SENSTAR™

Senstar Symphony SDK

7.6.x

Developer Guide

Contents

Introduction....3
Quick start....3
Requirements and recommendations....3
Portability....3
Server API overview....4
Sample code....5
Managed libraries....9
Media-streaming libraries....10
Persistence....11
Add streaming video....12
Enable panel switching....13
Get an image....14
Decoration options....14
Activate a relay....15
Enable a camera tour....16
Wrap an OCX in a DLL....17
Integrate a third-party video analytic....21
Legal information....25

Introduction

The Symphony SDK provides sample code that helps you to build applications to extend the functionality of and integrate third-party products with the Symphony Server.

The Symphony Client uses these same methods to ensure quality and feature completeness. Sample applications are included for a wide variety of use cases. Reference documentation is provided in `<SDK>/Documentation/readme.html`.

Note: You might need to restart Symphony services before a Symphony SDK application works correctly.

Quick start

Running a Symphony SDK application

1. Create an Senstar Xnet account.
2. Download and install the Symphony Server and Symphony Client.
3. Use a Web browser to access the Symphony Server configuration interface and add a camera to the Symphony Server.
4. Download and install the Symphony SDK. If you install the Symphony SDK on a computer without the Symphony Server or Symphony Client, open a command prompt in the Symphony SDK bin folder (`C:\Program Files (x86)\Senstar\Symphony SDK\bin` by default) as an administrator and run the following command:

   ```
   for %x in (*.dll *.ax *.ocx) do regsvr32 /s %x
   ```

5. Run `LiveStreamTest.exe` in the Symphony SDK to see live video from the Symphony Server.

Compiling a Symphony SDK application

1. Open `SDK.sln` in Microsoft Visual Studio.
2. Right-click **LiveStreamTest** in the Solution Explorer and click **Set as Startup Project**.
3. Press F5.

Requirements and recommendations

- The managed interfaces must be from executables that target .NET 4.5.2 or later.
- The recommended IDE is Microsoft Visual Studio 2012 or later.
- The recommended programming language is C#.
- The developer should be familiar with the Symphony Server.
- It is recommended that you install the Symphony Server and the Symphony SDK on different computers.

Portability

It is possible to integrate live video from the Symphony Server directly to non-Windows platforms using GStreamer and the RESTful Symphony Mobile Bridge interface.

The portable Symphony Mobile Bridge interface is not as full featured as the Windows accessible SeerWS interface. If you need functionality in SeerWS from non Windows platforms, then you must develop your own layer that provides this interface. Ported implementations of .NET will work (such as Mono), but the referenced .NET libraries link to Windows style DLLs that require deeper platform equivalence.

Server API overview

Sample code

Symphony Server farm connectivity

Use cases	SDK projects
Connect to a Symphony Server farm	NotificationMonitorExample FarmTest

Video

Use cases	SDK projects
Get a list of cameras	LiveStreamTest FarmTest
Connect and render live video	LiveStreamTest
Connect and render historical video	LiveStreamTest
Get a historical JPG	AlarmHandlerExample
Export video	ExportVideo

PTZ cameras

Use cases	SDK projects
Control PTZ on a video device	LiveStreamTest FarmTest
Load guard tour locations	LiveStreamTest FarmTest
Go to a specific guard tour location	LiveStreamTest FarmTest

Events

Use cases	SDK projects
Trigger an alarm	AddAlarmToCamera FarmTest
Listen for alarms	AlarmHandlerExample MobileBridge

Use cases	SDK projects
Get a list of alarms	LiveStreamTest
Mark an alarm	LiveStreamTest
Listen for farm events	NotificationMonitorExample NotificationMonitors MobileBridge

Farm settings

Use cases	SDK projects
Retrieve farm settings	WebService
Execute a web service method	WebServiceConsole

Recording

Use cases	SDK projects
Start/stop recording	LiveStreamTest FarmTest

Symphony Client control

Use cases	SDK projects
Switch cameras in the Symphony Client interface using the Symphony Server API	VideoWall FarmTest
Switch cameras in Symphony Client interface using the Symphony Client API	VideoWallClient

Reports

Use cases	SDK projects
Get a heatmap	GetDensityImageJpg

Security

Use cases	SDK projects
Change the currently active security profile	SetActiveProfile
Add/remove users	UserManagement

Video analytics

Use cases	SDK projects
Get live XML metadata from a running analytics engine	XMLStream
Analyze video in Symphony from a third-party video system	AlgorithmIntegration
Load a Symphony UI component in a third-party application	PluginHost
Incorporate your own analytics engine	AlgoSample
Incorporate your own rules engine	AlgoSample TestRuleEngine
Incorporate your own analytics engine configuration	AlgoSampleConfiguration
Incorporate your own rules engine configuration	AlgoSampleConfiguration

