Control Center Integration Software Development Kit (ISDK) Guide

Everbridge Suite

Contents

Control Center Integrations Software Development Kit (IDSK)

Control Center is a PSIM software-based integration and management platform. It connects and manages disparate building and security technologies such as video surveillance, life critical systems, radar, analytics, HVAC, PIDS, GPS tracking and GIS mapping.

The Control Center Integration Software Development Kit (ISDK) is a set of programs and related files that enable you to develop new connectors that let Control Center communicate with specific security devices in your security solution.

The Control Center connector allows you to implement API and protocol functions implemented by a specific security device, providing you with the ability to control and monitor that device in Control Center. This allows you coordinated control and monitoring of disparate devices through Control Center, improving performance of your security solution.

Product Naming Changes

The following table describes product name changes.

Previous Name	New Name
IPSecurityCenter	From version 5.25 onwards, IPSecurityCenter was renamed Control Center.
DDK	From version 5.30 onwards, Driver Development Kit was renamed Integrations Software Development Kit.
Driver	From version 5.30 onwards, drivers were renamed connectors.
Addon	From version 5.30 onwards, addons were renamed extensions.

Setting Up Control Center ISDK Environment

The following software must be installed on the machine where you are developing your driver and where you are going to install Control Center ISDK.

CAUTION: Uninstall any old versions before installing new versions.

- Visual Studio. Make sure the following is installed as part of Visual Studio:
  - VS 2019 Entity Framework Powertools
  - VS 2019 DSL
- Reshaper (latest version)
- StyleCop for Resharper

Setting up Resharper and StyleCop in Visual Studio

To do this:

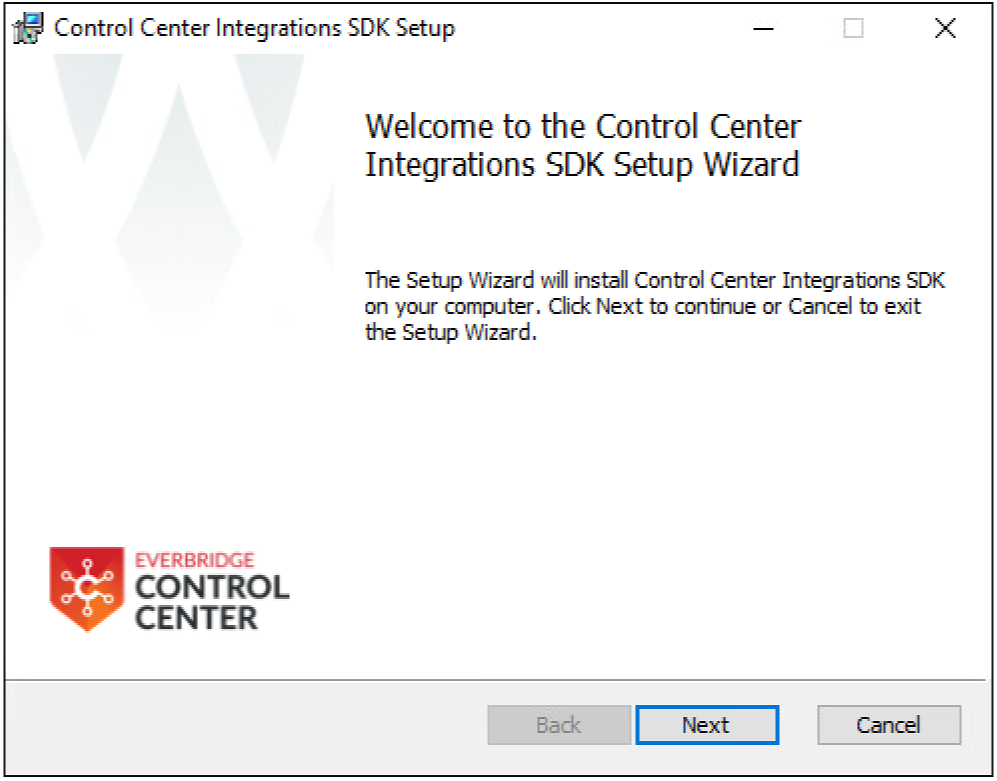
1. From Visual Studio, select **Resharper** > **Manager Options**.
2. Add layers for:
   - **CNL.Resharper**
   - **CNL.StyleCop**
3. Uncheck the **StyleCop.StyleCop** extension.
4. Setup the Visual Studio Reshaper context menus by selecting, **Resharper** > **Options** > **Keyboards & Menus** and deselect **Hide overridden Visual Studio** option.

Installing Control Center ISDK

To install Control Center ISDK :

1. Make sure your system meets the requirements, see Setting Up Control Center DDK Environment.
2. Close all instances of Visual Studio.
3. Browse to the location of your Control Center ISDK installation package
4. Double-click **Everbridge.ControlCenter.ISDK.Install.msi**. The **Control Center Integrations SDK Setup Wizard** displays.

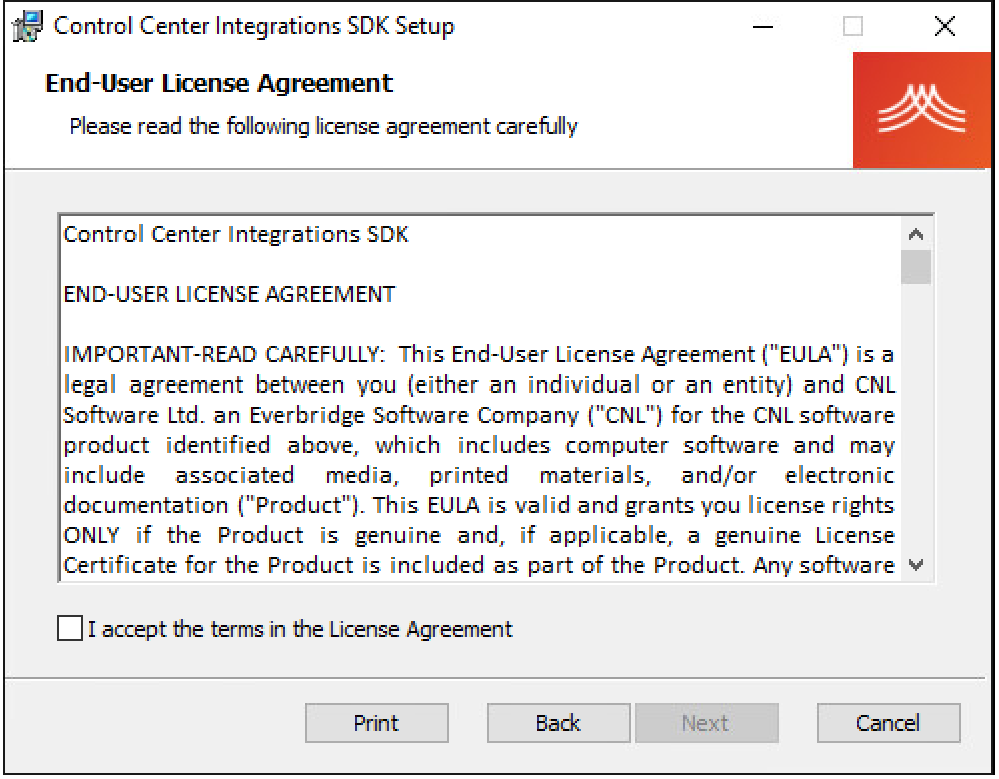
5. From **Welcome**, select **Next**.



Welcome to the Control Center Integrations SDK Setup Wizard

The Setup Wizard will install Control Center Integrations SDK on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

6. From **End-User License Agreement**, select **I accept the terms in the License Agreement**.



End-User License Agreement

Please read the following license agreement carefully

Control Center Integrations SDK

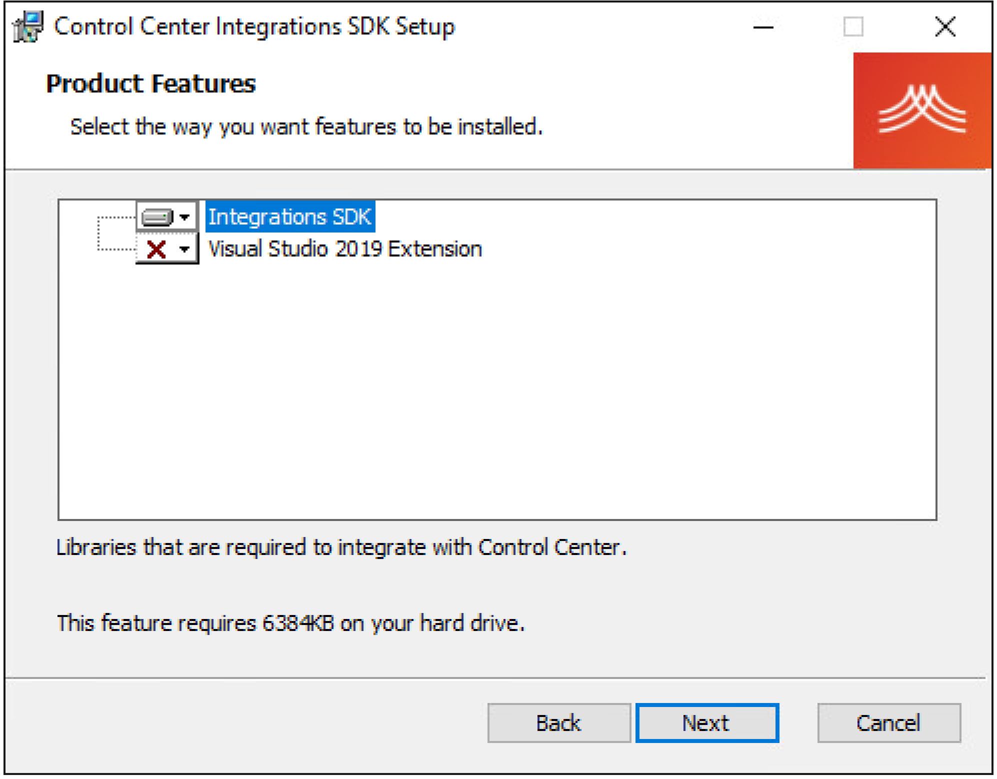
END-USER LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or an entity) and CNL Software Ltd. an Everbridge Software Company ("CNL") for the CNL software product identified above, which includes computer software and may include associated media, printed materials, and/or electronic documentation ("Product"). This EULA is valid and grants you license rights ONLY if the Product is genuine and, if applicable, a genuine License Certificate for the Product is included as part of the Product. Any software

□ I accept the terms in the License Agreement

7. Select **Next**.

8. From **Product Features**, select the way you want features to be installed.



Control Center Integrations SDK Setup

Product Features

Select the way you want features to be installed.

- Integrations SDK
- ☒ Visual Studio 2019 Extension

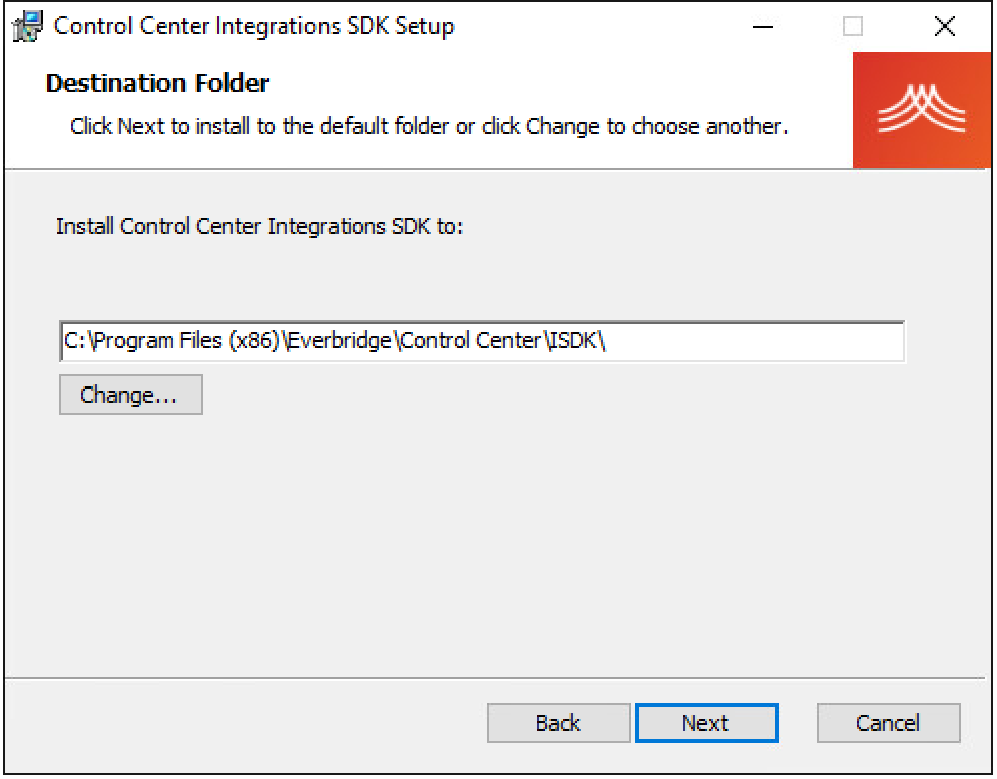
Libraries that are required to integrate with Control Center.

This feature requires 6384KB on your hard drive.

NOTE: The wizard prompts you to close Visual Studio, if you have Visual Studio open, as you cannot proceed with the installation if Visual Studio is open.

9. Select **Next**.

10. From **Destination Folder**, you can either accept the default installation folder or select **Change** and browse to a new location.



Control Center Integrations SDK Setup

Destination Folder

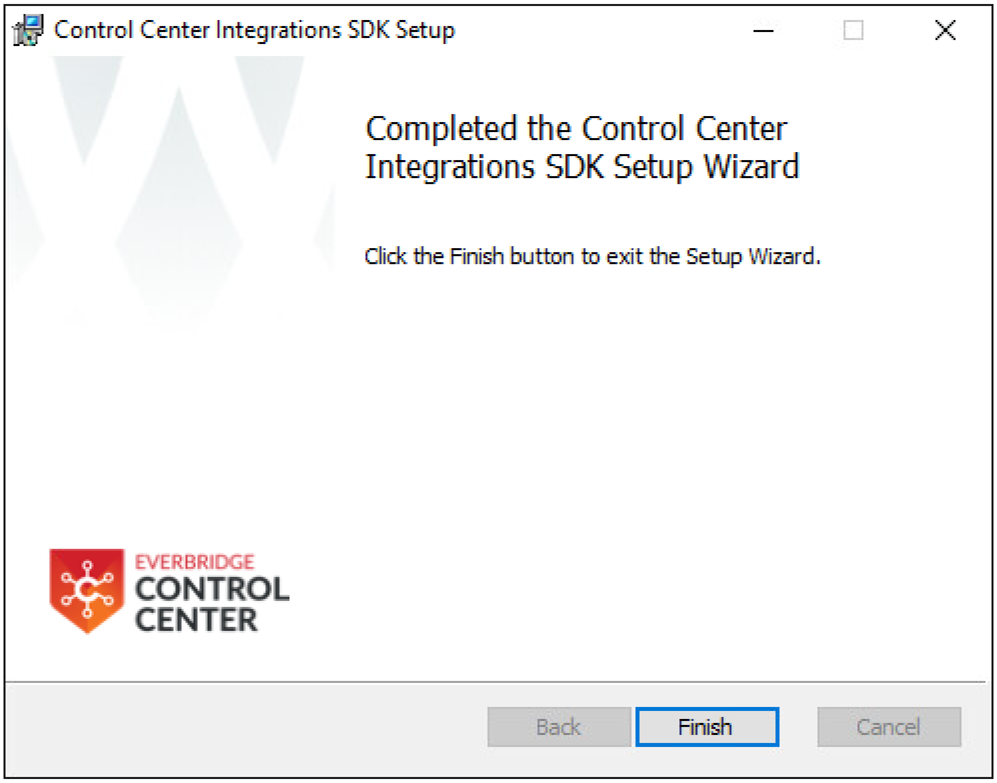
Click Next to install to the default folder or click Change to choose another.

Install Control Center Integrations SDK to:

C:\Program Files (x86)\Everbridge\Control Center\ISDK\

11. Select **Next**.
12. From **Ready to install Control Center Integrations SDK**, select **Install**.

13. Once Control Center Integrations SDK is installed, select **Finish** to close the **Control Center Integrations SDK Setup Wizard**.



Control Center Integrations SDK Setup

Completed the Control Center Integrations SDK Setup Wizard

Click the Finish button to exit the Setup Wizard.

Installing Connectors in Control Center

An overview of the process for installing device drivers into a Control Center solution, is described below.

1. The device driver is installed using the **Device Driver Manager** option within the **System Configuration** window. The driver package is then sent to the **Server** service.
2. The **Server** service receives and loads the newly installed device driver and then informs the **Notification** service of the update.
3. The **Notification** service then notifies the Connection Manager that a new device driver is available for download.
4. Once the Connection Manager has downloaded the new device driver, it instructs the **Notification** service that the new driver has been loaded and read for use.
5. The **Notification** service then notifies all clients in the solution that a new driver has been loaded.
6. Any clients without a copy of the new device driver download the new driver from the **Server** service.

Control Center Connector Architecture

Connectors integrate with various 3rd party systems (aka subsystems).

Each connector is released as a driver package (a file with .ipscdriver extension). Driver package contains:

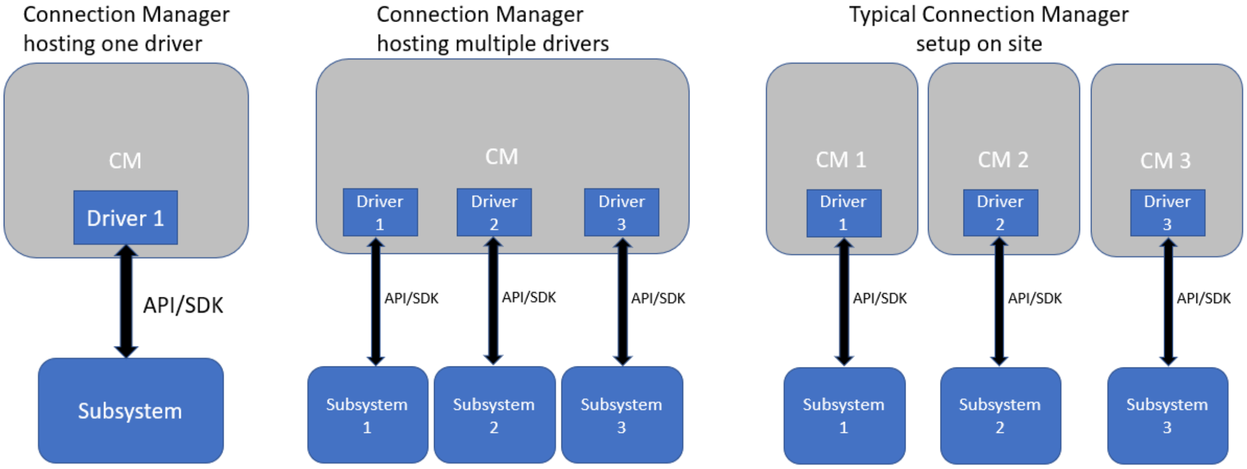
- Driver DLL
- ISDK libraries
- 3rd party SDK files
- log4Net DLL to log messages

All the driver packages are loaded by Connection Manager services. Everbridge recommends that you have multiple Connection Manger services, with one Connection Manager service per driver.

After driver package is installed, it is copied into two folders on a PC hosting Connection Manager service:

- `C:\ProgramData\Everbridge\ControlCenter\Connection Manager\Default\Packages` - driver package copies
- `C:\ProgramData\Everbridge\ControlCenter\Connection Manager\Default\Extracted` - extracted (unzipped) driver packages

Connection Manager - driver hosting



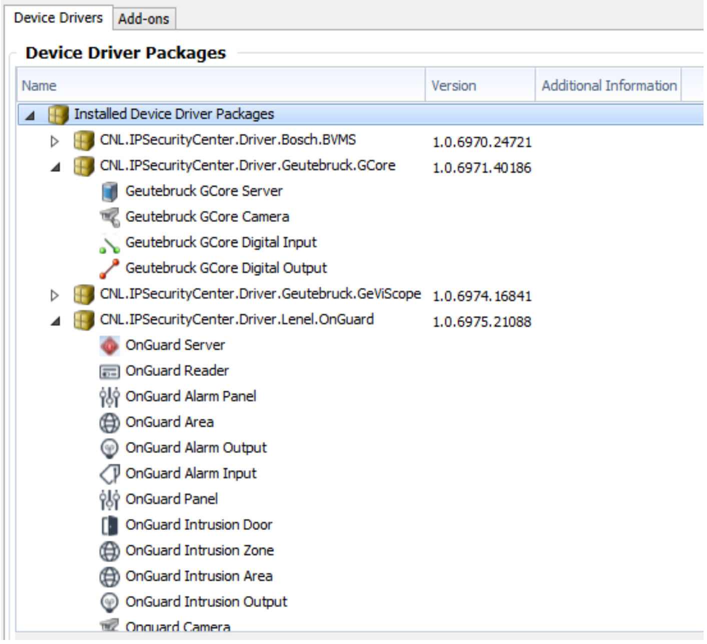
Control Center Connector Structure

All devices consist of the following:

NOTE: The states, properties, events and methods that a device has depends on the type of device.

Concept	Description
Types	The type of device.
States	The state of the device. For example, camera states may include Online, Offline, Failed, Warning, Connecting and so on.
Properties	The properties you may need to use on a device.
Events	The events you can action on a device from your driver.
Methods	The commands a driver can send to a subsystem device.

Each Control Center connector defines one or more Control Center device types.



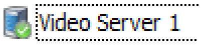
Name	Version	Additional Information
Installed Device Driver Packages		
CNL.IPSecurityCenter.Driver.Bosch.BVMS	1.0.6970.24721	
CNL.IPSecurityCenter.Driver.Geutebruck.GCore	1.0.6971.40186	
Geutebruck GCore Server		
Geutebruck GCore Camera		
Geutebruck GCore Digital Input		
Geutebruck GCore Digital Output		
CNL.IPSecurityCenter.Driver.Geutebruck.GeViScope	1.0.6974.16841	
CNL.IPSecurityCenter.Driver.Lenel.OnGuard	1.0.6975.21088	
OnGuard Server		
OnGuard Reader		
OnGuard Alarm Panel		
OnGuard Area		
OnGuard Alarm Output		
OnGuard Alarm Input		
OnGuard Panel		
OnGuard Intrusion Door		
OnGuard Intrusion Zone		
OnGuard Intrusion Area		
OnGuard Intrusion Output		
Onguard Camera		

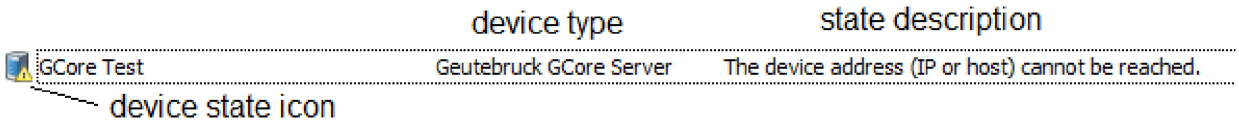
Notes:

- Each device type describes a 3rd party entity:
  - physical device
  - service
  - server
  - physical/logical 3rd party entity (input, output, LED and so on)
- Each Control Center device has properties, methods (actions), and events. See About Devices for more information.
- Control Center devices can be connectable. This means a Control Center device can create a connection to a 3rd party server or an individual device.

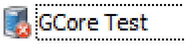
Connectable Devices

Connectable devices have properties that store the 3rd party device connection details. For example, IP address, port number, username, password and so on. Everbridge recommends that drivers have only one connectable device connecting to the main 3rd party server. A connectable device connects and authenticates via the 3rd party server when a Control Center user enables the device. See About Installing Drivers for more information about enabling devices.

Once connected, the device goes to **Online** state  Video Server 1 . If the device fails to connect or loses connection, the devices goes to **Failed** state.



GCore Test	Geutebruck GCore Server	The device address (IP or host) cannot be reached.

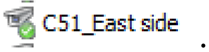
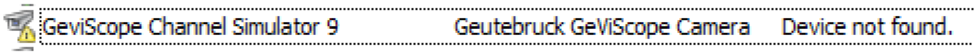
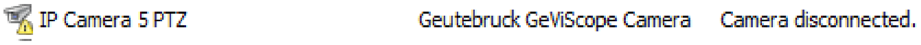
To disconnect, a Control Center user must disable the server device. The driver logs out from the server device goes to **Disabled** state  GCore Test .

Non-Connectable Devices

Other non-connectable devices are usually created automatically by the main server device, once the driver connects to the subsystem. Within Control Center, a server device is called a parent device and the automatically created devices are called child devices. Non-connectable devices do not have connection properties.

When a child device is enabled, it does not connect to a subsystem directly but uses the existing connection session created by the parent device.

The driver checks the child device's current status in the subsystem and does the following:

- If the device is available in the subsystem and the device is healthy (in other words, in a working state, connected, with no faults), it should be in an **Online** state  C51_East side .
- If the device is unavailable (in other words, it is disabled in the subsystem or removed from the subsystem configuration), it should be in a **Failed** state, with a description that describes the reason for the failure. For example, **Device not found**.
  
  GeviScope Channel Simulator 9 | Geutebruck GeViScope Camera | Device not found.
- If the device is available but it is faulty, it should be in a **Failed** state with a state description describing the reason for the fault.
  
  IP Camera 5 PTZ | Geutebruck GeViScope Camera | Camera disconnected.
- If the device is available and it supports custom states (see Custom States), it should go to the correct custom state matching the actual device state in the subsystem.

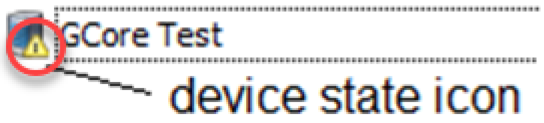
State Propagation Logic

It is important to understand how device states propagate between parent and child devices.

When a connectable device (typically a server device) changes its state, its child devices go to the same state (including the state icon and the state description).

Standard Control Center Device States

In Control Center, the current state of the device is displayed.



The following table describes Control Center device states and their meaning.

State	Icon	When this state occurs	Meaning
Disabled	⊗	Device is disabled by a user	- Connectable devices - device is disconnected from subsystem and will not raise any events and no methods can be triggered on the device. - Non-connectable devices (child devices) - device behavior is ignored by Control Center. It will not raise any events and no methods can be triggered.
Offline	✖	Connection Manager service stopped or crashed	Connection Manager is offline. **Note**: do not use this state represent offline devices.

Online	✓	• Connectable devices: ○ Device was previously disabled and is now enabled and connected ○ Device was disconnected and has automatically reconnected. • Non-connectable devices: ○ Device was prviously disabled and is enabled and parent device is connected. ○ Device is enabled. There was a fault on the corresponding subsystem device and the fault was removed. For example, camera was reconnected.	• Connectable devices - the driver has successfully connected and authenticated with the remote subsystem server using the connection details on the device. • Non-connectable devices - the devices is healthy (no faults) and it is configured in the subsystem.
Failed	⚠	• Connectable devices. Driver cannot connect to subsystem or lost connection with the subsystem • Non-connectable devices: ○ There is a fault on the device ○ This device is not found on the subsystem ○ Parent device has lost connection to the subsystem	• Connectable devices - driver cannot connect to subsystem server or lost connection with the server: there can be number of issues: ○ SDK is not installed ○ invalid connection details ○ faulty remove server • Non-connectable (child devices)- ○ There is a fault on the device (e.g. camera is disconnected) ○ This device is not found on the subsystem (subsystem configuration changed so this device was removed or disabled) ▪ Parent device has lost connection to the subsystem

Warning	⚠	Non-connectable child devices - enable the device while the parent device is disabled	Do not confuse Warning state with Failed state. Warning state should not be raised by the driver itself, but by Connection Manager in standard scenarios like, for example, parent device is disabled.
Connecting	⌛	Connectable devices - enable a previoulsy disabled server device	Device is currently attempting to connect to remote subsystem server

There are exceptions to this rule.

- Disabled devices remain in **Disabled** state.
- if a child device is also a connectable device, it does not set its states according to the parent device. In other words, the state will not propagate from a connectable parent device to a connectable child device.
- If a driver has a multi-level parent-child hierarchy (for example, Server → Recorder → Camera), the states do not propagate automatically from the parent to a 'leaf' (the device on the lowest level). It only propagates one level down, to an immediate child device.
- You have applied the property **DeviceOverridesChildOnlineState** to the child device when, as the name implies, state propagation is suspended and you need to manage the state of the child devices.

The following table describes the default state propagation rules.

NOTE: Sometimes a connector can override some of these.

Server Device State	**Resulting Child Device State**
Disabled	Warning (the device's parent has been disabled)
Online	Online
Connecting	No Change
Failed	Failed
Custom	No Change

Video Connector Architecture

Video connectors are different to non-video drivers. A connector that can display video has its package loaded both into a Control Center server and a Control Center client.

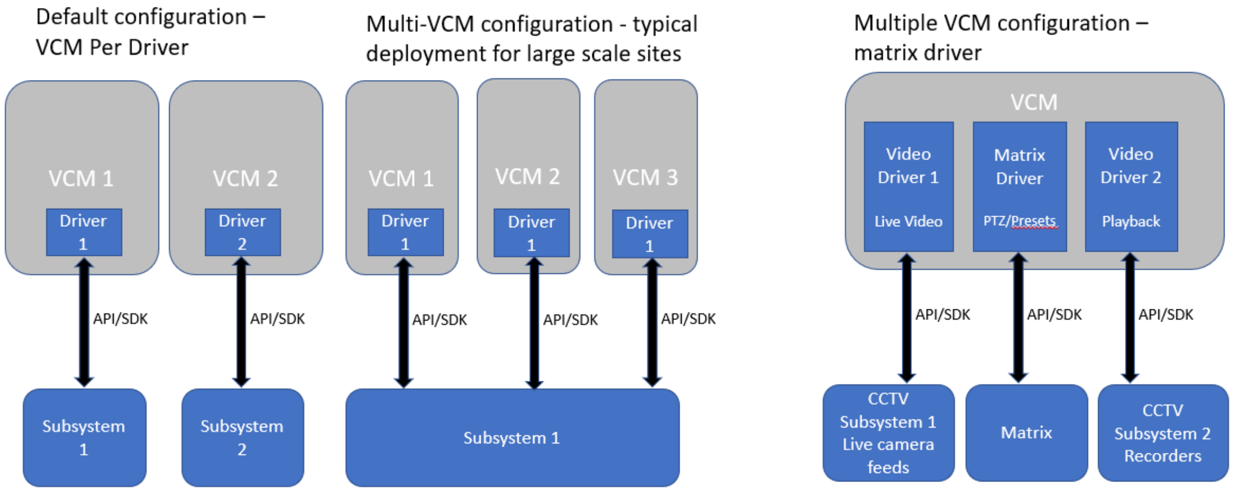
On a Control Center server side, the connector package is used by 2 services:

- A video driver is hosted by a Connection Manager service, like all the non-video connectors. This connector instance is responsible for:
    - Monitor connectivity and state of all subsystem devices
    - Receive events and alarms from the subsystem
    - Optionally manage the native alarms: acknowledge or close them
    - Reflect on configuration changes if the subsystem can report it
    - Get camera snapshot by time stamp from a visual response plan (VRP). See *Control Center Reference Guide* for more information.)
    - Select a PTZ camera preset (from a VRP)
- A video connector is also hosted by a Video Export service. This connector instance is handling video export functionality, export a recorded video from a given camera to a file.

Control Center client runs a separate Windows process called Video Control Manager (or VCM). The connector is hosted by a VCM on a Control Center client. This connector instance is responsible for:

- Displaying live video feeds
- Displaying playback feeds
- PTZ and Preset functionality for PTZ cameras
- Saving video snapshots
- Optional extra features (depending on the subsystems available capabilities):
    - Change video resolution
    - De-warp camera image
    - Audio in/out
    - Digital zoom

VCM configurations can vary and can be set for each Control Center client machine.



Interactions with Microsoft Eco-system

The core infrastructure of Control Center Connection Manager uses the following Microsoft technologies:

- .Net 4.5
- .Net 4.7.2
- Windows Communications Foundation (WCF)
- Microsoft Messaging Queue (MSMQ)
- Microsoft SQL Server
- C# programming Language

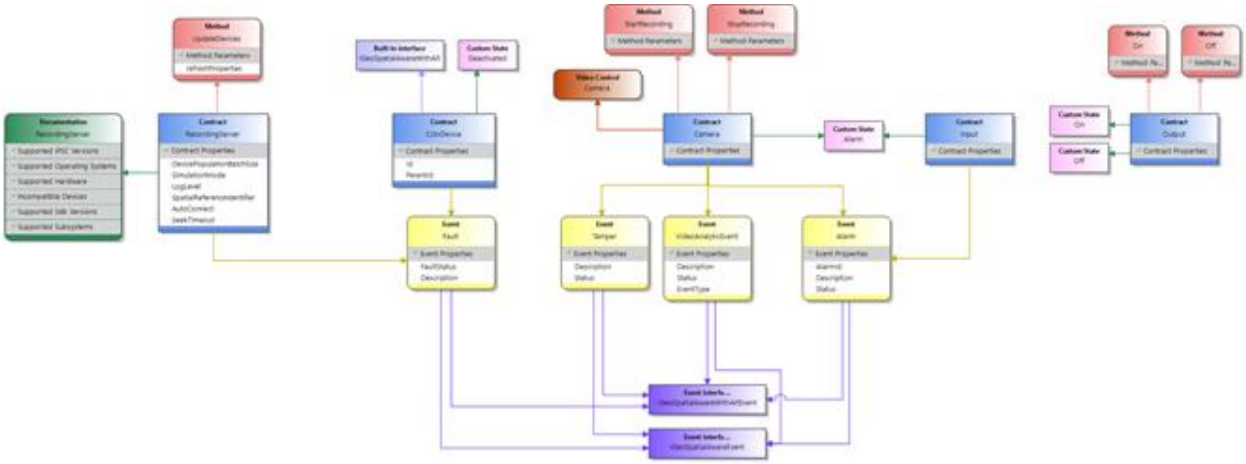
You can use other technologies and versions of .Net that are compatible with this infrastructure, but the documentation supplied with the connector (in other words, RDIN) must document the required technologies and that you must install them on every system that the connector is installed on.

Using NVR Connector Template

The NVR Connector template defines standard functionality for a Control Center CCTV subsystem. The NVR connector template makes it faster and easier for you to develop and test your CCTV connectors.

To use the template, create a new project in Visual Studio and select NVR Template as your project type.

Following is the NVR Connector Template designer diagram.



NVR Template Connector Terminology

The NVR connector template has the following terminology.

CCTV Term	NVR Connector Template Term	Description
NVR/DVR	Recording Server	Recording device manager recordings of one or more video cameras.
Device/asset Disabled	Deactivated	Devices/assets disabled in the Recorder are shown in Deactivated custom state in Control Center.
Device/Asset	Asset	A physical or logical entity in CCTV subsystem.
Alarm	Alarm	Alarm that can occur on a video camera or an input. (This is not a Control Center alarm).
Fault/Failure	Fault	Device malfunction that can occur on any CCTV asset including recorders.
Tamper	Tamper	Camera was tampered with.

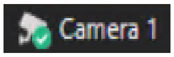
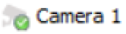
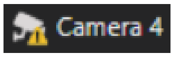
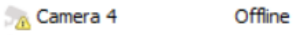
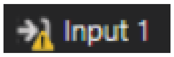
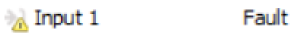
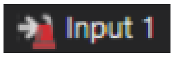
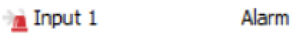
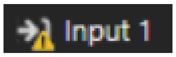
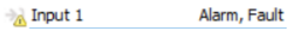
NVR Connector Template Feature List

The NVR Connector Template provides the following features.

- Live Video
- PTZ
- Playback: Seek: Playback Loop
- Events: Fault, Alarm, Tamper, Video Analytic events

CCTV Device States

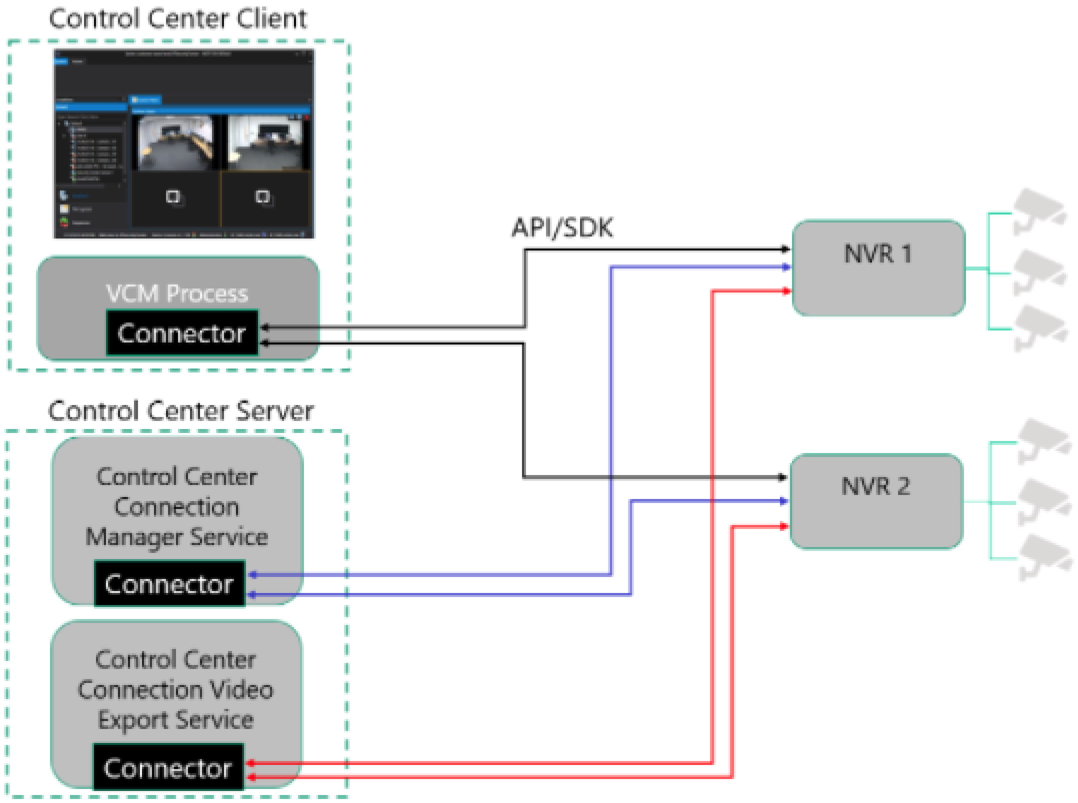
The following table describes how the common CCTV device states and how they are displayed in Control Center.

Scenario	State	Description	Main GUI	System Configuration GUI
Device online, no faults/alarms	Online	(Empty)	 Camera 1	 Camera 1
Camera Offline	Failed	*Offline*	 Camera 4	 Camera 4 Offline
Device with Fault	Failed	Fault	 Input 1	 Input 1 Fault
Device in Alarm (no faults)	Alarm	Alarm	 Input 1	 Input 1 Alarm
Device in Alarm and Fault	Failed	Alarm, Fault	 Input 1	 Input 1 Alarm, Fault

Connecting NVR Connector Template to Server

The following diagram describes how, using the NVR Connector Template, the connector establishes connection directly to video recorders (DVRs/NVRs).

NOTE: The diagram assumes only one connection is made to the same recording server (NVR) from every Control Center Connection Manager service. If you create multiple instances representing the same recording server, the connection session is share across the multiple instances. Secondly, if mulitple cameras are displayed from the same recording server, the connection to the server is shared across the VCM process where the connector is hosted on Control Center client.



Using Alarms with NVR Connector Template.

Cameras and Inputs can receive Alarm events. The NVR Connector Template assumes that:

- Every camera/input can receive multiple alarms,
- Every alarm has a unique ID that is passed in the Alarm ID property in a Alarm event. (This can be set as an Alarm ID text box in a CCTV Simulator).
- A camera or input has a boolean alarm state:
  - **True** - asset in alarm,
  - **False** - asset is not in alarm.

- Devices in Control Center must always reflect the current asset alarm state. In other words, the Control Center device must have an alarm custom state. See CCTV Device States.

Use Case Scenarios

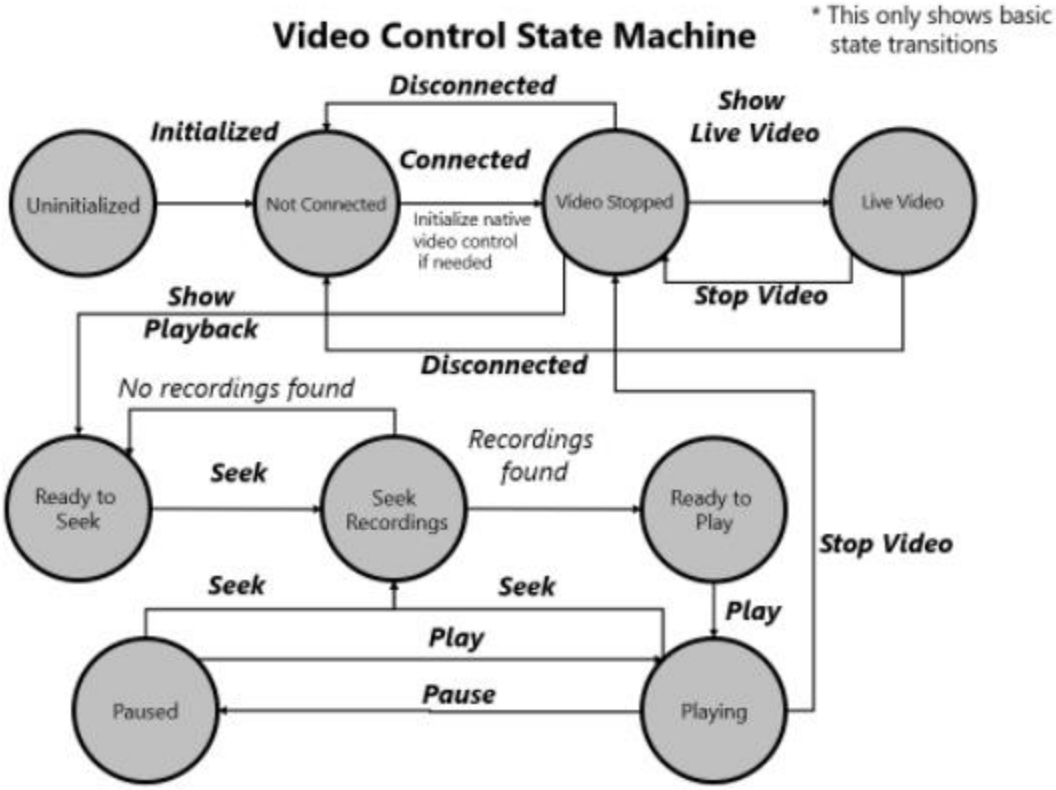
The following table describes some common use case scenarios.

NOTE: This table assumes the asset in question is online, enabled and no faults are reported.

