allowing the plug-in to proceed.

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

Note: Only plug-ins with a specific type of Plug-in key will be allowed to make IOCCHostSerenity.GetAuthToken() and IOCCHostSerenity.GetBaseURI() API calls.

4.0 Plug-In Discovery

Plugins are discovered using .NET reflection.

Installing a Plugin

To install a plugin, the dll or exe containing the class which inherits from IPlugin must be copied to a directory beneath the “Plugins” directory for the Ops Center Client. The location pointing to the installation directory for the Ops Center Client will be stored in a registry key.
For now this registry key points here. (ProgramData\Pelco\OpsCenter\Plugins). Be sure to get the appropriate path from the registry key in case it changes in the future.

For ANYCPU and 64bit installers the key can be found here…
HKLM\Software\WOW6432\Pelco\OpsCenter
Key=InstallDir Value=The Installation Directory Type=String

For 32bit installers look here...
HKLM\Software\Pelco\OpsCenter
Key=InstallDir Value=The Installation Directory Type=String

Thus the plugins can be found under <<InstallDir>>\Plugins\. It is recommended that each company create their own subfolder and folder for each plugin.
e.g. <<InstallDir>>\Plugins\CompanyName\PluginName\PluginAssembly.dll(exe)

IPlugin attributes used for Discovery:

Field	Type	Required	Description
Name	String	Yes	A friendly name for the Plugin
Version	String	Yes	The Plugin version number. (NOTE: This is NOT the version of the interface that this plug-in uses. This IS the version of this plugin.)
Description	String	Yes	A brief description of the Plugin
Overlay	Boolean	Yes	This field determines if the purpose of the Plugin is to overlay onto a video feed. Default is False.
RequiresCredentials	Boolean	Yes	Indicates that this plug-in requires credentials.
PluginID	String	Yes	Id that the Ops Center uses to identify where credentials go.

5.0 Temporary Files and Persistent Configuration Data

Temporary Files

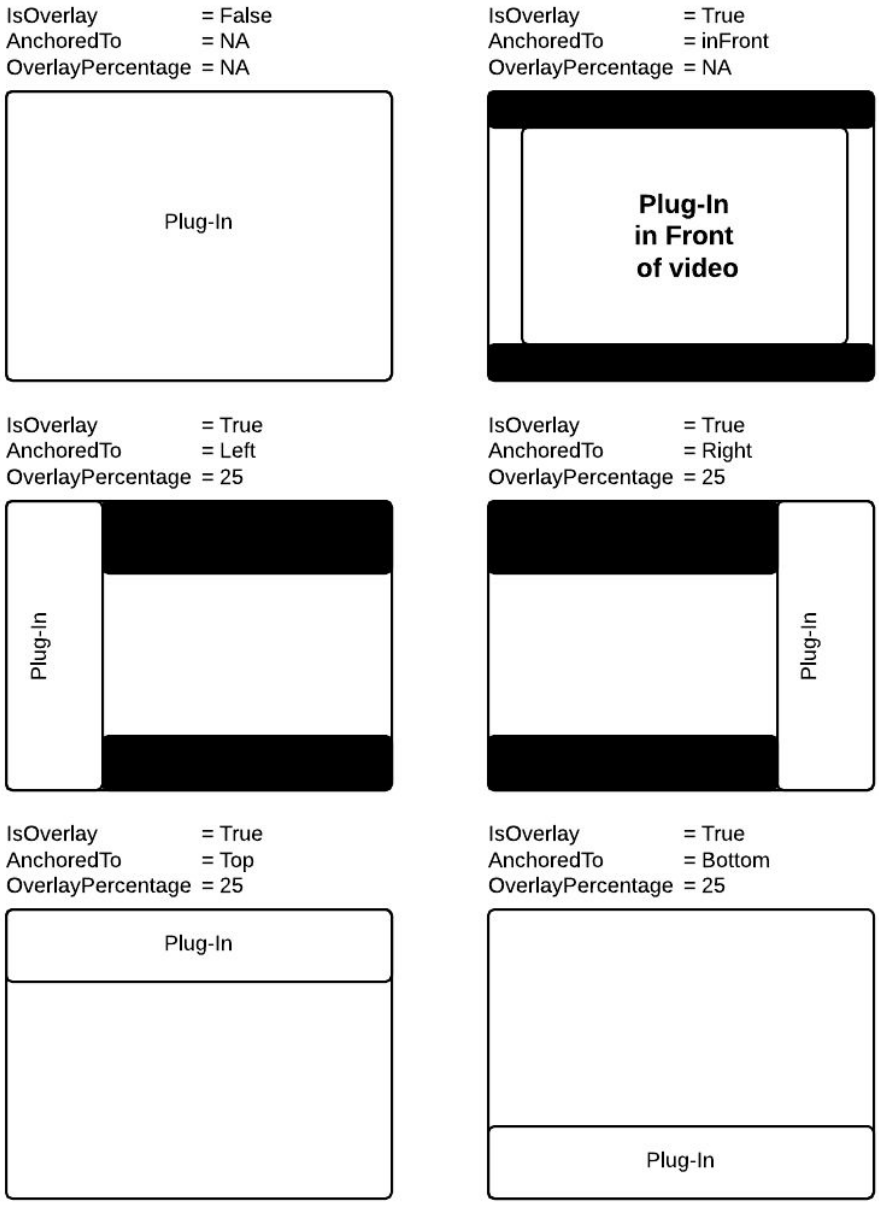
1. A plugin MAY use a users/<currentUser>/AppData/Roaming/<pluginName>/<UUID> directory that it creates to store temporary data while the plugin is running.
2. A plugin MUST NOT rely on this temporary data being available when it starts up as it may be started up on a different Ops Center or the temporary data folders may have been deleted.
    a. (i.e. Persistent Configuration Data MUST NOT be stored here.)

Persistent Configuration Data:

The Ops Center supports saving plugin state as part of saving a workspace. For example if a 4 up workspace contains a plug-in in one of its cells the Ops Center, when appropriate, will call the IOCCPlugin1.GetPluginState() to retrieve the plug-in's current state 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 Ops Center, the Ops Center 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.