Timeline

Use cases	SDK projects
Get timeline information	TimelineGetter

Navigation

Use cases	SDK projects
Use navigation buttons to control video	FarmTest

DIO rules

Use cases	SDK projects
Import/export DIO rules from/to a CSV file	AddDIORulesFromCSV ExportDIORulesToCSV

Hardware devices

Use cases	SDK projects
Get a list of all hardware devices	PSA
Add a new hardware device	PSA

Camera usage

Use cases	SDK projects
View camera-use information	ClientBandwidth

Direct file access (persistence)

Use cases	SDK projects
Show detailed footage file frame information	PrintFootageFile
Show detailed timeline file information	PrintSignalsFile
Show detailed metadata file information	PrintSignals2File
Show detailed export file information	PrintAiraFile

Managed libraries

The Symphony SDK uses the following managed libraries:

Component	Description
`Seer.Connectivity.dll`	• Library for connecting to the Symphony Server (usually used indirectly by samples)
`Seer.BaseLibCS.dll`	• `Seer.BaseLibCS.Communication.ServerConnectionManager` for client-server connectivity (usually used indirectly by samples) • `Seer.BaseLibCS.AiraWS.Signals` for Web service interface • `Seer.BaseLibCS.CameraMessageStruct` for structure definitions • `Seer.BaseLibCS.Utils.MulticastMessage` for constants
`Core.dll`	• Source part of the Symphony SDK • `Seer.SDK.SDKFarm` for farm connectivity and event stream notifications
`Seer.Farm.dll`	• `Seer.Farm.DeviceModel.Client` for client-side device operations and information • `Seer.Farm.DeviceModel.Server` for server-side device operations and information
`Internationalization.dll`	• `Seer.Internationalization.Translator` for language translation into all supported languages
`NotificationMonitors.dll`	• Source part of the Symphony SDK • Interface to the event stream that comes from the Symphony Server

The following managed libraries are dependencies of the above libraries and likely do not need to be used by your code directly. However, you might need to add them as references to your project. If they do need to be added, the compiler will fail with a message stating what reference needs to be added.

- `Seer.Common.dll`
- `DeviceModel.Dio.dll`
- `DeviceModel.Security.dll`
- `Farm.Security.dll`
- `SecurityLib.dll`

The following managed libraries enable access to video streaming.

- `AxVideoRecvCtrl.dll` wraps `VideoRecvCtrl.dll`
- `BaseIDL.dll` is a set of COM interfaces that aid in streaming video

Media-streaming libraries

Component	Description
`AxVideoRecvCtrl`	• ActiveX control that supports the Dispatch interface • Can be used by simple applications such as VB or Web-based applications
`IVideoRecvCtrl`	• A COM interface exposed by `AxVideoRecvCtrl` • Supports richer API when combined with `INetworkEndpoint` and `IHistoricalSeek` • This interface is demonstrated by the LiveStreamTest sample • This is the recommended interface
`OCXDll.dll`	• Library that allows users ti dynamically create, use, and destroy `AxVideoRecvCtrl`
`OCXExe.exe`	• An executable that shows how to use `OCXDll.dll` • Useful if you need to support multiple NVRs and need a library with a common API for each NVR

Persistence

Files

Files	Description
_FootageArchive (`*.dat`)	• Video • Audio • Overlays (decorations)
_Signals (`*.dat`)	• Timeline data
_Signals2 (`*.dat`)	• Metadata (usually from video analytics)
Exported movie files (`*.aira`)	• Video • Audio • Overlays (decorations)

Database

Data	Description
Settings	• Most settings (except users, groups, and servers)
Servers	• All servers and server states in a farm
Alarms	• All alarms
Senstar Xnet data	• License information
ObjectCounts	• Count of objects per minute (video analytics)
LineCounts	• Count of line crossings per minute (video analytics)

You can use `Dbupdater.exe` from the command line to run SQL against the database.

```
dbupdater "select top 10 * from settings order by sequenceid desc"
```

Shows the last ten settings that have changed.

```
dbupdater "select * from servers"
```

Shows all server addresses and states in the farm.

```
dbupdater "select count(*) from alarms"
```

Shows the current count of all alarms.

All database times are UTC encoded as the number of seconds (or milliseconds) since 1970. Run `dbconfiguration.exe` to view or change the current database connection string.

Add streaming video

You can use the Symphony SDK to add streaming video to a project.