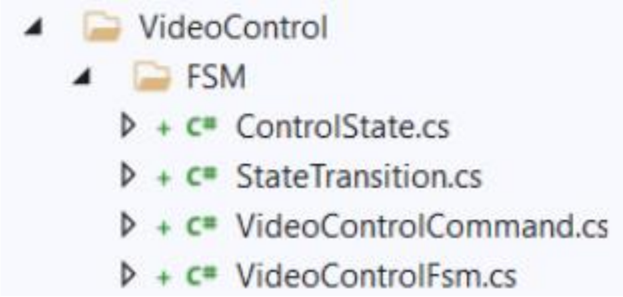
Scenario	Expected Behavior	Events Raised
New alarm on asset, asset is in **Alarm** state	• Connector receives **Alarm** event with **AlarmStatus = Start**. • Device goes to **Alarm** state.	1. **Alarm** event with property **Status** is **Start** 2. State Change event
Alarm ends on an asset. In other words, the asset is not in an **Alarm** state any more.	• Connector receives **Alarm** event with **AlarmStatus = Stop** and the **Alarm** state flag on the asset reports as **False** • device goes to **Online** state	1. **Alarm** event with property **Status** is **End** 2. State Change event
Another alarm on asset that is already in **Alarm** state. For example, it could be a repeated alarm or a different alarm.	• Connector receives Alarm event with AlarmStatus = Start, • device state remains unchanged	**Alarm** event with property **Status** is **Start**
(unlikely to happen) Alarm ends while the asset is already not in Alarm state	• Connector receives Alarm event with AlarmStatus = End • device state remains unchanged	**Alarm** event with property **Status** is **End**

State Machine (FSM)

A Video Control Machine (VCM) tile has its own state machine and API that has to be synchronized with the Software Development Kit (SDK) video player. The SDK video player has its own state logic. FSM helps to synchronize these two.



The FSM implementation is located in `VideoControl\FSM`.



VideoControl
FSM
ControlState.cs
StateTransition.cs
VideoControlCommand.cs
VideoControlFsm.cs

FSM in Camera Video Control Class Use Cases

Below are two examples of typical use cases of the FSM in `Camera Video Control` class.

1. Switch the FSM state after satisfying the following condition.

```
if (!_videoControl.PlayLiveVideo(out var error))
 {
     throw new FatalDriverException(error);
 }
 //assume successfully streaming live video, can switch FSM to live
Video state
 _fsm.ProcessCommand(VideoControlCommand.ShowLiveVideo);
```

2. Make sure certain section of code is valid for the current FSM state.

```
if (!_fsm.IsValidCommand(VideoControlCommand.ShowPlayback))
 {
     return;
 }
```

Implementing Live Video

The video control implements a `Switch Camera` interface to optimize displaying a sequence of cameras on the same video tile.

The NVR connector implements a `LifeTime Manager` pattern to cache connections to recorders.

Using Playback

When using playback, the NVR Connector Template assumes:

- the subsystem can search for existing recordings and return a list of playback chunks (which allows the connector to display them on the Time bar).
- the recordings are managed by the recorder, and not by cameras, so it is possible to show playback videos even from cameras which are currently offline.
- all the recording queries are designed passing the parameters and returning results using UTC time, so the connector does not need to convert to/from Local Time of the recorder. The conversions between the connector UTC time and Control Center Client local time is done by Control Center outside of the connector.

The connector manages the playback results cache to optimize the recordings search, similar to connectors like March Networks and HuperLab HuperVision.

The NVR Connector Template's Seek algorithm logic is that if there are no recording chunks within 3 hours (hard-coded) from the seek time (time selected on the Time Bar/Calendar Control or the 'Now' time when switching from Live Video), the video control displays a **No recordings found** message. A Security Operator cannot manipulate playback (play or pause) when a seek operation has failed. The Security Operator has to try to seek again until a recording is found.

Understanding Playback Speeds

In Playback mode, implemented speeds are: -4, -2, 1, 2, 4 where 1 is a normal default speed. In Paused mode, the implemented speeds are: -0.5, -0.2, -0.1, 0, 0.1, 0.2, 0.5.

NOTE: The speeds implemented in Video Control Simulator are not precise. In other words, the speed x4 does not necessarily plays 4 times faster and so on.

Playback is automatically restored to default x1 speed after being paused.

About CameraVideoControl.cs

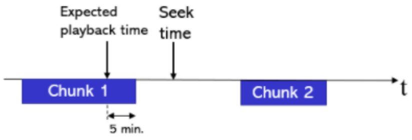
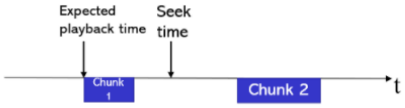
`CameraVideoControl.cs` manages StorageTimer to automatically populate the last time bar chunk to simulate continuous recording. This feature demonstrates a common workaround when a third party SDK cannot supply the exact list of recordings.

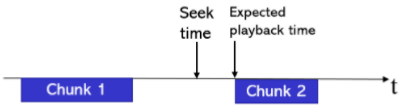
The seek algorithm implemented in the `CameraVideoControl` class does not include seek results validation. This is in case some SDKs return results irrelevant to the requested seek time, as this should be done in the SDK session wrapper implementation.

Playback Scenarios

The following table describes some common playback scenarios.

Scenario	Comment	Expected Behavior
Display a camera after connection to parent video server was lost and then restored.	The server device state may be restored later than the connection (depending on the **Retry Interval** setting in the recorder device), so it is possible to successfully display video while the parent server and the camera still appear in **Failed** State.	Display video (live or play back) is the camera is online.
Display a camera while parent video server is **Disabled**.	Due to ISDK limitation, the video tile cannot be notified when a parent server device is **Disabled**, so the convention is to display video if the actual recorder is online and the camera is **Enabled** and online.	Display video (live or play back) is the camera is online.

Play back time when switching to Playback mode and recording is in progress.	It is not practical to try rewinding video to present time as it takes time to record and buffer video. The exact timing is unpredictable as it is dependent on a recorder model and the network speed so rewinding to present time usually fails. Rewinding to a very recent time, for example, few seconds back, may succeed, but causes the driver to stutter as the video immediately plays to the end, then tries to seek for more video, loads only few seconds, seeks again and so on. To prevent this, most drivers try to rewind to the last 15-30 seconds instead.	Once switched to playback mode, the camera plays from (**DateTime.Now** is 15 seconds).
Recorder does not bring back recordings list or returns them after a long time.	The template defines a maximum time allowed to seek, preventing the Tile from hanging. This is needed for SDKs that do not implement this internally. This is set in the Seek Timeout property on the parent server device.	If the SDK returns no results (or fails to rewind) after the time defined by Seek Timeout, a **No Recordings found** message is displayed.
Recording Seek algorithm - seek for a time between two recording chunks a. The seek time is closer to the previous chunk and the chunk is longer than 5 minutes. b. The seek time is closer to the previous chunk and the chunk is shorter than 5 minutes. c. The seek time is closer to the next chunk.	The video control should try to play back the closest available time to the requested seek time.	If the SDK supports the smart seek, in othr words, finds the closest available time itself, the outcome depends on the SDK. The logic implemented in the NVR Connector Template is as follows: a. Play the latest 5 minutes of the previous chunk:  Expected playback time Seek time Chunk 1 Chunk 2 t 5 min. b. Play the previous chunk from the start:  Expected playback time Seek time Chunk 1 Chunk 2 t

		c. Play the next chunk from the start:  Seek time Expected playback time Chunk 1 Chunk 2 t
Seek (rewind) in progress	Native video controls may behave differently during the rewind process. If the video search and rewind process takes a long time and the native behavior is inconsistent, it may be required to hide the native control from the Security Operator, displaying an overlay panel displaying a **Seek in progress** message. In any case, it is preferable to show progress in the video tile during a long seek operation	Depends on the native video control.
Seek (rewind) when a recording is not available at the time selected. a. There is a recording chunk available within 3 hours of the seek time b. There is no recording chunks available within 3 hours of the seek time	Usually the preferred behaviour is to display some recording close to the requested seek time.	a. Play back the closest chunk available. If it is later than seek time, play from the start. If it is earlier, play the last 5 minutes of the chunk (or from the start, if the chunk is shorter than 5 minutes). b. Display message **No Recordings Found**. To continue, Security Operator has to try to rewind to another time.

Understanding Seek Results Cache

The Seek Results cache stores:

- results for previous seek operations, in other words, list of recordings previously found on the server.
- the times covered by the previous recording searches.

The Seek Results cache is needed to speed up the recording seek/rewind process. There can be many calls to rewind the playback just by dragging a teardrop along the timebar.

The cache is cleared when a video tile is closed and when a camera is switched to another one (for complex scenario such as: Display Live camera 1 → Switch to Playback → Switch back to Live Video → Switch to camera 2 Live Video → Switch to Playback on camera 2).

Special Cases

For the following cases, the NVR Connector Template behavior is as follows.

- The recording server is the parent device and its name/label is defined by the user. This means it does not get synched with the subsystem configuration (either simulated or the real one).
- When connection to a recorder is lost and later restored, while displaying live or playback video, the actual video may be restored before or after the states of the camera devices are restored. This is because the states restoration is done in server-side in Connection Manager and its timing depends on the **Retry Interval** property on the recording server device where as the restoration of the video is done in Control Center client as soon as the SDK signals the connection has been restored.

Limitations

The NVR Connector Template has the following limitations due to limitations with the VCM API.

1. If recordings do not exist inside the given Loop range, playback may get started on the chunk where recordings do exist, but outside the Loop boundaries.
2. Connectors have to always:
    - populate a chunk on a Time bar and
    - report at least one frame time while inside Seek(DateTime) VCM method.

   This is needed to be able to:

    1. implement Seek while the playback is paused
    2. scroll the Time bar to another time later (if the Seek time is far from the current playback time)

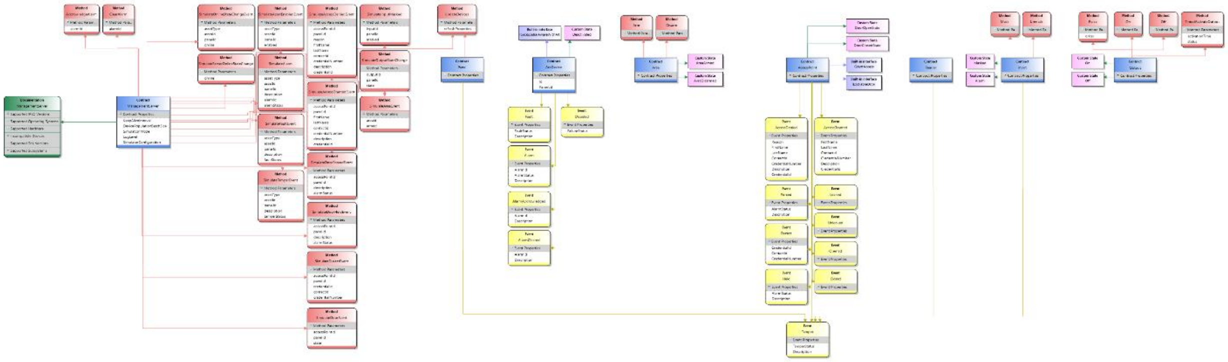
   The side effect of this is - the connector has to populate a fake chunk on a timebar before the seek results are known.

Using Access Control Connector Template

The Access Control Connector template (ACS Template) defines standard functionality for a Control Center ACS Connector which makes it faster and easier for you to develop and test your access control connectors.

To use the template, create a new project in Visual Studio and select ACS Template as your project type.

Following is the Access Control Connector Template designer diagram.



Access Control Template Connector Terminology

The Access Control connector template has the following terminology.

ACS Term	ACS Connector Template Term	Description
Door, Turnstile, Barrier, Gate	Access Point	Any point with restricted access where badge holders may want to access using authentication. For example, passing a badge/fob, biometric scanning, manual authentication and so on.
Panel/Door Controller	Panel	Hardware with Inputs/Outputs and connected readers.
User/Badge Holder	Contact	Person owning one or more credentials.
Badge/Card/Fob	Credential	Badge, card or another means of contact identification
Access Control System (ACS) Server	Management Server	Device representing the point of connection to the ACS server.
	Asset	A physical or logical entity in ACS subsystem
Alarm	Alarm	Alarm that can occur on any ACS asset. When an

		alarm occurs on an input, the input is set to Alarm custom state. **Note**: This is not the same as a Control Center alarm.
Input Masked/Inhibited	Input Masked	No alarms are detected on the input while it is in a Masked state. To detect the alarms the masked input needs to be unmasked.
Momentary Unlock, REX button pressed	Grant Access	Unlock a door (or open a barrier, depending on access point) for a short amount of time (usually pre-configured in the ACS).
Output	Output	Physical output (usually a two-state relay) or a logical output. It is assumed an output can be either on or off.

Access Control Device States

An access control connector can have the following states:

- Device States
- Area States
- Door States
- Input States
- Output states

Device States

An access control connector can have the following states for most of its device types.

Scenario	**State**	**State Description**
Device online	Online	(empty)
Device offline	Failed	Offline
Device deactivated	Deactivated	Deactivated

Area States

An access control connector can have the following area states.

Scenario	**State**	**State Description**
Area disarmed	Disarmed	Armed
Area armed	Armed	Disarmed

Door States

An access control connector can have the following door states.

Scenario	Locked	Open	Forced	Held	Disabled	Device State	State Description
default state (locked and closed)	1	0	0	0	0	Closed	Closed, Locked
unlocked	0	0	0	0	0	Closed	Closed, Unlocked
unlocked and open	0	1	0	0	0	Open	Open
forced and closed (usually unexpected)	1	0	1	0	0	Failed	Forced, Closed
Forced and Open	1	1	1	0	0	Failed	Forced
held and open	0	1	0	1	0	Failed	Open too long
forced and held open	1	1	1	1	0	Failed	Forced, Open too long
disabled in access control system	x	x	x	x	1	Deactivated	Deactivated

Input States

An access control connector can have the following input states.

Alarm	Masked	Device State
0	0	Online
0	1	Masked
1	0	Alarm
1	1	N/A

Output States

An access control connector can have the following output states.

Scenario	State	State Description
Output is on	On	On
Output is off	Off	Off

Alarms

Access control assets can have alarms in Control Center. Alarms have a unique ID.

The lifecycle of an alarm in an access control system is as follows.

Alarm created→ **Alarm acknowledged** → **Alarm cleared** (Removed from the access control sub-system)

Server Device Methods

The following server device methods are implemented in the Access Control Connector Template.

- Acknowledge Alarm
- Clear Alarm

Events

The following events are implemented in the Access Control Connector Template.

Alarm

- Alarm - event is raised when a new alarm is created (Alarm Status = Start) or when an alarm is no longer triggered (Alarm Status = End).
- Alarm Acknowledged - event is raised when a previously raised alarm is acknowledged.
- Alarm cleared - event is raised when a previously acknowledged alarm has been cleared from the system.

Some example scenarios are described below.

Scenario	Access Control Connector Template Behavior
ACS asset triggers an alarm	`Alarm` event is raised on corresponding device with `Status = Start`.
The ACS asset stops triggering the alarm	`Alarm` event is raised on corresponding device with `Status = End`.
A Control Center operator acknowledges the alarm	`Alarm Acknowledged` event is raised on the same device that previously raised the `Alarm` event.
A Control Center operator clears the alarm	`Alarm Cleared` event is raised on the same device that previously raised the `Alarm Acknowledged` event.

Fault

A Fault event means there is a fault/malfunction in the ACS asset. There can be multiple faults for an asset.

A Control Center device with a Fault appears in a Failed state, and its state includes **Fault**.

If an asset is deactivated or offline, the **Fault** state is ignored until the asset is enabled and back online.

Tamper

Tamper means the ACS subsystem has detected that someone was tampering with some hardware. There can be different types of tampers for an asset.

A Control Center device with at least one Tamper appears in Failed state, and its state includes **Tamper**.

If an asset is either deactivated or offline, the **Tamper** state is ignored until the asset is enabled and back online.

Access Control System Connector Functionality

Following are the properties, methods, events and interfaces for each of the elements that make up the Access Control System connector template.

Management Server

The following tables describe the properties, methods, events and interfaces for the management server.

Properties

The following table describes the management server properties.

Name	Type	Description	Default Value & Ranges
Keep Alive Interval	int	Time interval in seconds between web service connectivity checks.	Default: 10 Min: 0 Max: None
Device Population Batch Size	int	Maximum devices allowed to populate at a time.	Default: 50 Min: 20 Max: 100
Simulation Mode	bool	When true, the connector simulates the subsystem instead of connecting to a real one.	Default: None Min: None Max: None
Log Level	LogLevel	Logging level of the driver.	Default: None Min: None Max: None
ACS Simulator Configuration	AcsConfiguration	Editable ACS simulator configuration, only in use in Simulation mode.	Default: None Min: None Max: None
User Name (ISecureDevice)	String	The user name for the device.	Default: None Min: None Max: None
Password (ISecureDevice)	String	The password for the device.	Default: None Min: None Max: None

Timeout (INetworkedDevice)	TimeSpan	The timeout period to use when connecting to the physical device. Specify a zero period of time (00:00:00) to never timeout.	Default: 00:01:00 Min: None Max: None
Retry Interval (INetworkedDevice)	TimeSpan	The amount of time to wait before attempting reconnection to a device after the connection has timed out or failed. Specify a zero period of time (00:00:00) to attempt reconnection instantly after a connection failure.	Default: 00:01:00 Min: None Max: None
IP (INetworkedDevice)	String	The IP address for the device.	Default: None Min: None Max: None
Port (INetworkedDevice)	Int32	The port the device listens on.	Default: None Min: None Max: None

Methods

The following table describes the functional methods for management server.

Name	Description	Returns	Operator Action	Parameters			
				Name	Type	Description	Default Value & Ranges
Clear Alarm	Clears an existing alarm. An alarm can be cleared once it is acknowledged.	bool	False	Alarm Id	string	Alarm Identifier	Default: None Min: None Max: None
Acknowledge Alarm	Acknowledges an alarm that has been received.	bool	False	Alarm Id	string	Alarm Identifier	Default: None Min: None Max: None
Update Devices	Update the devices to match the current configuration on ACS server.	bool	False	Refresh Properties	bool	synchronize device properties and labels	Default: None Min: None Max: None

The following table describes the simulate methods for management server. These simulate methods are used for testing and are only available in simulation mode.

Name	Description	Returns	Operator Action	Parameters			
				Name	Type	Description	Default Value & Ranges
Simulate Online State Change Event	Simulates Online State Change API event	void	False	Asset Type	AssetType	Asset type the event is simulated for, cannot be used for Inputs and Outputs	Default: None Min: None Max: None
				Asset Id	string	The ID of the asset event is raised for.	
				Panel Id	string	The panel the asset is on, use only for Readers and Access Points	
				Online	bool	The online state to be simulated	
Simulate Server Online State Change	Simulates ACS server Online State Change API event	void	False	Online	bool	The online state to be simulated	Default: None Min: None Max: None
Simulate Asset Enabled Event	Simulates Asset Enabled API event,	void	False	Asset Type	AssetType	Asset type the event is simulated for	Default: None Min: None Max: None
				Asset Id	string	The ID of the asset event is raised for	
				Panel Id	string	The panel the asset is	

						on: used for Readers, Access Points, Inputs and Outputs	
				Enabled	bool	The enabled state to be simulated	
Simulate Alarm	Simulates Alarm	void	False	Asset Type	AssetType	Asset type the event is simulated for	Default: None Min: None Max: None
				Asset Id	string	The ID of the asset event is raised for	
				Panel Id	string	The panel the asset is on: used for Readers, Access Points, Inputs and Outputs	
				Description	string	Event description	
				Alarm Id	string	Alarm Identifier	
				Alarm Status	AlarmStatus	Alarm state	
Simulate Fault Event	Simulates Fault API event	void	False	Asset Type	AssetType	Asset type the event is simulated for	Default: None Min: None Max: None
				Asset Id	string	The ID of the asset event is raised for	
				Panel Id	string	The panel the asset is	

						on: used for Readers, Access Points, Inputs and Outputs	
				Description	string	Event description	
				Fault Status	AlarmStatus	Fault state	
Simulate Tamper Event	Simulates Tamper API event	void	False	Asset Type	AssetType	Asset type the event is simulated for, can select Panel, Access Point, Reader, Input	Default: None Min: None Max: None
				Asset Id	string	The ID of the asset event is raised for	
				Panel Id	string	The panel the asset is on: used for Readers, Access Points and Inputs	
				Description	string		
				Tamper Status	AlarmStatus	Tamper state	
Simulate Access Denied Event	Simulates Access Denied API event	void	False	Access Point Id	string	The ID of the access point event is raised for	Default: None Min: None Max: None
				Panel Id	string	The panel the access point belongs to	
				Reason	string	The reason	

						for the access denial	
				First Name	string	Contact first name	
				Last Name	string	Contact last name	
				Contact Id	string	Contact Identifier	
				Credential Number	string	Credential used to gain access	
				Description	string	Event description	
				Credential Id	string	The Credential ID	
Simulate Access Granted Event	Simulates Access Granted API event	void	False	Access Point Id	string	The ID of the access point event is raised for	Default: None Min: None Max: None
				Panel Id	string	The panel the access point belongs to	
				First Name	string	Contact first name	
				Last Name	string	Contact last name	
				Contact Id	string	Contact Identifier	
				Credential Number	string	Credential used to gain access	
				Description	string	Event description	
				Credential Id	string	The Credential ID	

Simulate Door Forced Event	Simulates Door Forced API event, works only in Simulation mode	void	False	Access Point Id	string	The ID of the access point event is raised for	Default: None Min: None Max: None
				Panel Id	string	The panel the access point belongs to	
				Description	string	Event description	
				Alarm Status	AlarmStatus	Alarm state	
Simulate Door Held Event	Simulates Door Held API event	void	False	Access Point Id	string	The ID of the access point event is raised for	Default: None Min: None Max: None
				Panel Id	string	The panel the access point belongs to	
				Description	string	Event description	
				Alarm Status	AlarmStatus	Alarm state	
Simulate Duress Event	Simulates Duress API event	void	False	Access Point Id	string	The ID of the access point event is raised for	Default: None Min: None Max: None
				Panel Id	string	The panel the access point belongs to	
				Credential Id	string	The Credential ID	
				Contact Id	string	Contact Identifier	
				Credential Number	string	Credential used to gain access	

Simulate Input Masked	Simulates Input Masked	void	False	Input Id	string	The ID of the Input event is raised for	Default: None Min: None Max: None
				Panel Id	string	The panel the Input belongs to	
				Masked	bool	Set to: True - mask, False - unmask	
Simulate Output State Change	Simulates Output State Change	void	False	Output Id	string	The ID of the Output event is raised for	Default: None Min: None Max: None
				Panel Id	string	The panel the Input belongs to	
				State	OnOff Status	Output state	
Simulate Door Event	Simulates Door common API event (locked, unlocked, open, closed)	void	False	Access Point Id	string	The ID of the access point event is raised for	Default: None Min: None Max: None
				Panel Id	string	The panel the access point belongs to	
				State	Simulated Access PointState	Access point state	
Simulate Area Event	Simulates Area armed or disarmed API event	void	False	Area Id	string	The ID of the area event is raised for	Default: None Min: None Max: None
				Armed	bool	Area state	

Events

The following table describes the properties for the Custom State Changed (RaiseCustomStates) event.

Name	Type	Description
Interface Identifier	Guid	Gets the identifier of the interface that has changed state
Custom State	ICustomState	The state that the device has changed to
Message	String	Gets the error message, if any, relating to the state change
Is Child State Change	Boolean	Indicates whether the state change applies only to non-networked devices connected to the interface identified by the Interface Identifier property
Device Identifier	Guid	The identifier of the device that raised the event
Date	DateTime	The UTC date and time the event was raised

Interfaces

The management server has the following interfaces:

- ISecureDevice
- INetworkedDevice
- IRaiseCustomStates

Access Point

The following tables describe the properties, methods, events, interfaces, and custom states for access point.

Properties

Access Point has the following properties.

Name	Type	Description	Default Value & Ranges
ID	string	Unique identifier	Default: None Min: None Max: None
Parent ID	string	The parent device ID	

Methods

Access Point has the following methods.

Method	Description	Returns
Lock Door (ILockableDoor)	Lock Door	Boolean
Unlock Door (ILockableDoor)	Unlock Door	Boolean
Grant Access (IGrantAccess)	Grant Access	Boolean

Events

Access Point has the following events.

Event	Description	Properties		
		Name	Type	Description
Access Denied	Access to the access point was denied	Reason	string	The reason for the access denial
		First Name	string	Contact first name
		Last Name	string	Contact last name
		Contact Id	string	Contact Identifier
		Credential Number	string	Credential used to gain access
		Description	string	Event description
		Credential ID	string	The Credential ID
Access Granted	Access was granted to the access point	First Name	string	Contact first name
		Last Name	string	Contact last name
		Contact Id	string	Contact Identifier
		Credential Number	string	Credential used to gain access
		Description	string	Event description
		Credential ID	string	The Credential ID
Forced	Access Point was forced open	Alarm Status	AlarmStatus	The alarm state
		Description	string	Event description
Held	Access point was held open for too long	Alarm Status	AlarmStatus	The alarm state
		Description	string	Event description
Duress	Duress was	Credential ID	string	The Credential ID

	signaled on the access point	Contact Id	string	Contact Identifier
		Credential Number	string	Credential used to gain access
Tamper	Tamper state change	Alarm Status	AlarmStatus	The alarm state
		Description	string	Event description
Locked	The access point is locked	n/a		
Unlocked	The access point is unlocked	n/a		
Open	The access point has been opened	n/a		
Closed	The access point has been closed	n/a		
Fault	Fault state change	Alarm Status	AlarmStatus	Fault state
		Description	string	Event description
Disabled	Disabled state change	Failure Status	AlarmStatus	Failure state
Alarm	Alarm state change	Alarm ID	string	Alarm ID
		Alarm Status	AlarmStatus	Alarm state
Alarm Acknowledged	Alarm has been acknowledged	Description	string	Event description
		Alarm ID	string	Alarm ID
		Description	string	Event description
Alarm Cleared	Alarm has been cleared from the system	Alarm ID	string	Alarm ID
		Description	string	Event description

Interfaces

Access point has the following interfaces

Name	Description
ILockableDoor	Interface for a door that can be locked
IGrantAccess	Device that supports granting access

Custom States

Access point has the following custom states.

- Door closed
- Door open

Output

The following tables describe the properties, methods, events, and custom states for output.

Properties

Output has the following properties.

Name	Type	Description	Default Value & Ranges
ID	string	Unique identifier	Default: None Min: None Max: None
Parent ID	string	The parent device ID	

Methods

Output has the following methods.

Name	Description	Returns	Operator Action	Parameters			
				Name	Type	Description	Default Value & Ranges
On	Switch the output on	bool	False	n/a			
Off	Switch the output off	bool	False	n/a			
Pulse	Switches the output on for a short period of time	bool	False	Order	PulseOrder	The pulse order (on, then off or vice versa)	Default: Done Min: None Max: None
Timed Activate Output	Switches the output on for a given period of time	bool	False	Activation Time	int		Default: None Min: 1 Max: None
				Status	OnOffStatus		Default: None Min: None Max: None

Events

Output has the following events.

Event	Description	Properties		
		Name	Type	Description
Fault	Fault state change	Alarm Status	AlarmStatus	Fault state
		Description	string	Event

				description
Disabled	Disabled state change	Failure Status	AlarmStatus	Failure state
Alarm	Alarm state change	Alarm ID	string	Alarm ID
		Alarm Status	AlarmStatus	Alarm state
Alarm Acknowledged	Alarm has been acknowledged	Description	string	Event description
		Alarm ID	string	Alarm ID
		Description	string	Event description
Alarm Cleared	Alarm has been cleared from the system	Alarm ID	string	Alarm ID
		Description	string	Event description

Custom States

Output has the following custom states.

- On
- Off

Input

The following table describe the properties, methods, events, and custom states for Input.

Properties

Input has the following properties.

Name	Type	Description	Default Value & Ranges
ID	string	Unique identifier	Default: None Min: None Max: None
Parent ID	string	The parent device ID	

Methods

Input has the following methods.

Name	Description	Returns	Operator Action
Mask	Mask the input so no alarms are raised	bool	false
Unmask	Unmask the input so alarms can be raised	bool	false

Events

Input has the following events.

Event	Description	Properties		
		Name	**Type**	**Description**
Tamper	Tamper state change	Alarm Status	AlarmStatus	The alarm state
		Description	string	Event description
Fault	Fault state change	Alarm Status	AlarmStatus	Fault state
		Description	string	Event description
Disabled	Disabled state change	Failure Status	AlarmStatus	Failure state
Alarm	Alarm state change	Alarm ID	string	Alarm ID
		Alarm Status	AlarmStatus	Alarm state
Alarm Acknowledged	Alarm has been acknowledged	Description	string	Event description
		Alarm ID	string	Alarm ID
		Description	string	Event description
Alarm Cleared	Alarm has been cleared from the system	Alarm ID	string	Alarm ID
		Description	string	Event description

Custom States

Input has the following custom states.

- Masked
- Alarm

Area

The following tables describe the properties, methods, events, and custom states for Area.

Properties

Area has the following properties.

Name	Type	Description	Default Value & Ranges
ID	string	Unique identifier	Default: None Min: None Max: None
Parent ID	string	The parent device ID	

Methods

Area has the following methods.

Name	Description	Returns	Operator Action
Arm	Arm the area	bool	false
Disarm	Disarm the area	bool	false

Events

Area has the following events.

Event	Description	Properties		
		Name	Type	Description
Fault	Fault state change	Alarm Status	AlarmStatus	Fault state
		Description	string	Event description
Disabled	Disabled state change	Failure Status	AlarmStatus	Failure state
Alarm	Alarm state change	Alarm ID	string	Alarm ID
		Alarm Status	AlarmStatus	Alarm state
Alarm Acknowledged	Alarm has been acknowledged	Description	string	Event description
		Alarm ID	string	Alarm ID
		Description	string	Event description
Alarm Cleared	Alarm has been cleared from the system	Alarm ID	string	Alarm ID
		Description	string	Event description

Custom States

Area has the following custom states.

- Area armed
- Area disarmed

Panel

The following tables describe the properties, methods, and events for Panel.

Properties

Panel has the following properties.

Name	Type	Description	Default Value & Ranges
ID	string	Unique identifier	Default: None Min: None Max: None
Parent ID	string	The parent device ID	

Methods

Panel has the following methods.

Name	Description	Returns	Operator Action	Parameters			
				Name	Type	Description	Default Range & Values
Update Devices	Update the devices to match the current configuration on ACS Server	bool	false	Refresh Properties	bool	Synchronize device properties and labels	Default: None Min: None Max: None

Events

Panel has the following events.

Event	Description	Properties		
		Name	Type	Description
Tamper	Tamper state change	Alarm Status	AlarmStatus	The alarm state
		Description	string	Event description
Fault	Fault state change	Alarm Status	AlarmStatus	Fault state
		Description	string	Event description
Disabled	Disabled state change	Failure Status	AlarmStatus	Failure state
Alarm	Alarm state change	Alarm ID	string	Alarm ID
		Alarm Status	AlarmStatus	Alarm state
Alarm Acknowledged	Alarm has been acknowledged	Description	string	Event description
		Alarm ID	string	Alarm ID
		Description	string	Event description
Alarm Cleared	Alarm has been cleared from the system	Alarm ID	string	Alarm ID
		Description	string	Event description

Reader

The following table describe the properties and events for Reader.

Properties

Reader has the following properties.

Name	Type	Description	Default Value & Ranges
ID	string	Unique identifier	Default: None Min: None Max: None
Parent ID	string	The parent device ID	

Events

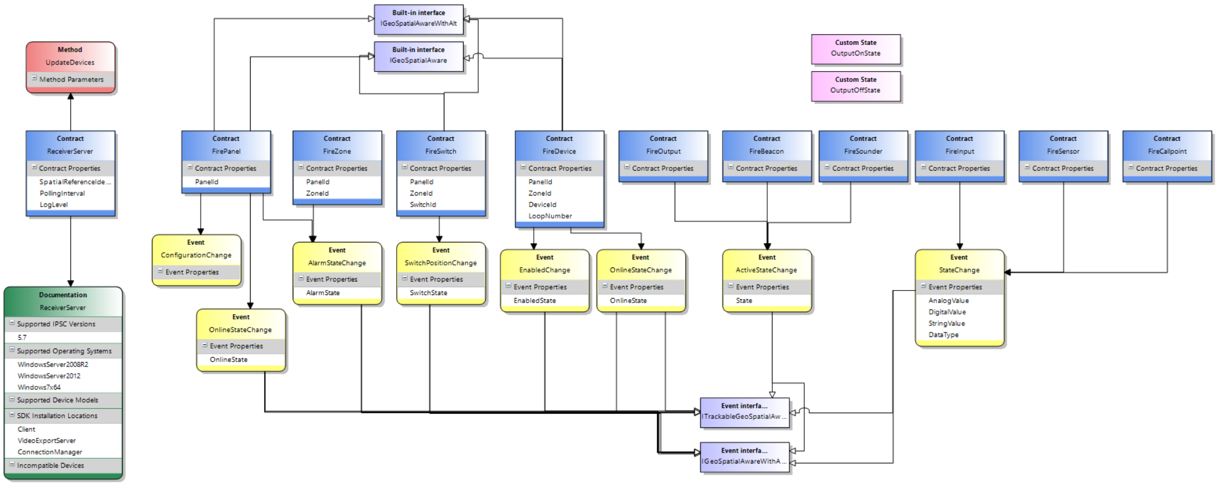
Reader has the following events.

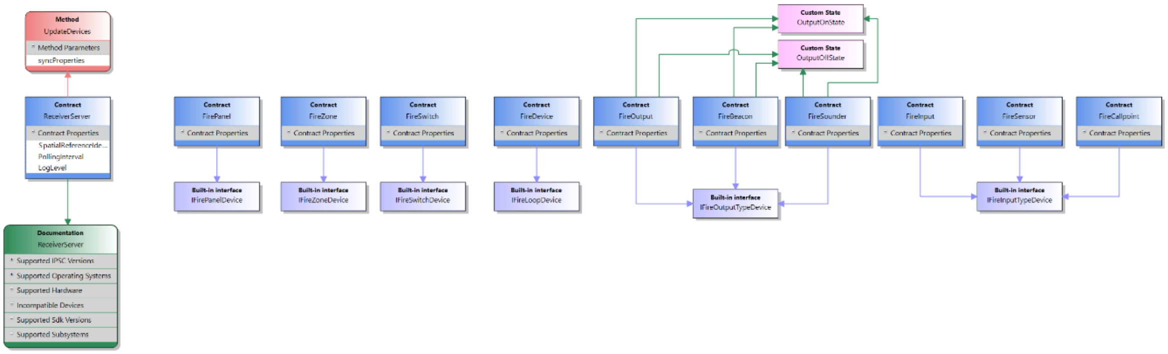
Event	Description	Properties		
		Name	Type	Description
Tamper	Tamper state change	Alarm Status	AlarmStatus	The alarm state
		Description	string	Event description
Fault	Fault state change	Alarm Status	AlarmStatus	Fault state
		Description	string	Event description
Disabled	Disabled state change	Failure Status	AlarmStatus	Failure state
Alarm	Alarm state change	Alarm ID	string	Alarm ID
		Alarm Status	AlarmStatus	Alarm state
Alarm Acknowledged	Alarm has been acknowledged	Description	string	Event description
		Alarm ID	string	Alarm ID
		Description	string	Event description
Alarm Cleared	Alarm has been cleared from the system	Alarm ID	string	Alarm ID
		Description	string	Event description

Using Fire Panel Connector Template

You can quickly and easily create Fire Panel connectors using the Fire Panel template. Using the standard functionality provided by the Fire Panel template makes it is faster and easier for you to develop and test your Fire Panel connector.

Illustrated below are the Fire Panel connector designer diagrams.





Fire Panel Template Connector Structure

The Fire Panel connector template has the following structure.

- Receiver Server
    - Fire Panel
        - Fire Zone
        - Fire Switch
        - Fire Devices

Contracts

All contracts that represent physical devices implement IGeoSpatialAware and IGeoSpatialAwareWithAlt interfaces. These are used for devices that can report their position. The specific driver implementation may not have these, but they are supplied in the template.

Contract	Description
ReceiverServer	This is the server device, to which multiple fire panels can be connected. It is the parent device for only Fire Panels. It contains code for populating the Panels and then asking Panels to populate their children. The device population code conforms to latest ISDK standards and happens asynchronously, with cancellation tokens included. It also contains logic for propagating device states when the fire panel gets disabled/enabled, which is necessary due to all other devices being children of the panel device. It contains one method contract - UpdateDevices. This method uses the latest data from the API and updates the labels and similar information on every device, as well as adding any new devices.
FirePanel	This contract represents an actual Fire Panel. This device is the parent of all the other devices, which represents the actual physical system. It contains code that handles population of all other devices, handles orphaned devices and propagates device states.
FireZone	This represents the concept of a zone in a fire panel. While this is not a physical device, all fire panels include zones to which other devices are assigned. To avoid adding another level to the structure, simply tie the devices to zones by adding a property ZoneId to devices.
FireSwitch	This contract represents the internal switches in the fire panels.
FireDevice	This is a base class for all the other loop fire devices. It is hidden as a contract and cannot be instantiated directly. It has code that allows setting and updating the state.
Other	The other devices only contain code that makes raising events on them easier. • FireOutput • FireBeacon • FireSounder • FireInput • FireSensor • FireCallpoint

Events

Almost all events implement IGeoSpatialAwareEvent and IGeoSpatialAwareWithAltEvent interfaces. These interfaces provide functionality that is required by Control Center to draw the event on a specific place on the map. If the Fire

Panel system does not supply this information, any information related to these interfaces can be left out.

Interfaces

The following interfaces are available. Each interface represents the minimum set of properties, events and methods that each fire device must have.

Interface	Description	Properties	Events
IFirePanelDevice	Implements • IGeoSpatialAware • IGeoSpatialAwareWithAlt	PanelId	• ConfigurationChange • OnlineStateChange • AlarmStateChange
IFireZoneDevice		• PanelId • ZoneId	AlarmStateChange
IFireSwitchDevice	Implements • IGeoSpatialAware • IGeoSpatialAwareWithAlt	• PanelId • ZoneId • SwitchId	SwitchPositionChange
IFireLoopDevice	Implements • IGeoSpatialAware • IGeoSpatialAwareWithAlt	• PanelId • ZoneId • DeviceId • LoopNumber	• EnabledChange • OnlineStateChange
IFireInputTypeDevice			FireInputStateChange
IFireOutputTypeDevice			FireOutputStateChange

Incoming Data Model

The template uses mocked data types. To make the development of the connector easier, you should get as close as possible to those data types in the API implementation.

Additional data can always be added on top, but if the data model implementation uses the same class names and property names, not much of the device population code needs to be edited.

Panel

```
private class NativePanel
        {
            public string PanelId { get; set; }
        }
```

Zone

```
private class NativeZone
         {
             public string PanelId { get; set; }
             public string ZoneId { get; set; }
         }
```

Switch

```
private class NativeSwitch
         {
             public string PanelId { get; set; }
             public string ZoneId { get; set; }
             public string SwitchId { get; set; }
         }
```

Device

```
private class NativeDevice
         {
             public string PanelId { get; set; }
             public string ZoneId { get; set; }
             public string DeviceId { get; set; }
             public int LoopNo { get; set; }
             public DeviceType DeviceType { get; set; }
         }

```

Connector Project Structure

Everbridge recommends that your driver projects, under the driver root folder, have the following folder structure:

- Author.CC.Driver.Manufacturer.Product.sln - the connector solution
- Author.CC.Driver.Manufacturer.Product - the connector project folder
- Author.CC.Driver.Manufacturer.Product.Spec - the connector Unit Test project folder
- Author.CC.Driver.Manufacturer.Product.TestApp - the connector test application folder

Connector Name

Everbridge recommends that your connector names have the following format: [*Author*].CC.Driver.[*Manufacturer*].[*Product*]

- *Author* - the company that wrote the connector
- *Manufacturer* - the manufacturer of the subsystem (for example, Bosch, Milestone)
- Product - the subsystem name and optionally, its version (for example, MAP5000, ProWatch, OnGuard and so on.)

For example, **EVBG.Control Center.Driver.Bosch.BVMS**

The connector name must be set as:

- the connector solution name
- the connector project name
- the default project namespace
- in Assembly settings:
  - The assembly name
  - The assembly title
  - The assembly product

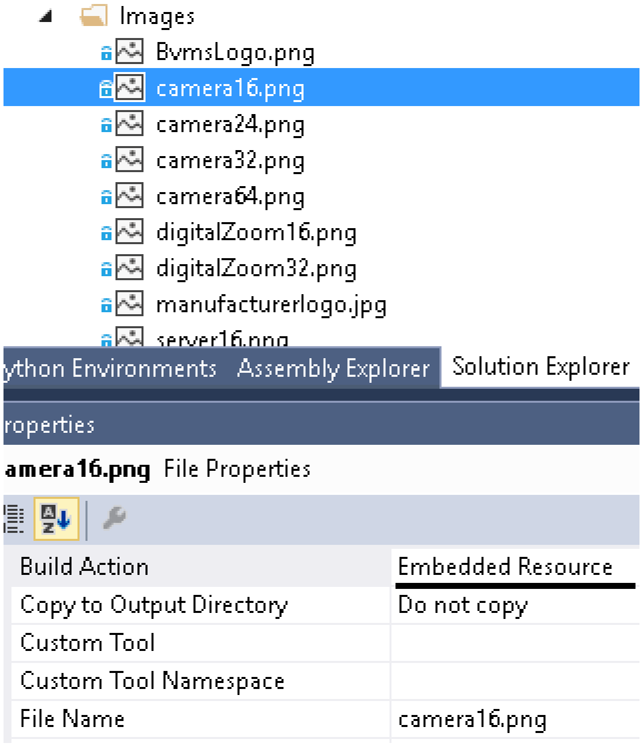
These should be set in **Project** → **Settings** in Visual Studio.

Connector Project Files

Every connector project has the following files:

- `Images\` folder. Contains all the graphics used by the driver.

CAUTION: Every icon file must have its **Build Action** set as **Embedded Resource**.

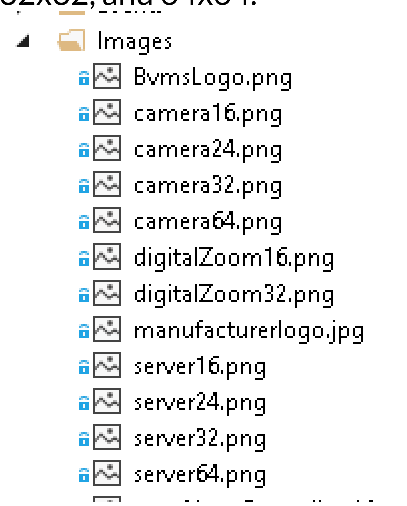


- Images
  - BvmsLogo.png
  - camera16.png
  - camera24.png
  - camera32.png
  - camera64.png
  - digitalZoom16.png
  - digitalZoom32.png
  - manufacturerlogo.jpg
  - server16.png

amera16.png File Properties

Build Action	Embedded Resource
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	
File Name	camera16.png

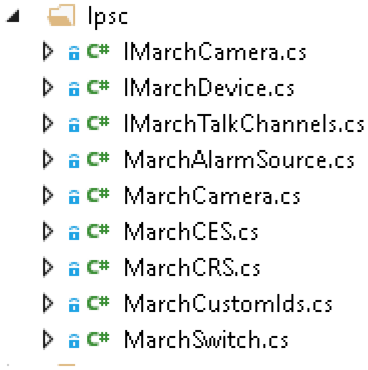
- Device type icons. Each device type has to have four icons of sizes: 16x16, 24x24, 32x32, and 64x64.



```
Images
    BvmsLogo.png
    camera16.png
    camera24.png
    camera32.png
    camera64.png
    digitalZoom16.png
    digitalZoom32.png
    manufacturerlogo.jpg
    server16.png
    server24.png
    server32.png
    server64.png
```

- Custom State icon. If the connector implements custom states, every state must have a unique icon. All the icons must be of size 16x16. There are some standard custom states, for example, TamperState. When using custom states, you do not need to provide icons.
- Operator Action icons. If the connector implements any operator actions, every such action must have a unique icon. All the icons must be of size 16x16. When using Operator Actions available via built-in interfaces, you do not need to provide icons.
- Video Operator Action icons. If the connector implements any Video Operator actions, every such action must have an icon of sizes 16x16. Some Video Operator actions may also require an icon size 32x32.
- Pictures. The following pictures need to be embedded into the auto-generated driver documentation.
    - `configurationdiagram.png`. A diagram describing connectivity and integration with the subsystem. It should mark the protocols/SDKs used and show the main connected parties and subsystem key elements and is integrated automatically in the generated RDIN.
    - `deviceWizardAddServerDevice1.png` and `deviceWizardAddServerDevice2.png`. These show how a new server device is added into Control Center.
    - `manufacturerlogo.jpg` This shows the manufacturer logo and is shown both in the documentation and in Control Center.
    - `productlogo.jpg` This shows the product picture or logo and is shown both in the documentation and in Control Center.