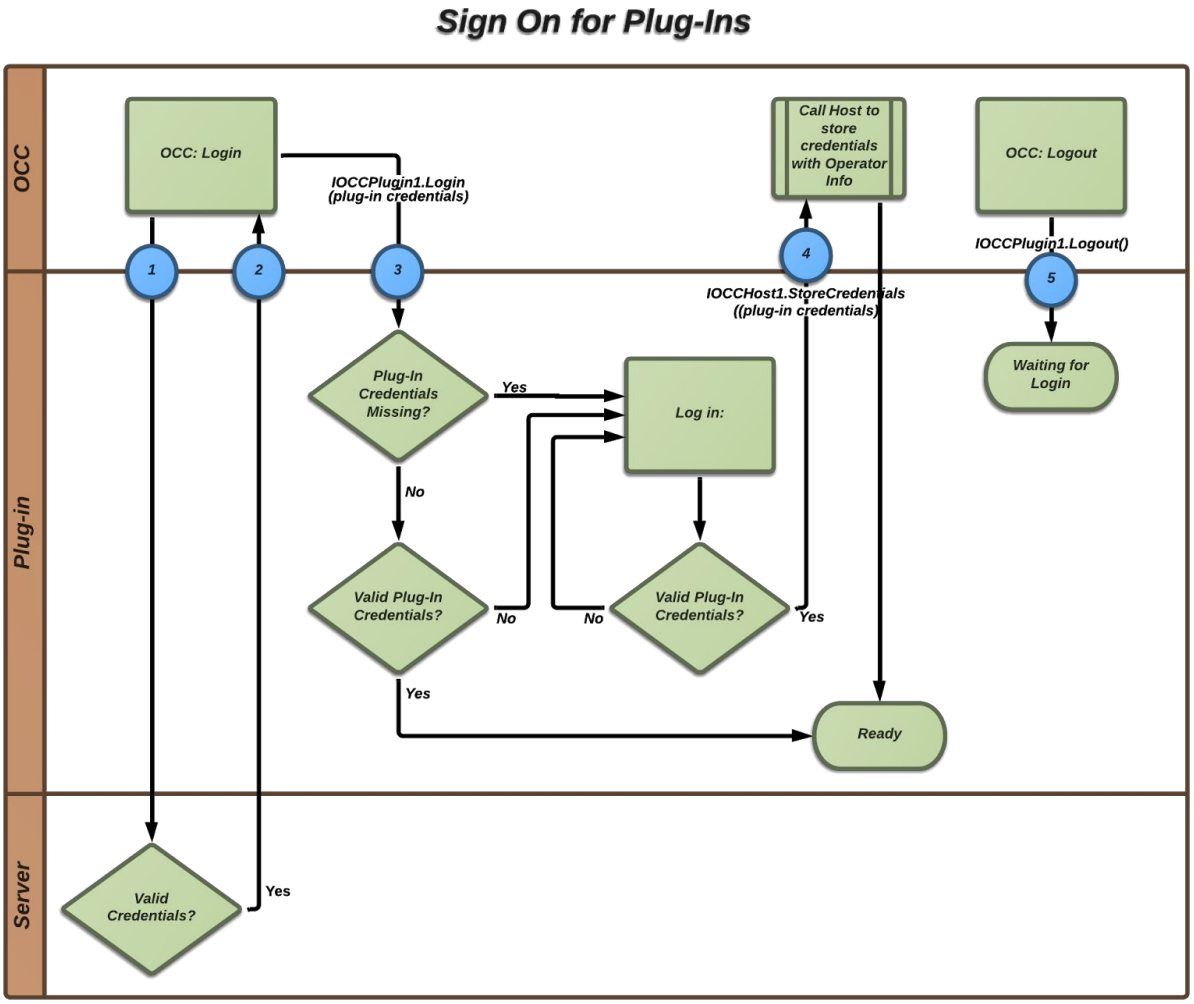
6.0 Overlay Plug-Ins

Plug-ins may be configured, (By returning true for IsOverlay) to be an Overlay Plugin or not. An Overlay plugin is one that occupies the same cell as a video stream. These types of plug-ins are intended to annotate the video stream in some way.
An overlay plugin may change its configuration at run time by calling IOCCHostOverlay.SetOverlayAnchor();



7.0 Sign On for Plug-Ins

A plug-in that needs to authenticate its users so that it can know what kinds of data that a given user has access to will follow the steps in the diagram below. These steps minimize the number of times a user has to log-into plug-ins. Note: Steps 1 and 2 are Ops center steps. They are included in the diagram to give the plug-in writer some context about when it will receive Login calls.



This approach requires the Ops Center store the plug-in credentials “with” the operator’s account.

Important Note:

1. **To do this the Ops Center will create a DataObject and associate it with the Phoenix Client and the current user. This will prevent other clients that don’t care about plug-in credentials from seeing them.**
2. **The DataObject:data field is wide-open to the client as long as its serializable and under**