1. Open the design view of the form to which you want to add the control.
2. From the **Tools** menu, click **Add/Remove Toolbox Items**.
3. Click the **COM Components** tab.
4. Select the **VideoRecvCtrl Control** box.
5. Click **OK**.
   The ActiveX control appears in the toolbox in the **Windows Forms** section.
6. Drag the ActiveX control onto the design view.

Enable panel switching

You can use the Symphony SDK to enable panel switching in the Symphony Client interface.

1. Perform one of the following tasks:
   - To switch video panels using the Symphony Server, use the FarmTest or VideoWall application.
   - To switch video panels using the Symphony Client, use the VideoWallClient application.
2. Ensure that the user has Video Wall Change Panel permissions.
3. To register the Symphony Client in which the video panel switch will occur as a video wall client, complete the following steps:
   a) In the Symphony Client interface, click **Settings** > **Video Wall**.
   b) Click the **Video Wall Client Configuration** tab.
   c) Click **Register current Symphony Client**.
4. To get the panel name, complete the following steps:
   a) In the Symphony Client interface, right-click the panel and click **Settings**.
   b) Click the **Child** tab to display the panel name.

      If you use the VideoWall application in the command prompt and the panel name includes a space, surround the panel name with double quotes (e.g., "*panel name*").
5. Use one of the following IP addresses:
   - In the FarmTest and VideoWall applications, use the IP address of the Symphony Server.
   - In the VideoWallClient application, use the IP address of the Symphony Client.

Get an image

You can use the Symphony SDK to get a JPEG image from a project.

1. Use the `GetJPEG` command to generate a JPEG image and return its URL.
2. Use the `GetJPEGImage` command to get the byte array of a JPEG image for a specific time and camera.
3. Use the `GetJPEGImage3` command to extend the `GetJPEGImage` command and add parameters for the decorations.
4. Use the `GetJPEGImage4` command to extend the `GetJPEGImage3` command and add parameters for the font used on the image.

Decoration options

The following parameters determine the decorations that JPEG images include.

```
DecorationOptions options = new DecorationOptions(15, 0, true, 0, true);
// Pass decorations. EncodedDecorationOptions to GetJPEG* functions

// You will want to use the following DecorationOptions constructor
public DecorationOptions(int decorations, int dateFormat, bool in24HourNotation, uint
 streamIndex, bool canViewPrivateVideo)
{
    // BITWISE OR together the following to select decoration
    // 1 = rectangles.
    // 2 = messages.
    // 4 = time.
    // 8 = paths.
    _rawDecoration = decorations;

    // SET to one of the following
    // 0 = yyyy/MM/dd
    // 1 = dd/MM/yyyy
    // 2 = MM/dd/yyyy
    _dateFormat = dateFormat;

    // True for 24 hour time format. False for 12 hour time format.
    _in24HourNotation = in24HourNotation;

    // 0 based stream number. The Symphony Client UI is 1 based.
    _streamIndex = streamIndex;
}
```

Activate a relay

To activate or deactivate a relay, call the following Web service method:

```
public void PerformAction(string username, string password, string sActionReq)
```

To activate a relay, `sActionReq` uses the following format:

```
<action><ServerIP>10.234.8.30</ServerIP><camera><On Relay='1' DeviceID='2'/></camera></action>
```

To deactivate a relay, `sActionReq` uses the following format:

```
<action><ServerIP>10.234.8.30</ServerIP><camera><Off Relay='1' DeviceID='2'/></camera></action>
```

Enable a camera tour

To enable or disable a camera tour, use the following SQL command:

```
declare @xml xml, @cameraId int, @tourName varchar(250), @disabled int;
set @cameraId = 3;
set @tourName = 'Camera Tour 2';
set @disabled = 1;
select @xml = CAST(v as xml) from settings where Type = 'Camera' and Section = 'Camera' and K =
 'cameraTour' and ID = @cameraId;
set @xml.modify('replace value of (/TourGroup/cameraTour[@name=sql:variable(''@tourName'')]/
disable/text())[1] with sql:variable(''@disabled'')');
update Settings set V = CAST(@xml as nvarchar(max)) where Type = 'Camera' and Section = 'Camera'
 and K = 'cameraTour' and ID = @cameraId;
```

To run a batch file on the Symphony Server, pass the following command to the DB Updater utility:

```
dbupdater "declare @xml xml, @cameraId int, @tourName varchar(250), @disabled int;
set @cameraId = 3;
set @tourName = 'Camera Tour 2';
set @disabled = 1;
select @xml = CAST(v as xml) from settings where Type = 'Camera' and Section = 'Camera' and K =
 'cameraTour' and ID = @cameraId;
set @xml.modify('replace value of (/TourGroup/cameraTour[@name=sql:variable(''@tourName'')]/
disable/text())[1] with sql:variable(''@disabled'')');
update Settings set V = CAST(@xml as nvarchar(max)) where Type = 'Camera' and Section = 'Camera'
 and K = 'cameraTour' and ID = @cameraId;"
```