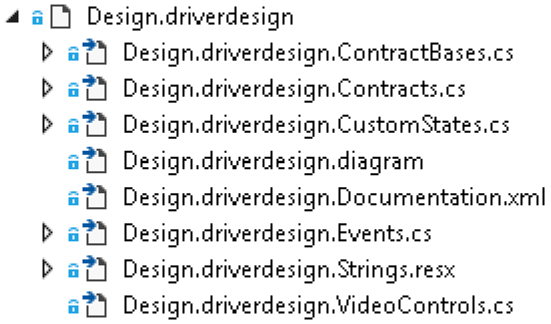
- `Control Center\` folder (sometimes named `Contracts\`, although Everbridge does not recommend this). This contains classes implementing device contracts and other related types: one contract (and file) per device type

  ```
  Ipsc
      IMarchCamera.cs
      IMarchDevice.cs
      IMarchTalkChannels.cs
      MarchAlarmSource.cs
      MarchCamera.cs
      MarchCES.cs
      MarchCRS.cs
      MarchCustomIds.cs
      MarchSwitch.cs
  ```

- `app.config.` This defines the .NET framework version and optionally can define some dependent assemblies' versions. It can also contain web service definition, bindings and so on that the driver connects to.
- `configurationdiagram.png`. A diagram describing the connectivity and integration with the subsystem. It should mark the protocols/SDKs used and show the main connected parties and subsystem key elements and is integrated automatically in the generated RDIN.
- `*.resx` Resource files. To allow for connector localization support, all the text constants displayed in UI must be placed in a resource file. Typical files are:
  - `ErrorMessages.resx`. Error messages of different kind.
  - [*SystemName*]`Messages.resx`. Other UI messages which are not errors.
- `DeviceDefaults.cs`. Implements a pattern to retrieve default device property values.
- `GlobalSuppressions.cs`. A class which gets automatically written when a developer decides to suppress a code analysis error, selecting an option, **Global suppression** file instead of **In source**.
- Link to a `key.snk` file. The files used to sign the driver assembly. It must be signed to produce a driver package. The key should be taken from the current ISDK branch: `C:\Source\DeviceDrivers\` *`BranchName`*`\key.snk`. For example, for trunk drivers branch, this is `C:\Source\DeviceDrivers\Trunk\key.snk`
- [*SystemName*]`CameraVideoControl.cs`. Custom Control - *for video drivers only*: Implementation of the video tile displayed in Control Center.
- `Documentation` folder including the automatically generated driver documentation files in MS Word format.

Additionally, any connector implemented with Connector Designer Visual Studio Extension includes the following files under `Design.driverdesign`:



```
◢ Design.driverdesign
    ▷ Design.driverdesign.ContractBases.cs
    ▷ Design.driverdesign.Contracts.cs
    ▷ Design.driverdesign.CustomStates.cs
      Design.driverdesign.diagram
      Design.driverdesign.Documentation.xml
    ▷ Design.driverdesign.Events.cs
    ▷ Design.driverdesign.Strings.resx
      Design.driverdesign.VideoControls.cs
```

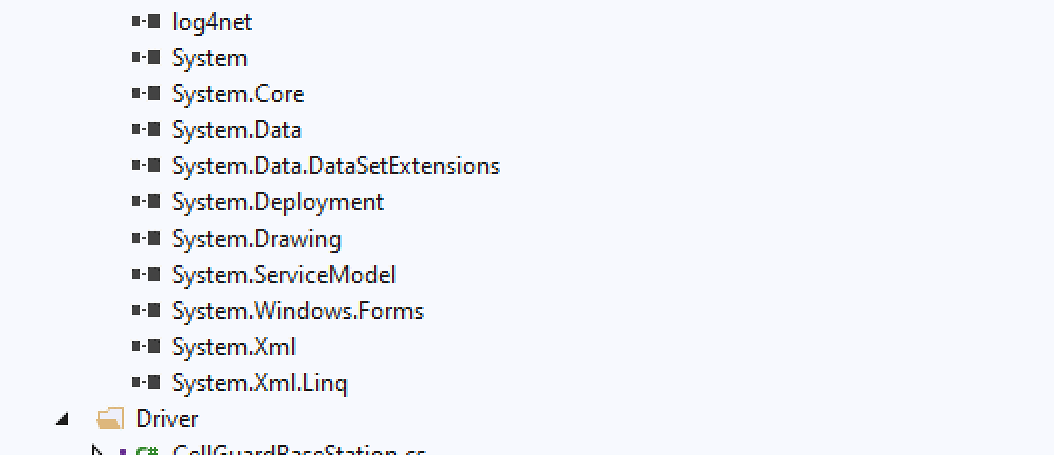
All these files and classes are automatically generated each time the Connector Design surface is saved.

- `Design.driverdesign.ContractBases.cs`. Device Contracts' base classes.
- `Design.driverdesign.Contracts.cs`. Interfaces defining the device Contracts.
- `Design.driverdesign.CustomStates.cs`. Custom states used by the driver.
- `Design.driverdesign.diagram`. The Driver Designer block diagram defining the driver components: Contracts, Methods, Events, Custom States and so on.
- `Design.driverdesign.Events.cs`. Device events implementation.
- `Design.driverdesign.Strings.resx`. Resource file with all the names and descriptions of devices, their properties, events, states and so on.
- `Design.driverdesign.Events.cs`. Device events implementation.
- `Design.driverdesign.VideoControls.cs`. Driver Video Control partial class which should be extended by the [*SystemName*] `CameraVideoControl.cs`. Implementation.

The driver project references:

- `CNL.ControlCenter.Driver.dll`. The main ISDK DLL, located in: `C:\Source\DeviceDrivers\ `*`DDKBranch`*`\ThirdParty\CNL\DDK`
- `CNL.ControlCenter.Driver.Extensions`, `CNL.ControlCenter.Driver.Utility`, `CNL.ControlCenter.Driver.Video.Matrix.dll`. Optional ISDK file references, located in the same folder as the main ISDK DLL.
- `log4net.dll`. Log4Net DLL used for logging, located in: `C:\Source\DeviceDrivers\ `*`DDKBranch`*`\ThirdParty`

- Standard .NET minimum references set



```
log4net
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Deployment
System.Drawing
System.ServiceModel
System.Windows.Forms
System.Xml
System.Xml.Linq
Driver
CellGuardBaseStation.cs
```

- Other 3rd party references which may be used by the driver, located in

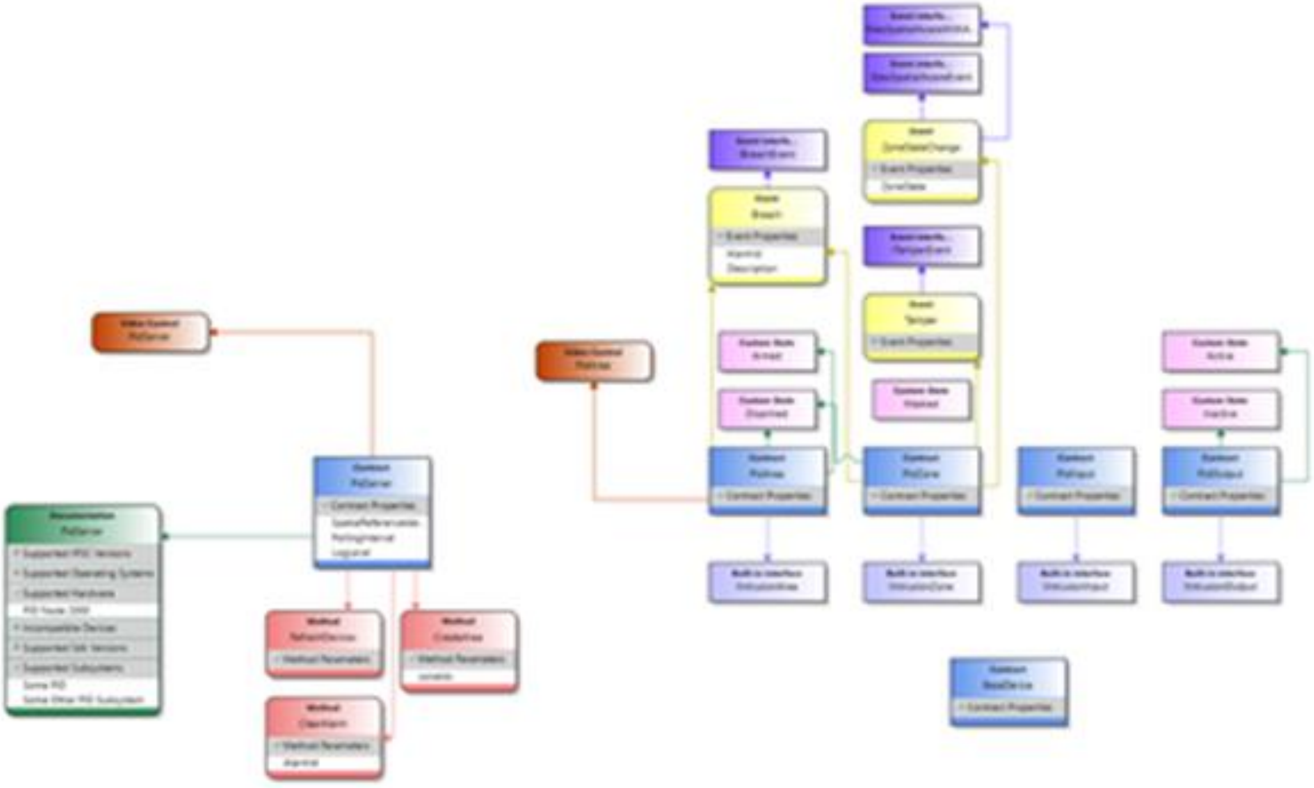
  `C:\Source\DeviceDrivers\{ISDK Branch}\ThirdParty.` The available libraries include Reactive Extensions, EntityFramework, CsvHelper.dll, Newtonsoft.Json.dll and more.
- The references needed for the subsystem SDK to work.

Using Connector Design Surface

You can design your drivers using the driver design surface.

In Solution Explorer, double-click a .driverdesign file to open the main diagram window. From here, you can create and link:

- contracts
- methods
- events
- custom states
- ISDK interfaces
- video control
- documentation



Connections

The connections between shapes are color coded.

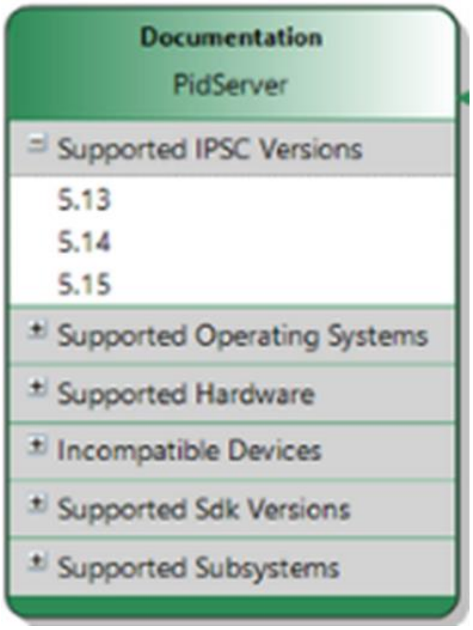
Color	Description
Green	Used for documentation. For custom states, the connection is only used for documentation purposes. Custom states are not constrained to specific contracts.
Red	Connects a method to a contract.
Brown	Connects a video control to a contract.
Yellow	Connects an event to a contract.
Grey/Blue	Connects a built-interface to a contract.
Purple	Connects an event interface to an event.

Shapes and Shape Properties

If a property affects documentation, it is marked green.

Documentation Shape

The documentation shape can be connected to a single contract (preferably the main parent of all other devices). There should only ever be one of these shares per driver. However, you are allowed to have multiples, in case you have broken the previous shape and want to create a new one (and use the old one for guidance).



Documentation

PidServer

⊟ Supported IPSC Versions

5.13

5.14

5.15

⊞ Supported Operating Systems

⊞ Supported Hardware

⊞ Incompatible Devices

⊞ Supported Sdk Versions

⊞ Supported Subsystems

Documentation Shape Properties

Property	Description
Device	
Authentication Method	Select how a connector authenticates with the subsystem. Available options are: `None`, `Basic`, `Windows`, `Windows Credentials`.
Integration Diagram Image	This is a relative path to the image, for example, `images\configuratinodiagram.png`. The image is shown in **Connector Features**. Create a diagram of how the driver interacts with the subsystem, including protocols used and similar information.
Online State Method	Select how a driver determines the online state of the subsystem. Available options: `None`, `Socket`, `Ping`, `SdkOrQueryDevice`. Most drivers use `SdkOrQueryDevice`.
Product Name	This is the product name. It is used on the title page of all 3 generated documents. Do not include the manufacturer name here, only the product name.
Driver	
Default Ports	This property opens a collection editor. Inside the editor default ports that the connector uses can be entered, along with their description and type. A default port has these properties: • **Port** - the Port number or range. • **Port Type** - They type of port. o TCP o UDP o Both • **Usage** - what is the port used for.
Known Issues	This property opens a collection editor. A known issue has these properties: • **Item** - A description of the known issue. This information is used in the Known Limitations section in the Functionality document. • **Name** - This property is only used in the Connector Designer itself, to help recognize collection items.
Additional Details Document (*optional*)	Enter a path to a .docx file that is to be included at the end of the functionality document. The path is relative to the root directory of the project. Do not worry about the styling in the document. RDIN style rules are applied automatically.

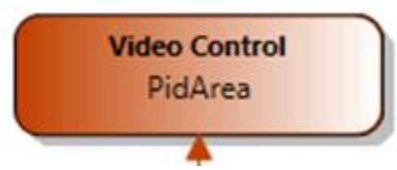
Installation Guide	
Installation Additional Details Document	Enter a path to a .docx file that is to be included at the end of the Installation Guide document starting from section 2. The path is relative to the root directory of the project. RDIN style rules are applied automatically. Example path: `Documentation\HoneywellProWatch_Configuration.docx`
New Device	Relative path to the screenshot displaying on the first page of **Add Device**.
Wizard Image	Wizard where user selects the new parent device type to create. Example path: **Images\deviceWizardAddServerDevice1.png**
New Device Wizard Image 2	Relative path to the screenshot displaying on the first page of the **Add Device Wizard** where user completes the parent device properties. Example path: `Images\deviceWizardAddServerDevice2.png`
Video	
Web Client	If this connector is supported by the Control Center Web Client.
Supported CC Version	
Version	Enter a minimum Control Center version that is supported by the connector. The versions should be entered separately. Start from the first version that supports the ISDK version you are building against.
Supported Operating System	
Capacity	Select whether the operating system is supported on server side, client side or both.Options available: `None. ClientSide. ServerSide` **NOTE:** None means the operating system is supported on both client and server side.
Operating System	Select each operating system that is supported by the driver. Normally, these will correspond with the operating systems supported by Control Center, unless an ISDK does not support one of them.
Supported Hardware	
Firmware Or Software Version	Enter the firmware/software version of the hardware that is supported by the driver.
Model	Enter each hardware device model that is supported by the driver. **CAUTION:** Only enter the top-level devices, such as: Recording Servers, Access Control Nodes and similar. There is no need to provide an infinite list of devices.

Incompatible Device (*optional*) - only add devices here if there are certain known hardware devices that are supported by the subsystem that will not work with the driver.	
Name	Name/model of the incompatible device.
Supported SDK Version	
Name	Name of the SDK.
Sdk Installation Location	Opens a collection editor In the editor you can add multiple install locations. Options available: `None`, `Client`, `Server`, `VideoExportServer`, `ConnectionManagerStreamingServer`
Sdk Limitations	Opens a collection editor. In the editor you can add SDK limitations. A limitation has these properties: • **Item** - The limitation itself. The information is used in SDK Details table, Limitations section. • **Name** - This property is only used in the Driver Designer itself, to help recognize collection items.
Version	Enter the supported SDK version range.
Supported Subsystem	
Additional Info Document	Path to a document containing additional information about the subsystem. You can insert subsystem diagrams, building blocks and explanations to include in the Functionality document.
Description	Description of the subsystem.
Document Links	Any reference documents, such as SDK/API documentation and so on.
Name	Name of the subsystem.
Versions	Compatible subsystem versions.

Video Control Shape

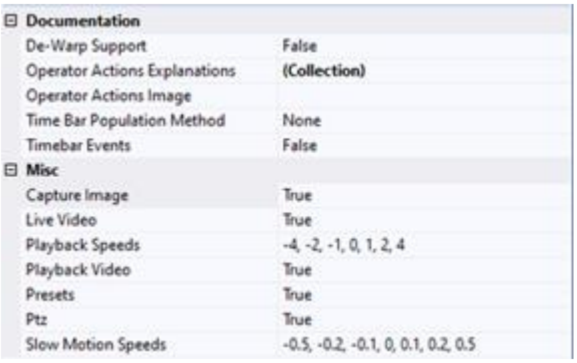
Video Control shape is used for video connectors. It generates VideoControl and adds a custom attribute to the connected contract.

NOTE: In earlier releases, of the ISDK, this shape was associated with the server device. Now, Everbridge recommends that you associate it with it an actual video device.



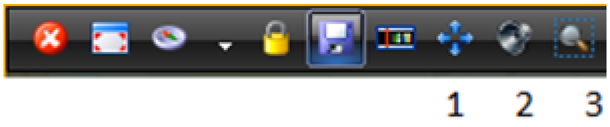
Video Control Shape Properties

Video Control shape properties are visible in the **Video** section of the **Functionality** document for the device the shape is connected to.



⊟ **Documentation**	
De-Warp Support	False
Operator Actions Explanations	**(Collection)**
Operator Actions Image	
Time Bar Population Method	None
Timebar Events	False
⊟ **Misc**	
Capture Image	True
Live Video	True
Playback Speeds	-4, -2, -1, 0, 1, 2, 4
Playback Video	True
Presets	True
Ptz	True
Slow Motion Speeds	-0.5, -0.2, -0.1, 0, 0.1, 0.2, 0.5

Properties that only affect documentation

Property	Description
Operator Actions Image	A path (relative to project root) to the image that contains an image of numbered video operator actions. If no custom operator actions have been added, this is not necessary. 
Operator Actions Explanation	Opens a collection editor. In the editor, you can add an explanation for each of the video operator actions. Please order them according to the numbering in the image.
Timebar Population method	How is the timebar populated. Options available: • `None` - Timebar is not populated. • `AssumeStorage` - Timebar is fully populated regardless of the recordings that exist. • `QueryDevice` - Timebar displays the actual recordings available.

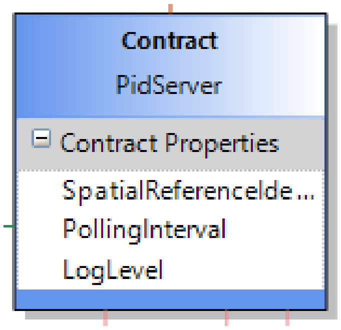
Timebar Events	Whether the driver supports timebar events. These are available from ISDK 3.3.
De-Warp Support	Whether the driver supports de-warping.

Properties that affect code and documentation

Property	Description
Capture Image	If true, generated video control implements **ICapture** interface for saving snapshots.
Live Video	If true, generated video control implements **ILiveVideoControl** interface. You can only use this if the driver supports live video.
Playback Speeds	Playback speeds that the driver (and the SDK) supports. You can only use this if the driver supports playback video.
Playback Video	If true, generated video control implements **IPlaybackVideoControl** interface. You can only use this if the driver supports playback video.
Presets	If true, generated video control implements **IPresets** interface. You can only use this if the driver supports PTZ presets.
Ptz	If true, generated video control implements **IPtz** interface. You can only use this if the driver supports PTZ presets.
Slow Motion Speeds	Slow motion playback speeds that the driver (and the SDK) supports.

Contract Shape

Contract shape is used to define devices. It can have multiple other shapes connected.



Contract Shape Properties

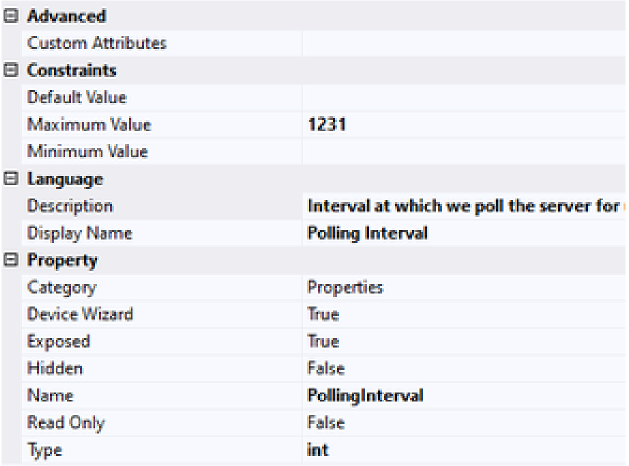
NOTE: Only the properties that affect documentation are described.

Property	Description
Contract	
Custom Attributes	Allows you to apply any attributes to the contract.
Base Class	Used in case this device contract should inherit from another contract. Base class for this contract. If a base class is provided, the Dispose method pattern is not generated in the interface, as it should be inherited from the base class.
Connectable Device	If true, contract implements the **IConnectableDevice** interface which provides Connect and Disconnect methods as well as Timeout and Retry Interval properties.
Hidden	If true, this device will not show up in any of the documentation. Useful for base classes.
Name	Name of the generated class/interface. Use this name to reference the contract in source code.
Networked Device	If true, contract implements the **INetworkedDevice** interface which provides IP and Port. **INetworkedDevice** inherits from **IConnectableDevice**, so if this is true, Connectable Device can be set to false.
Raises Custom State	If true, methods that allow you to raise custom states are generated in the contract.
Secure Device	If true, contract implements the **ISecureDevice** interface which provides Username and Password properties.
Manufacturer	
Manufacturer Description	Description of the manufacturer that can be found in Control Center.
Manufacturer Image	Image that can be found in Control Center connector information page
Manufacturer Image Caption	
Manufacturer Name	For the contract that is connected to the documentation shape, this property gets included in the title page, under manufacturer name and title of the document.
Manufacturer Support	URL that can be found in Control Center connector information page.

Url	
Manufacturer Url	URL that can be found in Control Center connector information page.
Product	
Product Category	For the contract that is connected to the documentation shape, product category is on the title page. It may also be used for licensing
Product Description	Description of the product that can be found in Control Center.
Product Image	URL that can be found in Control Center connector information page
Product Image Caption	
Product Name	Name of the device, used in the heading of each device in functionality document. Examples: BVMS Camera, SymmetryDoor and so on.
Product Url	URL that can be found in Control Center connector information page.
Resources	
Product Image (x * x)	Icons of the device that will be used in Control Center.
Video	
Presets	If true, contract implements the **IPresetsDevice** interface which provides PresetSelected event, PresetsSupported property and SelectPreset method. When the PresetsSupported property is set to True, the Video Control Tile menu includes the preset selector button.
Presets Server	If true, contract implements the **IPresetsServer** interface which provides a SelectPreset method.
PTZ	If true, contract implements the **IPtzDevice** interface which provides a**PtzSupported** property. This interface does not allow a PTZ control, currently a PTZ control is only allowed on Video Controls. When the PTzSupported property is set to True, the mouse cursor becomes an arrow when hovering over Video Control. The PTZ commands are sent to the video control on mouse clicks and scrolls.
Video Export	
File Extension	This property is deprecated.
Maximum Exports	This property is deprecated.
Video Export	This property is deprecated.

Contract Property

Select a property name in the Contract shape to edit the property look and behavior.



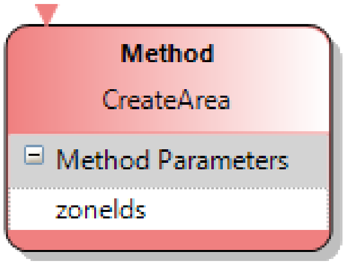
⊟ **Advanced**	
Custom Attributes	
⊟ **Constraints**	
Default Value	
Maximum Value	**1231**
Minimum Value	
⊟ **Language**	
Description	**Interval at which we poll the server for**
Display Name	**Polling Interval**
⊟ **Property**	
Category	Properties
Device Wizard	True
Exposed	True
Hidden	False
Name	**PollingInterval**
Read Only	False
Type	**int**

Properties	Description
Custom Attributes	Allows to apply any attributes to the property. See Custom ISDK Attributes.
Default Value	Used in the documentation for default value of the property, but also sets a default value in Control Center, through [Default Value] attribute.
Maximum Value	ONLY used in the documentation.
Minimum Value	ONLY used in the documentation.
Description	Description of the property. Accurate and full descriptions areencouraged.
Display Name	The name that is visible in documentation and in Control Center.
Category	Category in which the property is visible in Control Center.
Device Wizard	Whether to put the property into device wizard which pops up when a new device is to be created.
Exposed	Whether to write the property as operation contract, which allows access to the property externally through WCF.
Hidden	If true, the property will be hidden in Control Center and in documentation.

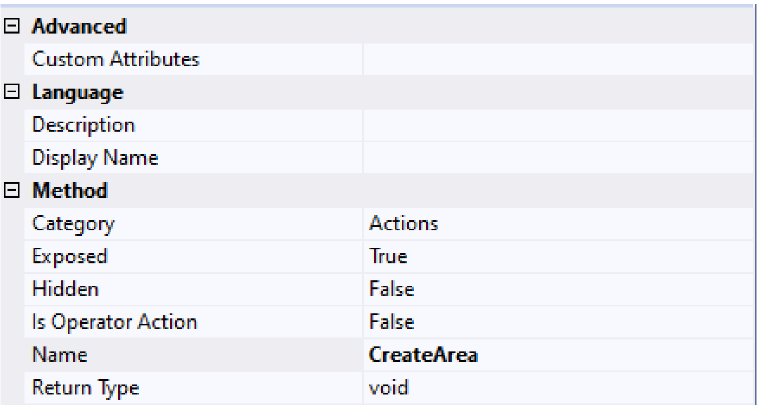
Name	Name of the generated property. Use this name to reference the property in source code.
Read Only	Whether to allow writing to the property in Control Center.
Type	Type of the property.

Method Shape

Method shape is used to define methods of a device contract. They are implemented as operation contracts and are also available through WCF.



Method Shape Properties

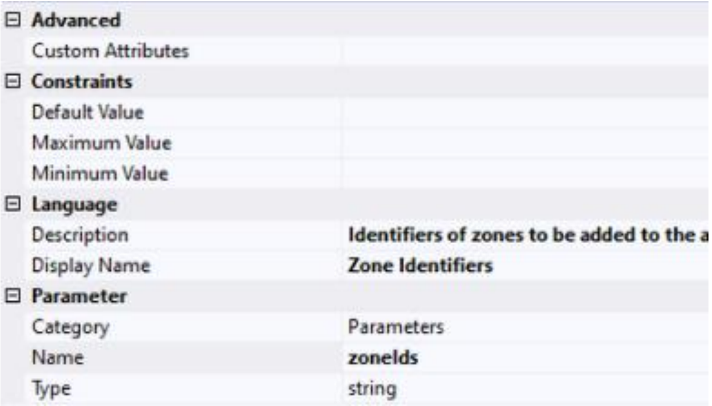


⊟ **Advanced**	
Custom Attributes	
⊟ **Language**	
Description	
Display Name	
⊟ **Method**	
Category	Actions
Exposed	True
Hidden	False
Is Operator Action	False
Name	**CreateArea**
Return Type	void

Property	Description
Custom Attributes	Allows you to apply any attributes to the method. See [Custom ISDK Attributes]().
Description	Description of the method. Accurate and full descriptions encouraged.
Display Name	The name that is visible in documentation and in Control Center.
Category	Category in which the method is visible in Control Center. Normally, it should be set to **Actions** category.
Exposed	Whether the method should be written as Operation Contract, to allow access externally through WCF.
Hidden	If true, the method is hidden in Control Center and in documentation.

Is Operator Action	Whether to make the method an operator action. (Show method in the right click menu available to operators in Control Center).
Name	Name of the generated method. Use this name to reference the method in source code.
Return Type	Return type of the method.

Method Parameter Properties

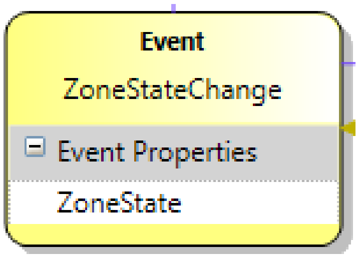


⊟ **Advanced**	
Custom Attributes	
⊟ **Constraints**	
Default Value	
Maximum Value	
Minimum Value	
⊟ **Language**	
Description	**Identifiers of zones to be added to the a**
Display Name	**Zone Identifiers**
⊟ **Parameter**	
Category	Parameters
Name	**zoneIds**
Type	string

Property	Description
Custom Attributes	Allows to apply any attributes to the parameter. See Custom ISDK Attributes.
Default Value	(*Not available*) Used in the documentation for default value of the property, but also sets a default value in Control Center, through [Default Value] attribute.
Maximum Value	ONLY used in the documentation.
Minimum Value	ONLY used in the documentation.
Description	Description of the parameter. Accurate and full descriptions encouraged.
Display Name	The name that is visible in documentation and in Control Center.
Category	Category into which the parameter is placed in Control Center. Normally, this should be set to **Parameters** category.
Name	Name of the generated parameter. Use this name to reference the parameter in the code.
Type	Type of the parameter.

Event Shape

Event shape is used to define events that a device contract can raise.



Event

ZoneStateChange

⊟ Event Properties

ZoneState

Event Shape Properties

Property	Description
Custom Attributes	Allows you to apply any attributes to the event. See Custom ISDK Attributes.
Name	The name of the generated event and event arguments classes. Use this to reference the even in source code.
Description	Description of the event. Accurate and full descriptions encouraged.
Display Name	The name that is visible in documentation and in Control Center.

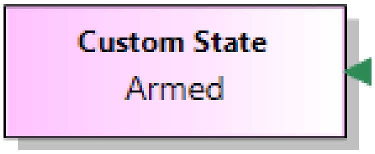
Event Property Properties

Property	Description
Custom Attributes	Allows you to apply any attributes to the property. See Custom ISDK Attributes.
Description	Description of the property. Accurate and full descriptions encouraged.
Display Name	The name that is visible in documentation and in Control Center.
Category	The category in Control Center into which the property is placed. Normally, this should be set to **Properties** category.
Name	Name of the generated property. Use this name to reference the property in source code.
Type	Type of the property.

Custom State Shape

Custom state shape is used to define a custom state. They are not limited to any contracts. For documentation purposes. Everbridge recommends you connect custom states to contracts to show where they are being used.

CAUTION: The custom shape must be connected/mapped to a contract to show up in documentation.

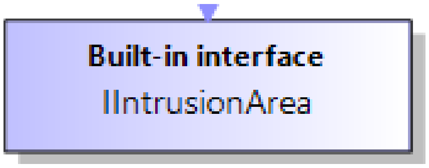


Custom State Shape Properties

Property	Description
Description	Description of what the custom state represents. Used only in documentation.
Display Name	The name that is visible in documentation and in Control Center.
Icon	Icon of the custom state that is visible in documentation and Control Center
Name	The name of the generated custom state class. Use this name to reference custom state in the code.

Built-in Interface Shape

The built-in interface shape is used to allow contracts to implement interfaces that are defined in the DDK. The interfaces can contain properties, methods and events.



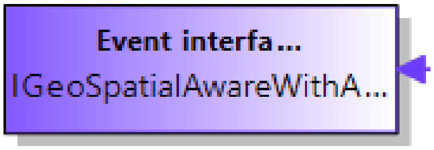
Property	Description
Interface Type	ISDK interface selected from the list.

Built-in Interface Shape Properties

None

Event Interface Shape

The Event Interface shape is used to allow events to implement interfaces that are defined in the ISDK, such as GeoSpatial aware events. These interfaces provide events with properties.



Event Interface Shape Properties

None

Custom ISDK Attributes

Usage of most C# attributes are allowed. For an extensive list of them see https://docs.microsoft.com/en-us/dotnet/api/system.attribute?redirectedfrom=MSDN&view=netframework-4.8 This section defines the attributes that can be entered in **Custom Attributes** field. There are more ISDK attributes. However, they are controlled by properties and in most cases should not be used manually.

NOTE: You may need to specify the full namespace when using these attributes, like this: `[CNL.IPSecurityCenter.Driver.Attributes.Validation.IntegerConstraint(MinValue = 1, MaxValue = 8)]`

Property Value Validation

Custom Attribute	Description
HostNameIPConstraintAttribute	Checks for either a Hostname or IP pattern before accepting the entry. Usage: `[HostNameIPConstraint]`
IntegerConstraintAttribute	Constraints an integer value. Usage: `[IntegerConstraint(MinValue = 1, MaxValue = 8)]`
PortConstraintAttribute	Constraints the entry to an integer from 0 to 65535. Usage: `[PortConstraint]`
StringConstraintAttribute	Constraints a string value. Usage: `[StringConstraint("Message Shown When Value Invalid", AllowNull = false, AllowEmpty = false, RegularExpression = @"^([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])")]`

TimeSpanConstraintAttribute	Constraints a TimeSpan value. Usage: `[TimeSpanConstraint(MinValue = "00:00:10", MaxValue = "00:10:00")]`

Contract Custom Attributes

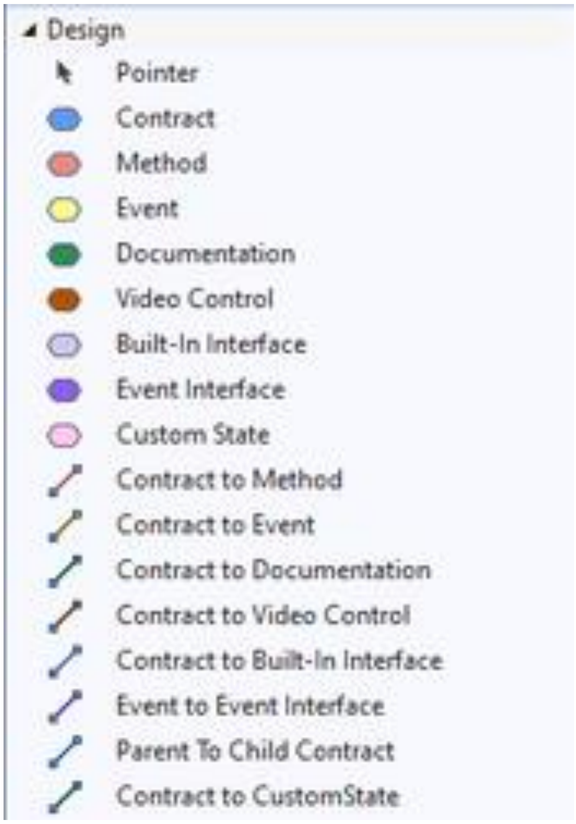
Custom Attribute	Description
DeviceOverridesChildOnlineState	Stops Connection Manager from automatically setting all child devices to online state when parent comes online. This allows individual control of device states

Other Attributes

Custom Attribute	Description
SupportedPreviousDriverAttribute	When there are serialization changes between driver versions, this attribute can be used to convert objects of the old driver to the new driver.

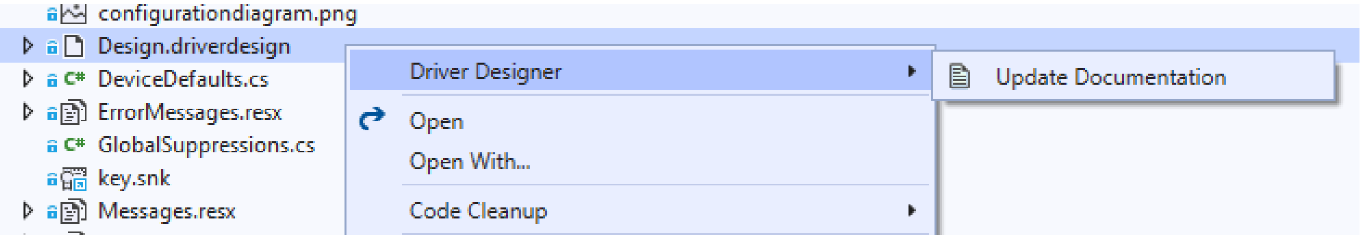
Toolbox

Toolbox contains a list of items that you can add to the diagram.



Right Click Menu

The right click menu contains additional commands that can be executed on the Connector Designer. Currently, there is a single extra command, but this may be extended in the future.



Update Documentation

Running Update Documentation should generate a new xml file that can be opened using Microsoft Word (not tested below 2010 version).

Documentation Generation Failures

Sometimes documentation generation fails. In Visual Studio 2019, if upon generating documentation, you see errors in the Error List, or your generated xml file contains **ErrorGeneratingOutput**, please restart your visual studio (all instances) and try again.

Device Contract

A device contract must be defined for every device type supported by the driver. The contract is an interface used as the WCF service extension. A device contract is defined by a C# interface and its implementation.

NOTE: The device types are called 'contracts' because Connection Manager exposes WCF 'Connection Manager' service where different types are presented as the service contracts.

A device contract is typically implemented by 3 classes:

1. Interface derived from IDevice. These classes are automatically generated by a driver designer in `Design.driverdesign.Contracts.cs.` For example,

   ```
   public partial interface IFusionCatalystServer : IDevice
   ```

2. Class derived from the Device DDK base class. These classes are automatically generated by a driver designer in `Design.driverdesign.ContractBases.cs`. For example,

   ```
   public abstract class FusionCatalystServerBase : Device, IDisposable
   ```

3. Device class implementing the interface in step 1 and derived from the base class in step 2. This class is written by the driver developer implementing the relevant business logic. For example,

```
public class FusionCatalystServer : FusionCatalystServerBase,
IFusionCatalystServer, IDeserializationCallback
```

Device Contract Class Format

Constructor

You must implement a parameterless constructor to create the device manually by right-clicking **New** > **Device** on menu.

CAUTION: Never implement protected Serialized class members as this leads to serialization problems when new devices are populated.

Private Fields

Most private fields need to be marked as non-serialized.

```
[NonSerialized]
 private ILog _log;
```

Such fields need to be initiated not just in a constructor, but also in a special method `InitializeFields()` called when the device is been deserialized when the connector is been loaded in the Connection Manager.

For example, a typical non-connectable device contract is shown below.

```
[Serializable]
     [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
     public class AccessPoint : AccessPointBase, IAccessPoint,
IDeserializationCallback
     {
         [NonSerialized]
         private ILog _log;

         //this is called when a new device is created in Control Center
         public AccessPoint()
         {
             Interfaces.Add(new DeviceInterface(DeviceInterfaceType.Door,
"Door Output", "1"));
             InitializeFields();
         }

         //this is called when device is deserialized from Database
         public void OnDeserialization(object sender)
         {
             InitializeFields();
         }

         private void InitializeFields()
         {
             //initialize any non-serialized fields here
             _log = LogManager.GetLogger("Access Point");
//subscribe to EnabledChanged to set the device to correct state
             EnabledChanged += AccessPointEnabledChanged;
         }
     }
```

In some special cases, usually when a device needs to persist its state even when Connection Manager is offline, the field can be declared without the `[NonSerialized]` attribute, so it will be serialized into Connection Manager database. These properties should be initialized in constructor and not in the `InitializeFields()`.

```
[Serializable]
     [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
     public class VideoCamera : VideoCameraBase, IVideoCamera
     {
         private PresetCollection _presets;

         public VideoCamera()
         {
             _presets = new PresetCollection();
      InitializeFields();
         }
      }
```

Drivers Public Methods

`Connect()` is called when a Connectable device is **Enabled** in Control Center or a Connectable device has not connected (the device has not reported **Online** state) during the time period set by the **Timeout** property.

There are four basic elements typically present in `Connect()` method implementation:

1. Initialization and connection to the subsystem.
2. Get the list of relevant subsystem devices, and populate the relevant Control Center child devices.
3. Subscribe to events and alarms.
4. Start monitoring the connection with the subsystem, if not automatically provided by the subsystem API/SDK.

`Disconnect()` is called when a Connectable device is **Disabled** in Control Center. Implement resources design patterns here, and not in `Dispose()`

Connectable Device Contract Class Implementation

1. Create a Contract on Driver Design surface.
2. Add and connect relevant Methods, Events, Custom States, and Interfaces.
3. Create device contract class.
4. Implement `Connect()` method.
5. Implement `Disconnect()` method.
6. Raise the events created in 1.
7. Implement the methods created in 1.
8. Set device states.

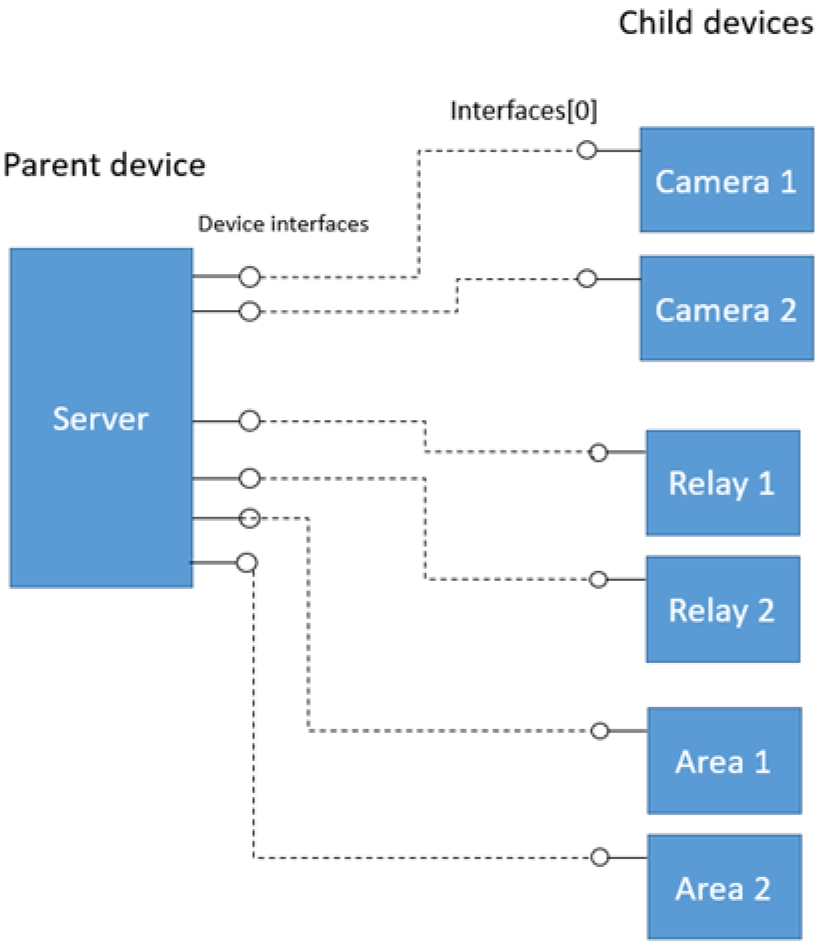
Non-Connectable Device Contract Class implementation

1. Create a Contract on Driver Design surface
2. Add and connect relevant Methods, Events, Custom States, and Interfaces.
3. Implement device contract class.
4. Handle EnabledChanged event.
5. Raise the events created in 1.
6. Implement the methods created in 1.
7. Set device states.