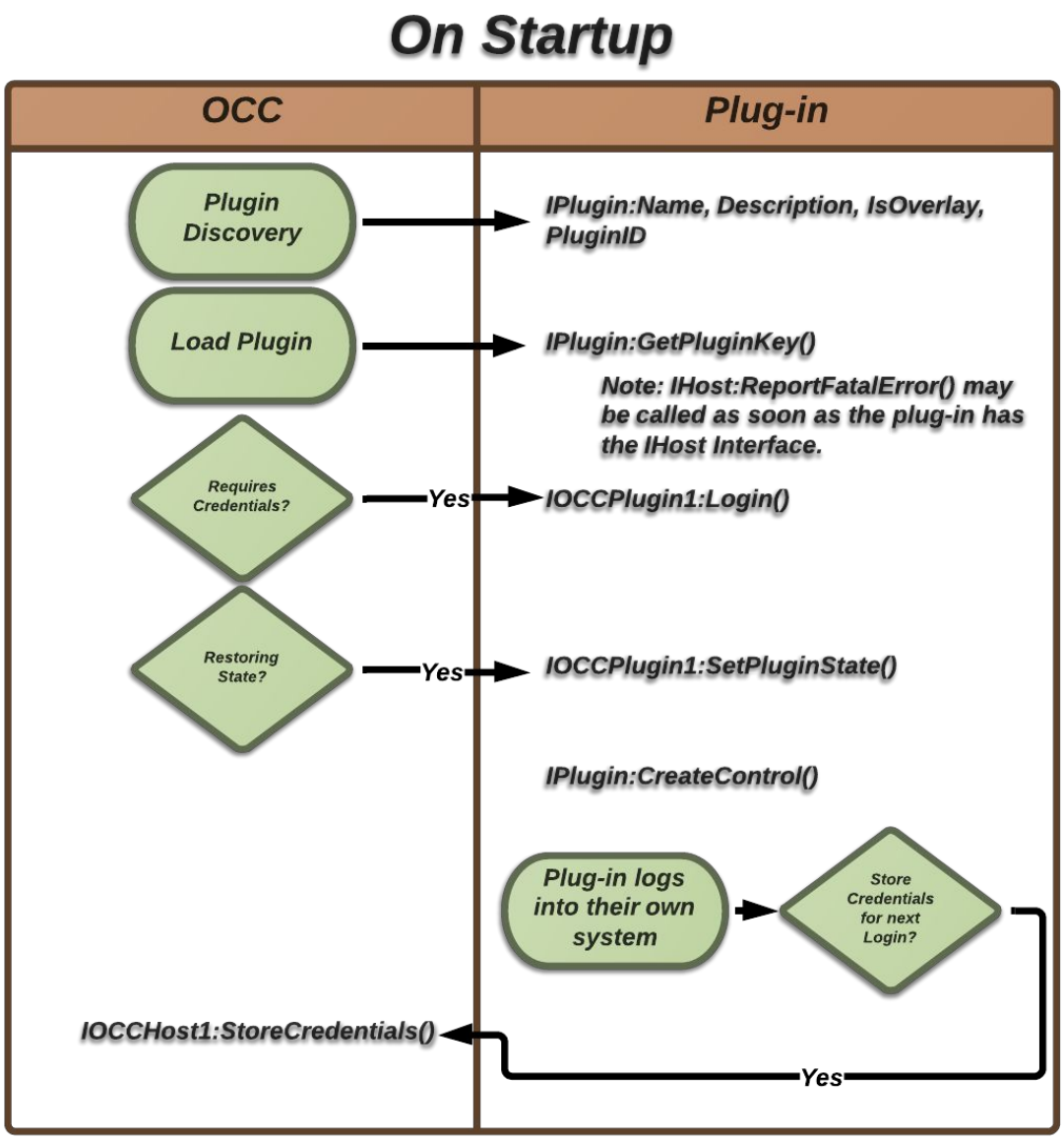
1MB.

3. **The Ops Center Client should wrap the data handed to it from the plug-in so that if future DataObject use happens we can differentiate what kinds of DataObject:data we are looking at.**

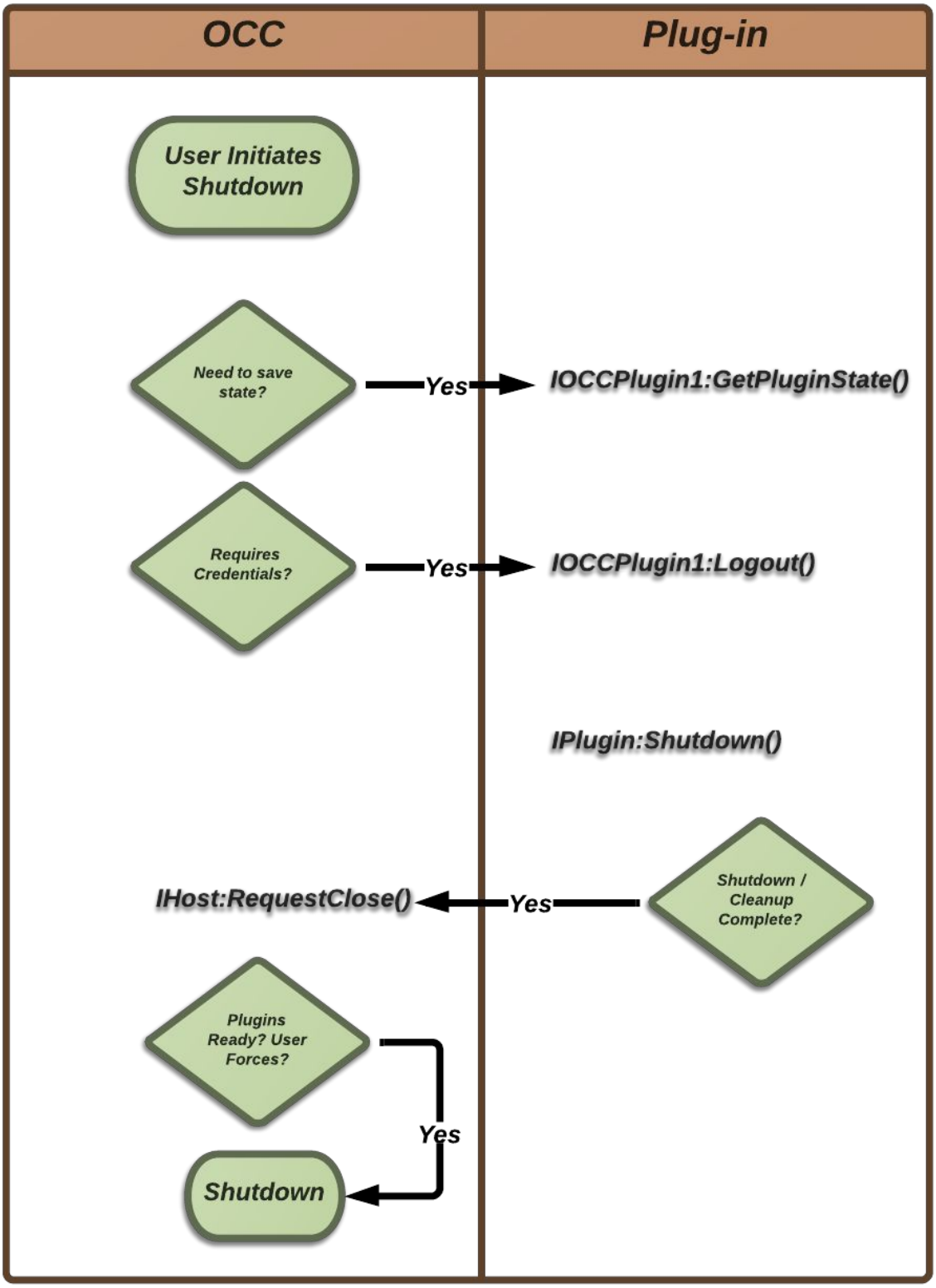
If this is the first time logging in, there will be no plug-in credentials. The Plug-in will request them from the user and then call the host to store them with the currently logged in operator's account.
If this is a subsequent login then the plug-in credentials will be provided. It's up to the plugin 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 plug-in is good to go. If the credentials are not valid, then the plug-in will report that the user is not authorized for this plug-in, and give them an opportunity to log-in.