Wrap an OCX in a DLL

1. Create an MFC DLL project:
   a) Click **File** > **New** > **Project**.
   b) Expand **Visual C++** and click **MFC**.
   c) Click **MFC DLL**.
   d) Type `OCXDLL` in the **Name** field and click **OK**.
   e) Click **Finish**.
2. Add a dialog to the DLL:
   a) In the **Resource View** window, right-click **OCXDLL**.
   b) Click **Add** > **Resource**.
   c) Select **Dialog** and click **New**.
   d) In the dialog, delete the **OK** and **Cancel** buttons.
3. Change the default properties of the dialog:
   a) Right-click the dialog.
   b) Click **Properties**.
   c) Change the border to none.
   d) Change **No Fail Create** to **True**.
   e) Change **Style** to **True**.
4. Add a dialog class:
   a) Right-click the dialog.
   b) Click **Add class**.
   c) Type `OCXDLLDialog.h` in the **Name** field and click **Finish**.
   d) Add `#include "resource.h"` in `OCXDLLDialog.h`.
5. Add the OCX control:
   a) Select the dialog tab.
   b) In the **Toolbox** window, right-click **Choose Toolbox Items**.
   c) Click **COM Components** > **VideoRecvCtrl Control**.
   The file is `VideoR~2.OCX` and is located in the `aira\bin` directory.
   d) Drag the control on to the dialog and re-size it to fill the space.
6. Add a variable to the OCX:
   a) Right-click the **VideoRecvCtrl** control.
   b) Click **Add Variable**.
   c) Change **Access** to protected.
   d) In the **Variable Name** field, type `m_OCX`.
   e) Click **Finish**.
7. Add play and stop functions:
   a) In the `OCXDLLDialog.h` file, add the following as public declarations:

```
HRESULT Play(unsigned long IP, unsigned short port, LPCTSTR user, LPCTSTR pass,
 __int64 historicalTime);
HRESULT Stop(void);
```

   b) In the `OCXDLLDialog.cpp` file, add the following methods:

```
HRESULT OCXDLLDialog::Play(unsigned long IP, unsigned short port, LPCTSTR user,
 LPCTSTR pass, __int64 historicalTime)
{
    HRESULT hr;
    if (FAILED(hr = m_OCX.Init()))
        return hr;
    if (FAILED(hr = m_OCX.Connect(IP, port, user, pass, historicalTime)))
        return hr;
```

```
    if (FAILED(hr = m_OCX.StartVideo()))
        return hr;
    return S_OK;
}

HRESULT OCXDLLDialog::Stop(void)
{
    HRESULT hr;
    if (FAILED(hr = m_OCX.StopVideo()))
        return hr;
    if (FAILED(hr = m_OCX.Destroy()))
        return hr;
    return S_OK;
}
```

8. In the `OCXDLL.cpp` file, add a call to AfxEnableControlContainer() in InitInstance().
9. Create an interface for the VideoCreate, VideoDestroy, VideoPlay, and VideoStop functions.
   a) In the **Solution Explorer**, right-click **Header Files**.
   b) Click **Add** > **New Item**.
   c) Add a new header file called `OCXDLLInterface`.
   d) Copy the following code into the `OCXDLLInterface` header file:

```
#pragma once
#ifdef OCXDLL_EXPORTS
#define OCXDLL_API __declspec(dllexport)
#else
#define OCXDLL_API __declspec(dllimport)
#endif
#include "OCXDLLDialog.h"
OCXDLL_API OCXDLLDialog*VideoCreate (CWnd *parent);
OCXDLL_API HRESULT VideoDestroy(OCXDLLDialog *wnd);
OCXDLL_API HRESULT VideoPlay (OCXDLLDialog *wnd, unsigned long IP, unsigned short
 port, LPCTSTR user, LPCTSTR pass, __int64 historicalTime);
OCXDLL_API HRESULT VideoStop (OCXDLLDialog *wnd);
```