Populating Child Devices

Once a connectable device has established a connection with a subsystem, the next step is to retrieve a list of relevant physical entities (cameras, doors) or logical entities (inputs, areas) and create Control Center devices connected to the parent device.

Devices use interfaces to connect to other devices. Each Control Center device can have a list of interfaces.



Device population needs to occur in the following scenarios:

1. Parent connectable device is **Enabled** and successfully connected to the subsystem. For example, server device is connected to an NVR and needs to populate camera devices.
2. For drivers with multi-tier device hierarchy, a non-connectable device is **Enabled**. For example, in an ASC driver a Door Controller device is **Enabled** and needs to populate the doors connected to this controller.
3. A standard method, **Update Devices**, sometimes called **Repopulate Devices** or **Sync Devices**, is invoked on a parent device.

NOTE: Each child device Contract class must implement a parameterless Constructor, otherwise Control Center cannot create a new device using the Device Wizard in Control Center.

Populating Single Child Device

```
var customIdentifier = input.ID; //custom identifier must be unique, 
typically provided by a native SDK;

 if (!Interfaces.Contains(customIdentifier))
 {
     var inputDevice = GetConnectedDevice<HuperInput>(customIdentifier);
     if (inputDevice == null)
     {
         // creating a new input device
         inputDevice = new HuperInput
         {
             Label = input.Name,
             Id = input.Id
         };

         try
         {
             // Creating the device interface and connecting it to the server 
interface
             var serverInput = new DeviceInterface(DeviceInterfaceType.Other, 
inputDevice.Label, customIdentifier);
             this.Interfaces.Add(serverInput);
             serverInput.Connect(inputDevice.Interfaces.First());
         }
         catch (ArgumentException ex)
         {
             _log.Error($"{IP}: Failed to populate Input device - 
{ex.Message}", ex);
         }
         catch (InvalidOperationException ex)
         {
             _log.Error($"{IP}: Failed to populate Input device - 
{ex.Message}", ex);
         }
     }
 }
```

Populating Multiple Child Devices

```
var newConnections = new InterfaceConnectionCollection();
            var devicesAdded = false;
            foreach (var camera in sdkCameras)
            {
                var customIdentifier = camera.ID; //custom identifier must be
unique, typically provided by a native SDK
                var cameraDevice =
GetConnectedDevice<AxisCamera>(customIdentifier);
                if (cameraDevice != null)
                {

                    //The camera device already exists in the system
                    _log.Info("Skipping camera {0}", camera.ID);
                    //update the camera device' properties if needed and save
them by invoking OnPropertyChanged on the device
                }
                else
                {
                    try
                  {
                        cameraDevice = new AxisCamera
                       {
                            Label = camera.Name
                        };
                        if (!Interfaces.Contains(customIdentifier))
                        {
                           AddToConnections(customIdentifier, cameraDevice,
newConnections);
                            devicesAdded = true;
                        }
                    }
                    catch (ArgumentException ex)
                    {
                       throw new
FatalDriverException(ErrorMessages.FailedToPopulateCamera.CurrentFormat(ex.Me
ssage));
                    }
                    catch (InvalidOperationException ex)
                    {
                        throw new
FatalDriverException(ErrorMessages.FailedToPopulateCamera.CurrentFormat(ex.Me
ssage));
                  }
                }
            }
            // Adds all the new connections to the database in one go
            if (devicesAdded)
            {
                Interfaces.AddAndConnectRange(newConnections);
            }
        //create a new interface on parent device
        private static void AddToConnections(string customIdentifier, IDevice
device, InterfaceConnectionCollection newConnections)
        {
            if (device == null || newConnections == null)
            {
                return;
            }
            var serverInput = new DeviceInterface(DeviceInterfaceType.Video,
device.Label, customIdentifier);
            newConnections.AddAndConnect(serverInput, device.Interfaces[0]);
        }
```

Notes:

- If the population of devices takes significant time and you put it on a background Task remember to provide for task cancellation if the server is taken offline. See Populating Devices as a Background Task
- Also remember to block multiple instances of the task, if an update capability is provided as an exposed method.

Populating Large Number of Devices

Populating many devices at once is a relatively expensive SQL operation and may get a SQL Transaction timeout in Connection Manager. This means only part of the device set gets populated and may lead to inconsistencies in the database. The solution is to populate devices in small batches, so each small population transaction is successful.

Populating Devices as a Background Task

Although your development environment may have only a few devices to work against, your production environment may have many hundreds of devices/sensors. This can lead to the connect/population of devices taking many minutes, possibly, causing the Connection Manager to fail the device.

A workaround is to pass the population of devices onto a background task, leave it to complete and indicate the device as **Online** as soon as its connected to the subsystem (rather than waiting until all the devices are populated).

Notes:

- if no error handling/checking is implemented in the background task then device population can fail with no indication of the failure. In other words, not all devices are created/populated.
- no checking of the status of background task means:
    - if the device is taken offline, the background device creation task continues.
    - Changing the device state rapidly (for example, pressing F12 multiple times to enable/disable a parent device) can cause multiple background device creation tasks to be active, leading to duplication of devices in the system.

Populate Child Devices With a Task Cancellation

In the parent device class, add the following:

```
[NonSerialized]
 private CancellationTokenSource tokenSource;
 [NonSerialized]
 private CancellationToken cancelToken;
 [...]
 /// <summary>
 /// Connects to the physical device.
 /// </summary>
 [SuppressMessage("Microsoft.Design",
"CA1031:DoNotCatchGeneralExceptionTypes")]
 public override void Connect()
 {
     try
     {
         CheckDisposed();
         var username = DeviceDefaults.DefaultUsername(this);
         var port = DeviceDefaults.DefaultPort(this);
 [...]
         log.InfoFormat(CultureInfo.CurrentCulture,
ErrorMessages.ConnectingText, username, IP, port);
         lock (lockInstance)
         {
             Disconnect();
             //
             // Should never get to this state
             // but just in case
             //
             if (tokenSource != null)
             {
                 tokenSource.Cancel();
                 tokenSource.Dispose();
             }
             tokenSource = new CancellationTokenSource();
             cancelToken = tokenSource.Token;
             if (string.IsNullOrEmpty(IP))
             {
                 throw new
ArgumentException(ErrorMessages.IPAddressNotSpecified);
             }
 [...]
             log.DebugFormat(CultureInfo.InvariantCulture, "Last Event
Received: {0}", LastEventReceived);
             if (RetrieveOfflineEvents &&
(!string.IsNullOrEmpty(LastEventReceived)))
             {
               Task.Run(() => GetOfflineEvents(cancelToken), cancelToken);
             }
             Task.Run(() => PopulateDevices(cancelToken), cancelToken);
 [...]
         }
     }
     catch (DeviceException ex)
     {
         log.Error(ex.Message, ex);
         OnStateChanged(DeviceState.Failed, ex.FullMessage);
         Disconnect();
     }
     catch (Exception ex)
   {
         log.Error(ErrorMessages.DeviceConnectionFailed, ex);
         OnStateChanged(DeviceState.Failed,
ErrorMessages.DeviceConnectionFailed + Environment.NewLine + ex.Message);
         Disconnect();
     }
 }
 /// <summary>
 /// Disconnects from the physical device.
 /// </summary>
 public override void Disconnect()
 {
     PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(string.Empty));
     CheckDisposed();
     log.InfoFormat(CultureInfo.CurrentCulture, ErrorMessages.Disconnecting,
DeviceDefaults.DefaultUsername(this), IP, DeviceDefaults.DefaultPort(this));
 [...]
     lock (lockInstance)
     {
         //
         // Cancel any running background task
         //
         tokenSource?.Cancel();
 [...]
     }
     //
     // and destroy the token source/token from the system
     //
     tokenSource?.Dispose();
     tokenSource = null;
     log.InfoFormat(CultureInfo.CurrentCulture, ErrorMessages.Disconnected,
DeviceDefaults.DefaultUsername(this), IP, DeviceDefaults.DefaultPort(this));
 }
 /// <summary>
 /// Populates the devices connected to the server.
 /// </summary>
 private void PopulateDevices(CancellationToken token)
 {
     try
     {
         //
         // Was cancellation already requested?
         //
         if (token.IsCancellationRequested)
         {
             log.InfoFormat("Task {0} was cancelled before waiting for
network data.", MethodBase.GetCurrentMethod().Name);
             token.ThrowIfCancellationRequested();
         }
         //        // if you split the population into additional methods
remember to hand the token through to those and check
```

```
         // at each stage for termination so as to terminate the task as 
quickly as possible, otherwise
         //
         // Foreach device
         //     is cancelation requested?
         //       break out the task
         //     else
         //       add device
         //

         [...]
     }
     catch(Exception ex)
     {
         //
         // report something here
         //
         [...]
     }
 }
```

Repopulating a Deleted Device

You can repopulate a child device that has been previously deleted. As the parent device still has the Interface created for the deleted child device, repopulating the child device means:

1. Create a new child device object
2. Connect the interface on the parent server to the first Interface on the new child device:
   `Interfaces[customIdentifier].Connect(cameraDevice.Interfaces[0]);`

```
var newConnections = new InterfaceConnectionCollection();
       var devicesAdded = false;
            foreach (var camera in sdkCameras)
            {
                var customIdentifier = camera.Id;
                var cameraDevice = 
GetConnectedDevice<AmsCamera>(customIdentifier);
                if (cameraDevice == null)
                {
                    // Create new device
                    cameraDevice = new AmsCamera
                    {
                        Label = camera.Name
                    };
                    if (Interfaces.Contains(customIdentifier))
                    {
                        //repopulate the camera device
                        Interfaces[customIdentifier].Connect(cameraDevice.Int
erfaces[0]);
                   }
                    else
                    {
                        var serverInterface = new
DeviceInterface(DeviceInterfaceType.Video, cameraDevice.Label, 
customIdentifier);
                        newConnections.AddAndConnect(serverInterface, 
cameraDevice.Interfaces[0]);
                        devicesAdded = true;
                    }
                }
                //set additional device properties if needed
            }
            if (devicesAdded)
            {
                Interfaces.AddAndConnectRange(newConnections);
            }
```

Navigating Device Hierarchy

Get Child device

There are two methods available to get a connected child device from a parent device.

1. `T GetConnectedDevice<T>(string customIdentifier)` where T : IDevice
   Returns null if no connected device is found.

   Example of usage: in the parent device class run:

   ```
   Camera cameraDevice = 
   GetConnectedDevice<AccessControlController>(customId);
   ```

   Runs Stored Procedure *Read_DeviceChildrenByCustomIdentifier* on CM Database, selects the first child device which custom ID is as given. This means that custom ID must be unique for its parent device, in other words, custom ID is not necessarily globally unique.

2. `T GetConnectedDevice (DeviceInterface deviceInterface)` where T : IDevice

   Returns null if no connected device is found.

   NOTE: This method is rarely used.

   Example of usage:

   ```
   Camera cameraDevice = GetConnectedDevice<Camera>(Interfaces[0]);
   ```

Get Parent Device

```
T GetConnectedParentDevice<T>() where T : IDevice
```

If the device has no parent throws `NullReferenceException`.

Usage: in the child device class run:

```
var parentDevice = GetConnectedParentDevice<VideoServer>();
```

NOTE: if the camera has more than one parent device, the method will return the parent added the first. This should not normally happen, but it can be achieved, for example, by manually connecting Device Interfaces using the Manage Device Connections option in Control Center's System Configuration.

Get Device Custom Identifier (String) from Device GUID

```
private string GetCustomIdentifier(Guid deviceIdentifier)
 {
    var serviceFactory = new ServiceFactory();
    var deviceDescriptorFactory =
serviceFactory.GetService<IDeviceDescriptorFactory>();
    var deviceDescriptor = deviceDescriptorFactory.Create(deviceIdentifier);
    if (deviceDescriptor.Interfaces.Count > 0 &&
deviceDescriptor.Interfaces[0].ConnectedInterfaces.Count > 0)
    {
        return
deviceDescriptor.Interfaces[0].ConnectedInterfaces[0].CustomIdentifier;
    }
    return null;
 }
```

Get Device from Device GUID

```
Guid deviceId = ...
 var serviceFactory = new ServiceFactory();
 var deviceRepository = serviceFactory.GetService<IDeviceRepository>();
 var cameraDevice = deviceRepository.Read<NextivaCamera>(deviceId);
```

Check whether the device is connected to another device (checking its list of Interface connections):

```
var serviceFactory = new ServiceFactory();
 var deviceDescriptorFactory =
serviceFactory.GetService<IDeviceDescriptorFactory>();
 var deviceDescriptor = deviceDescriptorFactory.Create(deviceIdentifier);
 var connectionInfo = deviceDescriptor.SimpleConnectionInformation; return
connectionInfo.Count > 0;
```

Find a parent device by a child device GUID (parent device on another driver)

```
private static Device FindParentVideoServer(Guid playbackCameraIdentifier)
 {
    var factory = new ServiceFactory();
    var descriptorFactory = factory.GetService<IDeviceDescriptorFactory>();
    DeviceDescriptor cameraDescriptor = null;
    try
    {
      cameraDescriptor = descriptorFactory.Create(playbackCameraIdentifier);
    }
    catch (NullReferenceException)
    {
       throw new ConfigurationException("Cannot find video playback camera
specified in the configuration.");
```

```
    }
    var serverIdentifier =
descriptorFactory.Create(playbackCameraIdentifier).SimpleConnectionInformatio
n[0].ParentIdentifier;
    var deviceFactory = factory.GetService<IDeviceRepository>();
    return (Device)deviceFactory.Read<IDevice>(serverIdentifier);
 }
```

Another example, used in any CCTV driver in *Initialize()* method:

```
private DeviceConnectionInformation _connectionInformation;
 private IVideoServer _server;
 public void Initialize(Guid deviceIdentifier, IDeviceDescriptorFactory
deviceDescriptorFactory, IDeviceRepository deviceRepository)
 {
    var cameraDescriptor = deviceDescriptorFactory.Create(deviceIdentifier);
    var connectionInformation =
cameraDescriptor.SimpleConnectionInformation.GetByParentType(typeof(IVideoSer
ver));
    _server =
deviceRepository.Read<IVideoServer>(connectionInformation.ParentIdentifier);
 }
```

Device Interfaces

Each Control Center device has a collection of Control Center Device Interfaces. A Control Center Device Interface models a logical or physical connection to another device.

It is represented in the Control Center ISDK as a `DeviceInterface` class. The `DeviceInterface` class has the following properties:

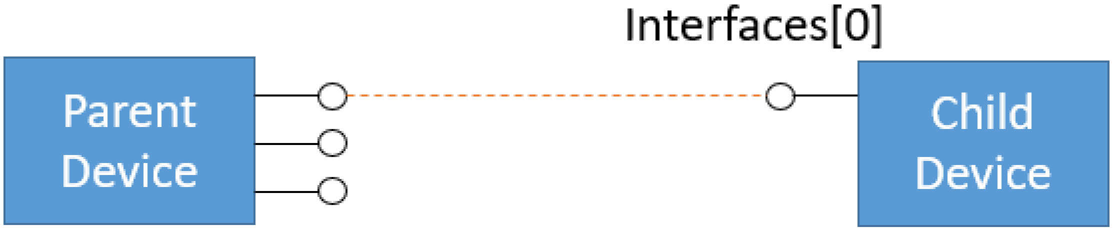
Property	**C# Type**
Identifer	GUID
Label	string
CustomIdentifier	string
Type	DeviceinterfaceType

Device Connection

To connect 2 Control Center devices, each device must have a Control Center Device Interface (see Device Interfaces for more information) and there should be a connection between the two Control Center Device Interfaces.

Notes:

- The custom identifier may not necessarily be the same on both Control Center Device Interfaces.
- Both Control Center Device Interfaces must have the same Type (for example, `DeviceInterfaceType.Door`, `DeviceInterfaceType.Video` and so on).



Typically, a device connection is needed when child devices are populated by a parent server device. For example, in a VMS system, a parent device is a VMS server and a child device is an NVR or DVR or a camera.

The following example connects 2 devices in an Access Control system. The child device is a door.

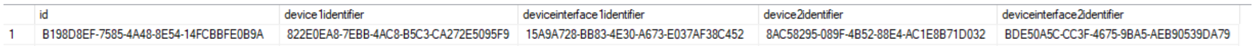
Child Device Class

```
public Door()
          {
                Interfaces.Add(new DeviceInterface(DeviceInterfaceType.Door,
"Door Output", "1"));
          }
```

Server (Parent) Device Class

```
Device doorDevice = ...  //either create new device or get an existing
Control Center device
          //custom Identifier must be a unique identifier typically provided
by the subsystem SDK/API
          var customIdentifier = doorId.ToString();
          var serverInput = new DeviceInterface(DeviceInterfaceType.Door,
doorDevice.Label, customIdentifier);
          this.Interfaces.Add(serverInput);
          serverInput.Connect(doorDevice.Interfaces[0]); //connect up the 2
Device Interfaces
```

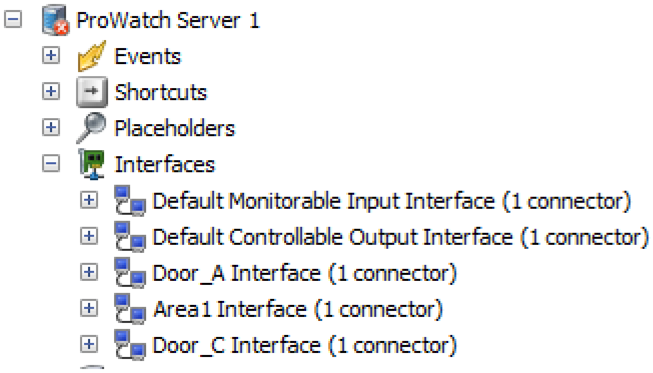
The Interface connection is saved in Connection Manager database, in `DeviceInterfaceConnection` table:



	id	device1identifier	deviceinterface1identifier	device2identifier	deviceinterface2identifier
1	B198D8EF-7585-4A48-8E54-14FCBBFE0B9A	822E0EA8-7EBB-4AC8-B5C3-CA272E5095F9	15A9A728-BB83-4E30-A673-E037AF38C452	8AC58295-089F-4B52-88E4-AC1E8B71D032	BDE50A5C-CC3F-4675-9BA5-AEB90539DA79

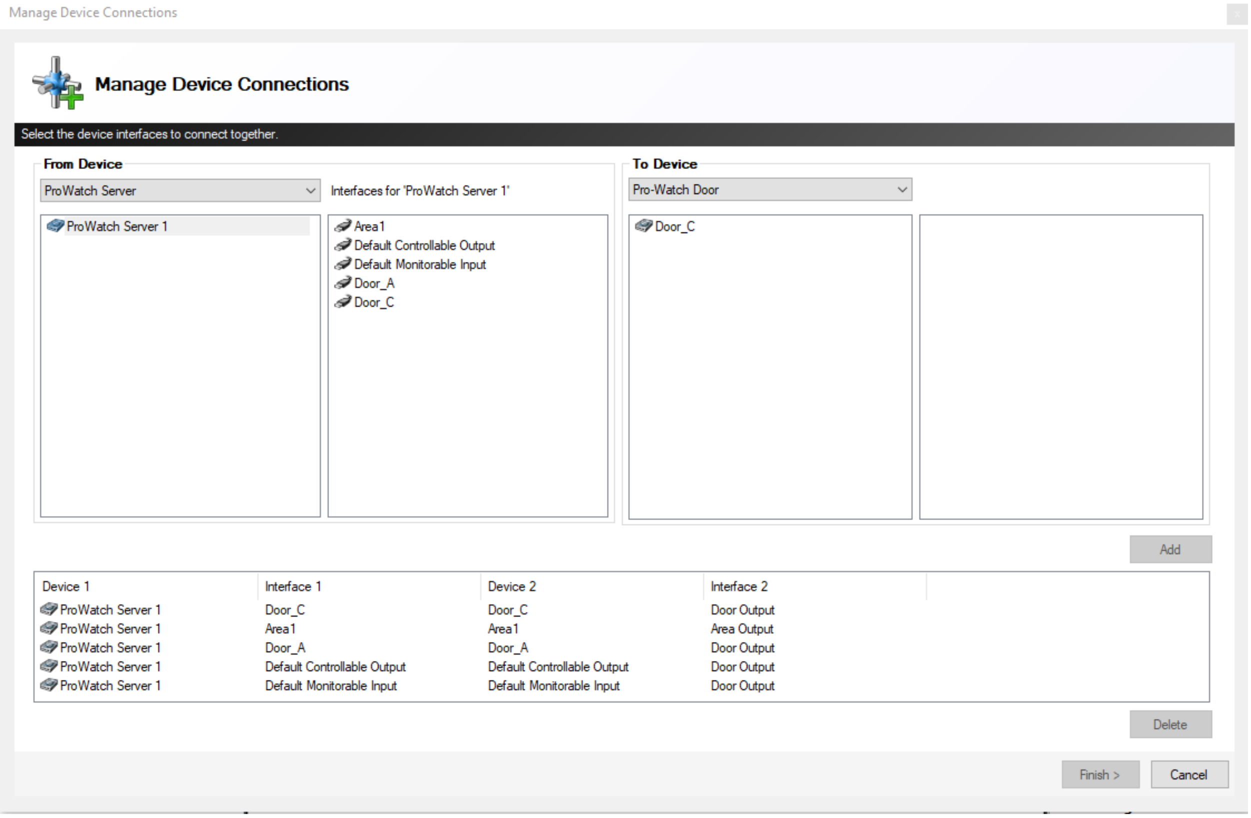
To view and manipulate Device Interface connections in Control Center

1. By expanding the **Device Interfaces** node in **System Configuration**.



- ProWatch Server 1
  - Events
  - Shortcuts
  - Placeholders
  - Interfaces
    - Default Monitorable Input Interface (1 connector)
    - Default Controllable Output Interface (1 connector)
    - Door_A Interface (1 connector)
    - Area1 Interface (1 connector)
    - Door_C Interface (1 connector)

2. In **System Configuration**, right-click on your device and select **Manage Device Interfaces.**



Manage Device Connections

Manage Device Connections

Select the device interfaces to connect together.

From Device

ProWatch Server

ProWatch Server 1

Interfaces for 'ProWatch Server 1'

- Area1
- Default Controllable Output
- Default Monitorable Input
- Door_A
- Door_C

To Device

Pro-Watch Door

Door_C

Device 1	Interface 1	Device 2	Interface 2
ProWatch Server 1	Door_C	Door_C	Door Output
ProWatch Server 1	Area1	Area1	Area Output
ProWatch Server 1	Door_A	Door_A	Door Output
ProWatch Server 1	Default Controllable Output	Default Controllable Output	Door Output
ProWatch Server 1	Default Monitorable Input	Default Monitorable Input	Door Output

Connectivity Monitoring

Connectable devices, in other words, Control Center devices which implement a device interface, must implement some connectivity check logic to report when the corresponding physical device or server is disconnected or re-connected.

Reflecting Current Device State

Devices must always reflect their current state.

- On connection
- On change state event
- On re-enabling the device. **Important**: in Control Center 4.9 the device goes to Online by default. To workaround this, subscribe to `EnabledChanged` event on the ISDK base class device. Example:

```
private void InitializeFields()
        {
            EnabledChanged += MxProDevice_EnabledChanged;
        }
        protected override void Dispose(bool disposing)
        {
            if (disposing)
      {
                ...
                EnabledChanged -= MxProDevice_EnabledChanged;
            }           base.Dispose(disposing);
        }
        private void MxProDevice_EnabledChanged(object sender,
EventArgs e)
        {
           if (!Enabled)
            {
                return;
            }
            //must run in a separate thread otherwise the delay won't
have any effect
            Task.Run(() =>
            {
              //resolve race condition: wait until Connection Manager
sets the device to Online state
               Thread.Sleep(DeviceEnabledDelay);
               //update the current device state
               InitDeviceState();
            });
        }
```

- Device was removed from the 3rd party. The corresponding Control Center device must be in Failed state and have a description of **Device doesn't exist** or **Device not found**. The work around is to compare the list of Control Center devices and the list of the 3rd party devices. The `.Except() LINQ` method gives you the orphaned devices. Example:

```
private void UpdateOrphanedDevices(IEnumerable<string>
knownDevicesCustomIds)
    {
            //all the devices which are not in the collection of
knownDevicesCustomIds don't represent any Pro-Watch entity - set to
Failure
            var orphanedCustomIds = Interfaces.Select(each =>
each.CustomIdentifier).Except(knownDevicesCustomIds);
```

```
            foreach (var customId in orphanedCustomIds)
            {
                string id;
            var deviceType = 
SateonCustomIds.ParseDeviceCustomId(customId, out id);
                IDevice device = GetConnectedDevice(customId);
                var sateonDevice = device as ISateonDevice;
                if (sateonDevice != null)
                {
                    sateonDevice.SetState(DeviceState.Failed, 
ErrorMessages.DeviceDoesntExist);
               }
            }
     }
```

- Device was renamed in the 3rd party. In this scenario there are two possible workarounds:
    - If an SDK supports events about devices been renamed - automatically rename Label property of relevant Control Center devices
    - If an SDK doesn't support such events - add a separate method `UpdateDevices()` which polls all the devices' current name and properties and updates them in Control Center. **Note**: the device Contract must implement the `INotifyPropertyChanged` interface to update Properties

Reporting Child Device States

There are two ways of reporting child device states in Control Center ISDK.

- Find the child device Interface. Example:

```
var cameraInterface = Interfaces.FirstOrDefault(interf => 
interf.CustomIdentifier == deviceId);
       if (cameraInterface != null)
       {
          OnStateChanged(cameraInterface, DeviceState.Failed, 
CustomErrorMessages.CameraConnectionStateDisconnected);
       }
```

- Get the actual child device and raise a public method on it.
    - Get the device in the parent device class. Example:

```
var device = GetConnectedDevice<IDevice>(customId);
        if(device != null)
        {
           device.SetState(DeviceState.Failed, 
ErrorMessages.DeviceNotFound);
        }
```

-
    - The public method in the device Contract class:

```
public void SetState(DeviceState state, string message)
        {
            if (!Enabled)
            {
                return;
            }
            if (_session != null)
            {
               _log.Debug(MxProMessages.SettingDeviceState.Curren
tFormat(Label, state, message));
            }
            if (string.IsNullOrEmpty(message))
            {
                OnStateChanged(state);
            }
            else
            {
                OnStateChanged(state, message);
            }
    }
```

Custom States

In addition to the built-in standard states, a device can also expose custom states, for example, 'door locked', 'zone armed'. A device can only have one current state. If a door is set to a custom state 'door locked' it will no longer be online or failed and these states have to be assumed.

There is no way to retain custom states information after reconnecting to a 3rd party system if your API does not support current state polling.

NOTE: You should not store a state cache in a database as the states may become outdated while a device is offline.

The example below implements custom states without using a connector designer surface.

Implement a CustomStateChanged event in the device contract class.

```
 [field: NonSerialized]
   public event EventHandler<CustomStateChangedEventArgs> CustomStateChanged;
   private void OnCustomStateChanged(CustomStateChangedEventArgs e)
   {
            if (e == null)
            {
                throw new ArgumentNullException("e");
            }
            if (CustomStateChanged != null)
            {
                CustomStateChanged.Invoke(this, e);
            }
        }
    }
    public void RaiseCustomStateChanged(ICustomState state, string message)
    {
            if (state == null)
            {
                throw new ArgumentNullException("state");
            }
            OnCustomStateChanged(new CustomStateChangedEventArgs(Identifier,
state, message));
    }
```

Implement the following methods in the device contract class:

```
public void RaiseStateChanged(DeviceState state, string message)
   {
       OnStateChanged(state, message);
   }
```

Implement your Custom States - class per state

Example using System;

```
namespace CNL.ControlCenter.Driver.Verint.Nextiva.Ipsc.States
 {
    /// <summary>
    /// The recording off state.
    /// </summary>
    [Serializable]
    public class OfflineState : ICustomState
    {
        /// <summary>
        ///     Gets the end user displayable name for the state
        /// </summary>
        public string DisplayName
        {
            get { return "Offline"; }
        }
        /// <summary>
        ///     Gets the icon
        /// </summary>
        public string Icon
        {
            get { return
"CNL.ControlCenter.Driver.Verint.Nextiva.Images.CameraOfflineState.png"; }
        }
    }
 }
```

Custom State Race Condition

There are 3 scenarios that can cause a device state not to be updated.

- Trying to update individual child devices after a parent device goes Online.
- Re-enabling a device
- Fast state updates

To workaround this, remember the last state change on each device. If the current state came too soon, add a time delay to let the previous state change, finish processing).

Example:

```
[NonSerialized]
        private DateTime _lastStateUpdate;
        private void InitializeFields()
        {
            ...
            _lastStateUpdate = DateTime.MinValue;
        }
        private void UpdateCurrentState(AcsInput<string> input)
        {
            //prevent state update race condition when adjacent state updates
arrive
            if ((DateTime.Now - _lastStateUpdate).TotalMilliseconds < 500)
            {
                _log.Debug("Input '{0}': wait for {1} msec. before state
update".InvariantFormat(Label, SateonSession.CustomStateUpdateDelayMsec));
                Thread.Sleep(SateonSession.CustomStateUpdateDelayMsec);
            }
            //set the current state here
            _lastStateUpdate = DateTime.Now;
        }
```

Device Properties

NOTE: You can set connector properties manually but Everbridge recommends that you use the Design Surface.

Supported Property Types

NET types: uint, short, byte and any 64 bit type are not supported.

Custom type properties can be defined but you must provide the full type name.

Default Property Values

You can set default value property values in connector design surface, but it only works if the custom attributes property is not set.

You can also set it manually by using the standard .NET Custom attribute.

```
[DefaultValue(2000)]
```

or use a different overload `DefaultValue(type, string)`:

```
[DefaultValue(typeof(TimeSpan),"00:00:01")]
```

You can set a default DateTime property value. For example:

```
[DefaultDateTime(DateTimeOrigin.Now, DateTimeOperation.Subtract, 0, 10, 0)]
 DateTime from
 [DefaultDateTime(DateTimeOrigin.Now)]
 DateTime to
```

To configure the default value displayed in Device Wizard, set the value directly in the class constructor.

Add a New Property

To manually add a property to a device, declare the type and property name, and then add the following attribute lines above the declaration.

```
[DisplayName("<Name  of declared Variable>")]
 [Description {"description of what the property does/defines>"}]
 [CategoryProperties]
 <property Type> <Property name>;
```

Make a Property Read Only

To make a device property read only, add the Attribute to the Custom Attributes property:

```
[System.ComponentModel.ReadOnly(true)]
```

Saving and Persisting a Property

To save a device property programmatically, the device Contract class must implement the `INotifyPropertyChanged` interface:

```
[Serializable]
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
    internal class GalaxyOutput : GalaxyOutputBase, IGalaxyOutput,
IDeserializationCallback, IGalaxyDevice, INotifyPropertyChanged
    {
      ...
      [field: NonSerialized]
      public event PropertyChangedEventHandler PropertyChanged;
      ...
      /// <summary>
      /// Raises PropertyChanged event which causes the recently updated
properties saved into Database.
      /// </summary>
      public void SaveChangedProperties(PropertyChangedEventArgs e)
      {
            if (PropertyChanged != null)
            {
                PropertyChanged.Invoke(this, e);
            }
      }
    }
```

Saving all the properties can be implemented like this:

```
public void SaveChangedProperties()
        {
            if (PropertyChanged != null)
            {
                PropertyChanged.Invoke(this, new
PropertyChangedEventArgs(string.Empty));
            }
        }
```

Validating Property Values

Property values should be validated in two places:

1. In code
2. In the Property Grid. Assign Custom Attributes property in the driver designer. Examples:
    - Integer:

      ```
      [CNL.IPSecurityCenter.Driver.Attributes.Validation.IntegerConstra
      int(MinValue=0, MaxValue=int.MaxValue)]
      ```

    - String:

      ```
      [CNL.IPSecurityCenter.Driver.Attributes.Validation.StringConstrai
      nt("The API key must not be empty",AllowNull = false,AllowEmpty =
      false)]
      ```

    - IP:

      ```
      [CNL.IPSecurityCenter.Driver.Attributes.Validation.StringConstrai
      nt("The 'Local Address' property must be set to a valid IP4
      address",

       AllowNull = false, AllowEmpty = false, RegularExpression =
      @"^([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(\.([0-
      9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3}$")]
      ```

    - GUID:

      ```
      [CNL.IPSecurityCenter.Driver.Attributes.Validation.StringConstrai
      nt("The 'Recording ID' parameter must be in a form of a valid
      GUID string: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",

      AllowNull = false,AllowEmpty = false,RegularExpression = @"\b[a-
      fA-F0-9]{8}(?:-[a-fA-F0-9]{4}){3}-[a-fA-F0-9]{12}\b")]
      ```

    - TimeSpan:

      ```
      [CNL.IPSecurityCenter.Driver.Attributes.Validation.TimeSpanConstr
      aint(MinValue="0:0:1", MaxValue="1:0:0")]
      ```

    - Port (integer):

      ```
      [CNL.IPSecurityCenter.Driver.Attributes.Validation.PortConstraint
      ]
      ```

Detecting Property Value Changes

There is no direct way to detect device property values in the connector designer. You must implement a property in code.

Below is some sample code taken from FusionFC4000 connector.

1. Expand the device interface (add partial class):

```
public partial interface IFusionCatalystWebSource
    {
        [CategoryProperties]
        [DeviceWizard]
        [DisplayName("CNL.ControlCenter.Driver.Jupiter.FC4000.Design.dr
iverdesign.Strings", "DisplayNameUrl",
typeof(IFusionCatalystWebSource))]
        [Description("CNL.ControlCenter.Driver.Jupiter.FC4000.Design.dr
iverdesign.Strings", "DescriptionUrl",
typeof(IFusionCatalystWebSource))]
        [System.ComponentModel.DefaultValue(0)]
        string Url
        {
          [OperationContract]
           get;
```

2. Implement the property in the device class:

```
[OperationContract]
           set;
       }
    }
    public class FusionCatalystWebSource : FusionCatalystWebSourceBase,
IFusionCatalystWebSource ...
    {
        private string _url;
        public override string Url
        {
            get
            {
                return _url;
            }
            set
            {
                _url = value;
              //custom code here

            }
        }
    }
```

Device Public Methods

Listed below are the device Public Methods.

`Connect()`	This method is called when a connectable device is **Enabled** in Control Center or a connectable device has not connected (in other words, a connectable device has not reported an **Online** state) during the time period set by the **Timeout** property. There are four basic elements typically present in Connect() method implementation: • Initialization and connection to the subsystem. • Get the list of relevant subsystem devices, and populate the relevant Control Center child device Subscribe to events and alarms. • Start monitoring the connection with the subsystem, if not automatically provided by the subsystem API/SDK.
`Disconnect()`	This method is called when a connectable device is **Disabled** in Control Center, implement resources design patterns here, and not in `Dispose()`.
`Dispose()`	This method is only called when a device is deleted in Control Center (it is not called at any other time, even when Connection Manager is shutting down).

Device Method Name Limitations

Method display names (Display Name property) cannot include characters: '-', '/', '(', ')'

Methods with these characters cannot be called from Response Plans.

Device Methods Parameter Types

Device methods supports standard .NET types:

- int
- string
- double (shown in VRPs as Decimal)
- boolean
- DateTime

The following types are not supported:

- short
- long
- byte
- uint

Complex, 64bit and custom types are not supported.

You must never expose native SDK types in Control Center.

To pass a Control Center device (for example, the ISDK device Contract) as a method parameter, the parameter of type Guid must be defined with `Custom Attribute [DeviceIdentifier]`.

```
bool StartDecoder(
            [DeviceIdentifier(typeof(INextivaDecoder))]
             Guid decoderIdentifier,
             [DeviceIdentifier(typeof(INextivaCamera))]
             Guid cameraIdentifier);
```

To pass a file path as a parameter it is worth implementing access to a File Browser editor. To do this, add the Editor Custom Attribute as follows:

```
[System.ComponentModel.Editor(typeof(FileBrowserEditor),
typeof(UITypeEditor))]
```

Device Method Return Types

Only basic .NET types are currently supported as connector device methods return types.

To return a picture:

Possible solution: return byte[] then, assign the .Image property of an Image Control on a Control Center GUI.

Special case: Herta driver: use a Plugin to decode a picture of Base64 format.

connector methods can return a List of basic .NET types. A Response Plan then can iterate over the list items and process them.

Hide a Method From a Property Grid

Sometimes device methods need to be hidden from the UI. Usually it is internal methods (for example, for testing purposes) or obsolete methods which cannot be removed due to possible Serialization problems for previously deployed connectors.

- To hide a method on connector design surface, set **Hide** property to true on the method shape.
- To hide a method on connectors which do not have a connector design surface, add the `[System.ComponentModel.Browsable(false)]` custom attribute to the method definition in device interface class.

Provide a List of Items

Sometimes the connector needs to provide a list of certain items: layouts, devices, 3rd party users and so on.

Implementation options:

1. The connector implements method `GetItems()` returning A C# List<> (List<string>, List<int>, or List<*supported basic type*>).
2. As an option, the connector can also implement an event `ItemsFound` passing a property of type List<>
3. Commissioner creates a custom GUI displaying the list in a Combobox, for example.

4. In the GUI create a VRP logic for OnLoad Event. In the event call the `GetItems()` on the server VRP Variable, then store the result in another VRP Variable, then use the Iterate Collection shape and on every iteration call .AddItem to the Combobox.

Operator Actions

A connector method can be exposed as an operator action, which means this method is available to the user in a context menu in the display GUI. In the example below, the Integriti Door device has three operator actions:

- Lock Door
- Timed Access
- Unlock Door

To expose a method as an operator action:

1. Set the Is Operator Action property to True
2. Chose a 16x16 unique icon to be shown for the Operator Action in the Context Menus. Add the icon to Images\ folder of the driver project, assign Build Action to Embedded Resource.
3. Assign the icon to the Action by setting custom attribute on the method. For example,

```
[DisplayImage(DisplayImageSize.Image16x16,
@"CNL.ControlCenter.Driver.Hanwha.NVR.Images.AlarmInputOn.png",
typeof(IAlarmInputDevice))]
```

CAUTION: You cannot re-use the icons which were already used in this driver, for Custom States.

If there is a requirement to provide different access levels to different actions, the Category property should be assigned accordingly.

Connector Event Properties

You should avoid properties of type string (except the user-friendly text descriptions), and instead use a strong typed approach if possible.

If an API sends a string property, check what the possible values are, then report it as an enum. (String properties with undefined values should not be used because you cannot build any rules around them in Commissioning).

You must not report unparsed, raw data as an event property unless there is a special need for it. This is because there will be no parsing at the Commissioning stage.

You must always report timestamps as DateTime in UTC format.

Usually there is no need to report an event timestamp as a separate property.

- If the 3rd party provides the native timestamp, use the overload: OnDoorForced(new DoorForcedEventArgs(this, nativeTimestampUtc))
- if the 3rd party doesn't provide a timestamp - use the overload: OnDoorForced(new DoorForcedEventArgs(this)) (the event time will be automatically assigned to DateTime.UtcNow)

You must never expose native SDK types in Control Center.

If the subsystem reports null value in an expected field of type string, set the corresponding event property to string.Empty: event fields which are null are hidden in Control Center which may be misleading.

Raising Connector Events

Avoid caching/serializing 3rd party alarms or events in the connector unless there is a very good reason for it.

Avoid serializing 3rd party event ID counters (something like int _lastEventID)

Dealing with repeated alarms/events from a 3rd party system:

- If the alarms have unique IDs, track (but do not serialize) the last received ID.
- If the subsystem reports the Timestamp, you can filter out the events with Timestamp older than the last received).
- Everbridge recommends that you cache the current devices' state and report the Alarm only if the state changes.

NOTE: Normally if the subsystem has a problem reporting repeated events, it is a bug in the subsystem and ideally should be fixed by the 3rd party.

Reporting Geographic Location

There are three ISDK Interfaces available:

1. `IGeospatialAwareEvent` to raise a geo-aware event to update dots on a map (now deprecated).
2. `ITrackableGeospatialAwareEvent`. This is an extended version of `IGeospatialAwareEvent` which has TrackId on it. Always use this instead of `IGeospatialAwareEvent`.
3. `IGeoSpatialTracking` to make a Control Center device trackable (see ISS demo driver).

CAUTION: Wherever you implement `IGeospatialAware` Interface, you must implement and assign the member: int `SpatialReferenceIdentifier` (as defined by spacialreference.org). Otherwise, Control Center does not plot the reported coordinates on the map.

- Add a property Spatial Reference Identfier to device Server Contract and set a Description: “A unique value used to unambiguously identify projected, unprojected, and local spatial coordinate system definitions.” Default value – 4326
- Assign this ID to `SpatialReferenceIdentifier` property of any event implementing `IGeospatialAware`

Exposing ENUMs

If a connector uses a custom enum type in its methods, events or properties, the type must be exposed to Control Center. To do this, add a custom attribute `CNL.ControlCenter.Driver.Attributes.Description` to the enum.

The following example uses `CNL.ControlCenter.Driver.Attributes;`

```
...
 [Description("Output State")]
 public enum MxProOutputActivationState
 {
      Unknown = 0,
      Activated,
      Deactivated,
 }
```

NOTE: There is currently no way to customize the values of the enum, so you need to make sure they are self-explanatory.

Testing enums within Control Center:

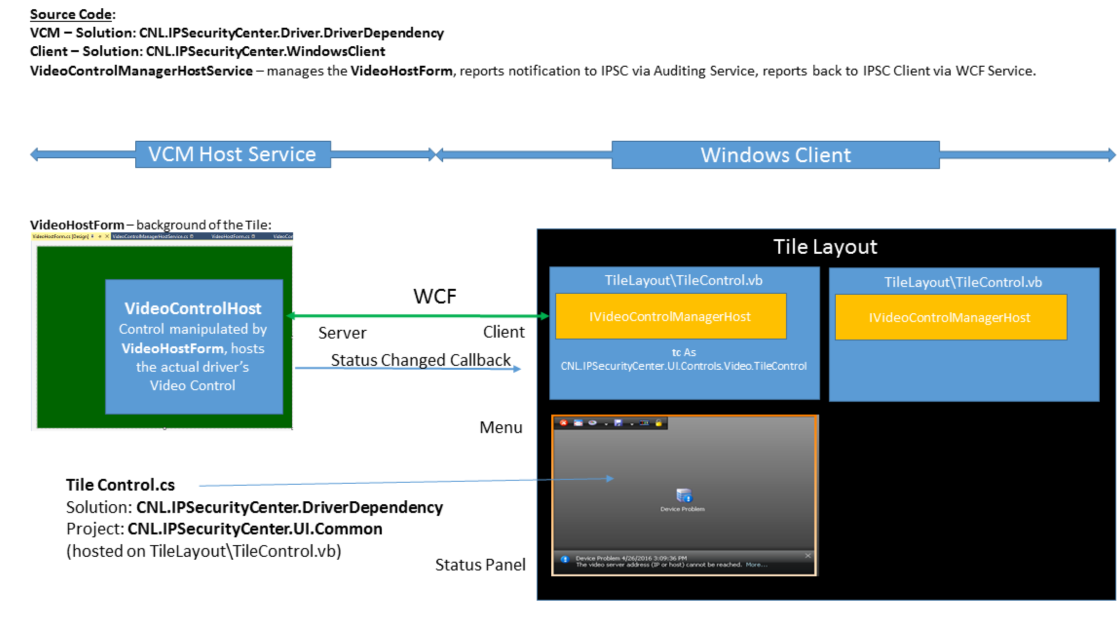
1. Create a new VRP.
2. Create a new variable of type Enum.
3. Select the driver from the drop down list. The second drop-down field must list the available enums.