8.0 Sequence Diagrams

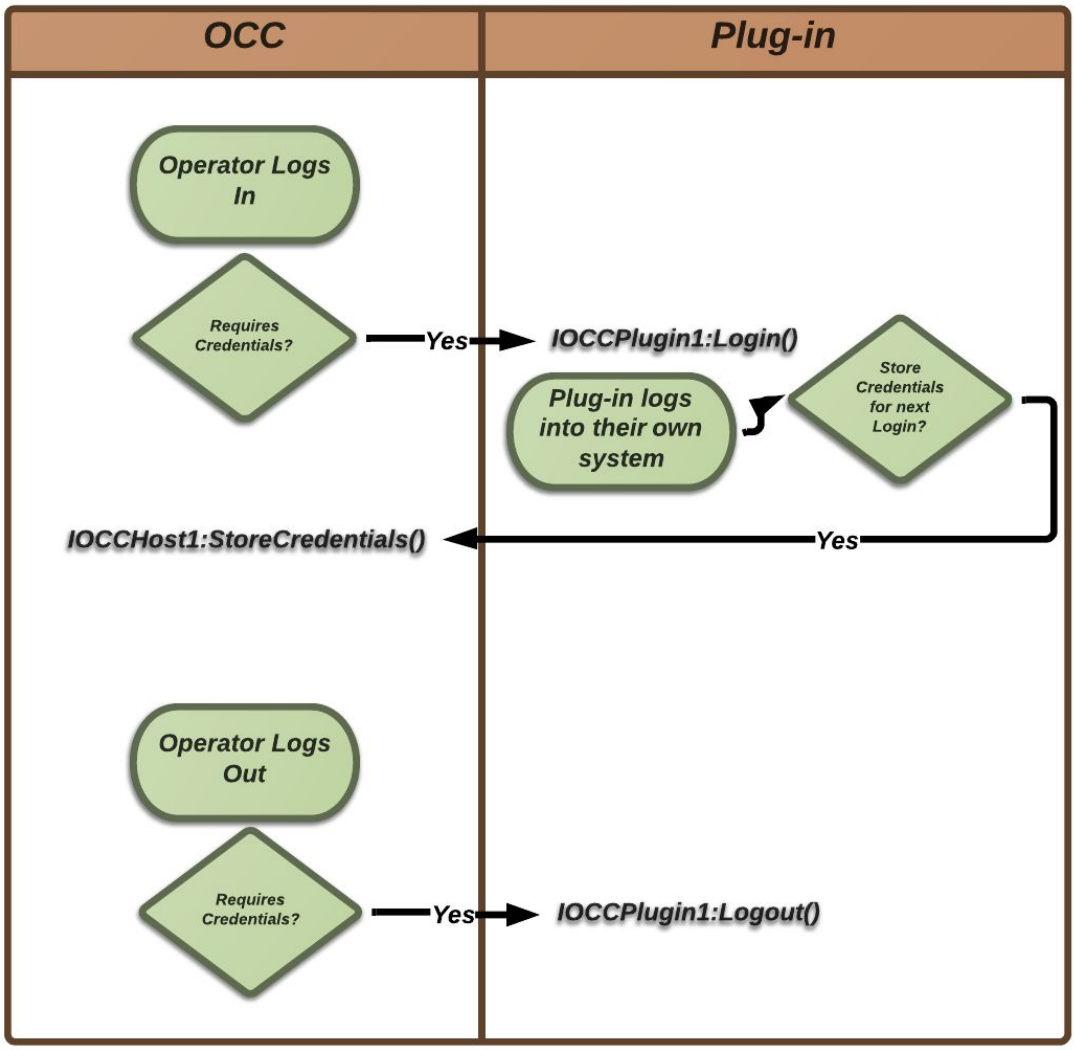
Below are sequence diagrams that show what methods are called and in what order.