Developing Video Connectors

You can configure a video control manager (VCM) in which to run your connectors and display their UI component (typically a control from the target system's SDK that shows video).

As with other components that load connectors, a VCM's primary purpose is to isolate other processes from third-party SDK/API instability and unreliability. It has a WCF interface, allowing clients to tell which driver to load, go to playback, and so on. Calls go back to the client control code on another WCF interface describing state changes.

The timebar shown in playback mode is owned by the VCM.



Source Code:
VCM – Solution: CNL.IPSecurityCenter.Driver.DriverDependency
Client – Solution: CNL.IPSecurityCenter.WindowsClient
VideoControlManagerHostService – manages the **VideoHostForm**, reports notification to IPSC via Auditing Service, reports back to IPSC Client via WCF Service.

VideoHostForm – background of the Tile:

Tile Control.cs
Solution: **CNL.IPSecurityCenter.DriverDependency**
Project: **CNL.IPSecurityCenter.UI.Common**
(hosted on TileLayout\TileControl.vb)

Device Problem 4/26/2016 3:09:36 PM
The video server address (IP or host) cannot be reached. More...

VCM Host Service
Windows Client
VideoControlHost
Control manipulated by VideoHostForm, hosts the actual driver's Video Control
WCF
Server
Client
Status Changed Callback
Menu
Status Panel
Tile Layout
TileLayout\TileControl.vb
IVideoControlManagerHost
tc As CNL.IPSecurityCenter.UI.Controls.Video.TileControl
TileLayout\TileControl.vb
IVideoControlManagerHost
Device Problem

Populating Buttons and Controls

- Presets list is populated by calling `GetPresets()` on a camera device implementing **IPresetsDevice**.
- PTZ controls are shown if the property **PTZ Supported** is set to **true** on the displayed camera device (same for Preset controls).

VideoControlHost.cs

```
public bool PtzSupported
       {
            get
            {
                var ptzDevice = _displayedDevice as IPtzDevice;
                return PtzControl != null && (ptzDevice != null &&
ptzDevice.PtzSupported);
            }
        }
        public bool PresetsEnabled
        {
            get
            {
                var ptzDevice = _displayedDevice as IPresetsDevice;
                return PtzPresetControl != null && (ptzDevice != null &&
ptzDevice.PresetsSupported);
            }
        }
```

Video Operator Action buttons

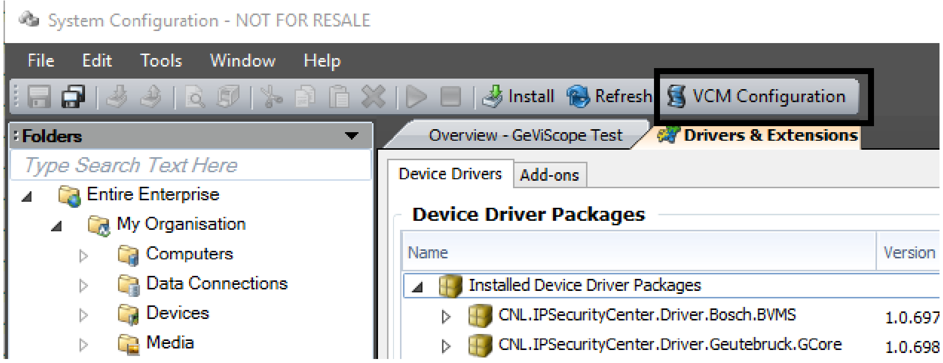
Project: `CNL.IPSecurityCenter.UI.Common`
Class: `TileControl.cs`

```
public partial class TileControl : UserControl
 {
     public IList<ToolStripButton> OperatorActionButtons { get; private set;
}
     ...
     public ToolStripButton AddOperatorActionButton(string text, string
toolTip, Image image, string methodName)
     {
        var button = AddToolStripButton(text, toolTip, image);
        OperatorActionButtons.Add(button);
        return button;
     }
 }
```

VCM Configuration

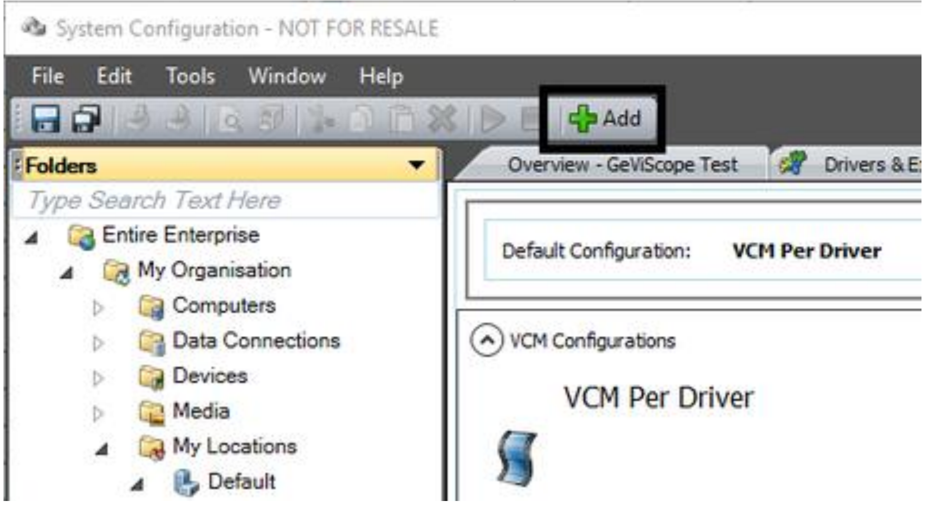
VCM configuration allows you to assign driver video controls to be hosted in various VCMs.

1. From **System Configuration**, select **Drivers & Extensions** > **VCM Configuration**.



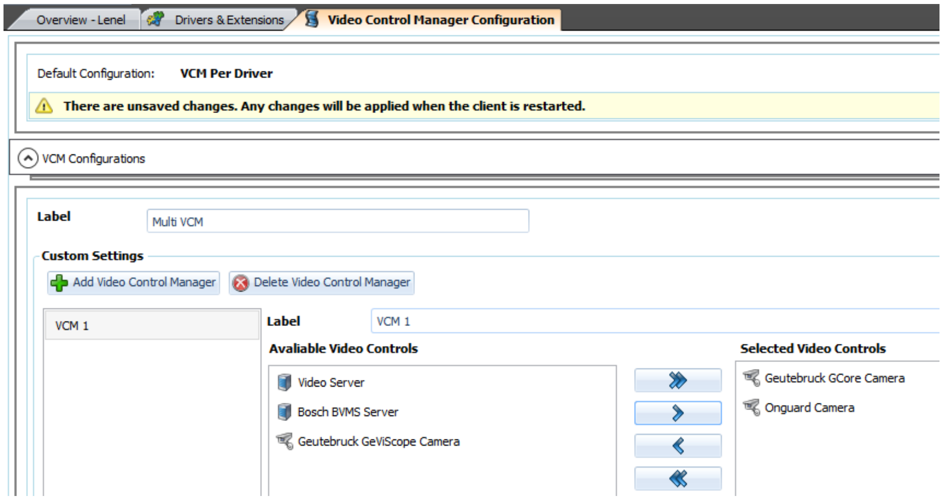
The default configuration is called **VCM Per Driver**. This means that each driver with a Video Control runs in a separate VCM process (if you have 3 video connectors and your Control Center client is set to use **VCM Per Driver** configuration, the Control Center client runs 3 VCM processes, one for each connector).

2. Select **Add**. Create a new VCM configuration



Default Configuration: **VCM Per Driver**

3. Fill in the VCM details:
   a. Type a name for the VCM configuration.
   b. Select **Add Video Control Manager** to add a new VCM to configuration.
   c. Rename the VCM or leave the default label **VCM 1**.
   d. Assign driver Video Controls to this VCM by clicking > , so all the hosted Video Controls appear on the right-hand side:



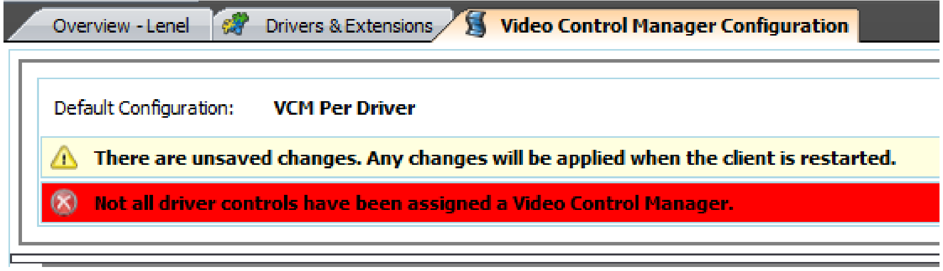
Default Configuration: **VCM Per Driver**

⚠ **There are unsaved changes. Any changes will be applied when the client is restarted.**

Label Multi VCM

Label VCM 1

4. Add & and configure more VCMs if needed. You can only save a VCM configuration once all the Video Controls are assigned. In other words, each Video Control is hosted on at least one VCM. If you have not assigned a video control to a VCM configuration, a **Not all connector controls have been assigned a Video Control Manager** error message displays when you try to save.

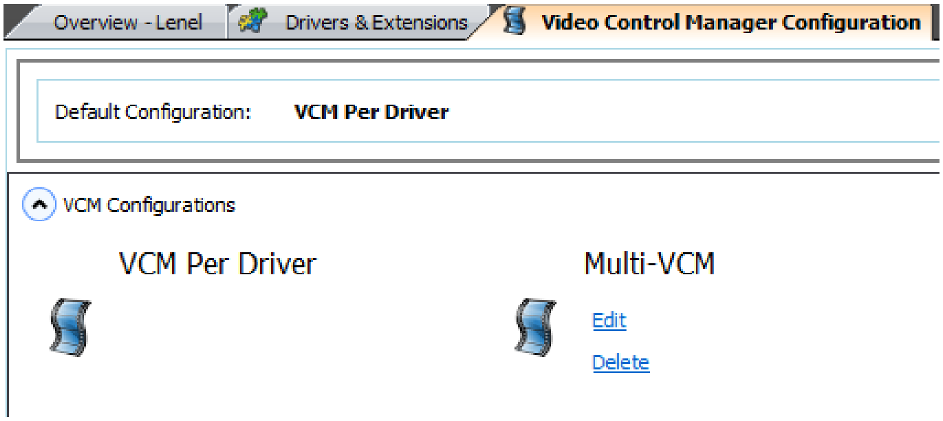


Default Configuration: VCM Per Driver

⚠ There are unsaved changes. Any changes will be applied when the client is restarted.

☒ Not all driver controls have been assigned a Video Control Manager.

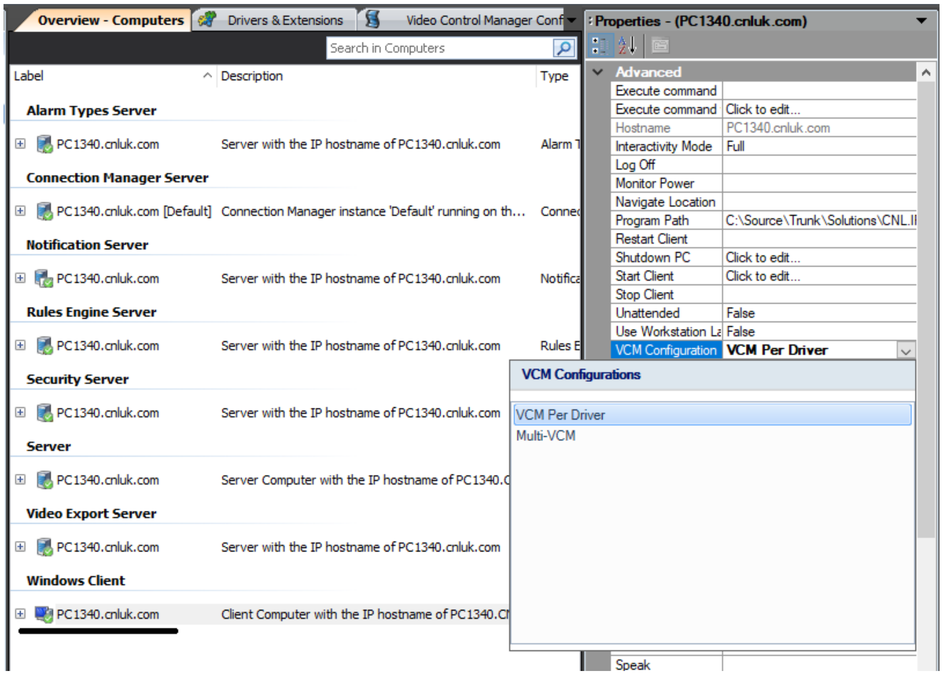
The new VCM Configuration now appears in the list:



Default Configuration: VCM Per Driver

5. In **System Configuration**, double-click the **Computers** folder.
6. Select a Control Center client instance to configure.

7. From the **VCM Configuration** drop-down list, select a VCM configuration .



Overview - Computers | Drivers & Extensions | Video Control Manager Conf

Search in Computers

Label	Description	Type
Alarm Types Server		
PC1340.cnluk.com	Server with the IP hostname of PC1340.cnluk.com	Alarm T
Connection Manager Server		
PC1340.cnluk.com [Default]	Connection Manager instance 'Default' running on th…	Connec
Notification Server		
PC1340.cnluk.com	Server with the IP hostname of PC1340.cnluk.com	Notifica
Rules Engine Server		
PC1340.cnluk.com	Server with the IP hostname of PC1340.cnluk.com	Rules E
Security Server		
PC1340.cnluk.com	Server with the IP hostname of PC1340.cnluk.com	
Server		
PC1340.cnluk.com	Server Computer with the IP hostname of PC1340.c	
Video Export Server		
PC1340.cnluk.com	Server with the IP hostname of PC1340.cnluk.com	
Windows Client		
PC1340.cnluk.com	Client Computer with the IP hostname of PC1340.Cl	

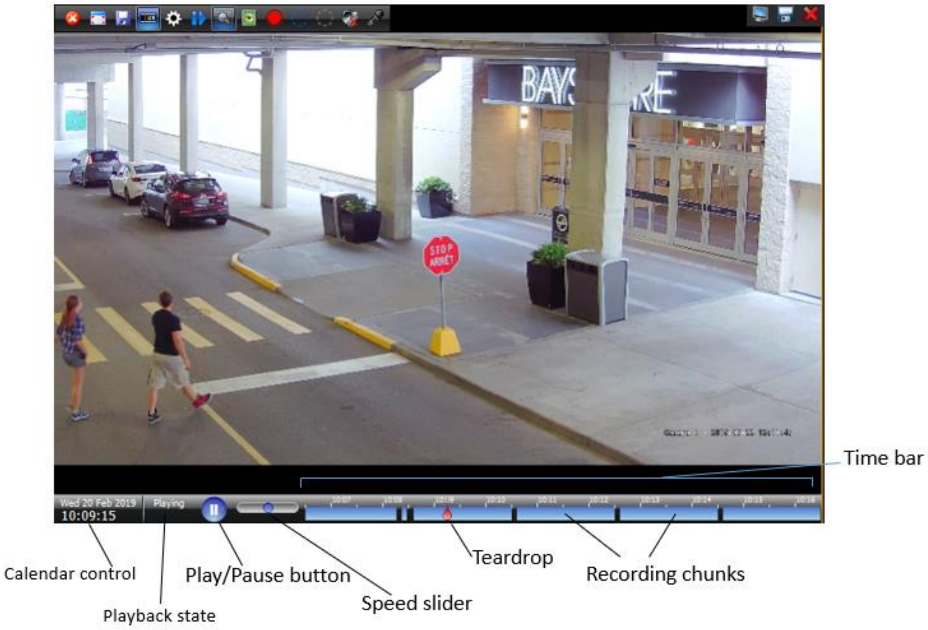
Properties - (PC1340.cnluk.com)

Advanced	
Execute command	
Execute command	Click to edit...
Hostname	PC1340.cnluk.com
Interactivity Mode	Full
Log Off	
Monitor Power	
Navigate Location	
Program Path	C:\Source\Trunk\Solutions\CNL.II
Restart Client	
Shutdown PC	Click to edit...
Start Client	Click to edit...
Stop Client	
Unattended	False
Use Workstation La	False
VCM Configuration	**VCM Per Driver**
Speak	

VCM Configurations

VCM Per Driver
Multi-VCM

Video Tile Control



Basic Features of a CCTV Connector

Server-side

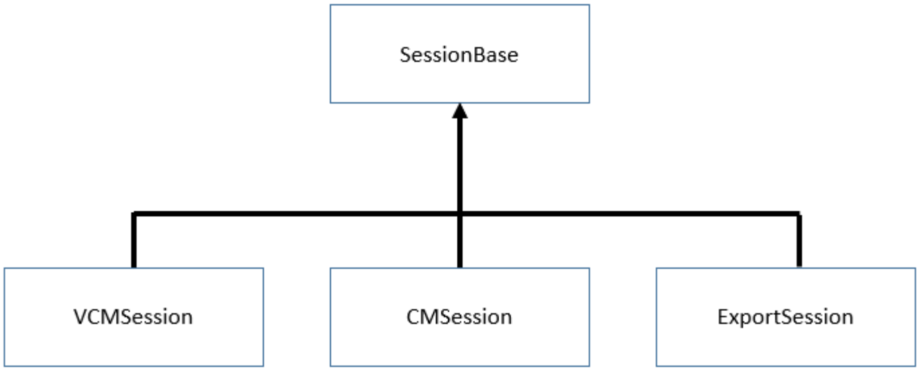
- Connection
- Device population (cameras, Inputs and Outputs for DVR driver, Recorders for VMS driver)
- Select Pre-set
- Alarm handling (Acknowledge, Close and so on.)
- Snapshot
- Switch outputs
- Events: motion detection, online states, alarms

Client-side

- Live Video
- Playback
  - Seek
  - Play, Pause
  - Playback loop
- Switch camera
- PTZ
- Pre-sets
- Snapshot
- Video Operator Actions
  - Digital Zoom
  - Focus
  - 360 De-warp
  - Audio In/Out
  - Video resolution selector
- Lifetime Manager

SDK Session Implementation

- **SessionBase** - common functionality: connect, disconnect, get devices
- **CMSession** - derived from SessionBase, implements server-side connector features
- **VCMSession** - derived from SessionBase, implements client-side connector features and a reference counter
- **ExportSession** - derived from SessionBase, implements video export



Connector Patterns

Over a period of development, Everbridge have created a range of recommended patterns to use in the development of third-party integrations.

Safe Timer

A Wrapper for the self-restarting timer safe from locking the timer thread when trying to dispose the timer during a timer tick.

Typical usage is for a storage timer to auto-populate the Playback Time Bar

```
private SafeTimer _storageTimer;

     ...

       _storageTimer = new SafeTimer(true, PlaybackTimerInterval, "Timebar
Timer");

     ...

        private void StartStorageTimer()

        {

            if (!_storageTimer.Enabled)

            {

              _storageTimer.Elapsed += StorageTimerTick;

                _storageTimer.Enabled = true;
            }
        }
        private void StopStorageTimer()
        {
            if (_storageTimer != null && _storageTimer.Enabled)
            {
                _storageTimer.Elapsed -= StorageTimerTick;
                _storageTimer.Enabled = false;
            }
        }
        private void DisposeStorageTimer()
        {
            StopStorageTimer();
            Task.Run() =>
            {
               _storageTimer?.Dispose();
               _storageTimer = null;
            }
        private void StorageTimerTick(object sender, EventArgs e)
        {
             //TODO required processing
        }
```

Assembly Redirection

Dynamically load the 3rd party SDK DLLs in runtime subscribing to `AppDomain.CurrentDomain.AssemblyResolve`. This is used in two cases:

- To automatically load the latest version of the SDK to make the driver compatible with multiple SDK versions and minimize the upgrade effort
- To prevent copying the DLLs locally to the `Bin\` folder of the Connection Manager or VCM

Used in drivers: March Networks, Genetec, Avigilon

The code using Assembly Redirection must be refactored so that the classes where the Redirection occurs does not reference any SDK types. These must be offloaded by using wrapper classes or Interfaces.

If the SDK is C++ based or a .NET wrapper around C++ libraries, the Assembly Redirection does not work. However, the driver can try to load the SDK Assemblies directly from the SDK install folder:

```
Assembly.LoadFrom(@"C:\GEVISOFT\GeViProcAPINET_4_0.dll");
```

Generic Pool

See drivers: Verint Nextiva, MxPro5

Generic Poller

This is useful in large scale systems where SDK does not provide users with connectivity monitoring and you want to implement a polling thread. On sites with hundreds of servers, it is not a good idea to run a thread per server as this leads to overload and thread starvation. Instead, use the global poller which uses one thread for all the servers/devices.

Usage example:

```
private void Connect()
       {
            var poller1 = GenericPoller<string>.Instance("Connectivity");
            var poller2 = GenericPoller<int>.Instance("Cameras");
            poller1.PollingInterval = 1000;
            poller2.PollingInterval = 300;
      //Initialize connectivity poller
            poller1.AddItem(new PollItem<string>("1", PollConnectivity,
500));
            poller1.AddItem(new PollItem<string>("2", PollConnectivity,
500));
      //Initialize cameras poller
            poller2.AddItem(new PollItem<int>(1, PollCameras, 500));
            poller2.AddItem(new PollItem<int>(2, PollCameras, 500));
      //stop polling Camera 1
            poller2.RemoveItem(2);
     //stop pollers
            poller2.Dispose();
            poller1.Dispose();
       }
       private void PollConnectivity()
       {
          ...
       }
       private void PollCameras()
       {
          ...
       }
```

Playback FSM

Some video playback systems have a complex set of steps to move between video playback modes based on previous state and potential failure mode. This Finite State Machine class allows for the definition of these steps and correct step based on previous known state. Example code is provided in Appendix A

Connection Monitors

Classes providing generic way of monitoring subsystem availability (by Ping, TCP, HTTP or SDK).

- NetworkMonitor - part of the ISDK, Reference assembly: `CNL.ControlCenter.Driver.Utility.dll`
- PingMonitor - part of the ISDK, Reference assembly: `CNL.ControlCenter.Driver.Utility.dll`

  NOTE: Ping is generally deprecated as a means of connection monitoring as it exposes an attack surface within the system. If used, it is recommended to have a property on the server to enable the functionality and to default it to disabled.

- TCP Monitor - Useful class to monitor a standard TCP connection (should be used if there is no SDK to provide you the connectivity).
- SDK-based Connection Monitor.

Network Socket Wrappers

`TcpClientWrapper` wraps the standard .NET `TcpClient` class. It connects and runs background thread continuously reading from the socket.

You can configure Encoding. Encoding has events Connected, Disconnected, DataReceived. Data is always received as byte[] and can be sent both as byte[] or string.

Example: Commend driver

Float Comparison

Comparisons of two float numbers can return invalid results, so it's better to compare this way:

```
private const float ThresholdMin = 0.00001F;
 public static bool Compare(float firstNumber, float secondNumber)
 {
    return Math.Abs(firstNumber - secondNumber) < ThresholdMin;
 }
 public static bool Compare(float firstNumber, int secondNumber)
 {
    return Math.Abs(firstNumber - secondNumber) < ThresholdMin;
 }
```

Example: IndigoVision driver, FloatExtensions class

Split Camel Case

If you need to provide a user-friendly description for an SDK enum type, you can use Regex to convert the enum:

```
internal static class GuardallShortenedEventCodes
 {
    internal enum GuardallShortenedEventCode : byte
    {
         Headcount = 20,
         CircuitAutocheckFail = 22,
         DefaultPinsClear = 55,
         ...
    }
    public static string GetTypeString(this GuardallShortenedEventCode 
eventCode)
    {
            switch (eventCode)
            {
                case GuardallShortenedEventCode.Headcount:
                    return "Number of activations of all circuits programmed 
with the head count option while the panel was unset";
                case GuardallShortenedEventCode.DefaultPinsClear:
                    return "Default PINs cleared";
                  ...
                default:
                    return SplitCamelCase(eventCode.ToString());
            }
    }
    public static string SplitCamelCase(string input)
    {
         return Regex.Replace(input, "([A-Z])", " $1", 
RegexOptions.Compiled).Trim();
    }
 }
```

Device Population

When developing a new connector, you must remember that although your development environment may have only a few devices to work against, your production environment may have many hundreds of devices/sensors. This can lead to the connect/population of devices taking many minutes and possibly causing Connection Manager to fail the device.

A suggested solution is to pass the population of devices onto a background task, leave it to complete and indicate the device as 'online'. This has some implications.

- If no error handling/checking is implemented in the background task, then device population can fail with no indication of the failure. In other words, not all devices are created/populated.

- No checking of the status of background task means:
    - if the device is taken 'off-line' the background device creation task continues.
    - changing the device state rapidly (such as, pressing F12 multiple times) can cause multiple background device creation tasks to be active. This can lead to duplication of devices in the system.

Device Patters Example Code

In the declarations for the device server, add the following:

```
[NonSerialized]
 private CancellationTokenSource tokenSource;
 [NonSerialized]
 private CancellationToken cancelToken;

[...]

/// <summary>
 /// Connects to the physical device.
 /// </summary>
 [SuppressMessage("Microsoft.Design",
"CA1031:DoNotCatchGeneralExceptionTypes")]
 public override void Connect()
 {
     try
     {
         CheckDisposed();
         var username = DeviceDefaults.DefaultUsername(this);
         var port = DeviceDefaults.DefaultPort(this);

[...]

log.InfoFormat(CultureInfo.CurrentCulture, ErrorMessages.ConnectingText,
username, IP, port);
         lock (lockInstance)
         {
             Disconnect();
             //
             // Should never get to this state
             // but just in case
             //
             if (tokenSource != null)
             {
                 tokenSource.Cancel();
                 tokenSource.Dispose();
             }
             tokenSource = new CancellationTokenSource();
             cancelToken = tokenSource.Token;
             if (string.IsNullOrEmpty(IP))
             {
                 throw new
ArgumentException(ErrorMessages.IPAddressNotSpecified);
             }

[...]
log.DebugFormat(CultureInfo.InvariantCulture, "Last Event Received: {0}",
LastEventReceived);
             if (RetrieveOfflineEvents &&
(!string.IsNullOrEmpty(LastEventReceived)))
             {
                 Task.Run(() => GetOfflineEvents(cancelToken), cancelToken);
             }
             Task.Run(() => PopulateDevices(cancelToken), cancelToken);

[...]

}
     }
     catch (DeviceException ex)
     {
         log.Error(ex.Message, ex);
         OnStateChanged(DeviceState.Failed, ex.FullMessage);
         Disconnect();
     }
     catch (Exception ex)
     {
         log.Error(ErrorMessages.DeviceConnectionFailed, ex);
         OnStateChanged(DeviceState.Failed,
ErrorMessages.DeviceConnectionFailed + Environment.NewLine + ex.Message);
         Disconnect();
     }
 }
 /// <summary>
 /// Disconnects from the physical device.
 /// </summary>
 public override void Disconnect()
 {
     PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(string.Empty));
     CheckDisposed();
     log.InfoFormat(CultureInfo.CurrentCulture, ErrorMessages.Disconnecting,
DeviceDefaults.DefaultUsername(this), IP, DeviceDefaults.DefaultPort(this));

[...]

lock (lockInstance)
     {
         //
         // Cancel any running background task
         //
         tokenSource?.Cancel();

[...]

}
     //
     // and destroy the token source/token from the system
     //
     tokenSource?.Dispose();
     tokenSource = null;
     log.InfoFormat(CultureInfo.CurrentCulture, ErrorMessages.Disconnected,
DeviceDefaults.DefaultUsername(this), IP, DeviceDefaults.DefaultPort(this));
 }
 /// <summary>
 /// Populates the devices connected to the server.
 /// </summary>
 private void PopulateDevices(CancellationToken token)
 {
     try
     {
         //
         // Was cancellation already requested?
         //
         if (token.IsCancellationRequested)
         {
             log.InfoFormat("Task {0} was cancelled before waiting for
network data.", MethodBase.GetCurrentMethod().Name);
             token.ThrowIfCancellationRequested();
         }
         //
         // if you split the population into additional methods remember to
hand the token through to those and check
         // at each stage for termination so as to terminate the task as
quickly as possible, otherwise
         //
         // Foreach device
         //     is cancelation requested?
         //       break out the task
         //     else
         //       add device

[...]

}
     catch(Exception ex)
     {
         //
         // report something here
         //
         [...]
     }
 }
```

Connector Testing

To test a connector, you must think about testing:

- a connector can be successfully installed and uninstalled.
- a connector can successfully connect to a subsystem.
- all the required connector features.
- all operator actions can be carried out successfully.

Connector Testing Prerequisites

Before beginning your testing, complete the following prerequisites.

1. Install Control Center client and server.
2. Check the requirements for the connector.
3. Check the video subsystem, for example, check hardware manuals.

4. Set up the subsystem to test all the required features.
    - The subsystem is configured to raise all the events supported in the connectors.
    - For video connectors:
        - it has at least one PTZ camera with pre-configured presets.
        - it has at least 3 cameras configured:
            - one camera recording continuously.
            - one camera recording on motion or another event.
            - one camera that does not have any recordings.
5. Study the RDIN. Check the following sections: Installation, and Known Issues & Limitations.
6. Install all the connector prerequisites as described in the RDIN.
7. Install the SDK on relevant machines as described in the RDIN.
8. Install the connector in Control Center.

Connections and Online States

To make a connector connect to a subsystem:

1. Add a server device representing a subsystem server/service/panel.
2. Set the connection properties.
3. Enable the device.

If the connection is successful, the device goes to a **Connecting** state, and then to an **Online** state.

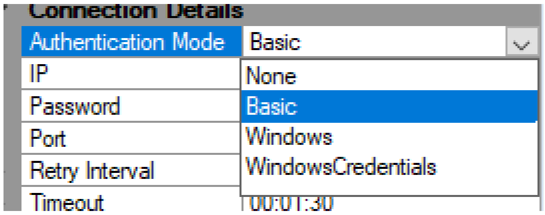
if the connector cannot connect, the device goes to a **Failed** state and the **State Description** explains the reason for the failure.

Connection properties may be slightly different from connector to connector, but a typical set of properties include:

- **IP** - IP address or host name of the subsystem.
- **Port** - (for TCP-based protocol/SDK). This can be set to a valid TCP port. It can also be set to 0. In this case, the connector should automatically use a default value (such as 80 or 443 or some system-specific port).
- **Username**
- **Password**
- **Timeout** - (1 minute by default): This is the period of time the Connection Manager is waiting for a device to go to an **Online** state after it's **Enabled**. If the device does not go **Online** in time, Connection Manager sets the device to a **Failed** state.
- **Retry Interval** - (1 minute by default): after the device is in **Failed** state, Connection Manager schedules automatic re-connection after the amount of time set in **Retry Interval**.

Windows Credentials/Single Sign On

Some subsystems have an option to use Windows credentials to connect. If several connection options are available, the driver should have a property called **Authentication Mode** (or similar) so a user can select an authentication type from the list.



Connection Details

Authentication Mode	Basic
IP	
Password	
Port	
Retry Interval	
Timeout	00:01:30

If Windows credentials are used, the **Username** property is typically in the format *Domain/ user* or *user@domain*.

Lifetime Manager

The Lifetime Manager is used in some CCTV drivers to allow faster camera display. It should only affect the first camera displayed after the Control Center client starts up.

To make the feature work, the main server device has a property, typically, a boolean property called **Auto Connect**, controlling whether the Control Center clienttries to connect to this server on start up.

Test Scenario	Expected Behaviour	Comments
• Control Center client starts up. • Subsystem server is available. • **Auto Connect** property is set to **True** on the server device.	The VCM automatically connects to the subsystem server.	This can be confirmed from VCM logs.
• Continued: drag a camera device to Display Area.	The camera is displayed within 1-2 seconds.	
• Control Center client starts up. • **Auto Connect** property is set to **False** on the server device.	The VCM does **not** connect to the subsystem server on start up.	This can be confirmed from VCM logs.
• Continued: drag a camera device to Display Area.	The camera is displayed within 10-20 seconds (depends on the subsystem and the network speed).	

• Control Center client starts up • The server device is **Disabled** in Control Center.	The VCM does **not** connect to the subsystem server on start up.	This can be confirmed from VCM logs.
• Continued: drag a camera device to Display Area.	The camera is displayed within 10-20 seconds (depends on the subsystem and the network speed).	
• Control Center client starts up. • Server device is **Disabled** in Control Center. • Enable the server device.	The VCM does not connect to the subsystem server on start up. It also does not try to connect when the server device is **Enabled**.	This can be confirmed from VCM logs.
• Control Center Client starts up. • Subsystem server is unavailable or connection details are invalid in the server device. • **Auto Connect** property is set to **True** on the server device.	The VCM automatically attempts to connect to the subsystem server. An option is to allow Lifetime Manager to implement a retry mechanism, similar to Connection Manager re-try mechanism.	There should be a limited number of re-connect attempts controlled by a server device property **Maximum Reconnects**.
• Control Center client starts up, subsystem server is available. • **Auto Connect** property is set to **True** on the server device. • Wait until the VCM has connected to the subsystem. • Disconnect the subsystem from network, then re-connect the network. • Wait until the server	The VCM should automatically re-connect to the subsystem when it becomes available. When the subsystem is back online, the camera camera should be displayed within 1-2 seconds.	

device returns to **Online** state. - Drag a camera device to Display Area.		
Start or re-start Control Center client, **Auto Connect** property is set to **True** on the server device, Display the camera device after the client has started up as soon as possible (simulate the situation where Lifetime Manager is still connecting to the subsystem, where the camera is being displayed).	The video tile will stay in **Connecting** state until VCM has finished logging in to the subsystem and the video is displayed successfully.	This scenario occurred in CBK with DvTel Latitude driver.
All of the above behavior should be tested with multiple VCMs configured.		

Device Population

For most connectors which support child devices, a server device is added manually. Once the device is **Enabled**, it connects to a subsystem and goes to **Online** state.

There are two basic options:

1. A connector queries the subsystem for available devices, then the child devices are populated automatically in Control Center.
2. If there is no way to query the subsystem, the device configuration is supplied manually by setting a server device property as a path to a configuration CSV file or a custom window, where configuration can be set up manually.

Typical Scenarios for Device Population

Test Scenario	Expected Behavior	Comments
First-time connection, create a new connectable parent (server) device and **Enable** it.	After connecting to the subsystem, the devices goes to **Online** state. The device then then goes to **Populating Devices** custom state, while populating the child devices. Finally, the device goes to the **Online** state, once all the child devices are populated.	

After its child devices are already populated and in **Disabled** state, enable parent (server) device.	Same as above.	
Enable parent (server) device after its child devices are already populated and are **Enabled** in Control Center.	Same as above. If some of the child devices can populate devices as well (for example Panels which in turn can populate I/Os and readers) these will be also populated if can successfully get a list of devices - the new devices	
After connected to the subsystem, delete a child device in Control Center, then re-enable the parent device.	The deleted child device should be re-populated in **Disabled** state.	
Run the **Sync Devices** (in other words, **Update Devices**) parent device method after successfully connected to the server.	The connector should query the subsystem and re-sync. The child devices should: • Populate any missing Control Center devices in **Disabled** state, • set any devices that were deleted in the subsystem to **Failed** state with description similar to **Device not found**.	If currently not connected to the server, the method must log an error and return **False**.
Run the **Sync Devices** (in other words, **Update Devices**) parent device method multiple times, quickly.	The connector must not allow more than one device population at a time. If the connector is already populating devices the repeated method call should result in a warning log message and return **False**. The original device population must complete uninterrupted.	
Quickly, disable and enable parent device multiple times.	As above but allow only one device population at a time.	
Test cancelling device population by enabling a parent device, waiting for the connector to start device population, and then disable the device	Device population must stop almost immediately and the parent server device goes to **Disabled** state.	

while still populating the devices.		

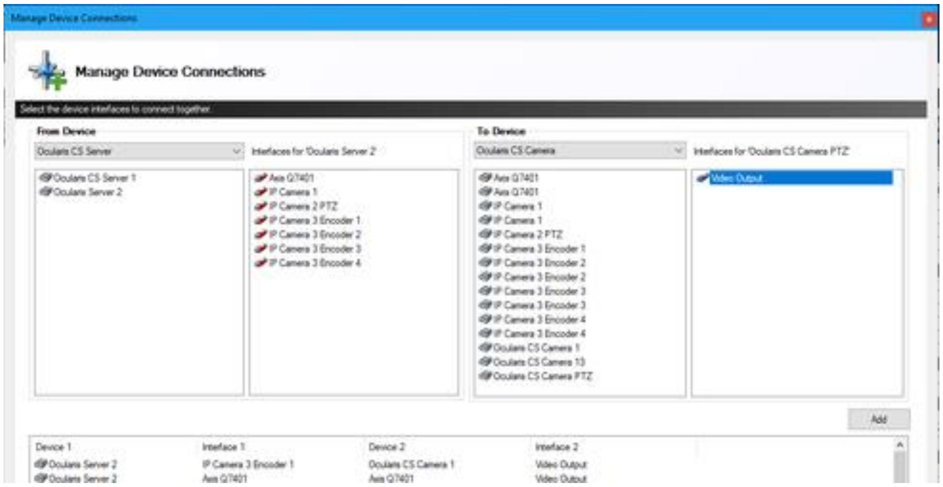
Notes:

- All new devices are populated in **Disabled** state.
- For all video camera devices, the connector should automatically detect whether a camera supports PTZ and set **PTZ Supported** and **Presets Supported** properties to the correct value.

Re-adding Child Devices Manually After Deletion

You should test manually reading child devices after deletion from a parent device.

1. Right-click server device and select **Manage Interfaces** to start **Device Connections** wizard.
2. Select the parent device, and select the interface of the previously deleted camera.
3. Select **Add** to connect the server device to the new camera device by linking the two selected interfaces.
4. Select **Finish** to close the wizard.



Device 1	Interface 1	Device 2	Interface 2
Ocularis Server 2	IP Camera 3 Encoder 1	Ocularis CS Camera 1	Video Output
Ocularis Server 2	Axis Q7401	Axis Q7401	Video Output

Device Properties

To view the properties of a connector, in **System Configuration**, select a device. The **Driver Properties** pane displays. Properties control the behavior of a device. Properties on a server device can affect the behavior of the whole driver.

Device Methods

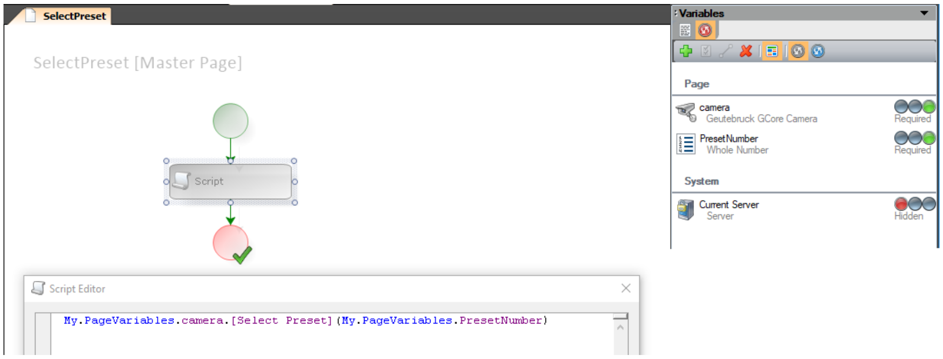
Methods are commands/actions available for a device. Device methods have to be asynchronous. Any returned data has to be raised in a separate event. Device methods return a unique ID. This is included in any event raised as a result of the method being invoked.

Methods can have parameters of various types and return a value which is typically a boolean variable. In other words, **True** if the command was successful, **False** otherwise.

If a command such as **Open Door** returns **True** it means the command was successfully sent to the subsystem. It does not mean the command was executed, in other words, the door is open. If the method returns **False**, it means the command could not be sent to the subsystem.

Invoke Device Methods

1. In the **Properties** pane in the **Actions** category, select the method. If the method has more than one parameters, a dialog displays and you must supply the parameter values.
2. Run a Response Plan with a Script shape. To check the return value, create a variable in a visual response plan (VRP) that represents the value, and assign the method result to the variable. For example, call **Select Preset** method on a camera device passing in the Preset Number. For this VRP, you need to create the variables **camera** and **PresetNumber**.



Page

camera
Geutebruck GCore Camera
Required

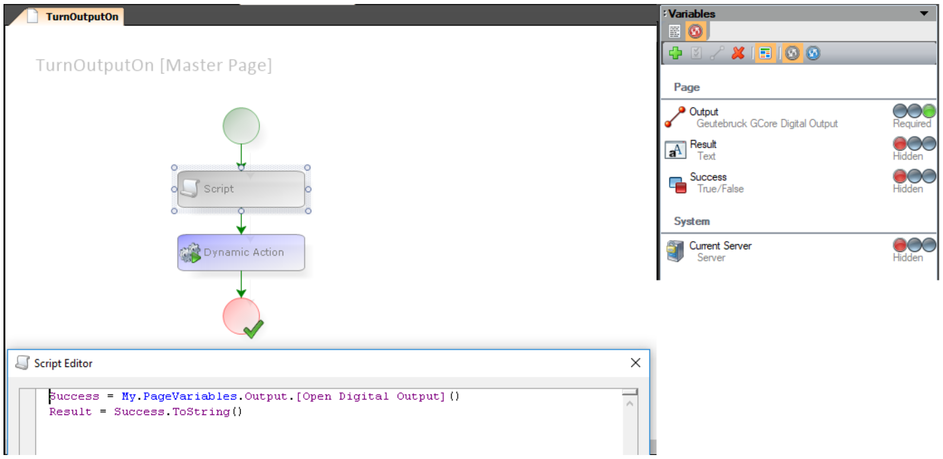
PresetNumber
Whole Number
Required

System

Current Server
Server
Hidden

```
My.PageVariables.camera.[Select Preset](My.PageVariables.PresetNumber)
```

In this example, call **Open Digital Output** method on an output device and print the return value.



Page

Output
Geutebruck GCore Digital Output
Required

Result
Text
Hidden

Success
True/False
Hidden

System

Current Server
Server
Hidden

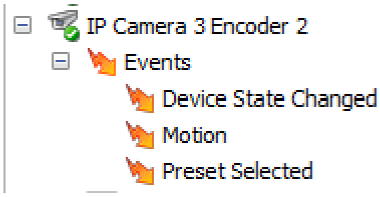
```
Success = My.PageVariables.Output.[Open Digital Output]()
Result = Success.ToString()
```

Device Events

Every device can raise events. Events are first reported by the Connection Manager service and then sent to the Rules Engine service to trigger response plans, if required.

Every device has one standard event, **Device State Changed**.

To see the available device events, select a device in **System Configuration**, and expand the **Events**.



IP Camera 3 Encoder 2
Events
Device State Changed
Motion
Preset Selected

Creating a VRP Triggered on Device Event

To test a device event you can create a VRP that is triggered when an event is raised.

1. Right-click on event and select **React to Event** > **Run response plan** > **Create New response plan**.
2. Navigate to **Services** and drag the Connection Manager service while testing the events and the response plan.
3. Verify the events are raised by the driver by going to **System Configuration** > **Services** folder. Double-click Connection Manager service to open the Event Viewer.
4. Make sure the VRP created for the event is triggered.
5. (Optional) Go to **System Configuration** > **Computers** folder. Double-click **Rules Engine Server** to open the Event Viewer.

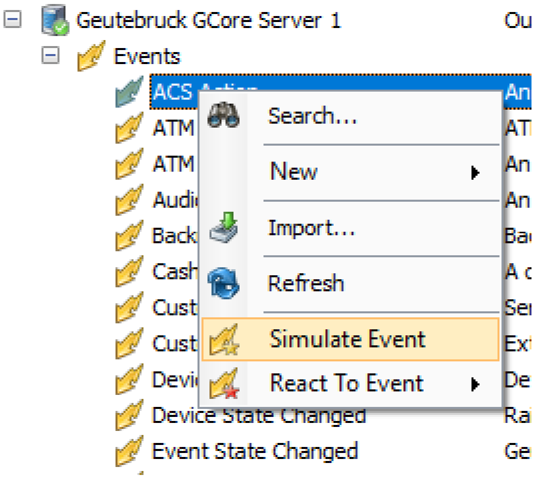
Event Properties

Event properties support basic types; DateTime, Integer, Double, String, custom Enum, Boolean.

Simulating Events

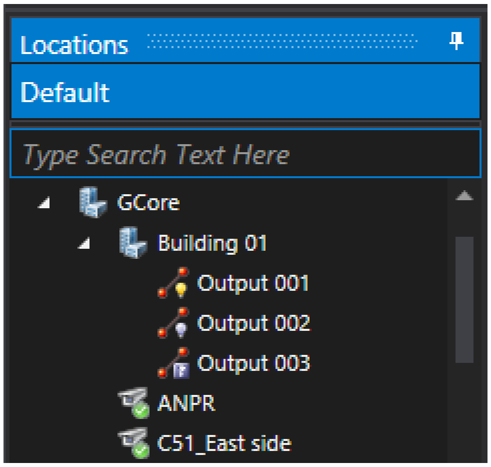
Control Center allows you to simulate connector events raised without any subsystem activity.

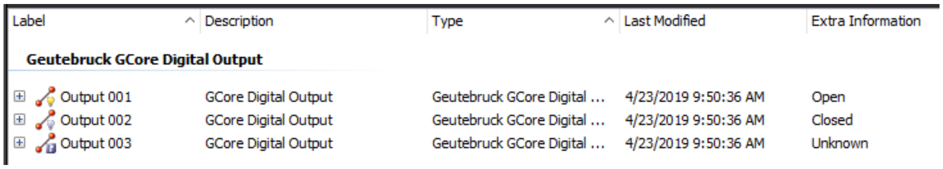
1. Go to **System Configuration**.
2. Right-click on an event and select **Simulate Event**.
3. If the event has custom properties, fill in the property values and select **OK**.



Device Custom States

In addition to standard online states (Online, Disabled, Offline, Failed, Warning), a device can implement one or more additional states called custom states . Typically, a custom state displays a description in addition to the icon.





Geutebruck GCore Digital Output

Label	Description	Type	Last Modified	Extra Information
Output 001	GCore Digital Output	Geutebruck GCore Digital ...	4/23/2019 9:50:36 AM	Open
Output 002	GCore Digital Output	Geutebruck GCore Digital ...	4/23/2019 9:50:36 AM	Closed
Output 003	GCore Digital Output	Geutebruck GCore Digital ...	4/23/2019 9:50:36 AM	Unknown

More examples:

- Omitted state
- Abnormal state
- Unset state

Typical Usage of Custom States

You can represent a current state of a relay output or a logical output device. Typically, the available states are **On** and **Off**. Where the current state cannot be polled by a connector, the device stays in an **Unknown** custom state until the first state update from the subsystem.

There are some states that it might be important for a customer to see, track and control. These are frequently implemented as custom states.

- Show current device faults. If there are multiple faults, the state description lists them all, for example, Door fault, Battery low and so on. If this is a compound device, for example, an ACS logical door which can have multiple readers, inputs and a door lock, the state description lists this. For example, **Reader IN: disabled, Lock fault**.
- Populating custom state can be used to show a user that the connector is busy populating child devices. This is applicable for connectors with child devices released for large scale projects where device population can take significant time.

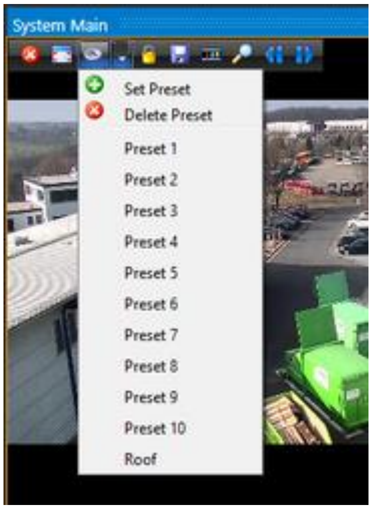
- States of devices representing logical subsystem entities like Zones, Areas or Groups that can be locked, unlocked, set/armed, unset/disarmed, disabled/omitted/inhibited.

Live Video

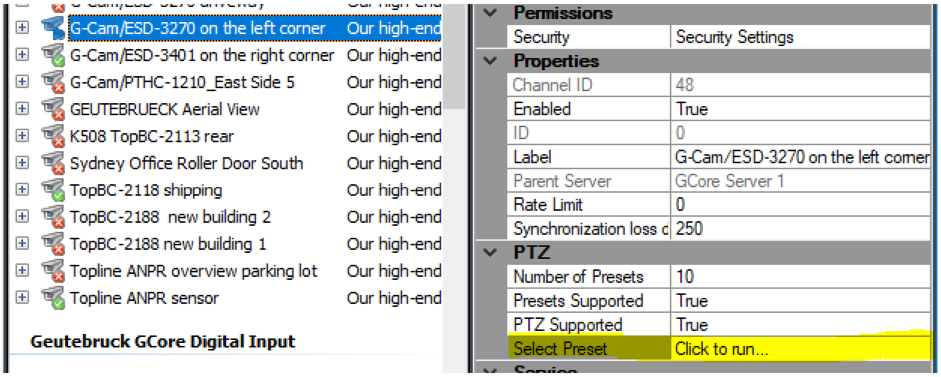
When you select a video, the video should be displayed, in a tile layout, for example, single tile, multiple tile, full screen or minimized, or in a sequence, using a short cut, or optical zoom and digital zoom. You also need to check that when you unplug the device, if it goes to offline state and if the video still plays. Finally, if you are able to play the video using a VRP.

Presets

The **Preset Selector** menu is available from the **Tile** menu.



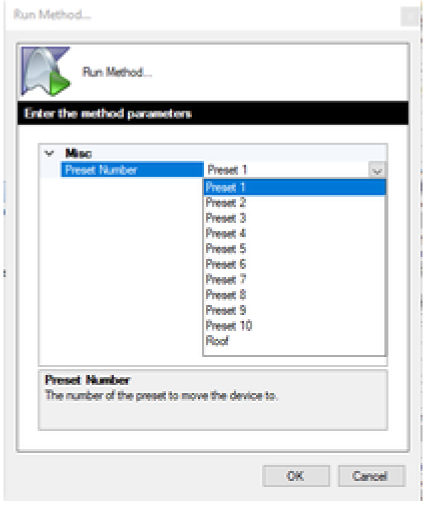
The same menu is available from **System Configuration** and selecting a camera device. You can use the **Select Preset** method from the **Properties** pane.



G-Cam/ESD-3270 on the left corner	Our high-end
G-Cam/ESD-3401 on the right corner	Our high-end
G-Cam/PTHC-1210_East Side 5	Our high-end
GEUTEBRUECK Aerial View	Our high-end
K508 TopBC-2113 rear	Our high-end
Sydney Office Roller Door South	Our high-end
TopBC-2118 shipping	Our high-end
TopBC-2188 new building 2	Our high-end
TopBC-2188 new building 1	Our high-end
Topline ANPR overview parking lot	Our high-end
Topline ANPR sensor	Our high-end

Geutebruck GCore Digital Input

Permissions	
Security	Security Settings
Properties	
Channel ID	48
Enabled	True
ID	0
Label	G-Cam/ESD-3270 on the left corner
Parent Server	GCore Server 1
Rate Limit	0
Synchronization loss c	250
PTZ	
Number of Presets	10
Presets Supported	True
PTZ Supported	True
Select Preset	Click to run...



Run Method...

Enter the method parameters

Misc

Preset Number: Preset 1

Preset Number
The number of the preset to move the device to.

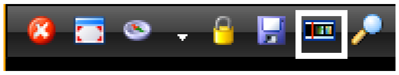
Note: Control Center ISDK and VCM have the following known limitations:

- It only allows to set Presets supported = True or False. In other words, if Presets are supported, Control Center assumes that presets can be Set (created/saved), Renamed, and Deleted. If the actual subsystem does not support it, the **Delete Preset** dialog still appears, but when you select **OK** to delete, nothing happens. Although, the driver might emulate the preset deletion overriding the SDK behavior.
- Even though the ISDK provides 2 independent boolean properties, Preset Supported and PTZ Supported, Preset controls only appear on a video tile if both properties are set to **True**. This makes the Preset Supported property obsolete.
- The Preset GUI is inconsistent. The **Preset Delete** dialog shows only **Preset Number**, whereas the drop-down menu shows only **Preset Labels**. **Set Preset** dialog shows both **Preset Number** and **Preset Label**.

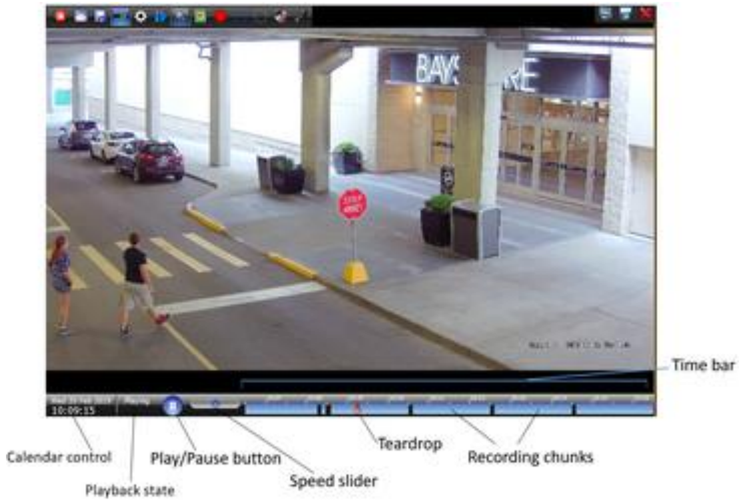
Playback

You can playback the video from a camera in Control Center. Drag a camera device into a Display Area. A Tile Layout is automatically generated to host the video. The camera is displayed in Live Video mode.

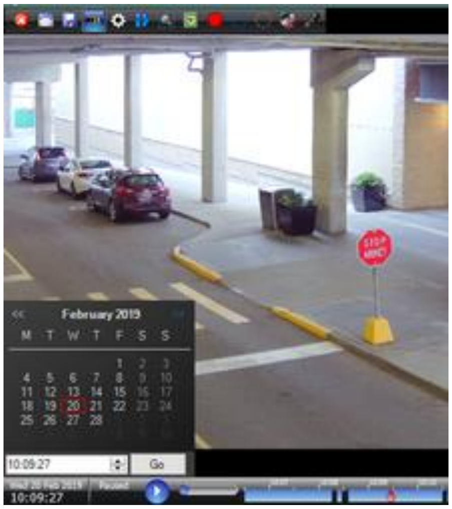
Select **Playback** on the tile menu to switch to Playback mode.



The tile menu has the following controls.



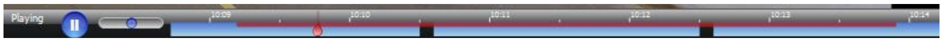
You can use the calendar control to seek playback for a particular date and time. Select a date and time and select **Go**.



Playback Loop

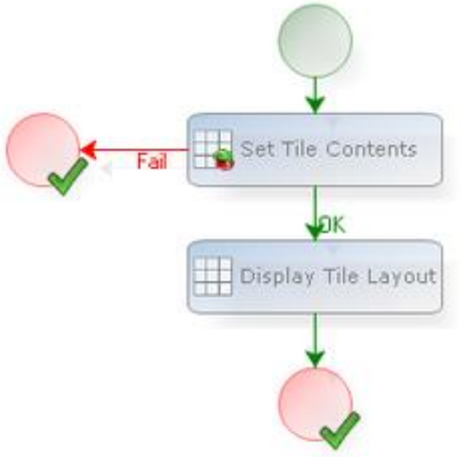
Everbridge recommends that you validate playback loops in time zones other than UTC+-0. Times shall always be described in UTC in response plans and always in LocalTime in the UI.

- **Start playback Loop** Using the mouse, mark a region on the timebar where you want to define a loop. When you release the mouse button, a context menu displays. Select **Start Video Loop**. The loop is marked in the timebar.



- **Stop playback Loop** Right-click the Loop marker in the timebar. In the context menu, select **Stop Video Loop.**

- **Display a camera in Playback mode from a VRP** Create or import a VRP that displays a camera in Playback mode.

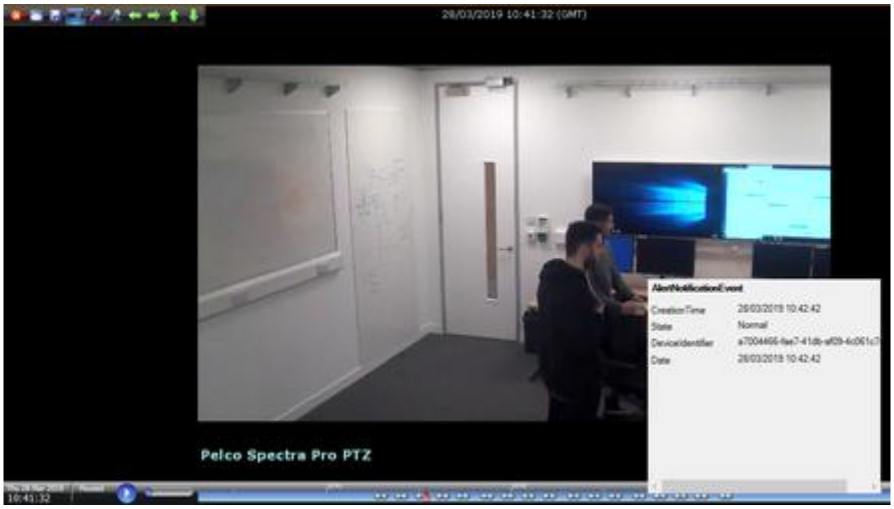


You can modify a VRP to enable you to test playback loops. Double-click a VRP to edit it in the VRP Editor.

  - To display the camera in **Playing** mode, select **Set Tile Contents** shape and set the **Paused** property to **False**.
  - To display the camera in **Paused** mode, select **Set Tile Contents** shape and set the **Paused** property to **True**.

Timebar Events

Video connectors can optionally make camera device events to appear on a timebar.



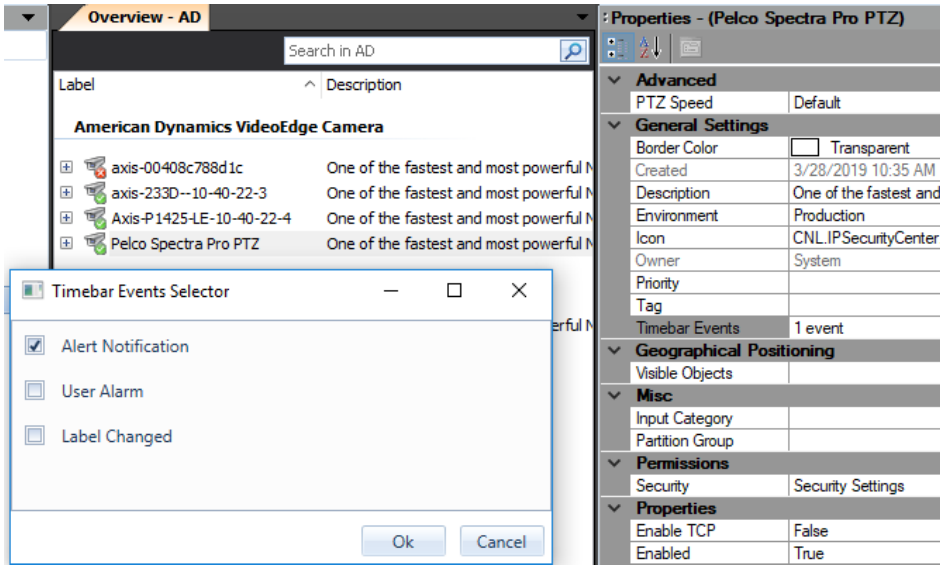
AlertNotificationEvent

CreationTime	28/03/2019 10:42:42
State	Normal
DeviceIdentifier	a7004466-fae7-41db-af09-4c061c7
Date	28/03/2019 10:42:42

The timebar events can be configured to:

- be logged to timebar.
- provide a user with an option to select or deselect an event type to appear on a timebar.

If an events appearance on a timebar is optional, these events can be configured by clicking the **Timebar Events** property on a camera device in the **Properties** pane in **System Configuration**.



American Dynamics VideoEdge Camera

Label	Description
axis-00408c788d1c	One of the fastest and most powerful N
axis-233D--10-40-22-3	One of the fastest and most powerful N
Axis-P1425-LE-10-40-22-4	One of the fastest and most powerful N
Pelco Spectra Pro PTZ	One of the fastest and most powerful N

Timebar Events Selector

☑ Alert Notification

☐ User Alarm

☐ Label Changed

Properties - (Pelco Spectra Pro PTZ)

Advanced	
PTZ Speed	Default
General Settings	
Border Color	☐ Transparent
Created	3/28/2019 10:35 AM
Description	One of the fastest and
Environment	Production
Icon	CNL.IPSecurityCenter
Owner	System
Priority	
Tag	
Timebar Events	1 event
Geographical Positioning	
Visible Objects	
Misc	
Input Category	
Partition Group	
Permissions	
Security	Security Settings
Properties	
Enable TCP	False
Enabled	True

Summary of VRPs for Testing Playback

Test	VRP Name	Details
Show camera in Playback mode (playing).	Display Camera Feed Playback.xml	
Show camera in Playback mode (paused).	Display Camera Feed Playback Paused.xml	
Show camera in Playback Loop mode.	Display Camera Feed Playback Loop.xml	
Show multiple cameras in Playback.	Display Multiple Cameras in Playback Mode.xml	Create a folder in Control Center and copy camera devices to the folder. The Folder is set via the **DeviceFolder** variable. All the cameras will try to play from the same time, set by **StartDate** Variable.
Check memory leaks while re-displaying Tile Layout.	TileLayoutReload3.xml	You need to find or create a Tile Layout (2x2 or bigger) to be assigned to *tilelay* VRP variable.

Video Operator Actions

You can test the operator actions that can be performed on video. Optional video features are available as additional controls on the video tile menu.

Digital Zoom

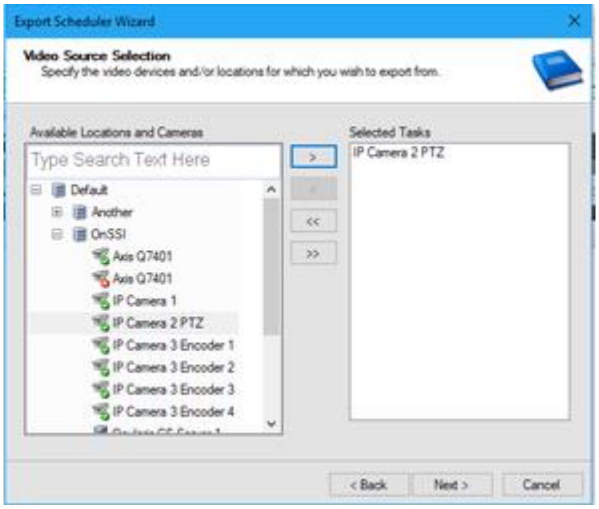
To test digital zoom:

- Digital Zoom button must appear in the video tile
- To switch Digital Zoom on, press the Digital Zoom button. A Digital Zoom icon appears in the lower left corner of the tile and the mouse cursor changes to a cross (+).
- To switch Digital Zoom off, press the Digital Zoom button. The Digital Zoom icon disappears and the mouse cursor changes back to a default cursor.
- Digital Zoom can be supported for both Live and Playback modes:
    - When Digital Zoom is on in Live mode and a user switches the tile to Playback, the Digital Zoom is automatically disabled.
    - When Digital Zoom is on in Playback mode and a user switches the tile to Playback, the Digital Zoom is automatically disabled.

Video Export

To schedule a new video export Job:

1. Start Video Export Wizard.
2. In Control Center client, go to **System > Video Export Scheduler**
3. Select **New > Next** and assign a name for your video export job.
4. Select **Next** > **Next** and choose a camera device to export from.
5. Select > to move the selected camera to the right pane.
6. Select **Next**.



Export Scheduler Wizard

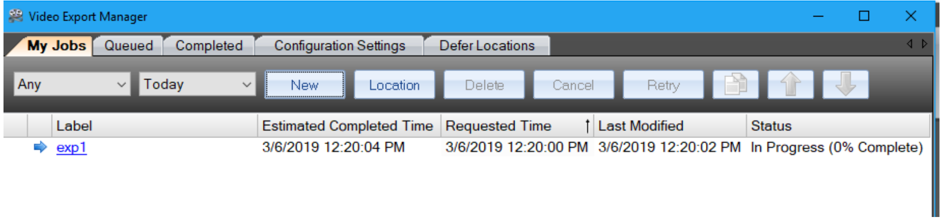
Video Source Selection

Specify the video devices and/or locations for which you wish to export from.

Selected Tasks: IP Camera 2 PTZ

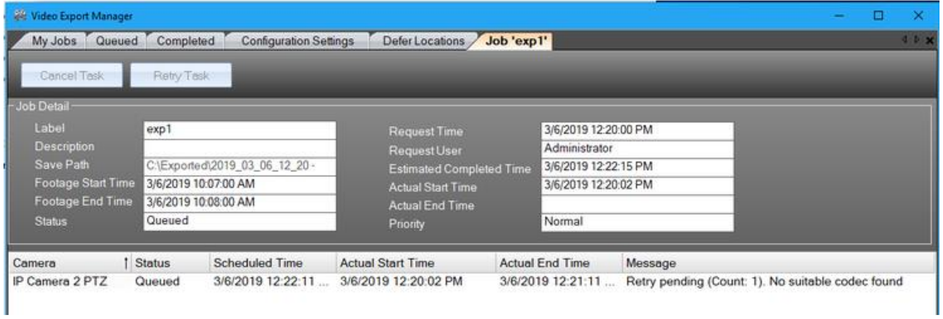
7. Select a start time and end time in local time (not in UTC).
8. Select **Path** to pick the folder to save the exported files to. The folder on Video Export Service machine is used/created.
9. Select **Next** in **Summary** page.

10. Select **Submit** to close the Wizard. A new Job appears in the Export Manager.



Label	Estimated Completed Time	Requested Time	Last Modified	Status
exp1	3/6/2019 12:20:04 PM	3/6/2019 12:20:00 PM	3/6/2019 12:20:02 PM	In Progress (0% Complete)

11. Double-click on the task to show the export task progress and information.
12. If an export task fails, an error message appears in the **Message** column.



Job Detail

Label	exp1	Request Time	3/6/2019 12:20:00 PM
Description		Request User	Administrator
Save Path	C:\Exported\2019_03_06_12_20 -	Estimated Completed Time	3/6/2019 12:22:15 PM
Footage Start Time	3/6/2019 10:07:00 AM	Actual Start Time	3/6/2019 12:20:02 PM
Footage End Time	3/6/2019 10:08:00 AM	Actual End Time	
Status	Queued	Priority	Normal

Camera	Status	Scheduled Time	Actual Start Time	Actual End Time	Message
IP Camera 2 PTZ	Queued	3/6/2019 12:22:11 ...	3/6/2019 12:20:02 PM	3/6/2019 12:21:11 ...	Retry pending (Count: 1). No suitable codec found

13. After an export task fails after 3 retries, the parent export job fails as well. You can r etry or cancel export tasks.

There are some known issues with video exports.

- Progress of export task is not shown when connectors report it (raising OnProgress() event).
- When a task is cancelled, <NULL> is displayed in the Wizard.
- The export file path is created on the Control Center client machine instead of the Video Export Service machine.

You can configure a Video Export Service to run on a separate Machine, rather than on a Control Center server machine.

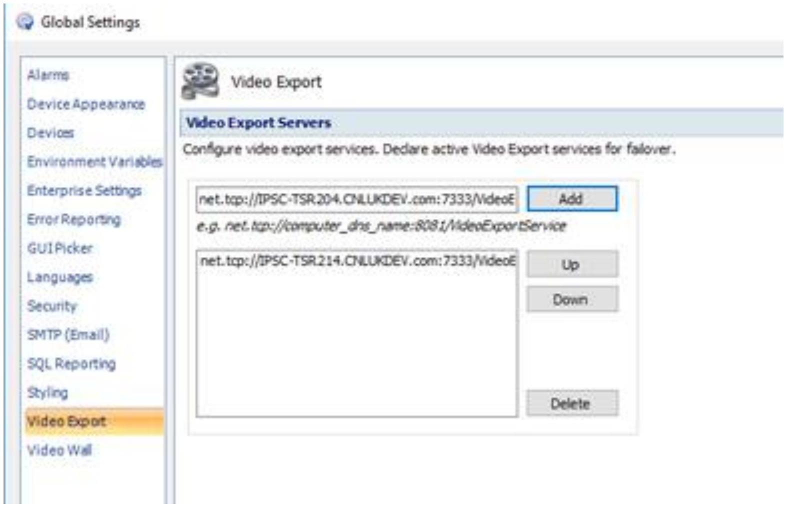
For this test you need 3 separate machines:

- Control Center client
- Control Center server
- Control Center VES

Do the following:

1. Run Control Center server installer.
2. Select **Custom installation**
3. Select only **Video Export service** to install:
   - enter the same service credentials used for other services on Control Center server machine.
   - enter the same SQL Database Instance as the other services on Control Center server machine.
4. On the VES machine, stop the VES if running.

5. Add the following: `<add key="CoreServerHostname" value="Control Center server machine host name"/>` to the `C:\Program Files (x86)\Everbridge\IPSecurityCenter\IPSecurityCenter Video Export Service\CNL.IPSecurityCenter.VideoExport.WindowsService.exe .config`.
6. In Control Center client, navigate to **System Configuration** > **Global Settings** > **Video Export**.
7. Add a connection to the new VES, providing a web service URL: `net.tcp://` *`VES machine host name`*`:7333/VideoExportService`



Configure video export services. Declare active Video Export services for failover.

net.tcp://IPSC-TSR204.CNLUKDEV.com:7333/VideoE

e.g. net.tcp://computer_dns_name:8081/VideoExportService

net.tcp://IPSC-TSR214.CNLUKDEV.com:7333/VideoE

8. Follow the driver RDIN Installation section to install the subsystem SDK on the VES machine, if needed.
9. Copy `\\` *`Control Center Server hostname`*`\c$\ProgramData\Everbridge\ControlCenter\Packages` to `C:\ProgramData\Everbridge\ControlCenter`. This is so the driver can load on the VES machine.
10. Start the Video Export service on VES machine.

Test SDK Sessions/Connections Release

On the server side, when Connection Manager is stopped and/or server device is **Disabled**, the connector is expected to:

- close all TCP connections
- close any SDK connections
- successfully log out from the subsystem releasing any 3rd party licenses if used.

On client side (video drivers), when all the video feeds are closed in Control Center client, the connector is expected to

- close all TCP connections
- close any SDK connections
- successfully log out from the subsystem releasing any 3rd party licenses if used.

This can be tested in 2 ways:

1. Run **netstat -a -b** command on native server Command Prompt to show all TCP connections.
2. Check there are no established TCP/UDP connections with Connection Manager machine. Tip: if there are too many connections, you can use filtering: *netstat -a -b | findstr "10.*"*
3. Use native software to show existing or recent SDK sessions, license counter or logs showing Control Center logging in/out.

Memory Leaks Detection

You can use the following software tools for memory tracking.

- **PerfMon** Windows tool. Set **Process** > **Private Bytes** counter for the tracked process.
- **ANTS Memory Profiler** Attach to the requested process. Take one Snapshot and then take several more snapshots during the test. Compare the memory consumption, available here: `\\fileserver\software-library\Internal Software\RedGate\DotNetDeveloperBundle`
- **Loupe Desktop** After the tested period, crash/stop the process. In Loupe, go to **Local Sessions** > **Control Center**, and double-click the recently finished session. Check the memory graph.
- **Process Explorer**
- **Task Manager** Visually check the RAM level (Memory (private working set) column). This is the least preferred tool as there is no way to record or display the memory consumption during the time period. You should only use this for short running tests.

Uninstall Connectors

For some test scenarios, such as connector version upgrade, it may be necessary to remove the current version of the connector from Control Center.

1. In **System Configuration**, delete all devices of the connector. If there are any dependencies preventing the devices being deleted (such as VRPs, Tile Layouts, Sequences using the devices), remove the dependencies and try deleting the devices again.
2. Close all instances of Control Center client.
3. On every machine that has Control Center client installed, delete the connector from the following folders:
    - `C:\ProgramData\Everbridge\IPSecurityCenter\Packages`
    - `C:\ProgramData\Everbridge\IPSecurityCenter\Extracted Packages`
    - `C:\ProgramData\Everbridge\IPSecurityCenter\Windows Client\Packages`
    - `C:\ProgramData\Everbridge\IPSecurityCenter\Windows Client\Extracted Package`

4. On Control Center server, stop the following services:
    - Control Center Connection Manager Service hosting the driver devices
    - Control Center Server
5. On Control Center server machine, delete the connector from folders:
    - `C:\ProgramData\Everbridge\IPSecurityCenter\Packages`
    - `C:\ProgramData\Everbridge\IPSecurityCenter\Extracted Packages`
    - `C:\ProgramData\Everbridge\IPSecurityCenter\Connection Manager\` *`CM name`*`\Packages`
    - `C:\ProgramData\Everbridge\IPSecurityCenter\Connection Manager\` *`CM name`*`\Extracted`
6. On Control Center server machine, re-start the following services:
    - Control Center Connection Manager Service hosting the driver devices
    - Control Center Server
7. Restart the Control Center client. The connector package should not appear in **System Configuration** > **Drivers & Extensions**.

All Connectors - Expected Functionality

The expected functionality of drivers may depend on the particular subsystem. The following table describes some standard test scenarios.

Test Scenario	Comment	Behavior 1	Behavior 2	Behavior 3
Disconnection while the connector is populating child devices **(only relevant for connectors with child devices)**	If the connector populates device synchronously (as implemented in most connectors), the connection may time out. If the device population is asynchronous, there may be synchronization problems, if user decides to re-enable server device or connection is broken while devices are being populated.	(March Network Command driver): after logging in to subsystem the connector goes Online briefly to stop CM timer and prevent connection timeout, then goes to Populating Devices Custom state to show the user the driver has not finished initializing.	Most connectors: device populates all devices and only then does the server device go to an Online state. This can result in Timeout. Re-connect attempts will end up with dead locking CM.	Some connectors: server device goes Online immediately after logging in to subsystem and then populates device s in the background. No timeouts in this case, but the connector must be able to stop populating devices and continue the next connection attempt. **Note:** Best

				practice is to populate devices in one call which is then queued in CM. You must make sure this application is atomic.
Offset is not displayed on specific events	If the connector has the option to give an offset to the server, it will not be displayed in all of the events.	(EAL driver) When Unlocked/Locked event in Rules Engine there are three visible events • DoorLockedEvent, • CustomChangedEvent, • DeviceStateChanged. The DoorLockedEvent and the CustomChangedEvent would show the set offset. DeviceStateChanged would show the local time.	This behavior is to be expected from all connectors.	

Video Connectors - Expected Functionality

The expected functionality of video connectors may vary slightly depending on the particular subsystem. The following table describes test scenarios to test behavior that may be different from standard.

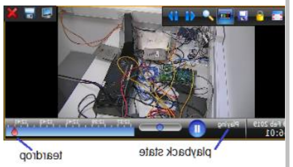
Test Scenario	Comment	Behavior 1	Behavior 2	Behavior 3
Disconnect a camera from the network (or, for analogue cameras disconnect it directly from the server hardware)	Some subsystems display a 'no video' icon on their video controls, some show a black screen, some cause a video freeze. As we want a standard	Display a standard "Video signal lost" error message on the video tile.		

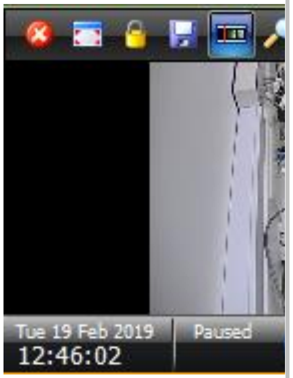
	behavior across connectors and a clear indication a signal was lost, preferable method is displaying a standard error message on the tile.			
Display a camera while parent video server is disconnected **(the server device is in *Failed* state)**	In some subsystems, it is still possible to display cameras despite the server (typically a VMS) being offline. The convention is, a driver must display a video if it can.	(Typical for DVRs with cameras physically connected to it) Display an error message in the tile saying "The video server is not connected"	(Typical for web service connection-less APIs) Display a camera regardless of parent server state.	
Restore connection to server while displaying a camera	If displaying a camera is dependent on the server connection, the connector has to manage a re-connection loop, and only try to restore the feeds once re-connected.	Re-connection implemented in the connector (example: Bosch BVMS, March Networks): The connector (in VCM) will eventually re-connect to the subsystem server and re-display the camera without user intervention.	Re-connection is not implemented in the connector (example: Verint Nextiva connector). The tile will remain displaying an error message until the camera is manually re-displayed (or with a VRP).	(Typical for web service connection-less APIs) Connector implements re-connection per camera. The camera keeps displaying video even with the server offline.
Display a non-video device in Tile Layout	Control Center allows non-video devices to be dropped on video tiles, and the connector must	Display an error message: "This device does not support video"		

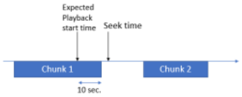
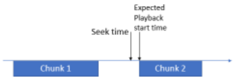
	cope with this.			
Native server configuration changes	Subsystems rarely provide API to inform about configuration changes. A user may need to either re-enable a server device or run a *Refresh Devices* server method to force downloading a new configuration.	SDK does not provide a way to detect configuration changes. A user has to re-enable a server device to get a new configuration.	SDK does not provide a way to detect configuration changes and the client wants to keep the driver connected. A server device implements *Refresh Devices* server method which forces the driver to re-connect and/or download the updated configuration.	SDK can detect configuration changes (example: Bosch BVMS). The configuration changes are detected automatically and no need for RefreshXXX methods on server device.
Re-adding a camera to subsystem	In some subsystems, a re-added camera has a new SDK ID which makes the connector treat it as a new camera. A new camera device is populated and the device that represented the camera earlier becomes unusable.	Subsystem assigned a new unique ID to a re-added/enabled camera. A new Control Center camera device is created for the re-added camera.	Subsystem assigned the same ID to a re-added/enabled camera. The same Control Center camera device is maintained for the re-added camera.	
Previous Frame and Next Frame Video Operator Actions	These two buttons are typically implemented for the SDKs which do not support slow motion (speeds 0.5, 0.2	Subsystem SDK supports slow motion: Slow motion is implemented. When playback is paused, it is possible to use the	Subsystem SDK doesn't support slow motion: Slow motion is not implemented. When playback is paused the speed slider has no	

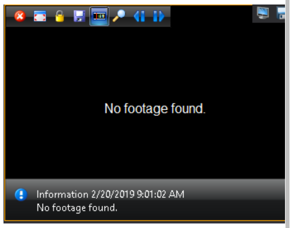
	and so on), but support playback of a next/previous frame.	speed slider control to change speeds < 1	effect. 2 buttons "Previous Frame" and "Next Frame" appear in the video tile menu.	
Video Operator Actions availability	The custom Video Operator Actions appear in every connector mode (Live, Playback, Paused) even though most of them only work in one mode. For example, Next Frame will only work in Paused mode.			
Audio Support	There is no standard implementation of Audio. However, usually there are two buttons: **Audio Mute** - toggles the camera microphone on/off (in other words, 'Audio In' feature), **Toggle Audio Out** - enable/disable streaming audio from Control Center client microphone to camera speaker.	March Networks implementation example: **Audio Mute:** When a camera with audio capability is displayed, the audio is automatically muted by default.		
Playback	It is not practical	Once switched to		

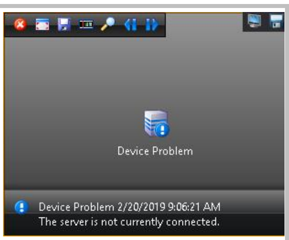
time when switching to Playback mode	to try rewinding video to present time as it takes time to record and buffer video. The exact timing is unpredictable as it's heavily dependent on a recorder model and the network speed. This means rewinding to present time would usually fail. Rewinding to a very recent time (few seconds back) may succeed, but causes the driver to stumble, as the video immediately plays to the end, then tries to seek for more video, load only few seconds, then seek again and so on. To prevent this, most drivers try to rewind to the last 15-30 seconds instead.	playback mode the camera plays from (Now - 15 seconds)		
Seek (rewind) when a recording is not found	Time taken to rewind a video is very dependent on SDK, hardware, and network so it can be unpredictable.	Connectors with **Seek Timeout** property, if recording is not found within a given time, an error message "Footage not found" is		

	To prevent a video tile seeking endlessly, many connectors introduce a **Seek Timeout** property on a server device.	displayed on the tile and the user can hide the error message and try seek again.		
Recorded video Seek (rewind)	Native video control may behave differently during rewind process. Some connectors show the rewind process on the video control. Some connectors do the search in the background and then the results are loaded into the video control. It is preferable to show progress in the video tile during a long seek operation.	Connector implements an overlay showing 'Seeking' message in the video tile. This is to hide any irrelevant video shown during the seek operation, especially for subsystems where seek may rewind to a wrong unpredictable time.	Native control shows the current progress during seek/rewind operation.	
Playing back video	1. The time in the bottom left corner shows the current playback time and gets updated every second (for the standard x1 speed). 2. The Teardrop is moving			

	along the time bar. It should always stay within a timebar chunk, never between chunks). 3. The state shown is Playing.			
Paused video	1. The time in the bottom left corner shows the current playback time. It doesn't get updated. 2. The Teardrop is not moving along the timebar. 3. The state shown is Paused.			
Seek algorithm - seek time is earlier than the first recording time available	The implementation depends on the SDK.	Connector can fetch the first recording available: The camera playback rewinds to the first available recording.	Connector cannot fetch the first recording available: The video tile displays error message saying, "Footage is not available" or "Recorded video not found"	
Seek	Ideally the	If the SDK supports	If the Seek	

algorithm - seeking for a time between two recording chunks	connector should try to play back the closest available time to the requested seek time.	the smart seek, in other words, finds the closest available time itself, the outcome totally depends on the SDK.	algorithm is implemented manually in the connector: 1. If the closest available time is a beginning of a chunk - play this chunk from the beginning 2. If the closest available time is an end time of a chunk - play the last 5-10 seconds of this chunk  Expected Playback start time Seek time Chunk 1 Chunk 2 10 sec.  Seek time Expected Playback start time Chunk 1 Chunk 2 3. A reasonable criterion for available video can be implemented, for example, play a video which is no more than 1 hour away from the desired seek time. If no recordings match the criterion, display an error message: "Footage not found" or similar.	

Playback seek error message	Error message should be displayed in the video tile if a user tries to rewind video to a time where there are no recordings (and no other recordings close enough to the seek time).	• Display the error message as an *Information message* so it can be hidden and a new seek operation can be done without closing the tile • if the native video control does not provide its own error GUI, it is better to show an overlay displaying the same error message. Do not show irrelevant footage as this may be confusing.  No footage found. Information 2/20/2019 9:01:02 AM No footage found. Below is an example of an error message to avoid. For example, it covers the whole tile, so a user cannot access the calendar control or timebar to seek again. The user has to close the tile.		

		 Device Problem Device Problem 2/20/2019 9:06:21 AM The server is not currently connected.		
'Seeking' Tile state	As seek or rewind video can take a long time, the tile should display a 'Seeking' state until the video is found and ready to play.	The player state should say: "Seeking" The tile itself may display an overlay with "Seeking..." message This is better than displaying a black screen or an irrelevant footage.		
Seek Timeout	Many connectors define a maximum time allowed to seek preventing the Tile to hang (this is needed for SDKs which don't implement this internally), this is set in *Seek Timeout* property on the parent server device	When seek starts the connector waits for seek results displaying 'Seeking' status in the Tile. If the SDK returns no results (or fails to rewind) for *Seek Timeout*, the "No footage found" error message is displayed.		
Scroll Teardrop on Time bar	In some connectors, the SDK does not allow you to cache seek results (or query the actual chunks available), so each rewind may take time.			