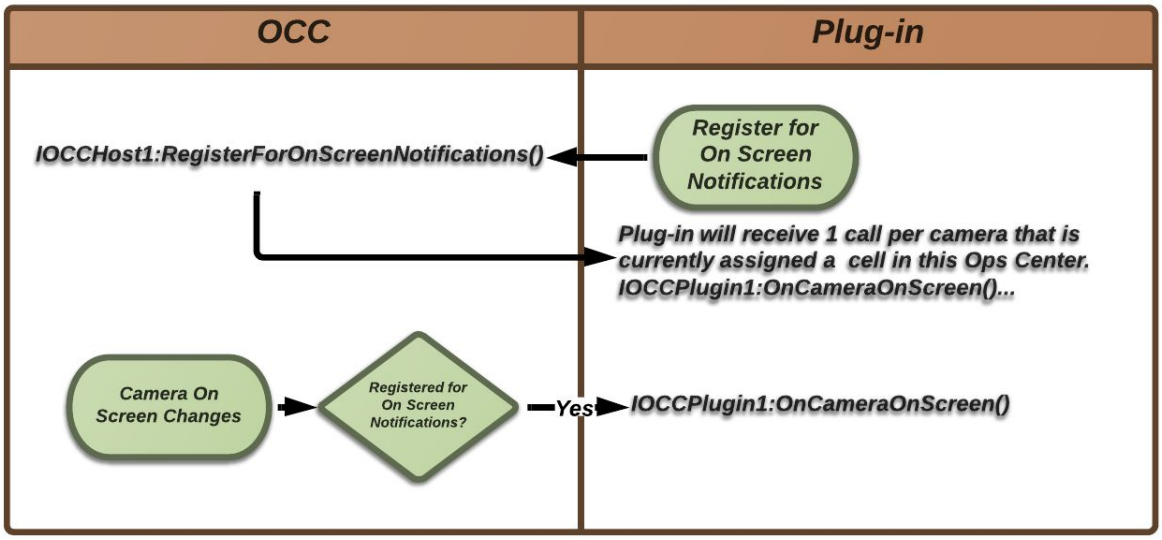
On Shutdown



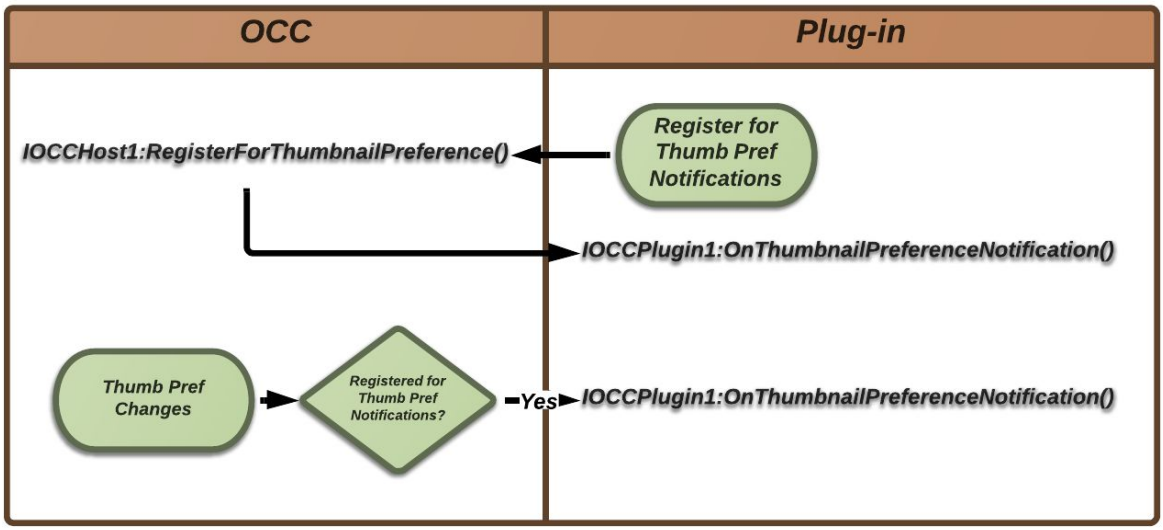
On Operator Login/Logout of OCC



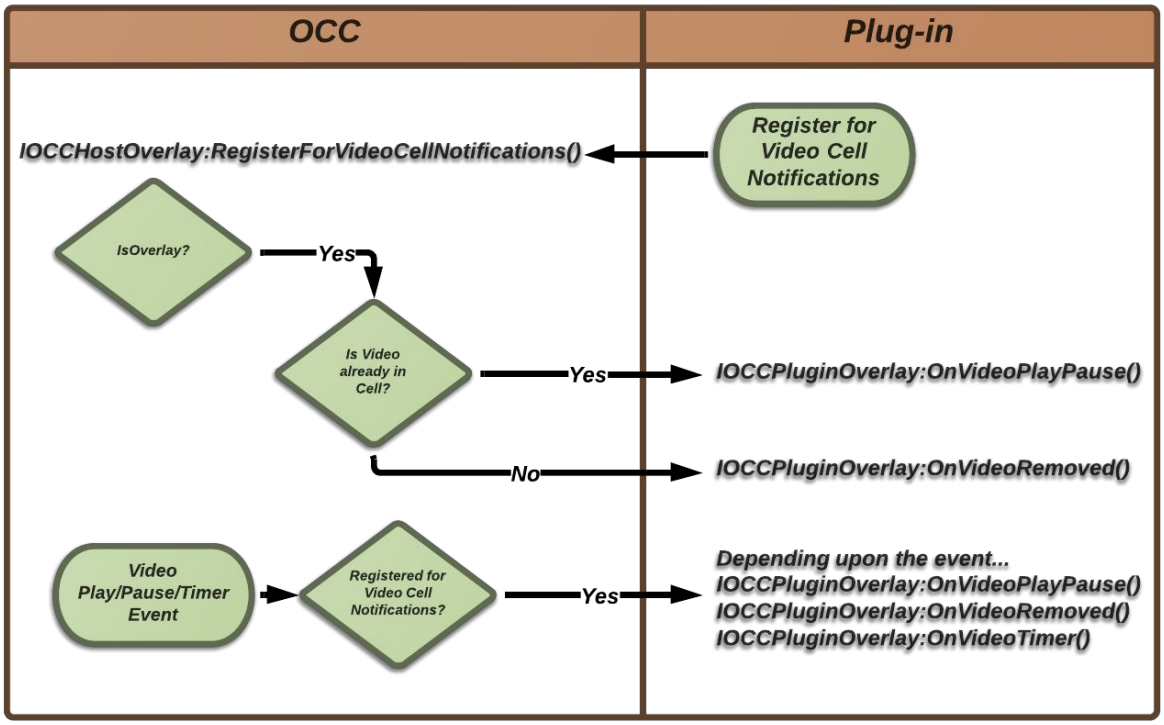
Registering for Camera On Screen Notifications



Registering for Thumbnail Pref Notifications



Registering for Video Cell Notifications



9.0 Class and Interface Definitions

Here are some tips and tricks that will help your plug-in to perform optimally.

1. Be prepared to have the host call into your plugin as soon as your plugin's constructor returns. It will do this to send you information that you will need in order to set up or restore your plug-in's state before it actually becomes visible to the user. (To discover what plug-in APIs can happen before the host calls CreateControl() on your plug-in please refer to the Sequence Diagrams.

2. 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 may make the Ops Center appear to be frozen.
    a. For example:
    b. Do NOT do synchronous server side communication inside an IPlugin or IOCCPlugin call. If the server is slow responding, your plug-in and possibly the entire Ops Center will appear to be frozen. Once again, do the minimum and return.