Video Export Task fails	Video Export Job or Task is scheduled and then failed.	Correct error message is displayed in *Message* column. The folder in the path picked in the export wizard is not created.		
Video Export Task is cancelled by user	Video Export Job or Task is scheduled and then cancelled by a user.	No error messages are displayed (or it can display a status message: "Cancelled by user"). The folder in the path picked in the export wizard is not created.		
Video export when IPSC Client and subsystem are in one Timezone, and IPSC Server is in another timezone	Video exported according to the local time set in the Export Wizard.	The file title includes timestamp in UTC.	If a timestamp is displayed in the actual video file it should be in UTC. Example: OnSSI Ocularis	

Example FSM Implementation

```
using System;
 using System.Collections.Generic;
 using System.Globalization;
 using System.Threading.Tasks;
 using CNL.IPSecurityCenter.Driver.Utility.Threading;using log4net;

 namespace CNL.IPSecurityCenter.Driver.Verint.Nextiva.Ipsc.PlaybackFsm

 {
    internal class PlaybackFsm : IDisposable
    {
       private struct StateTransition
```

```
      {
         private readonly EPlaybackStates _currentState;
         private readonly EPlaybackFsmCommand _command;
         public StateTransition(EPlaybackStates state, EPlaybackFsmCommand
command)
         {
         _currentState = state;
         _command = command;
         }
         //need this because this object is used as a dictionary key public
override int GetHashCode()
         {
         return 17 + 31 * _currentState.GetHashCode() + 31 *
_command.GetHashCode();
         }
         //need this because this object is used as a dictionary key public
override bool Equals(object obj)
         {
         var other = (StateTransition)obj;
         return this._currentState == other._currentState && this._command ==
other._command;
         }
         }
         protected ILog _log;
         private string _deviceLabel;
         /// <summary>
         /// Seek Time passed by command, saved in this temporary variable
because the seek command might be rejected
         /// </summary>
         private DateTime _seekTimePending;
         /// <summary>
         /// Time to play from in the end of successful Seek query
         /// </summary>
         private DateTime _startPlaybackTime;
         private SafeTimer _seekTimer;
         private float _speed;
         //the stata machine truth table to easily locate valid state
transitions
         private Dictionary<StateTransition, Action> _truthTable;
         private event EventHandler<FsmSeekEventArgs> CmdRequest;
         /// <summary>
         /// Fired when seek operation is failed (due to SDK reply or time
out).
         /// </summary>
         public event EventHandler SeekFailed;
         public EPlaybackStates CurrentState { get; private set; }
         /// <summary>
         /// Gets or sets the last user play/pause command.
         /// </summary>
         public bool IsPaused { get; set; }
         /// <summary>
         /// Gets seek time of the current/last seek operation
         /// </summary>
         public DateTime SeekTime { get; private set; }
         /// <summary>
         /// Amount of video in mimutes loaded per query each way - for a
given SeekTime, FSM will seek for media from -
```

```
         LoadMediaRangeMinutes to LoadMediaRangeMinutes
         /// </summary>
         public int LoadMediaRangeMinutes { get; private set; }
         public DateTime ActualStartTime { get; private set; }
         public DateTime ActualEndTime { get; private set; }
        /// <summary>
        /// Gets or sets the timeout for Seek operation
        /// </summary>
        public int SeekTimeoutMsec
       {
           get { return _seekTimer.IntervalMilliseconds; }
           set { _seekTimer.IntervalMilliseconds = value; }
       }
       public PlaybackFsm(bool isPaused, string deviceLabel)
       {
          _deviceLabel = deviceLabel;
          _seekTimer = new SafeTimer(false, 10000, "Seek Timer");
          _seekTimer.Elapsed += OnSeekTimeout;
          const int throttleDelayMs = 130;
          CmdRequest += CreateThrottledEventHandler(ThrottleInvoker,
TimeSpan.FromMilliseconds(throttleDelayMs));
          Reset(isPaused);
          _truthTable = new Dictionary<StateTransition, Action>
         {
                { new StateTransition(EPlaybackStates.SeekFailed,
EPlaybackFsmCommand.Seek), SeekInit }, //initialize
                  new seek operation
                { new StateTransition(EPlaybackStates.SeekInit,
EPlaybackFsmCommand.LoadMedia), LoadMedia }, //the media is
                  not loaded yet - load it
                { new StateTransition(EPlaybackStates.SeekInit,
EPlaybackFsmCommand.Play), SeekAndStartPlayback }, //the
                  media is loaded & validated already - start playback (play
or pause)
                { new StateTransition(EPlaybackStates.MediaLoaded,
EPlaybackFsmCommand.ValidateMedia), ValidateMedia }, //the
                  media is loaded, but not validated - validate it
                { new StateTransition(EPlaybackStates.MediaLoaded,
EPlaybackFsmCommand.Seek), SeekOverride }, //start a new
                  seek query while another one is already in progress
                { new StateTransition(EPlaybackStates.MediaLoaded,
EPlaybackFsmCommand.ChangeSpeed), SaveSpeed }, //save the
                  speed so playback starts at that speed when we start it
                { new StateTransition(EPlaybackStates.MediaValidated,
EPlaybackFsmCommand.Play), StartPlayback }, //the media
                  is loaded & validated - start playback (play or pause)
                { new StateTransition(EPlaybackStates.Playback,
EPlaybackFsmCommand.Seek), SeekInit }, //new Seek
request                     while playing
                { new StateTransition(EPlaybackStates.Playback,
EPlaybackFsmCommand.Pause), Pause }, //pause the playback
                { new StateTransition(EPlaybackStates.Playback,
EPlaybackFsmCommand.ChangeSpeed), ChangeSpeed }, //change
                  the playback speed
                { new StateTransition(EPlaybackStates.Pause,
EPlaybackFsmCommand.Seek), SeekInit }, //new Seek request while
                  paused
                { new StateTransition(EPlaybackStates.Pause,
EPlaybackFsmCommand.Play), Resume }, //resume the paused
                  playback
                { new StateTransition(EPlaybackStates.Pause,
EPlaybackFsmCommand.ChangeSpeed), ChangeSpeed }, //change the
                  playback speed
                { new StateTransition(EPlaybackStates.Playback,
EPlaybackFsmCommand.Play), Resume }, //Enable Pause for Web
                  Client
                };
          }
          public EPlaybackStates ProcessCommand(EPlaybackFsmCommand cmd, bool
throttle, DateTime seekTime =
          default(DateTime), float speed = 1.0f)
          {
                var transition = new StateTransition(CurrentState, cmd);
                if (!_truthTable.ContainsKey(transition))
                {
                       _log.WarnFormat("{0}: Illegal Command '{1}' for State
{2}", _deviceLabel, cmd, CurrentState);
                       return EPlaybackStates.Illegal;
                }
                else
                {
                       Action action = _truthTable[transition];
                       if (action != null)
                       {
                             var args = new FsmSeekEventArgs(cmd, seekTime,
speed, action);
                             if (throttle)
                             {
                                    if(CmdRequest != null)
                                    CmdRequest.Invoke(this, args);
                             }
                             else
                            {
                                    ThrottleInvoker(this, args);
                            }
                       }
                 }
                 return CurrentState;
        }
        /// <summary>
        /// Resets the state machine to initial state values
        /// </summary>
        public void Reset(bool isPaused)
        {
            CurrentState = EPlaybackStates.SeekFailed;
            IsPaused = isPaused;
            _speed = 1f;
            _seekTimer.Enabled = false;
            ActualStartTime = DateTime.MinValue;
            ActualEndTime = DateTime.MaxValue;
            _startPlaybackTime = DateTime.MinValue;
            LoadMediaRangeMinutes = 60;
        }
        private void SeekInit()
        {
            _log.DebugFormat("{0}: SeekInit", _deviceLabel);
            SwitchToState(EPlaybackStates.SeekInit);
            SeekTime = _seekTimePending;
           _startPlaybackTime = SeekTime; //by default will play from the 
desired seek time
           if (IsRelevantMediaLoaded())
       {
              _log.DebugFormat("{0}: media is already loaded", _deviceLabel);
               ProcessCommand(EPlaybackFsmCommand.Play, false);
           }
           else
           {
              _seekTimer.Enabled = true;
              LoadMedia();
           }
       }
       private bool IsRelevantMediaLoaded()
       {
           if (ActualStartTime == DateTime.MinValue)
           {
               return false;
           }
           return SeekTime >= ActualStartTime && SeekTime <= ActualEndTime;
           }
       //Fired when media cannot be validated - meaning the media 'loadled' 
is invalid and cannot be played back
       private void OnSeekTimeout(object sender, EventArgs args)
       {
           if (CurrentState == EPlaybackStates.SeekInit || CurrentState == 
EPlaybackStates.MediaLoaded
           || CurrentState == EPlaybackStates.MediaValidated)
       {
           _log.DebugFormat("{0}: Seek timed out", _deviceLabel);
           _seekTimer.Enabled = false;
           OnSeekFailed(true);
           }
           else
           {
                _log.WarnFormat("{0}: Seek time out was ignored! FSM State: 
{1}", _deviceLabel, CurrentState);
           }
       }
       //called when Seek process fails or timed out
       private void OnSeekFailed(bool fireEvent)
      {
               _log.InfoFormat("{0}: Seek failed - no data available", 
_deviceLabel);
               ActualStartTime = DateTime.MinValue;
               ActualEndTime = DateTime.MaxValue;
              //NOTE: the native pause causes exception in SDK 6.4 SP3, but 
might be still relevant in 6.4 SP1
              SwitchToState(EPlaybackStates.SeekFailed);
              if (fireEvent && SeekFailed != null)
              {
              SeekFailed.Invoke(this, EventArgs.Empty);
              }
      }
      //Cancel existing Seek process
      private void SeekAbort()
      {
             _log.DebugFormat("{0}: aborting the current Seek",
_deviceLabel);
             OnSeekFailed(false);
      }
     //Start a new Seek after aborting an Seek in progress
     private void SeekOverride()
     {
            SeekAbort();
            SeekInit();
     }
     private void StartPlayback()
     {
          if (IsPaused)
          {
               Pause();
          }
          else
          {
               Play();
          }
     }
     //called in SeekInit -> Playback transition (media is loaded already)
     //need to update time on video control before playback
     private void SeekAndStartPlayback()
     {
          NativeSeek(_startPlaybackTime);
          StartPlayback();
     }
     private void Play()
     {
         _log.DebugFormat("{0}: Play", _deviceLabel);
         SwitchToState(EPlaybackStates.Playback);
         NativeChangeSpeed(_speed);
         NativeSeek(_startPlaybackTime);
         NativePlay();
     }
     private void ChangeSpeed()
     {
         _log.DebugFormat("{0}: ChangeSpeed", _deviceLabel);
         SwitchToState(EPlaybackStates.Playback);
         NativeChangeSpeed(_speed);
     }
     private void SaveSpeed()
     {
         _log.Debug($"{_deviceLabel}: SaveSpeed ({_speed})");
     }
     private void Pause()
     {
         _log.DebugFormat("{0}: Pause", _deviceLabel);
         SwitchToState(EPlaybackStates.Pause);
         NativePause();
     }
     private void Resume()
     {
        _log.DebugFormat("{0}: Resume", _deviceLabel);
        SwitchToState(EPlaybackStates.Playback);
        NativeChangeSpeed(_speed);
        NativeResume();
     }
     private void LoadMedia()
     {
        _log.DebugFormat("{0}: LoadMedia", _deviceLabel);
        if (NativeLoadMedia(SeekTime.AddMinutes(-LoadMediaRangeMinutes),
SeekTime.AddMinutes(LoadMediaRangeMinutes)))
     {
             SwitchToState(EPlaybackStates.MediaLoaded);
             ProcessCommand(EPlaybackFsmCommand.ValidateMedia, false);
        }
        else
        {
            _log.ErrorFormat("{0}: Load media at {1} has failed",
_deviceLabel, SeekTime);
            OnSeekFailed(true);
        }
     }
     private void ValidateMedia()
     {
            NativeValidateMedia(SeekTime);
     }
     ////-- Native method stubs
     protected virtual void NativeResume()
     {
            _log.DebugFormat("{0}: NativeResume", _deviceLabel);
     }
     protected virtual void NativePlay()
     {
           _log.DebugFormat("{0}: NativePlay", _deviceLabel);
     }
     protected virtual void NativePause()
     {
          _log.DebugFormat("{0}: NativePause", _deviceLabel);
     }
     /// <summary>
     /// Load recorded media from recorder device
     /// </summary>
     protected virtual bool NativeLoadMedia(DateTime fromTime, DateTime
toTime)
     {
         _log.DebugFormat("{0}: Loading recorded media from: {1} to: {2}",
_deviceLabel, fromTime, toTime);
          return true;
     }
     /// <summary>
     /// Check the loaded media is relevant to the initial user query~
     /// </summary>
     protected virtual void NativeValidateMedia(DateTime seekTime)
     {
          _log.DebugFormat("{0}: NativeValidateMedia", _deviceLabel);
     }
     /// <summary>
     /// Update native video control player with desired playback time
     /// </summary>
     protected virtual void NativeSeek(DateTime startPlaybackTime)
     {
          _log.DebugFormat("{0}: NativeSeek - seek time: {1}", _deviceLabel,
startPlaybackTime);
     }
     protected virtual void NativeChangeSpeed(float speed)
     {
          _log.DebugFormat("{0}: NativeChangeSpeed to {1}", _deviceLabel,
speed);
     }
     /// <summary>
     /// Called when the media validation is performed asynchronously by SDK
     /// </summary>
     public void OnMediaValidated(DateTime actualStart, DateTime actualEnd,
DateTime startPlaybackTime)
     {
          //external method call - ensure we don't break the SM logic
          if (CurrentState == EPlaybackStates.MediaLoaded)
          {
                _log.DebugFormat(CultureInfo.InvariantCulture, "{0}:
Validated recorded media range from {1} to {2},
start                   playback from: {3}", _deviceLabel, actualStart,
actualEnd, startPlaybackTime);
                ActualStartTime = actualStart;
                ActualEndTime = actualEnd;
               _startPlaybackTime = startPlaybackTime;
               _seekTimer.Enabled = false;
                SwitchToState(EPlaybackStates.MediaValidated);
                ProcessCommand(EPlaybackFsmCommand.Play, false);
          }
     }
     /// <summary>
     /// Called when media validation fails (for example if SDK returns
irrelevant results or throws exceptions)
     /// </summary>
     public void OnMediaValidationFailure()
     {
         //external method call - ensure we don't break the SM logic
         if (CurrentState == EPlaybackStates.MediaLoaded)
         {
               _log.DebugFormat("{0}: Media Validation failed",
_deviceLabel);
               OnSeekFailed(true);
         }
     }
     private void SwitchToState(EPlaybackStates state)
     {
             _log.DebugFormat("{0}: Playback SM switching from {1} to {2}",
_deviceLabel, CurrentState, state);
             CurrentState = state;
     }
     [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Performance"
, "CA1822:MarkMembersAsStatic", Justification =
     "follow the event standard pattern")]
     private EventHandler<FsmSeekEventArgs>
```

```
CreateThrottledEventHandler(EventHandler<FsmSeekEventArgs> handler, TimeSpan
     throttle)
     {
            bool throttling = false;
            return (s, e) =>
           {
                 if (throttling)
                 {
                        _log.DebugFormat("Seek {0} was ignored due to
throttling logic", e.SeekTime);
                        return;
                 }
                throttling = true;
                Task.Delay(throttle).ContinueWith(_ => throttling = false);
           };
      }
      //The handler of CmdRequest event
      private void ThrottleInvoker(object sender, FsmSeekEventArgs args)
      {
           _log.DebugFormat("{0}: Playback FSM, State '{1}', Command '{2}'",
_deviceLabel, CurrentState, args.Command);
           _seekTimePending = args.SeekTime;
           _speed = args.Speed;
           args.FsmAction.Invoke();
      }
      public void Dispose()
      {
          if (_seekTimer != null)
          {
              _seekTimer.Elapsed -= OnSeekTimeout;
              _seekTimer.Dispose();
              _seekTimer = null;
          }
          CmdRequest = null;
          }
      }
 }
```

Control Center ISDK Compatibility

The Control Center ISDK is a set of tools and interfaces exposed in Control Center to create connectors. Essentially, it is a collection of types and interfaces related to connectors.

Control Center DDK uses a versioning scheme to describe how API versions are backwards-compatible with earlier versions of Control Center.

NOTE: IPSecurityCenter was renamed Control Center from version 5.25 onwards. From version 5.30 onwards, Driver Development Kit (DDK) was renamed Integrations Software Development Kit (ISDK).

Control Center ISDK starts at version 3.0. All subsequent versions are backward-compatible.

Each version of Control Center is compatible with one or more ISDK Versions.

Control Center	DDK Version												
	3.0	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.10	3.11	3.12
5.0.x	✓												
5.1.x	✓												
5.2.x	✓	✓											
5.3.x	✓	✓											
5.4.x	✓	✓	✓										
5.5.x	✓	✓	✓	✓									
5.6.x	✓	✓	✓	✓	✓								
5.7.x	✓	✓	✓	✓	✓	✓							
5.8.x	✓	✓	✓	✓	✓	✓							
5.9.x	✓	✓	✓	✓	✓	✓	✓						
5.10 5.10.1 5.10.2	✓	✓	✓	✓	✓	✓	✓	✓					
5.10.3 5.12	✓	✓	✓	✓	✓	✓	✓	✓	✓				
5.13	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
5.14.5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
5.18	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
5.19	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
5.20	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5.22	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5.23	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5.24	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5.25	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

ISDK Versions

The following sections list the changes in the APIs for each released version of the DDK.

ISDK 3.0

ISDK 3.0 succeeded ISDK 2.4. From version 3.0, connectors must be capable of being loaded into a 64-bit process to query their type information, as well as a 32-bit process. Therefore, connectors must be built for any CPU and must not expose any 32-bit-only types (such as types defined in a 32-bit-only 3rd party SDK, for example).

The `IVideoControlWithDynamicOperatorActions` interface definition was added. The `IVideoControlWithDynamicOperatorActions` is an optional interface implemented by a video control to expose additional actions other than the ones indicated statically (by `OperatorAction` attribute on its methods).

```
namespace CNL.IPSecurityCenter.Driver.Video.DynamicOperatorActions
 {
     /// <summary>
     /// Optionally implemented by a video control to expose additional
actions other than the
     /// ones indicated statically (by OperatorAction attribute on its
methods).
     /// </summary>
     public interface IVideoControlWithDynamicOperatorActions
     {
         /// <summary>
         /// Raised by the video control to specify what actions it supports
         /// </summary>
         event EventHandler<DynamicOperatorActionsChangedEventArgs>
DynamicOperatorActionsChanged;
         /// <summary>
         /// NB - currently ignored by IPSC!
         /// </summary>
         event EventHandler<DynamicOperatorActionStateChangedEventArgs>
DynamicOperatorActionStateChanged;
         /// <summary>
         /// Executes one of the actions supported by the control (according
to the most recent
         /// DynamicOperatorActionsChanged event raised.
         /// </summary>
         /// <param name='name'>The name of the action to execute</param>
         void ExecuteDynamicOperatorAction(string name);
     }
 }
```

ISDK 3.1

ISDK 3.1 contains some new interfaces that connectors can optionally implement.

IVideoControlWithDynamicOperatorActions

You can implement this interface for a video control to expose additional actions other than the ones indicated statically (by `OperatorAction` attribute on its methods).

```
{
     /// <summary>
     /// Optionally implemented by a video control to expose additional
actions other than the
     /// ones indicated statically (by OperatorAction attribute on its
methods).
     /// </summary>
     public interface IVideoControlWithDynamicOperatorActions
     {
         /// <summary>
         /// Raised by the video control to specify what actions it supports
         /// </summary>
         event EventHandler<DynamicOperatorActionsChangedEventArgs>
DynamicOperatorActionsChanged;
         /// <summary>
         /// NB - currently ignored by IPSC!
         /// </summary>
         event EventHandler<DynamicOperatorActionStateChangedEventArgs>
DynamicOperatorActionStateChanged;
         /// <summary>
         /// Executes one of the actions supported by the control (according
to the most recent
         /// DynamicOperatorActionsChanged event raised.
         /// </summary>
         /// <param name='name'>The name of the action to execute</param>
         void ExecuteDynamicOperatorAction(string name);
     }
 }
```

IDeviceOverridesLabel

Allows a device to stop the user changing its label. Normally, Control Center allows the user to change the label of a device. However, a connector may control the label. In this case, you do not want to allow a user to also change the label as the connector can overwrite the label at any time, without warning.

A likely pattern is that a VMS connector may want to allow the user to decide if camera device labels should be automatically updated from the external subsystem. In which case, the connector's main server device could have a boolean property called, for example, `UseSubsystemCameraLabels`, and the camera/child device can delegate its own `OverridesLabel` property to that setting.

```
// <summary>
    /// Allows a device to stop the user changing its label.
    /// </summary>
    public interface IDeviceOverridesLabel
    {
        /// <summary>
        /// If true, IPSC will not allow the user to change the label of this
device, as the
        /// device itself may change the label at any time due to changes in
the subsystem.
        /// </summary>
        bool OverridesLabel { get; }
    }
```

DeviceOverridesChildOnlineState (Attribute)

This attribute, when applied, allows a parent device to control the online state of its child devices. In other words, the devices do not automatically follow the enabled/disabled state of the parent device and may stay in an alternate state when a parent device has it state changed, rather that the default operation of following the parent device state. Child devices of a device are defined as all the devices connected to it that do not implement `IConnectableDevice`.

Normally, when a parent device is brought online, all its child devices have their online states set to online, although this happens after an unpredictable delay. Where the connector wishes to update its child device states to make them accurately reflect the states of whatever real-world devices they represent, it has previously been necessary to use a Thread.Sleep work-round to give Control Center enough time to finish setting the states to online.

Now, the server device's contract can optionally have the attribute `DeviceOverridesChildOnlineState` (no parameters). Only three state changes are affected: Online, Warning and Custom, as they can describe a 'healthy' state.

CAUTION: There are two places a connector has to implement special code if it adds this attribute to the server's contract:

1. In the server's `Connect` method, it has to update each child device's state.
2. On the child, it must handle the `EnabledChanged` event to also update that child device's state when the device is **Enabled**.

This table summarises whether a parent's state change is propagated to its children, depending on whether the default behavior has been overridden by applying this attribute to the server device class.

State	Default	Overridden
Connecting	No	No
Online	Yes	No
Warning	Yes	No
Custom	Yes	No
Disabled	Yes	Yes
Failed	Yes	Yes
Offline	Yes	Yes

ISDK 3.2

ISDK 3.2 implements the following interfaces.

SupportedPreviousDriverAttribute

When there are datatype changes in properties between connector versions, this attribute can be used to mitigate serialisation issues and allow objects conversion of types between the old connector and the new connector. The Connector developer documentation gives more detail on how to use this interface.

ISupportsPausingActivity

An interface to support notifying a connector when a tile is not active. When applied to a device, implementation of this interface indicates that the connector supports some form of pausing of its activity while still monitoring device state. It is intended to be used primarily by video connectors, although, any data steaming device may be a candidate to pause/resume the supplied data stream.

Typically, the functions of this interface are called when the UI framework knows that whatever the connector is doing is currently not visible to the user (for example, a video tile is not visible). Ideally, the pausing and resuming of activity should be implemented in as efficient a way as possible.

```
/// <summary>
    /// Implementation of this interface indicates that the driver supports
some form
    /// of 'pausing' of its activity whilst still monitoring device state and
the like.
    /// It intended to be used by Video Drivers to pause/resume display of
video. Typically
    /// the functions of this interface will be called when the UI framework
knows that
    /// whatever the driver is doing is currently not visible to the user
(e.g. video
    /// tile is not visible). Ideally the pausing and resuming of activity
should be
    /// implemented in as efficient a way as possible.
    /// </summary>
        public interface ISupportsPausingActivity
```

Motion JPEG support

A set of three interfaces indicating a device that is capable of producing data streams that meets the specification of Motion JPEG.

```
/// <summary>
    /// An open stream that is producing JPEG frames, implemented by a
driver.
    /// The caller to this interface is part of IPSC, not the driver.
    /// </summary>
    public interface IMotionJpegStream : IDisposable
    /// <summary>
    /// Implemented by a device (typically a camera) when it is capable of
streaming
    /// in MJPEG format.
    /// </summary>
    public interface IMotionJpegSource
    /// <summary>
    /// An object representing a single frame of video. Implement this
interface
    /// in a driver if you want to delay computing the frame image until it
is
    /// actually needed.
    /// </summary>
    public interface IMotionJpegFrame
```

ThrottledEventManager

ThrottledEventManager class added to support device connector event throttling. It creates an event aggregation manager, where you can define the rate at which events are sent to the rest of the Control Center systems. The throttling function and the event generated are defined by you and events, with potential information on items, such as number of events seen since last throttling action, can be added to the sent event.

```
namespace CNL.IPSecurityCenter.Driver.ThrottledEvents
 {
   /// <summary>
   ///
   /// </summary>
   public class ThrottledEventManager
   {
     private static readonly ThrottledEventManager _instance = new
ThrottledEventManager();
     private static readonly object _locker = new object();
     private static readonly int PollMSecsInterval = 100;
     private readonly Dictionary<Type, ThrottledEventManager.EventDefinition>
_eventDefinitions = new Dictionary<Type,
ThrottledEventManager.EventDefinition>();
     private Task _pollTask = (Task) null;
     private ThrottledEventManager()    {}
     public static ThrottledEventManager Instance =>
ThrottledEventManager._instance;
     public void RegisterThrottledEvent<TD, TA>(
       int maxEventsPerSec,
       ThrottledEventManager.AggregateFunction<TD, TA> aggregateFunc,
       ThrottledEventManager.RaiseEventAction<TD, TA> raiseEventAction)
       where TD : IDevice
       where TA : DeviceEventArgs
     { … }
     public void RaiseEvent<TD, TA>(TD sender, TA args)
       where TD : IDevice
       where TA : DeviceEventArgs
     { … }
     private void StartPollLoop()
     { … }
     private void PollTask()
     { … }
     public delegate bool AggregateFunction<TD, TA>(TD dev1, TA args1, TD
dev2, TA args2)
       where TD : IDevice
       where TA : DeviceEventArgs;
     public delegate void RaiseEventAction<TD, TA>(TD sender, TA args)
       where TD : IDevice
       where TA : DeviceEventArgs;
   }
 }
 namespace CNL.IPSecurityCenter.Driver.ThrottledEvents
 {
       public IDevice Sender { get; set; }
       public DeviceEventArgs Event { get; set; }
       public DateTime NextEventRaiseTime { get; set; }
       public bool Sent { get; set; }
 }
 namespace CNL.IPSecurityCenter.Driver.ThrottledEvents
 {
     private class EventDefinition
     {
       public IList<ThrottledEventManager.QueuedEvent> Queued =
                                         (IList<ThrottledEventManager.QueuedE
vent>)
                                             New
List<ThrottledEventManager.QueuedEvent>();
       public EventDefinition(
         int minMSecBetweenEvents,
         ThrottledEventManager.AggregateFunction<IDevice, DeviceEventArgs>
aggregateFunc,
         ThrottledEventManager.RaiseEventAction<IDevice, DeviceEventArgs>
raiseEventAction)
       { … }
       public int MinMSecBetweenEvents { get; }
       public ThrottledEventManager.AggregateFunction<IDevice,
DeviceEventArgs> AggregateFunc
            { get; }
       public ThrottledEventManager.RaiseEventAction<IDevice,
DeviceEventArgs> RaiseEventAction
            { get; }
     }
 }
```

ISDK 3.3

ISDK 3.3 implements two Event interfaces, both associated with displaying event information on the time-bar of the Video Connection Manager (VCM) display.

ITimebarDisplayAlwaysEvent

When a connector event implements this interface (and it only make senses for events on cameras), whenever this event is raised, it will always be displayed as a dot on a timebar, if that camera is being shown in playback mode.

```
/// <summary>
    /// This event will always be displayed as a dot on timebar in playback
mode
    /// </summary>
    [DesignerVisibleEventInterface]
    [DisplayName(DeviceConstants.ResourcePath,
'DisplayNameTimebarDisplayAlwaysEvent', typeof(ITimebarDisplayAlwaysEvent))]
    [Description(DeviceConstants.ResourcePath,  'DescriptionTimebarDisplayAlw
aysEvent',    typeof(ITimebarDisplayAlwaysEvent))]
    public interface ITimebarDisplayAlwaysEvent
```

ITimebarDisplayOptionalEvent

When a connector's event implements this interface (and it only makes sense only for events on cameras), whenever this event is raised, it can be displayed as a dot on timebar, if that camera is being shown in playback mode. A requirement of its use is that you have to configure on the camera object in its property grid what optional events should be displayed on the VCM Timebar. The property on a Camera is called 'TimebarEvents'

```
/// <summary>
    /// This event can be diplayed as a dot on timebar in playback mode
    /// </summary>
    [DesignerVisibleEventInterface]
    [DisplayName(DeviceConstants.ResourcePath,
'DisplayNameTimebarDisplayOptionalEvent',
       typeof(ITimebarDisplayOptionalEvent))]
    [Description(DeviceConstants.ResourcePath,
'DescriptionTimebarDisplayOptionalEvent',
       typeof(ITimebarDisplayOptionalEvent))]
    public interface ITimebarDisplayOptionalEvent
```

ISDK 3.4

This version implements a number of new interfaces and types to support display of position-aware entities on schematic scenes.
 These mirror the entities for geo-spatial connectors.

IPositionAware

When this interface is applied to a device or object it allows Control Center to plot the object on a schematic screen display.

```
namespace CNL.IPSecurityCenter.Driver.Types.PositionAware
 {
   /// <summary>Defines a position aware object</summary>
   [ServiceContract]
   [DesignerVisible]
   [DisplayName("CNL.IPSecurityCenter.Driver.Strings",
"DisplayNameIPositionAware", typeof (IPositionAware))]
   [Description("CNL.IPSecurityCenter.Driver.Strings",
"DescriptionIPositionAware", typeof (IPositionAware))]
   public interface IPositionAware : IDevice
   {
     /// <summary>
     ///     The Positional Reference Identifier that the coordinates are
using
     /// </summary>
     [CategoryPosition]
     [DisplayName("CNL.IPSecurityCenter.Driver.Strings",
"DisplayNamePositionalReferenceIdentifier", typeof (IPositionAware))]
     [Description("CNL.IPSecurityCenter.Driver.Strings",
"DisplayNamePositionalReferenceIdentifier", typeof (IPositionAware))]
     int PositionalReferenceIdentifier { [OperationContract] get;
[OperationContract] set; }
     /// <summary>The X Axis coordinate value.</summary>
     [CategoryPosition]
     [DisplayName("CNL.IPSecurityCenter.Driver.Strings", "DisplayNameX",
typeof (IPositionAware))]
     [Description("CNL.IPSecurityCenter.Driver.Strings", "DisplayNameX",
typeof (IPositionAware))]
     double X { [OperationContract] get; [OperationContract] set; }
     /// <summary>The Y Axis coordinate value.</summary>
     [CategoryPosition]
     [DisplayName("CNL.IPSecurityCenter.Driver.Strings", "DisplayNameY",
typeof (IPositionAware))]
     [Description("CNL.IPSecurityCenter.Driver.Strings", "DisplayNameY",
typeof (IPositionAware))]
     double Y { [OperationContract] get; [OperationContract] set; }
     /// <summary>The Z Axis coordinate value.</summary>
     [CategoryPosition]
     [DisplayName("CNL.IPSecurityCenter.Driver.Strings", "DisplayNameZ",
typeof (IPositionAware))]
     [Description("CNL.IPSecurityCenter.Driver.Strings", "DisplayNameZ",
typeof (IPositionAware))]
     double Z { [OperationContract] get; [OperationContract] set; }
   }
 }
```

IPositionAwareEvent

When applied against an event, IPositionAwareEvent provides location speed and heading information into Control Center allowing the system to update the schematic object information based on VRPs.

```
namespace CNL.IPSecurityCenter.Driver.Types.PositionAware
 {
   /// <summary>Defines a locatable event</summary>
   [DesignerVisibleEventInterface]
   [DisplayName("CNL.IPSecurityCenter.Driver.Strings",
"DisplayNamePositionAwareEvent", typeof (IPositionAwareEvent))]
   [Description("CNL.IPSecurityCenter.Driver.Strings",
"DescriptionPositionAwareEvent", typeof (IPositionAwareEvent))]
   public interface IPositionAwareEvent
   {
     /// <summary>
     ///     The Positional Reference Identifier for the system that the
coordinates are using
     /// </summary>
     [CategoryPosition]
     [DisplayName("DisplayNamePositionalReferenceIdentifier", typeof
(IPositionAwareEvent))]
     [Description("DisplayNamePositionalReferenceIdentifier", typeof
(IPositionAwareEvent))]
     int PositionalReferenceIdentifier { get; set; }
     /// <summary>The X Axis Value.</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameX", typeof (IPositionAwareEvent))]
     [Description("DisplayNameX", typeof (IPositionAwareEvent))]
     double? X { get; set; }
     /// <summary>The Y Axis Value.</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameY", typeof (IPositionAwareEvent))]
     [Description("DisplayNameY", typeof (IPositionAwareEvent))]
     double? Y { get; set; }
     /// <summary>The Z Axis Value.</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameZ", typeof (IPositionAwareEvent))]
     [Description("DisplayNameZ", typeof (IPositionAwareEvent))]
     double? Z { get; set; }
     /// <summary>Heading Value.</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameHeading", typeof (IPositionAwareEvent))]
     [Description("The heading of this track", typeof (IPositionAwareEvent))]
     double? Heading { get; set; }
     /// <summary>Speed Value.</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameSpeed", typeof (IPositionAwareEvent))]
     [Description("The speed of this track", typeof (IPositionAwareEvent))]
     double? Speed { get; set; }
   }
 }
```

IPositionAwareTracking

This is an extension to the IPositionAware interface and allows the object to provide a tracked position on the schematic scene

```
namespace CNL.IPSecurityCenter.Driver.Types.PositionAware
 {
   /// <summary>Defines a positional Tracking Object</summary>
   [ServiceContract]
   [DesignerVisible]
   [DisplayName("CNL.IPSecurityCenter.Driver.Strings",
"DisplayNameIPositionAwareTracking", typeof (IPositionAwareTracking))]
   [Description("CNL.IPSecurityCenter.Driver.Strings",
"DescriptionIPositionAwareTracking", typeof (IPositionAwareTracking))]
   public interface IPositionAwareTracking : IPositionAware, IDevice
   {
     /// <summary>Last update in UTC format</summary>
     [CategoryPosition]
     [DisplayName("CNL.IPSecurityCenter.Driver.Strings",
"DisplayNameLastUpdateUtc", typeof (IPositionAwareTracking))]
     [Description("CNL.IPSecurityCenter.Driver.Strings",
"DescriptionLastUpdateUtc", typeof (IPositionAwareTracking))]
     DateTime LastUpdateUtc { [OperationContract] get; [OperationContract]
set; }
     /// <summary>Last heading of the device</summary>
     [CategoryPosition]
     [DisplayName("CNL.IPSecurityCenter.Driver.Strings",
"DisplayNameHeading", typeof (IPositionAwareTracking))]
     [Description("CNL.IPSecurityCenter.Driver.Strings",
"DescriptionHeading", typeof (IPositionAwareTracking))]
     double? Heading { [OperationContract] get; [OperationContract] set; }
     /// <summary>Last heading of the device</summary>
     [CategoryPosition]
     [DisplayName("CNL.IPSecurityCenter.Driver.Strings", "DisplayNameSpeed",
typeof (IPositionAwareTracking))]
     [Description("CNL.IPSecurityCenter.Driver.Strings", "DescriptionSpeed",
typeof (IPositionAwareTracking))]
     double? Speed { [OperationContract] get; [OperationContract] set; }
     /// <summary>Raised when an object's Position has changed</summary>
     [DeviceEvent]
     [DisplayName("CNL.IPSecurityCenter.Driver.Strings",
"DisplayNamePositionChanged", typeof (IPositionAwareTracking))]
     [Description("CNL.IPSecurityCenter.Driver.Strings",
"DescriptionPositionChanged", typeof (IPositionAwareTracking))]
     event EventHandler<PositionChangedEventArgs> PositionChanged;
   }
 }
```

ITrackablePositionAwareEvent

This is an extension to the IPositionAwareEvent interface that adds a unique TrackId to an event and allows Control Center to maintain a track associated with changes in position of the specified object.

```
namespace CNL.IPSecurityCenter.Driver.Types.PositionAware
 {
   /// <summary>Defines a trackable event</summary>
   [DesignerVisibleEventInterface]
   [DisplayName("CNL.IPSecurityCenter.Driver.Strings",
"DisplayNameTrackablePositionAwareEvent", typeof
(ITrackablePositionAwareEvent))]
   [Description("CNL.IPSecurityCenter.Driver.Strings",
"DescriptionTrackablePositionAwareEvent", typeof
(ITrackablePositionAwareEvent))]
   public interface ITrackablePositionAwareEvent : IPositionAwareEvent
   {
     /// <summary>Id of a track</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameTrackId", typeof
(ITrackablePositionAwareEvent))]
     [Description("Identifier of this track", typeof
(ITrackablePositionAwareEvent))]
     string TrackId { get; set; }
   }
 }
```

PositionChangedEventArgs

Event arguments used in notifying Control Center that the device location has been updated.

```
namespace CNL.IPSecurityCenter.Driver.Types.PositionAware
 {
   /// <summary>
   /// Event Arguments used when a device has changed position
   /// </summary>
    public class PositionChangedEventArgs : DeviceEventArgs
   {
     /// <summary>Position Changed Event Args Constructor</summary>
     public PositionChangedEventArgs(IDevice device)
       : base(device.Identifier)
     {    }
     /// <summary>Position Changed Event Args Constructor</summary>
     public PositionChangedEventArgs(IDevice device, DateTime date) :
base(device.Identifier, date)
     {    }
     /// <summary>DateTime the movement event occured</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameLastUpdateUtc", typeof
(PositionChangedEventArgs))]
     [Description("The date time of this objects last movement", typeof
(PositionChangedEventArgs))]
     public DateTime UpdatedDateTime { get; set; }
     /// <summary>
     /// The Positional Reference Identifier for the system that the
coordinates are using
     /// </summary>
     [CategoryPosition]
     [DisplayName("DisplayNamePositionalReferenceIdentifier", typeof
(PositionChangedEventArgs))]
     [Description("The Positional Reference Identifier of the co-ordinates",
typeof (PositionChangedEventArgs))]
     public int PositionalReferenceIdentifier { get; set; }
     /// <summary>The current Y of the object</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameY", typeof (PositionChangedEventArgs))]
     [Description("The last Y of this object", typeof
(PositionChangedEventArgs))]
     public double Y { get; set; }
     /// <summary>The current X of the object</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameX", typeof (PositionChangedEventArgs))]
     [Description("The last X of this object", typeof
(PositionChangedEventArgs))]
     public double X { get; set; }
     /// <summary>The current Z of the object</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameZ", typeof (PositionChangedEventArgs))]
     [Description("The last Z of this object", typeof
(PositionChangedEventArgs))]
     public double Z { get; set; }
     /// <summary>The current heading of the object</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameHeading", typeof (PositionChangedEventArgs))]
     [Description("The last heading of this object", typeof
(PositionChangedEventArgs))]
     public double? Heading { get; set; }
     /// <summary>The current speed of the object</summary>
     [CategoryPosition]
     [DisplayName("DisplayNameSpeed", typeof (PositionChangedEventArgs))]
     [Description("The last speed of this object", typeof
(PositionChangedEventArgs))]
     public double? Speed { get; set; }
   }
```

ISDK 3.5

ISDK 3.5 implements the following attributes and interfaces.

x64BitCompatibilityAttribute

This attribute is applied to a video control / device class, to allow hosting in a 64-bit process if possible. This attribute should only be applied if the third-party SDK supports running as a 64-bit process.

```
[AttributeUsage(AttributeTargets.Class)]
    public sealed class x64BitCompatibilityAttribute : Attribute
    {
    }
```

IVideoControlLifetimeManager (aka Lifetime Manager)

Types that implement this are instantiated when the Video Control Manager starts-up. Everbridge recommends that this be used by CCTV drivers for pre-loading and caching connections during login before the first video is displayed. Caching SDK connections should be optional.

When this interface is implemented, it effectively leaves a permanent connection to the underlying Video system, even when there are no actively displayed video streams. The consequence of this is that no third-party SDK initialization is required on initial video display, removing any delay associated with that initialization.

NOTE: Be aware that, depending on the licensing model of the third-party SDK, one connection license is used at all times, for each client and VCM which may require the purchase of additional licenses.

```
public interface IVideoControlLifetimeManager : IDisposable
    {
        /// <summary>
        /// This method will be called when the Video Control Manager is
started during login and re-starting if crashed
        /// </summary>
        void Initialise(IDeviceRepository deviceRepository);
    }
```

ISwitchCamera

```
ISwitchCamera should be implemented on a video control to optimise
performance for switching cameras.

public interface ISwitchCamera
    {
        /// <summary>
        /// Called on the Video Control when a device of the same type is
being switched
        /// </summary>
        void SwitchCamera(Guid deviceIdentifier);
    }
```

ISDK 3.6

ISDK 3.6 implemented the following interfaces.

DeviceCategoryType

Three additional device categories to support licensing have been added to the device types.

```
/// <summary>
 /// The device that has Geographical positions
 /// </summary>
  GIS
 /// <summary>
 /// The device is a Perimeter Intrusion Detection System
 /// </summary>
 PIDS
 /// <summary>
 /// Hazmat And CBRNe
 /// </summary>
 CBRNe
```

BitArithmeticHelper

A helper class in CNL.IPSecurityCenter.Driver.Utility dll for various bit operations, it allows for the following fuctionalities:

- Merge arrays
- Set, check and reset a bit in a byte or in a byte array
- Create a short number out of 2 bytes
- Get a printable version for a byte array
- Create a copy of a part of an array

Operation Scheduler

Operation Scheduler has the following use cases.

- Implementation of asynchronous APIs/protocols where the subsystem notifies that the command/operation is completed. However, some commands must be executed in a strict synchronisation (because the subsystem may require it or the next operation is dependent on the previous one).
- Inconsistent asynchronous APIs/SDKs in which the context of a command has to be saved (as a command ID, for example).

Operation Scheduler has the following terms.

- **Operation** - an atomic unit of work, executed synchronously (only one Operation is executed at any given time). An Operation has a unique ID, a Timeout (setting the execution time limit), the execution result can be True on success or False on failure.
- **Scenario** - a sequence (linked list) of Operation instances, only one Scenario can execute at a time.
    - Operation Scheduler holds a queue of scheduled Operations.
    - Timeout must be set for each Operation.
    - Operation Scheduler must initialize a set of Operations and Scenarios.

  - If Scenario has operations of a same type, you must create a separate Operation object for each Operation.
  - Operations must not be shared between Scenarios, it can lead to parameters overriding whether an operation has failed or succeeded, and the Operation result may break current Scenario execution. If it's decided an Operation Failure doesn't cause a Scenario to abort, the Scenario is still reported as completed successfully.
  - You cannot cancel Operations or Scenarios after scheduling.

Operation Scheduler has the following list of classes.

- OperationScheduler
- Operation
- Scenario
- ScenarioEventArgs
- ScenarioStatus

ISDK 3.7

To support the independent ability of device connectors (sometimes called Matrix Connectors) to coordinate object detection and tracking the following capabilities were added to the ISDK.

DeviceInterfaceType Additions

Two new interface types introduced into the `DeviceInterfaceType` enum. The first is `VideoPlayback` which is intended for devices to advertise connections providing VideoPlayback features from another Matrix connector, and `PtzControl` providing Ptz and SlewToCue features.

```
// Summary:
 // The interface is a video playback.
 VideoPlayback = 14,
 // Summary:
 // The interface is a Camera Pan Tilt Zoom Control.
 PtzControl = 15
```

IOrientationAware & IGeoSpatialOrientationAware Interfaces

Two new interfaces support devices reporting their orientation. This allows devices to have dynamically drawn viewsheds within Geographical scenes within the application.

IOrientationAware

An interface for devices that are orientated based on relative orientation.