3. One obvious, or possibly not so obvious race condition that we get into with plug-ins happens when a plug-in spends a bunch of time in CreateControl(). What may happen when the plugin appears to be un-responsive (because its taking a long time to load) is that the user will give up on the plug-in and close it. This kind of a close may cause a hard shutdown of the plug-in, right when it's in the middle of trying to start up.
    a. The easiest way to avoid this is to respond quickly to CreateControl() without completely loading everything. Either put up some kind of "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, its just not done loading yet.

The PluginBase base class for the plugin as well as the above interfaces are described below.

PluginBase.cs

```
public abstract class PluginBase : MarshalByRefObject, IPlugin
{
    public abstract FrameworkElement CreateControl();

    public virtual object GetService(Type serviceType)
    {
        if (serviceType.IsAssignableFrom(GetType())) return this;
        return null;
    }

    public virtual void Dispose()
    {
    }

    public override object InitializeLifetimeService()
    {
        return null; // live forever
    }

    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.cs

```
public interface IPlugin : IServiceProvider, IDisposable
{
    /// <summary>
    /// Creates plugin's visual element; called only once in plugin's lifetime
    /// </summary>
    /// <returns>WPF framework element of the plugin</returns>
    FrameworkElement CreateControl();

    /// <summary>
    /// Vendor Specific Key given to add-in developer by Pelco
    /// </summary>
    /// <returns>string value of the plugin key</returns>
    string GetPluginKey();

    /// <summary>
    ///  Returns the plugin name
    /// </summary>
    /// <returns>string value of the plugin name</returns>
    string Name { get; }

    /// <summary>
    /// Description of the Plugin
    /// </summary>
    /// <returns>string value of the plugin description</returns>
    string Description { get; }

    /// <summary>
    /// Version of the plugin
    /// </summary>
    /// <returns>string value of the plugin version</returns>
    string Version { get; }

    /// <summary>
    /// Overlay plugins must return true.
    /// </summary>
    /// <returns>boolean value</returns>
    bool IsOverlay { get; }

    /// <summary>
    /// Uniquely identifies a vendor specific type of plugin.
    /// (EG. 1 for the first plugin type from this vendor, 2 for the next, etc...)
    /// </summary>
    /// <returns>string value of the plugin ID</returns>
    string PluginID { get; }

    /// <summary>
    /// Called before the Ops Center shuts down a plug-in.  (Note: The plugin should
    /// perform its cleanup and then call IOCCHost#.RequestClose().)
```

```
    /// </summary>
    void Shutdown();
}
```

IOCCPlugin1.cs

```
public interface IOCCPlugin1
{
    /// <summary>
    /// Occurs for thumbnail preference notification
    /// </summary>
    /// <param name="show">true = show thumbnails</param>
    void OnThumbnailPreferenceNotification(bool show);

    /// <summary>
    /// Occurs for camera on screen notifications
    /// </summary>
    /// <param name="Impl">datasource.id of this camera</param>
    /// <param name="onScreen">true = this camera is now up in the ops center,
    /// false = this camera was just removed from the Ops Center</param>
    void OnCameraOnScreeen(string cameraId, bool onScreen);

    /// <summary>
    /// Requests the Plugin's current state.  This state will be passed back to the plugin
    /// on startup so that the Plug-In can resume from it's last known state.  (This is necessary
    /// in support of saving multiple workspace configurations that may contain multiple
    /// mapping plug-ins on an particular Ops Center.)
    /// </summary>
    string GetPluginState();

    /// <summary>
    /// Sets the plug-ins state. (current map, zoom level, etc…)
    /// </summary>
    /// <param name="pluginState">state information that this plugin requires in order to set
    ///  a known state of the plugin.</param>
    void SetPluginState(string pluginState);

    /// <summary>
    /// Return true if this plugin requires credentials
    /// </summary>
    /// <returns>boolean value</returns>
    bool RequiresCredentials { get; }

    /// <summary>
    /// Called for plugins that set the "RequiresCredentials" field in the plugin.xml file to true.
    /// Lets the plug-in know that an operator has logged in and passes along plug-in credentials
    /// if they are available.
    /// </summary>
    /// <param name="credentials">plug-in credentials or null if none exist. e.g.
credentials=encrypt("userName:JohnDoe, password="secret")</param>
    void Login(string credentials);

    /// <summary>
    /// Called for plugins that set the "RequiresCredentials" field in the plugin.xml file to true.
```

```
        /// Lets the plug-in know that the operator has logged out.
        /// </summary>
        void Logout();
    }
```

IOCCPluginOverlay.cs

```
    public interface IOCCPluginOverlay
    {
        /// <summary>
        /// Called when the video that this plug-in is associated with begins playing or is paused.
        /// </summary>
        void OnVideoPlayPause(string dataSourceId, string number, bool live, bool playing,
System.DateTime utcTime);

        /// <summary>
        /// Called every 1 seconds with the current location in the video as a UTC time string
        /// when the video is playing.  Called each time the video is re-positioned when it is paused.
        /// </summary>
        /// <param name="utcTime">timestamp of current location in video stream</param>
        void OnVideoTimer(System.DateTime utcTime);

        /// <summary>
        /// Called when the video that this plug-in is associated with is removed from the ops center.
        /// If the cell is then populated with a different video you will receive an OnVideoPlayPause() call.
        /// </summary>
        void OnVideoRemoved();
    }
```

IOCCPluginReserved.cs

```
    public interface IOCCPluginReserved
    {
        /// Sets the datasource id of the camera.
        /// </summary>
        /// <param name="dataSourceID">Datasource id of the selected media source</param>
        /// <param name="number">Datasource.number field of the selected media source</param>
        void ReservedSetDataSource(string dataSourceID, string number);
    }
```

IOCCPluginJoystick.cs

```
public interface IOCCPluginJoystick
{
    /// <summary>
    /// These messages are sent periodically when the joystick is not at rest.
    /// Each axis holds a value that is mapped to the extents that were passed
    /// in through RegisterForJoystickNotifications();
    /// </summary>
    /// <param name="X">x axis value(Note: left or right on the 3d mouse)</param>
    /// <param name="Y">y axis value (Note: Push forward or pull backward on the 3d mouse)</param>
    /// <param name="Z">z axis value (Note: Twist the 3d mouse)</param>
    /// <param name="U">u axis value (Note: Pull up or press down on the 3d mouse)</param>
    void OnJoystickNotification(int X, int Y, int Z, int U);
}
```

IHost.cs

```
public interface IHost : IServiceProvider
{
    /// <summary>
    /// Reports fatal plugin error to the host; the plugin will be closed
    /// </summary>
    /// <param name="userMessage">Message explaining the nature of the error</param>
    /// <param name="fullExceptionText">Exception call stack as string</param>
    void ReportFatalError(string userMessage, string fullExceptionText);

    /// <summary>
    /// ID of the host process used by ProcessMonitor
    /// </summary>
    int HostProcessId { get; }

    /// <summary>
    /// A plug-in calls this method once it is ready to be unloaded.
    /// </summary>
    void RequestClose();

    /// <summary>
    /// Ask the Ops Center to make the plug-in full screen.  This will temporarily obscure other
    /// parts of the user interface that reside on the same monitor.
    /// Note: If the user presses the Escape key the plug-in should call the
    ///  ReturnFromFullScreen API.
    /// </summary>
    void RequestFullScreen();

    /// <summary>
    /// This will return the plugin back to its original cell if it is in Full Screen mode.
    /// </summary>
    void ReturnFromFullScreen();
}
```

LaunchContent.cs - For Drag / Drop operations.

```
namespace Pelco.Phoenix.PluginHostInterfaces
{
   /// <summary>
   /// Helper Methods to be used with the LaunchContent classes.
   /// </summary>
   public static class LaunchContentHelpers
   {
      /// <summary>
      /// Extension method to help determine if the DataObject contains
      /// valid LaunchContent
      /// </summary>
      /// <returns><c>true</c> if content is valid; otherwise, <c>false</c></returns>
      public static bool IsValidLaunchContent(this System.Windows.DragEventArgs e)
      {
         string[] formats = e.Data.GetFormats();
         object obj = e.Data.GetData(formats[0]);
         // Object contains a single valid item.
         if (typeof(LaunchContent).IsAssignableFrom(obj.GetType()))
         {
            return true;
         }
         // Object contains a list of valid items.
         if (obj is IList && (obj as IList).IsListOf(typeof(LaunchContent)))
         {
            return true;
         }
         // Object contains no valid items.
         return false;
      }

      /// <summary>
      /// Extension method to extract content from the DataObject.
      /// </summary>
      /// <returns>If the DataObject contains valid LaunchContent, the results will be added to a list.
Otherwise,
      /// the list will be empty.</returns>
      public static IList<LaunchContent> GetValidLaunchContent(this System.Windows.DragEventArgs e)
      {
         // Create the empty list.
         IList<LaunchContent> contents = new List<LaunchContent>();
         string[] formats = e.Data.GetFormats();
         object obj = e.Data.GetData(formats[0]);
         // Object contains a single valid item.
```

```
        if (typeof(LaunchContent).IsAssignableFrom(obj.GetType()))
        {
            // Add the single valid item.
            contents.Add(obj as LaunchContent);
        }
        // Object contains a list of valid items.
        if (obj is IList && (obj as IList).IsListOf(typeof(LaunchContent)))
        {
            // Add the list of valid item(s).
            foreach (LaunchContent content in (obj as IList))
            {
                contents.Add(content as LaunchContent);
            }
        }
        return contents;
    }

    public static bool IsListOf(this IList items, Type compareType)
    {
        if (items == null)
        {
            return false;
        }
        else
        {
            return items.GetType().GetProperty("Item").PropertyType.IsA(compareType);
        }
    }

    public static bool IsA(this Type thisType, Type compareType)
    {
        if (thisType == compareType) return true;
        var rv = compareType.IsAssignableFrom(thisType);
        return rv;
    }
}

/// <summary>
/// Basic LaunchContent Class. All implementations must derive from this abstract class.
/// </summary>
[Serializable]
public abstract class LaunchContent
{
    /// <summary>
    /// Reserved Parameters used for Ops Center purposes.
    /// </summary>
```

```
        public IDictionary<string, string> ReservedParams { get; set; }
    }

    /// <summary>
    /// Used to launch a single DataSource
    /// </summary>
    [Serializable]
    public class LaunchDataSource : LaunchContent
    {
        /// <summary>
        /// Unique ID of the DataSource
        /// </summary>
        public string DataSourceId { get; set; }
```

Warning this will NOT be implemented in the first release of OC that supports drag / drop for plugins.

```
        /// <summary>
        /// Amount of time, in seconds, for the video to loop. Zero means play forever.
        /// </summary>
        public TimeSpan? LoopDuration { get; set; }

        /// <summary>
        /// Start the video playing or paused.
        /// <c>true</c> for playing; otherwise, <c>false</c> for paused.
        /// </summary>
        public bool IsPlaying { get; set; }

        /// <summary>
        /// Time to start the video in Utc. If null, start the video live.
        /// </summary>
        public DateTime? StartTimeUtc { get; set; }
    }

    /// <summary>
    /// Used to launch a single Content Plugin.
    /// </summary>
    [Serializable]
    public class LaunchContentPlugin : LaunchContent
    {
        /// <summary>
        /// Optional Arguments to be sent to the plugin on load.
        /// </summary>
        public string PluginArgs { get; set; }

        /// <summary>
        /// Unique Id of the Plugin.
        /// </summary>
```

```
        public string PluginId { get; set; }

        /// <summary>
        /// Key of the Plugin Manufacturer.
        /// </summary>
        public string PluginKey { get; set; }
    }

    /// <summary>
    /// Used to launch a single Overlay Plugin.
    /// </summary>
    [Serializable]
    public class LaunchOverlayPlugin : LaunchDataSource
    {
        /// <summary>
        /// Optional Arguments to be sent to the plugin on load.
        /// </summary>
        public string PluginArgs { get; set; }

        /// <summary>
        /// Unique Id of the Plugin.
        /// </summary>
        public string PluginId { get; set; }

        /// <summary>
        /// Key of the Plugin Manufacturer.
        /// </summary>
        public string PluginKey { get; set; }
    }
}
```

IOCCHostRegisterForDragDrop.cs

```
public interface IOCCHostRegisterForDragDrop
{
        /// <summary>
        /// Tells the host that this plugin would like to start or stop
        /// receiving DragDrop notifications.
        /// </summary>
        /// <param name="send"> true=send notifications. false= do NOT send.</param>
        void RegisterForDragDrop(bool send);
}
```

IOCCHost1.cs