NOTE: The relative orientation of the device is considered the device's orientation when facing forward along a horizontal plane. An example would be for a camera, the lens facing 'forward' and the camera body being horizontal.

```
public interface IOrientationAware : IDevice
    {
        /// <summary>
        /// Raised when the device's orientation is changed. For initial
purposes, devices
        /// are not expected to exceed raising a maximum of 10 notifications
```

```
per second.
        /// When the device's <see cref='SupportsFieldOfView'/> is True, the
event MUST
        /// populate the Field of View associated fields.
        /// </summary>
        event EventHandler<DeviceOrientationChangedEventArgs>
DeviceOrientationChanged;
       /// <summary>
        /// Gets the current device relative orientation.
        /// </summary>
        [System.ComponentModel.Browsable(false)]
        DeviceOrientation CurrentOrientation { get; }
        /// <summary>
        /// Gets whether the <see cref='CurrentOrientation'/> also reports
the Field of View
        /// of the device using the <see cref='DeviceOrientation'/> derived
object.
        /// </summary>
        bool SupportsFieldOfView { get; }
        /// <summary>
        /// Gets the normalized percentage of a complete rotation supporting
0.0 to 1.0.
        /// The angle is measured from forward facing, in a clockwise
direction, meaning
        /// 0.25 is 90 degrees clockwise, 0.5 is 180 degrees and 1.0 is a
full rotation
        /// (360 degrees).
        /// </summary>
        double CurrentOrientationAzimuth { get; }
        /// <summary>
        /// Gets the normalized percentage of a complete rotation supporting
-0.25 to 0.25.
        /// The angle is measured from horizontal, starting in an upwards
direction, meaning
        /// 0.25 is 90 degrees from horizontal (vertically straight up), and
-0.25 is -90 or 270
        /// degrees from horizontal (vertically straight down).
        /// </summary>
        double CurrentOrientationElevation { get; }
        /// <summary>
        /// Gets the normalized percentage of a complete arc supporting 0.0
to 1.0. The range
        /// is considered centered relative to configured orientation. This
means a arc of 0.25
        /// would be a 90 degree horizontal arc centered at the associated
orientation, with
        /// +-45 degree field of view from the orientation.
        /// </summary>
        double? CurrentFieldOfViewAzimuthArc { get; }
        /// <summary>
        /// Gets the normalized percentage of a complete arc supporting 0.0
to 1.0. The range
        /// is considered centered relative to configured orientation. This
means a arc of 0.25
        /// would be a 90 degree horizontal arc centered at the associated
orientation, with
        /// +-45 degree field of view from the orientation.
        /// </summary>
        double? CurrentFieldOfViewElevationArc { get; }
        /// <summary>
        /// Gets the minimum usable range of the field of view in meters. 
This explicitly indicates
        /// that the field of view may not covered the area between the 
device and the minimum range.
        ///
        /// This may be calculated by the driver based on reasonable 
knowledge of the capabilities
        /// of the device (ie. the configured minimum range on a radar or the 
known focus point
        /// on a camera). This may also be calculated based on a static 
configuration exposed to
        /// the user and manipulated by the known state (ie. zoom) of the 
device.
        ///
        /// This is an optional field, even when the field of view is 
available.
        /// </summary>
        double? CurrentFieldOfViewMinimumRange { get; }
        /// <summary>
        /// Gets the maximum usable range of the field of view in meters.
        ///
        /// This may be calculated by the driver based on reasonable 
knowledge of the capabilities
        /// of the device (ie. the configured maximum range on a radar or the 
known focus point
        /// on a camera). This may also be calculated based on a static 
configuration exposed to
        /// the user and manipulated by the known state (ie. zoom) of the 
device.
        ///
        /// This is an optional field, even when the field of view is 
available.
        /// </summary>
        double? CurrentFieldOfViewMaximumRange { get; }
    }
```

IGeoSpatialOrientationAware

This interface provides the orientation alignment to the Geo-Spatial environment so the device can be displayed correctly orientated within geographical scenes and is required to be used alongside `IGeoSpatialAwareWithAlt` to correctly position the device.

```
public interface IGeoSpatialOrientationAware : IOrientationAware
    {
        /// <summary>
        /// Gets the normalized percentage of a complete rotation supporting 
0.0 to 1.0.
        /// The angle is measured from forward facing, in a clockwise 
direction, meaning
        /// 0.25 is 90 degrees clockwise, 0.5 is 180 degrees and 1.0 is a 
full rotation
        /// (360 degrees).
        ///
        /// Example for this is a camera base-orientated due east would have
a value of +0.25
        /// indicating the 'front' of the device is facing east and all
orientation is considered
        /// relative to this direction.
        /// </summary>
        double BaseOrientationYawFromNorth { get; set; }
         /// <summary>
        /// Gets the normalized percentage of a complete rotation supporting
-0.25 to 0.25.
        /// The angle is measured from the horizontal surface of the earth,
starting in an
        /// upwards direction, meaning 0.25 is 90 degrees from horizontal
(vertically straight
        /// up), and -0.25 is -90 or 270 degrees from horizontal (vertically
straight down).
        ///
        /// Limited to the range -0.25 and +0.25 - for inverted devices use
        /// <see cref='BaseOrientationRoll'/>. This value indicates the
elevation
        /// angle difference between the front of the device (relative to its
tilt point)
        /// and the surface of the earth. This is only required for cameras
not mounted on a
        /// flat levelled surface - ie. device mounted on sloped roof)
        /// </summary>
        double BaseOrientationPitch { get; }
        /// <summary>
        /// Gets the normalized percentage of a complete rotation supporting
-0.5 to 0.5.
        /// The angle is measured from the horizontal surface of the earth,
along the axis
        /// formed between the center and front of the device, meaning 0.25
is rolled 90
        /// degrees clockwise through this axis, and -0.25 is rolled 90
degrees anti-clockwise
        /// through this axis.
        ///
        /// Limited to the range -0.5 and +0.5. This value indicates the
elevation angle
        /// difference between the right of the device (relative to its tilt
point) and the
        /// surface of the earth. This is only required for cameras not
mounted on a flat
        /// leveled surface - ie. device mounted on side of a building or
inverted on ceiling)
        /// </summary>
        double BaseOrientationRoll { get; }
    }
```

Geo Spatial Extensions

Additional interface extensions are available for Geo Spatial and tracking devices.

IGeoSpatialAwareWithAlt

Extension to the IGeoSpatialAware adds the Altitude field to `IGeoSpatialAware`.

```
public interface IGeoSpatialAwareWithAlt
    {
        /// <summary>
        /// Altitude from Mean Sea Level (MSL) in meters.
        /// </summary>
        double? Altitude { get; set; }
    }
```

IGeoSpatialAwareWithAltEvent

Extension for `IGeoSpatialAwareEvent` adds Altitude and Vertical Rate fields to reported Geo Spatial events.

```
public interface IGeoSpatialAwareWithAltEvent
    {
        /// <summary>
        /// Altitude from Mean Sea Level (MSL) in meters.
        /// </summary>
        double? Altitude { get; set; }
        /// <summary>
        /// Vertical Rate Value in meters per second. This is the rate of 
altitude
        /// change where positive values means ascending and negative is 
descending.
        /// </summary>
        double? VerticalRate { get; set; }
    }
```

IRelativeGeoSpatialAwareEvent

Extension for `IGeoSpatialAwareEvent` where the reporting source also provides a Relative Position in reported Geo Spatial events.

```
/// </summary>
    /// Interface for devices that can PTZ to follow a track - Slew to Cue.
    /// </summary>
    public interface ISlewToCue : IDevice
    {
        /// <summary>
        /// Raised when the device starts following a track.
        /// </summary>
        event EventHandler<SlewToCueStartedEventArgs> SlewToCueStarted;
        /// <summary>
        /// Raised when the device stops following a track.
        /// </summary>
        event EventHandler<SlewToCueStoppedEventArgs> SlewToCueStopped;
        /// <summary>
        /// Start tracking the object assigned to the given track ID.
        /// </summary>
        /// <param name='trackId'>Track ID</param>
        /// <returns>True if started successfully, False if unsuccessful (for 
example due to invalid Track ID).</returns>
```

```
        bool StartSlewToCue(string trackId);
        /// <summary>
        /// Stops tracking the object which is currently being tracked by the
device.
        /// </summary>
        void StopSlewToCue();
    }
```

ISlewToCue Interface

An interface to support device which can slew to cue - automatically follow a track using PTZ. This has start/stop methods and events raised when the device starts/stops following a track.

```
public interface IRelativeGeoSpatialAwareEvent
    {
        /// <summary>
        /// Azimuth angle in decimal degrees relative to True North
        /// </summary>
        double? Azimuth { get; set; }
        /// <summary>
        /// Elevation angle in decimal degrees relative to the horizontal
plane of the
        /// surface of the earth.
        /// </summary>
        double? Elevation { get; set; }
        /// <summary>
        /// Range in meters.
        /// </summary>
        double? Range { get; set; }
    }
```

IRadar and IGeofence Interfaces

Interfaces to categorise two new device types; Radar devices and another interface to define a standard Geofence object.

```
IRadar
    /// <summary>
    /// Radar Device object type
    /// </summary>
    public interface IRadar : IDevice
    {
    }
```

The GeofenceGeometryType enum type supports Area, Line, and Point geometry types and, as of IPSecurityCenter 5.10, renders the associated GeofencePoints points as such. GeoPosition provides a basic Lat, Long, Alt? (in other words, nullableable) data type for passing WGS84 coordinates.

```
/// </summary>
    /// A Geofence Device object
    /// </summary>
    public interface IGeofence : IDevice
    {
        /// <summary>
        /// The reference identifier for the Geofence
        /// </summary>
        string GeofenceId { [OperationContract] get; [OperationContract] set;
}

        /// <summary>
        /// The geometry type associate with this geofence
        /// </summary>
        GeofenceGeometryType GeofenceGeometryType { [OperationContract] get;
[OperationContract] set; }

        /// <summary>
        /// The points that delimit the boundary of the zone with the last
point being
        /// connected to the first point.
        /// </summary>
        [System.ComponentModel.Browsable(false)]
        List<GeoPosition> GeofencePoints { get; set; }
    }
```

ISDK 3.8

ISDK 3.8 implemented IDeviceRepository Additions. The interface IDeviceRepository has been extended with a new method `ReadStale<TContract>(Guid identifier)`. This allows you to use the VCM (Video Display interface) to get all device properties in a single call to Connection Manager. The trade off is the properties are not live as they would be if you used the regular `Read<TContract>(Guid identifier)` where a proxy to the live instance in Connection Manager is returned.

```
/// <summary>
    /// Interface for repositories used for retrieving device objects.
    /// </summary>
    public interface IDeviceRepository : IDeviceService, IDisposable
    {
        /// <summary>
        /// Gets a device using the identifier for the device.
        /// </summary>
        /// <typeparam name='TContract'>The contract type for the device
service.</typeparam>
        /// <param name='identifier'>The identifier of the device.</param>
        /// <returns>Returns a static version of the device with all
properties pre-cached.</returns>
        [SuppressMessage('Microsoft.Design',
'CA1004:GenericMethodsShouldProvideTypeParameter')]
        TContract ReadStale<TContract>(Guid identifier);
    }
```

ISDK 3.9

DDK 3.9 extended the use of the `x64BitCompatibilityAttribute` attribute to allow the attribute to be set at the assembly level. This must be used to signify that the connector supports running in a 64-bit processes.

NOTE: NOTE: If the `x64BitCompatibilityAttribute` is applied to an assembly, then it is assumed the connector only supports 64-bit unless accompanied by the `x86CompatibilityAttribute` on the assembly. All connectors with no `x64BitCompatibilityAttribute` attribute on the assembly are assumed to have the `x86CompatibilityAttribute` and run in 32-bit processes. This means there is no requirement to update existing connectors. The effect of the `x64BitCompatibilityAttribute` attribute on a driver assembly is that it allows the driver support devices to work in a 64-bit Connection Manager (new to IPSecurityCenter 5.13) and, unless accompanied by the `x86CompatibilityAttribute,` the driver support devices will not work within the standard 32-bit Connection Manager.

x64BitCompatibilityAttribute Attribute

The `x64BitCompatibilityAttribute` has the following classes and assemblies.

- Class - Applied to Video Control/Device class in order for the device to be hosted within a 64-bit VCM. Only required if not applied to the assembly.
- Assembly - Applied to drivers that support running in 64-bit processes (both within VCM and within Connection Manager)

x86CompatibilityAttribute Attribute

The `x86CompatibilityAttibute` attribute has an assembly that is applied to connectors that support running in 32-bit processes (both within VCM and within Connection Manager). It is assumed for all drivers without `x64BitCompatibilityAttribute` attribute on the assembly. It must be added to any driver that has the `x64BitCompatibilityAttribute` attribute on the assembly that still supports running in a 32-bit process, otherwise the driver is assumed to be 64-bit compatible only.

ISDK 3.10

ISDK 3.10 adds the following new features.

Logging Utility

A new logging utility allows you to change logging level on the fly and on a per device basis.

Packaging Connector A With Dependencies Having White Spaces

Added support for packaging connector with dependencies having white spaces in the file name (by escaping spaces).

Common Fire Panel Additions

As part of creating a template to ease fire panel development, new built-in interfaces and enums have been added.

Built-in Interfaces	Properties	Events

`IFirePanelDevice`	PanelId (string)	`AlarmStateChange`
		`ConfigurationChange`
		`OnlineStateChange`

`IFireZoneDevice`	PanelId (string) ZoneId (string)	`AlarmStateChange`
`IFireSwitchDevice`	PanelId (string) ZoneId (string) SwitchId (string)	`SwitchPositionChange`

`IFireLoopDevice`	PanelId (string)
	ZoneId (string)
	DeviceId (string)
	LoopNumber (int)
		`OnlineStateChange`
		`EnabledChange`
		(Used to notify when the device has been disabled on the native system, not IPSecurityCenter)

`IFireOutputTypeDevice`		`FireOutputStateChange`
`IFireInputTypeDevice`		`FireInputStateChange`

Enumerations

The following enumerations are available.

Enumerations	Description
Alarm State	• Normal • InAlarm
DataType	Used to identify what kind of data an input is providing. • Analog • Digital • String
EnabledState	• Disabled • Enabled
OnlineState	• Offline • Online
OutputState	• On • Off
SwitchState	• On • Off

Event Properties

The following event properties are available.

Property	Description

`AlarmStateChange`	Used to notify when a device goes into/out of alarm. Can be raised against a Panel or a Zone device.
	This event implements `IGeoSpatialAwareEvent` and `IGeoSpatialAwareWithAltEvent` interfaces.
	`/// <summary>`
	`/// Current Alarm state of the device`
	`/// </summary>`
	`public AlarmState AlarmState { get; set; }`

`ConfigurationChange`	Used to notify when native configuration has changed.

`EnabledChange`	Used to notify when the enabled state of the device has changed (Device has been enabled natively, not in IPSecurityCenter). This event implements `IGeoSpatialAwareEvent` and `IGeoSpatialAwareWithAltEvent` interfaces.
	`/// <summary>`
	`/// Current enabled state of the device`
	`/// </summary>`
	`public EnabledState EnabledState { get; set; }`

`FireInputStateChange`	Notifies about the change of value in any of the input type devices. This event implements `IGeoSpatialAwareEvent` and `IGeoSpatialAwareWithAltEvent` interfaces.
	`/// <summary>`
	`/// Analog value provided by the device`
	`/// </summary>`
	`public float AnalogValue { get; set; }`
	`/// <summary>`
	`/// Digital value provided by the device`
	`/// </summary>`
	`public int DigitalValue { get; set; }`
	`/// <summary>`
	`/// String value provided by the device`
	`/// </summary>`
	`public string StringValue { get; set; }`
	`/// <summary>`
	`/// Type of data provided by the device`
	`(analog/digital/string)`
	`/// </summary>`
	`public DataType DataType { get; set; }`

FireOutputStateChange	Notifies about a change of value in an output device (Beacon on/off and so on.) This event implements IGeoSpatialAwareEvent and IGeoSpatialAwareWithAltEvent interfaces. /// <summary> /// Current state of the output /// </summary> public OutputState State { get; set; }
OnlineStateChange	Notifies about a change in the online state of a device. This event implements IGeoSpatialAwareEvent and IGeoSpatialAwareWithAltEvent interfaces. /// <summary> /// Raised when a device's online state changes /// </summary> public OnlineState OnlineState { get; set; }
SwitchPositionChange	Notifies about a change in the position of a switch. This event implements IGeoSpatialAwareEvent and IGeoSpatialAwareWithAltEvent /// <summary> /// Raised when a switch reports a changed position. /// </summary> public SwitchState SwitchState { get; set; }

ISDK 3.11

Deprecated

ISDK 3.12

ITrackSourceAwareEvent is implemented. ITrackSourceAwareEvent is an extended version of IGeoSpatialAwareEvent. This new event interface allows a device which does not generate tracks to raise a track-related event and still provide the Track Source Device so the correct track can be put into an alert state if this event causes an alert-state alarm.

```
/// <summary>
 /// Defines an event which is raised in relation to a track on the map, but
 /// is raised by a different device than the track source. This event
provides the
 /// Track Source as well as the Track Identifier so the correct track can be
alerted.
 ///
 /// It is expected that the event populates the geo-spatial properties of
the event
 /// based on the location of the track causing the event.
 ///
 /// An example of this used would be a geo-fence raising an event when a
track has
 /// approached/entered/exited it, and the track may need to be alerted along
with the
```

```
 /// geo-fence itself. The track identifier and the identifier of the radar
generating
 /// the track are included in the event so IPSecurityCenter can correctly
identify the
 /// associated track.
 /// </summary>
 [DesignerVisibleEventInterface]
 [DisplayName(DeviceConstants.ResourcePath,
"DisplayNameITrackSourceAwareEvent", typeof(ITrackSourceAwareEvent))]
 [Description(DeviceConstants.ResourcePath,
"DescriptionITrackSourceAwareEvent", typeof(ITrackSourceAwareEvent))]
 public interface ITrackSourceAwareEvent : IGeoSpatialAwareEvent
 {
     /// <summary>
     /// The identifier of the device which is generating the track
     /// </summary>
     [CategoryGeoSpatial]
     [DisplayName(DeviceConstants.ResourcePath,
"DisplayNameITrackSourceAwareEventTrackSourceDevice",
typeof(ITrackSourceAwareEvent))]
     [Description(DeviceConstants.ResourcePath,
"DescriptionITrackSourceAwareEventTrackSourceDevice",
typeof(ITrackSourceAwareEvent))]
     [DeviceIdentifier(typeof(IDevice))]
     Guid TrackSourceDevice { get; set; }
     /// <summary>
     /// The identifier of the track which created the event
     /// </summary>
     [CategoryGeoSpatial]
     [DisplayName(DeviceConstants.ResourcePath,
"DisplayNameITrackSourceAwareEventTrackId", typeof(ITrackSourceAwareEvent))]
     [Description(DeviceConstants.ResourcePath,
"DescriptionITrackSourceAwareEventTrackId", typeof(ITrackSourceAwareEvent))]
     string TrackId { get; set; }
 }
```

ISDK Event Interfaces

Connector device events may need to implement some standard interfaces. These standard interfaces are defined in the assembly, `CNL.IPSecurityCenter.Driver`.

Implementing an Event Interface in Connector Designer

1. In **Toolbox**, select **Event Interface** shape and drag into the workspace.
2. Click the **Event Interface** shape, select the **Interface Type** from the list.
3. In **Toolbox**, click **Event to Event Interface connector**.
4. Using the connector, connect the desired event to **Event Interface** shape.

The following sections list the current set of known Event interfaces and their expected use.

Event Interface	Description	Properties	Example
`IAccessEvents`	Deprecated - DO NOT USE		
`IDoorEvents`	Deprecated - DO NOT USE		

`IPositionAwareEvent`	Used when the device reports event with Schematic position available: X, Y, Z, Heading, Speed, and Positional Reference Identifier. Typically , applicable for radar-like systems which detect tractable targets.	Some of the properties are optional allowing nullable values as some subsystems may not provide all the coordinates.
		• double? X - optional X coordinate, the valid values are between 0.0 and 1.0
		• double? Y - optional Y coordinate, the valid values are between 0.0 and 1.0
		• double? Z - optional Z coordinate, denotes the floor within a multi-floor building. Z-value is set on a Location, so that trails with a matching z-value are plotted on that location
		• double? Heading - optional direction coordinate, angle relative to the vertical, not used in Control Center yet.
		• double? Speed - value reported by the 3rd party SDK, if this is unknown, Control Center
			Raising driver event in a driver.
			`OnTraceSchematicEvent(new TraceSchematicEventArgs(this)`
			` {`
			`   X = trace.PositionX,`
			`   Y = trace.PositionY,`
			`   PositionalReferenceIdentifier = 1,`
			` });`

		calculates the value automatically based on positions reported previously • int PositionalReferenceIdentifier - represents the coordinate system in use, currently the only valid value is 1	

`ITrackablePositionAwareEvent`	An extended version of `IPositionAwareEvent`
	Used when the device reports event with Schematic position available: X, Y, Z, Heading, Speed, Positional Reference Identifier, and TrackId.
	Typically, applicable for radar-like systems which detect tractable targets.
		Some of the properties are optional and consequently are nullable values as some subsystems may not provide all the coordinates.
		• double? X - optional X coordinate, the valid values are between 0.0 and 1.0
		• double? Y - optional Y coordinate, the valid values are between 0.0 and 1.0
		• double? Z - optional Z coordinate, denotes the floor within a multi-floor building. Z-value is set on a Location, so that trails with a matching z-value are plotted on that location
		• double? Heading - optional direction coordinate, angle relative to the vertical, not used in IPSC yet.
		• double? Speed -
			Raising driver event in a driver.
			`OnTraceSchematicEvent(new TraceSchematicEventArgs(this)`
			`{`
			`TrackId = trace.Id.ToString(),`
			`X = trace.PositionX,`
			`Y = trace.PositionY,`
			`PositionalReferenceIdentifier = 1,`
			`});`

		value reported by the 3rd party SDK, if this is unknown, IPSC will calculate the value automatically based on positions reported previously • int PositionalReferenceIdentifier - represents the coordinate system in use, currently the only valid value is 1 • string TrackId - unique target/track identifier	

`IGeoSpatialAwareEvent`	Used when the device reports event with Geographic position available: Latitude, Longitude, Heading, Speed, and SRID.
	Typically, applicable for radar-like systems which detect tractable targets.
		Some of the properties are optional and consequently nullable as some subsystems may not provide all the coordinates.
		• double? Latitude - optional geographical coordinate, the valid values are between -90.0 and 90.0
		• double? Longitude - optional geographical coordinate, the valid values are between -180.0 and 180.0
		• double? Heading - optional geographic direction coordinate, the angle from the geographic North, the valid values are between 0.0 and
			Raising driver event AlertGeoSpatialAlarm in Geospatial Alert driver.
			`OnAlertGeoSpatialAlarmEvent(new AlertGeoSpatialAlarmEventArgs(this, e.DateTimeOfAlarm)`
			`    {`
			`        SourceSystem = e.SourceSystem,`
			`        SourceId = e.SourceId,`
			`        Latitude = e.Latitude,`
			`        Longitude = e.Longitude,`
			`        Description = e.Description,`
			`        Url = e.Url,`
			`        TypeOfAlarm = e.TypeOfAlarm,`
			`        DeviceId = e.DeviceId,`
			`        SpatialReferenceIdentifier = SpatialReferenceIdentifier`
			`    }));`

		180.0, not used in Control Center yet • double? Speed - value reported by the 3rd party SDK, if this is unknown, Control Center calculates the value automatically based on positions reported previously • int SpatialReferenceIdentifier - use 4326 for longitude/latitude (based on World Geodetic System [WGS84]). A Spatial Reference System Identifier (SRID) is a unique value used to unambiguously identify projected, unprojected, and local spatial coordinate system definitions.	

`ITrackableGeoSpatialAwareEvent`	An extended version of `IGeoSpatialAwareEvent`
	Used when the device reports event with Geographic position available: Latitude, Longitude, Heading, Speed, SRID, and TrackId.
		• double? Latitude - optional geographical coordinate, the valid values are between -90.0 and 90.0
		• double? Longitude - optional geographical coordinate, the valid values are between -180.0 and 180.0
		• double? Heading - optional geographic direction coordinate, the angle from the geographic North, the valid values are
			`OnTraceUpdatedEvent(new TraceUpdatedEventArgs(this)`
			`{`
			`  TrackId = trace.Id.ToString(),`
			`  SpatialReferenceIdentifier = 4326,`
			`  Latitude = trace.PositionLatitude,`
			`  Longitude = trace.PositionLongitude,`
			`});`

	Typically, applicable for radar-like systems which detect distinct target movements, each target is identified by Track ID which is displayed in Control Center maps as an object with a trail path.	between 0.0 and 180.0, not used in IPSC yet • double? Speed - value reported by the 3rd party SDK, if this is unknown, Control Center calculates the value automatically based on positions reported previously • int SpatialReferenceIdentifier - use 4326 (based on World Geodetic System [WGS84]) for longitude/latitude. A Spatial Reference System Identifier (SRID) is a unique value used to unambiguously identify projected, unprojected, and local spatial coordinate system definitions. • string TrackId - unique target/track identifier	

`ITrackSourceAwareEvent`	An extended version of IGeoSpatialAwareEvent
	Used when the device reports event with Geographic position and a source device for the track: Latitude,
		• double? Latitude - optional geographical coordinate, the valid values are between -90.0 and 90.0
		• double? Longitude - optional geographical coordinate, the valid values are between -180.0 and 180.0
		• double? Heading - optional geographic
			`OnEnteringEvent(new EnteringEventArgs(this, track.CurrentPosition.EventTime)`
			`{`
			`TrackId = track.FriendlyTrackId,`
			`Longitude = track.CurrentPosition.Longitude,`
			`Latitude = track.CurrentPosition.Lat`

	Longitude, Heading, Speed, SRID, TrackId and TrackSourceDevice
	Defines an event which is raised in relation to a track on the map but is raised by a different device than the track source. This event provides the Track Source as well as the Track Identifier so the correct track can be alerted. It is expected that the event populates the geo-spatial properties of the event based on the location of the track causing the event. An example of this used would be a geo-fence raising an event when a track has approached/entered/exited it, and the track may need to be alerted along with the geo-fence itself. The track identifier and the identifier of the radar generating the track are included in the event so Control Center can correctly identify
		direction coordinate, the angle from the geographic North, the valid values are between 0.0 and 180.0, not used in IPSC yet
		• double? Speed - value reported by the 3rd party SDK, if this is unknown, IPSC will calculate the value automatically based on positions reported previously
		• int SpatialReferenceIdentifier - use 4326 (based on World Geodetic System [WGS84]) for longitude/latitude. A Spatial Reference System Identifier (SRID) is a unique value used to unambiguously identify projected, unprojected, and local spatial coordinate system definitions.
		• string TrackId - unique target/track identifier
		• Guid TrackSourceDevice - the device identifier of the device raising the track event to which this event is related (the radar-like device)
			`itude,`
			`Heading = track.CurrentPosition.Heading,`
			`SpatialReferenceIdentifier = 4326,`
			`Speed = track.CurrentPosition.Speed,`
			`TrackSourceDevice = track.ReportingSensor`
			`});`

	the associated track.		
ITimebarDisplayAlwaysEvent	When connector event implements ITimebarDisplayAlwaysEvent interface, this event appears on Timebar in Playback mode.		
ITimebarDisplay OptionalEvent	When connector event implements ITimebarDisplayOptionalEvent interface, this event appears on Timebar in Playback mode		

Utility Libraries

The ISDK provides a set of standard utility libraries designed to implement a fixed, tested, implementation of standard repetitive tasks.

The resource set lives

- DDL - `CNL.IPSecurityCenter.Driver.Utility`.
- root namespace - `CNL.IPSecurityCenter.Driver`

The sections below outline the signature of each of these utilities. During a code review, use of these utilities should be checked for and the review failed if they are not used in the appropriate locations.

Driver

`OrderedThreadPool` provides a background thread pool that processes work-units (methods) in a fixed sequence.

Driver.Editors

```
CollectionConverter<t>
 CollectionEditor Dialog<TItem,TCollection>
 CollectionEditor<TItem,TCollection,TValidator>
 CollectionListViewItem<T>
 DisplayIndexAttribute
 IListViewValidator<T>
 InvalidPropertyValueException
 Localization
 ShellFolders
```

Utility.Logging

Provides a wrapper around the Loupe Logging software that allows for dynamic, in application modification of the logging levels provided.
See Logging Utility below for more detail and how to apply the pattern.

Utility.Net

Utility	Description
Address	Static generalized class to pull back the local domain name, device name and address information. **NOTE:** If IPv6 is active that address may be returned.

INetwork Monitor	The expected set of functionalities required by network monitoring.
	`public interface INetworkMonitor : IDisposable`
	`    {`
	`      event EventHandler ConnectionFailed;`
	`      string Address { get; set; }`
	`      int Attempts { get; set; }`
	`      TimeSpan Interval { get; set; }`
	`      bool IsConnected();`
	`      void Start();`
	`      void Stop();`
	`    }`

Network Monitor	An extended abstract class that provides the network monitoring functionality and just requires the user to provide a method: `public abstract bool IsConnected();` which implements how the connection is tested and returns a boolean value `true` when connected, otherwise `false`.
PingMonitor	Although provided within the utility set, Everbridge strongly recommends that such functionality is not used to determine the availability of a sub-system. Most security policies view 'ping' as a security risk and disable it as a device detection method. As a

	consequence, no description of its interface or workings are provided. If any form of 'Ping' monitoring is used in a driver, you MUST provide a boolean property on the driver server to allow for the disablement of that functionality.
Port	Provides a method to find the first available (unused) UPD port within a range of port values, or throw an `InvalidOperationException` exception if none available. `Var PortId = Port,FindRandomFreeUdpPort(int rangeStart, int rangeEnd)` **NOTE:** OS security and network policies may block the returned port rendering it unusable. If using this method, it MUST be stated in the provided RDIN and checks on the security policy for the port range must be made.

Utility.Net.Sockets

A set of utility functions to provide a uniform access method to TCP data streams.

Utility	Description

`DataReceivedChunkByteTerminator`	Defines a default data packet terminator strategy based defining a byte array that indicates the end of a data packet, subsequent data is maintained in the buffer and future data appended to the end of the buffer.
	When initializing an instance of the class the following values should be set:
	`public byte[] Terminator { get; set; }`
	`public int BufferSize { get; set; }`
	Use the:
	• **Terminates** property to define a byte sequence that terminates a message packet
	• **BufferSize** property to indicate a maximum size of byte array to maintain (defaults to 1024 bytes)

`DataReceivedEventArgs`	Event object raised when a terminated data packet has been received
	`[Serializable]`
	`public sealed class DataReceivedEventArgs : EventArgs`
	`{`
	`private byte[] m_data;`

<table>
<tr><td></td><td><pre>
     private Encoding m_encoding;
     public DataReceivedEventArgs(byte[] data, string
source, Encoding encoding)
     {
       this.m_data = data;
       this.m_encoding = encoding;
       this.Source = source;
     }
     public byte[] GetData() => this.m_data;
     public string Source { get; private set; }
     public override string ToString() =>
this.m_encoding.GetString(this.GetData());
</pre></td></tr>
<tr><td>DataReceivedPass
Thru</td><td>An
IDataRecievedChunkStategy
derived class that as the name implies just returns each data packet received without any processing.</td></tr>
<tr><td>DisconnectedEvent
Args</td><td>An event object raised when the system detects a disconnection event from a data source, the event contains any exception information that is associated with the disconnection.</td></tr>
<tr><td>HandleClient
Connection</td><td>A fully defined server listening class for which the user needs to provide, or register to the following:<pre>
public HandleClientConnection(Encoding encoding)
 public event EventHandler<DataReceivedEventArgs>
DataReceived;
 public bool KeepLooping { get; set; }
</pre></td></tr>
<tr><td>IDataReceived
ChunkStrategy</td><td>Defines the interface for providing a incoming data packet received strategy that can be applied to an incoming data stream, before message packet events are raised.<pre>
public HandleClientConnection(Encoding encoding)
 public event EventHandler<DataReceivedEventArgs>
DataReceived;
 public bool KeepLooping { get; set; }
</pre></td></tr>
<tr><td>ITcpClientWrapper</td><td>Definition of the minimum functionality required for a TCP client connection implementation.<pre>
public interface ITcpClientWrapper : IDisposable
   {
     event EventHandler<EventArgs> Connected;
     event EventHandler<DisconnectedEventArgs>
Disconnected;
     event EventHandler<DataReceivedEventArgs>
DataReceived;
     string HostAddress { get; }
     int Port { get; }
     Encoding Encoding { get; }
</pre></td></tr>
</table>

<table>
<tr>
<td></td>
<td>

```
     bool IsConnected { get; }
     void Send(string data);
     void Send(byte[] data);
     void Connect(string hostAddress, int port);
     void Connect(string hostAddress, int port,
Encoding encoding);
     IDataReceivedChunkStrategy
DataReceivedChunkStrategy { get; set; }
     void Disconnect();
   }
```

</td>
</tr>
<tr>
<td>ITcpServer</td>
<td>

Define the standard set of methods and events that should be implemented for a TCP server object.

```
public interface ITcpServer
   {
     event EventHandler<DataReceivedEventArgs>
DataReceived;
     void Start(int port, Encoding encoding);
     void Stop();
     void SendMessage(byte[] message);
     IDataReceivedChunkStrategy
DataReceivedChunkStrategy { get; set; }
     event EventHandler Connected;
     event EventHandler Disconnected;
     event Action<string> ConnectionFailure;
   }
```

</td>
</tr>
<tr>
<td>TcpClientWrapper</td>
<td>

An implementation of the Utility defined interface, if no `IDataReceivedChunkStrategy`

is defined it uses the utility defined default of pass through. Otherwise, the user needs to define the Address/Port/Encoding for the connection and listen to the appropriate events.

</td>
</tr>
<tr>
<td>TcpServer</td>
<td>

Implementation of multi-port listening server. Allows the user to register the following events and process messages as appropriate. If no `IDataReceivedChunkStrategy` is defined it uses the utility defined default of pass through.

```
public event EventHandler Connected;
 public event EventHandler Disconnected;
 public event Action<string> ConnectionFailure;
 public event EventHandler<DataReceivedEventArgs>
DataReceived;
 public IDataReceivedChunkStrategy
DataReceivedChunkStrategy { get; set; }
```

</td>
</tr>
</table>

Utility.OperationScheduler

When an integration has a set of operations or states it runs through this utility provides a scheduling capability to create the list of operations, and run through them monitoring their state until completion.

Utility	Description

Operation	The base class of an operation for the scheduler to execute.
	The user should override the following two methods to provide the functionality to execute:
	`public abstract Operation Clone();`
	`public abstract void Execute();`
	and set the following values, either during object initialisation or by direct manipulation of the properties
	`protected Operation(string id, int timeout)`
	`protected Operation(string id, int timeout, Scenario parentScenario)`
	`public int Timeout { get; set; }`
	`public string Id { get; set; }`
	`public bool Success { get; set; }`
	`public Scenario ParentScenario { get; set; }`

Operator Scheduler	The actual class/object that works through each of the provided operational scenarios to exercise the required functionality.

Scenario	A wrapper class to hold the set of operations and their sequence for implementation.
	`ScenarioEventArgs`
	Event raised when a scenario operation, or complete scenario has completed.
	`public class ScenarioEventArgs : EventArgs`
	`{`
	`public string ScenarioId { get; private set; }`
	`public bool CompletedSuccessfully { get; private set; }`
	`public ScenarioEventArgs(string scenarioId, bool completedSuccessfully)`
	`{`
	`this.ScenarioId = scenarioId;`
	`this.CompletedSuccessfully = completedSuccessfully;`
	`}`
	`}`
	`ScenarioStatus`
	An enumerate returned by the operation Scheduler to indicate current status, the enumerate values are descriptive of their meaning and are not covered separately.

```
public enum ScenarioStatus
    {
      OperationCompleted,
      AbortScenario,
      ScenarioCompleted,
      ScenarioFailed,
      ScenarioTimeout,
      Error,
    }
```

Utility.Patterns

BitArithmeticHelper

A group of operations to allow for the manipulation of byte data.

```
public static class BitArithmeticHelper
   {
     public static void SetNthBit(byte[] array, int bitNo)
     public static void ResetNthBit(byte[] array, int bitNo)
     public static bool CheckNthBit(byte[] array, int bitNo)
     public static bool CheckNthBit(byte b, int bitNo)
     public static string ByteArrayToString(byte[] array)
     public static byte[] CopyArray(byte[] source, int startIndex, int 
length)
     public static string GetBitmapLogString(byte[] bitmap)
     public static byte[] MergeArrays(byte[] array1, byte[] array2)
     public static short ShortFromTwoBytes(byte msb, byte lsb) => 
BitConverter.ToInt16(new byte[2]
   }
```

GenericPool<T>

Depricated – .Net now provides equivalent functionality in the collections library.

Utility.Threading

Everbridge provides a standard timer class and pattern that should be followed. This deals with issues seen when terminating timer operations.

SafeTimer - Class

```
namespace CNL.IPSecurityCenter.Driver.Utility.Threading
 {
     public class SafeTimer : IDisposable
     {
         public event EventHandler Elapsed;
         public event EventHandler<SafeTimerExceptionEventArgs> 
ExceptionOccurred;
         public SafeTimer(bool repeat, int interval, string name){…}
         public string Name { get; private set; }
         public int IntervalMilliseconds { get; set; }
         public bool Repeat{ get…  set… }
         public bool Enabled {get…, set …   }
     }
 }
```

SafeTimerExceptionArgs

An exception class used in an event returned from the timer object that the user can examine if the timer object encounters an unexpected issue.

Video

Some video connectors do not provide the capability to capture video stream frames and save them as a still image on the system.

Everbridge provides a method that allows the user to capture a screen scrape from the VMC display and save that. However, Everbridge recommends that, wherever possible, the third-party supplied methods should be used in preference to this technique.

ImageCapture

This static class provides two methods that return a bitmap image object, based on the region requested, either through a windows Rectangle object or providing the initial x, y position and the width and height of the area to capture. Both methods require the Windows 'Handle' to the display area.

```
namespace CNL.IPSecurityCenter.Driver.Video
 {
   public static class ImageCapture
   {
     public static Image Capture(IntPtr handle, Rectangle region)
     public static Image Capture(IntPtr handle, int x, int y, int width, int
height)
   }
 }
```

Logging Utility

When developing and first deploying a device connector a significant amount of logging information is required to help characterize behavior and identify issues, and subsequently to deployment. If issues are reported this logging helps Everbridge identify the issue. Control Center uses Log4Net to provide that capability.

However, this level of logging information can often overwhelm the logging system and significantly slow the connector when the system is finally deployed.

Log4Net does provide a capability to change the logged information, but this change effects the whole of the Connection Manager Service and requires a configuration file change and restart of the Manager service.

This wrapper and the use of a configuration property on the connector allows for the dynamic control of the logging level on a 'per-device' basis without restart, does not affect other connectors attached to the Manager, nor does it require a restart of the Service. This will allow for PSG/Support to set an increased logging level when initially deploying or when an issue occurs, without changes to the connector code or Service configuration.

The usage of this wrapper is similar to how log4net is currently used, but also provides additional features for use during initial development.

LogLevel

LogLevel is an enum containing 5 levels of logging.

Logging Level	Description
Verbose	not used inside the logger, however, you can use this if you want to expose the data you are sending/receiving inside your communication level. You will have to implement separate checks for this.
Debug	Outputs logs of Debug, Info, Warn, Error and Fatal types
Info	Info will show logs of Info, Warn, Error and Fatal types.
Warn	Warn will show logs of Warn, Error, and Fatal types
Error	Error will show logs of Error and Fatal types

NOTE: Note: Fatal is not part of this enum, as connectors should not exhibit fatal faults – one that would crash the connection manager. Any fault that could be logged as fatal needs to be completely eradicated from the connector code before being released.

LogManagerStore

`LogManagerStore` is a static class that provides instances of DriverLogManager. It has the following methods.

Signature	Description
`DriverLogManager GetLogManager(Guid deviceId)`	Retrieves a `DriverLogManager` instance. The same instance should be used throughout the entire driver. It is recommended to use the server device id for deviceId, and pass the server id to any child device, so that it can retrieve the same instance of `DriverLogManager`.

DriverLogManager

`DriverLogManager` is a class that is used by client applications to request logger instances. It has the following methods.

Signature	Description
DriverLog GetLogger(type)	Retrieves a logger of the desired type. Returns a Log object that can be used in the same way as ILog of log4net.
LoadLog4NetConfig(filePath)	Loads a log4net configuration file to be used while developing the low-level code. Any calls to this method MUST be removed before loading the connector to Control Center, otherwise the driver will not work. This can be used to create local log files while developing, to have full trail of actions performed by the application.

It has the following properties.

Name	**Type**	**Description**
LogLevel	LogLevel	Set this property to the desired logging level for all loggers managed by this DriverLogManager.

DriverLog

Log class implements a log4net interface ILog. It has the same methods, so it can be used as a standard ILog implementation, however it has additional features beyond those of ILog. It has the following properties.

Name	Type	Description
LogLevel	LogLevel	Set this property to the desired logging level for this logger. Note: It is not recommended to change this property on the logger. Instead, set it on DriverLogManager from which the logger instance was retrieved.
Bool	OutputToConsole	Set this property to true if you want to output your logs to console. This is useful if you are starting to develop the low-level code of the driver and not using it in IPSC yet. This will essentially perform Console.WriteLine() but using the methods of ILog interface. After you are done with developing the low-level code, just set this property to false and your logs will be ready to use with the driver.

For other properties and methods please have a look at log4net ILog interface.

Example

CAUTION: For this to work properly, the same instance of `DriverLogManager` must be used across the entire connector. To achieve this, it is suggested to use the Identifier property of the Server device, pass that property to any child device (in a custom constructor) and use it with `LogManagerStore.GetLogManager(Guid deviceId)` to get a `DriverLogManager` instance.

Setting the Logging Level

Create a property on the server with name `LogLevel` of type `CNL.IPSecurityCenter.Driver.Utility.Logging.LogLevel` In the server class, overwrite the original get and set methods of this property as such

NOTE: A private variable is needed. This variable is serialized. Suggested name: `_logLevel):`

```
private LogLevel _loggingLevel;
 public new LogLevel LoggingLevel
 {
     get { return _loggingLevel; }
     set
     {
         if (_loggingLevel != value)
         {
             _loggingLevel = value;
             LogManagerStore.GetLogManager(Identifier).LogLevel = value;
         }
     }
 }
```

This updates the logging level for the specific `DriverLogManager` instance, which in turn updates the level for each logger that belongs to this manager.

Getting a Logger

To get a logger first define a private `DriverLog` variable. Then, inside the device's `InitializeFields()` method assign it `(_log)` as such:

Server Device

```
private void InitializeFields()
 {
     _lockInstance = new object();
     _log =
LogManagerStore.GetLogManager(Identifier).GetLogger(typeof(BriefCamServer));
     LogManagerStore.GetLogManager(Identifier).LogLevel = LoggingLevel;
 }
```

NOTE: Do not forget to set the logging level on the LogManager, otherwise logging level will reset to debug upon every Control Center restart. Only needed on the server device.

All Other Devices

```
private void InitializeFields()
 {
     if (_serverGuid == default(Guid))
     {
         _serverGuid = new Guid();
     }
     _log =
LogManagerStore.GetLogManager(_serverGuid).GetLogger(typeof(BriefCamCamera));
 }
```

NOTE: `InitializeFields()` should be called inside the constructor and inside `OnDeserialization()`

Using the Logger

To log things, use this as a log4net ILog object:

- `_log.Debug(message);`
- `_log.Debug(message, exception);`
- `_log.DebugFormat(formatString, args);`

Other methods from `ILog` such as Info, Warn, Error remain available.

NOTE: Since the Log4Net configuration tool is only a wrapper around log4net, using the methods of log4net for actual logging, if the configuration file for log4net is edited to change the internal logging level, this utility also conforms to those settings. For example, if log4net is set to only display Error level logs, even if the utility is set to Debug level, only Error level messages are logged.

Example Log4Net Configuration File

This file can be used with `LoadLog4NetConfig()` method in `DriverLogManager`.

```
 <?xml version="1.0" encoding="utf-8" ?>
 <configuration>
   <configSections>
     <section name="log4net" 
type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
   </configSections>
   <log4net>
     <appender name="LogFileAppender" 
type="log4net.Appender.RollingFileAppender">
       <param name="File" value="Logger1.log"/>
       <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
       <appendToFile value="true" />
       <rollingStyle value="Size" />
       <maxSizeRollBackups value="2" />
       <maximumFileSize value="10MB" />
       <staticLogFileName value="false" />
       <layout type="log4net.Layout.PatternLayout">
         <param name="ConVersionPattern" value="%d [%t] %-5p %c %m%n"/>
       </layout>
     </appender>
     <root>
       <level value="ALL" />
       <appender-ref ref="LogFileAppender" />
     </root>
   </log4net>
   <startup>
         <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
   </startup>
 </configuration>
```