```
public interface IOCCHost1
{
	/// <summary>
	/// Tells the host that this plugin would like to initiate streaming from this list of cameras
	/// </summary>
	/// <param name="cameraList">the List<DataSource.Id> of the cameras</param>
	void InitiateStreams(List<string> cameraList);

	/// <summary>
	/// Tells the host that this plugin would like to start or stop
	/// receiving ThumbnailPreference notifications
	/// Immediately after a call to RegisterForThumbnailPreference(true), the corresponding
	/// IOCCPlugin1.OnThumbnailPreferenceNotification() will get called to set the current state.
	/// </summary>
	/// <param name="send"> true=send notifications. false= do NOT send.</param>
	void RegisterForThumbnailPreference(bool send);

	/// <summary>
	/// Tells the host that this plugin would like to know when a camera is currently
	/// assigned a cell in the ops center.
	/// Immediately after a call to RegisterForOnScreenNotifications(true), the corresponding
	/// IOCCPlugin1.OnCameraOnScreen() will get called, possibly multiple times,
	///  to set the current state.
	/// </summary>
	/// <param name="send"> true=send notifications. false= do NOT send.</param>
	void RegisterForOnScreenNotifications(bool send);

	/// <summary>
	/// Causes the host to store plug-in credentials with the currently logged in operator’s account.
	/// Note: The credential string should not exceed 10k
	/// It is recommended that the credential string be a base64 encoded binary blob that contains
	/// an encrypted username/password.
	/// </summary>
	/// <param name="credentials"> plug-in credentials </param>
	void StoreCredentials(string credentials);
}
```

IOCCHostOverlay.cs

```
public enum AnchorTypes { inFront=0, left=1, top=2, right=3, bottom=4 };

public interface IOCCHostOverlay
{
	/// <summary>
	/// Tells the host that this plugin would like to start or stop
	/// receiving VideoCellNotifications  Immediately after a call to
	/// RegisterForVideoCellNotifications(true) for a cell that contains video the
	/// IOCCPlugin1.OnVideoPlayPause() will get called to set the current state.  If the cell has
	///  no video then IOCCPlugin1.OnVideoRemoved() will be called.
	/// </summary>
	/// <param name="send"> true=send notifications. false= do NOT send.</param>
	void RegisterForVideoCellNotifications(bool send);

	/// <summary>
	/// Overlay plug-ins call the host to tell it how they wish to be configured.
	/// Overlay plug-ins are allowed to call this api to change their configuration at any time.
	/// If this call is NOT made the default configuration is…
	///    anchoredTo = inFront
	///    percentage = NA, min = NA, max = NA
	/// If min = max = some value, then the width of the plugin will = some value.
	/// Note: When anchoredTo is inFront, all other parameters are ignored.
	/// </summary>
	/// <param name=”anchoredTo”> inFront,  left, top, right, bottom</param>
	/// <param name=”percentage”> 25.5= occupy 25.5% of the cell</param>
	/// <param name=”min”> min allowed width/height in pixels.</param>
	/// <param name=”max”> max allowed width/height in pixels.</param>
	void SetOverlayAnchor(AnchorTypes anchoredTo, double percentage, int min, int max);
}
```

IOCCHostSerenity.cs

```
 public interface IOCCHostSerenity
{
    /// <summary>
    /// Gets the authentication token that can be used by client.
    /// Note: Must have valid plug-in key that specifies “Use Serenity”
    /// <returns>The authentication token to be used when making Serenity calls</returns>
    /// </summary>
    string GetAuthToken();

    /// <summary>
    /// Gets the Base URI of Serenity host  for client to interact with.
    /// Note: Must have valid plug-in key that specifies “Use Serenity”
    /// <returns>endpoint of the Serenity System e.g.: https://{IP}:{PORT} </returns>
    /// </summary>
    string GetBaseURI();
}
```

IOCCHostJoystick.cs

```
   public interface IOCCHostJoystick
{
    /// <summary>
    /// Tells the host that this plugin would like to start or stop
    /// receiving JoystickNotifications.  Once the plugin has registered
    /// to receive joystick notifications, subsequent joystick changes will
    /// be sent to the plugin via OnJoystickNotification(...);
    /// </summary>
    /// <param name="send"> true=send notifications. false= do NOT send.</param>
    void RegisterForJoystickNotifications(bool send);

    /// <summary>
    /// Sets the limits of all four axis of the joystick.  The defaults are -100 to 100 for every axis.
    /// </summary>
    /// <param name="xMin">the value to send when the joystick is at its extent (EG. Pressed left)
</param>
    /// <param name="xMax">the value to send for the opposite extent.(EG. Pressed right)</param>
    /// <param name="yMin">the value to send when the joystick is at its extent (EG. forward) </param>
    /// <param name="yMax">the value to send for the opposite extent.(EG. backward)</param>
    /// <param name="zMin">the value to send when the joystick is at its extent (EG. twisted counter
clockwise) </param>
    /// <param name="zMax">the value to send for the opposite extent.(EG. twisted clockwise)</param>
```

```
        /// <param name="uMin">the value to send when the joystick is at its extent (EG. pulled up)
</param>
        /// <param name="uMax">the value to send for the opposite extent.(EG. pressed down)</param>
        void SetJoystickLimits(int xMin, int xMax, int yMin, int yMax, int zMin, int zMax, int uMin, int uMax);

        /// <summary>
        /// Tells the host that the user pressed the escape key.
        /// The host uses this information to deactivate an active cell
        /// which causes the joystick to be put into a navigation mode and joystick messages will be halted.
        /// until the cell is activated again for pan / tilt/ zoom behavior.
        /// </summary>
        void EscJoystick();
    }
```

IOCCHostSetVideoPosition.cs

```
    public interface IOCCHostSetVideoPosition
    {
        /// <summary>
        /// Allows an overlay plugin to tell the host to position the current video stream at a
        /// specific UTC time.  If that time is available the video will jump to that time and begin
        /// playback from there.
         /// <param name="utcTime">position in UTC time to play the video</param>
       void SetVideoPosition(System.DateTime utcTime);
    }
```