   e) In the **Solution Explorer**, right-click **Source Files**.
   f) Click **Add** > **New Item**.
   g) Add a C++ file called `OCXDLLInterface`.
   h) Copy the following code into the `OCXDLLInterface` C++ file:

```
#include "stdafx.h"
#include "OCXDLLInterface.h"
#include "OCXDLLDialog.h"

OCXDLL_API OCXDLLDialog*VideoCreate (CWnd *parent) {
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    OCXDLLDialog *wnd = new OCXDLLDialog();
    wnd->Create(OCXDLLDialog::IDD, parent);
    return wnd;
}

OCXDLL_API HRESULT VideoDestroy(OCXDLLDialog *wnd) {
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    wnd->DestroyWindow();
    delete wnd;
    return S_OK;
}

OCXDLL_API HRESULT VideoPlay (OCXDLLDialog *wnd, unsigned long IP, unsigned short
 port, LPCTSTR user, LPCTSTR pass, __int64 historicalTime) {
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    return wnd->Play(IP, port, user, pass, historicalTime);
}

OCXDLL_API HRESULT VideoStop (OCXDLLDialog *wnd) {
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    return wnd->Stop();
}
```

10. Add **OCXDLL_EXPORTS** to the list of preprocessor definitions for the project:
    a) Right-click the project.
    b) Click **Properties**.
    c) Expand **C/C++** and click **Preprocessor**.
    d) Add `OCXDLL_EXPORTS` to the preprocessor definitions.
11. Add a dialog-based MFC application project:
    a) Click **File** > **Add** > **New Project**.
    b) Expand **Visual C++** and click **MFC**.
    c) Click **MFC Application**.
    d) Type `OCXEXE` and click **OK.**
    e) Click **Next**.
    f) Change the application type to **Dialog based** and click **Finish**.
12. Add buttons:
    a) In the **Resource View** window, expand **OCXEXE** > **OCXEXE.rc** > **Dialog**.
    b) Double-click **IDD_OCXEXE_DIALOG**.
    c) Delete the following items:
        - **OK** button
        - **Cancel** button
        - **TODO: Place dialog controls here** text
    d) Increase the width of the dialog windows to fit two OCX controls.
    e) Using the toolbox, drag three new buttons to the dialog.
    f) Change the following properties for the new buttons:

```
Caption: Add Video
ID: IDB_ADD
```

```
Caption: Start
ID: IDB_START
```

```
Caption: Remove Video
ID: IDB_REMOVE
```

13. Add a variable to the dialog:
    a) Right-click the dialog.
    b) Click **Add Variable**.
    c) Select the **Control variable** box.
    d) In the **Variable Name** field, type `m_btnAdd`.
    e) Verify that the control ID is **IDB_ADD**.
    f) Click **Finish**.
14. To add the OnBNClickedAdd, OnBNClickedRemove, and OnBnClickedStart methods to the `OCXEXEDlg.cpp` file, double-click each of the new buttons.
15. In the `OCXEXEDlg.h` file:

    Add the required header files and std namespace:

```
#include <vector>
#include "..\OCXDLL\OCXDLLInterface.h"
using namespace std;
```

    Add the following protected member:

```
vector<OCXDLLDialog*> m_windows;
```

16. In the `OCXEXEDlg.cpp` file:

    Modify the OnBnClickedAdd, OnBnClickedRemove, and OnBnClickedStart methods as follows:

```
void COCXEXEDlg::OnBnClickedAdd()
{
    OCXDLLDialog *dlg = VideoCreate(NULL);

    dlg->SetParent(this);
    dlg->ShowWindow(SW_SHOW);
    dlg->MoveWindow((m_windows.size() & 1) * 250, (m_windows.size() & 2) * 50, 250, 100);
    m_windows.push_back(dlg);
}

void COCXEXEDlg::OnBnClickedRemove()
{
    if (m_windows.size() > 0) {
        OCXDLLDialog *dlg = m_windows.back();
        m_windows.pop_back();
        dlg->SetParent(NULL);
        VideoDestroy(dlg);
    }
}

void COCXEXEDlg::OnBnClickedStart()
{
    if (m_windows.size() > 0) {
        // This will connect to live video on localhost (127.0.0.1), camera #1, using user
 “admin” and password “admin”
        VideoPlay(m_windows.back(), 0x7F000001, 50010, _T("admin"), _T("A1crUF4="), 0);
    }
}
```

17. In the `OCXEXE.cpp` file, call `CoInitialize(NULL)` after `AfxEnableControlContainer()` in `InitInstance()`.
18. In the Solution Explorer:
    a) Right-click the **OCXEXE** project.
    b) Click **Project Dependencies**.
    c) Select **OCXDLL** as a dependency.
    d) Right-click the **OCXEXE** project.
    e) Click **Set as StartUp Project**.

    To connect to the camera, you might need to modify the camera details for the **OnBnClickedStart** method in the `OCXEXEDlg.cpp` file.

Integrate a third-party video analytic

You can use sample projects in the Symphony SDK to integrate a third-party video analytic with the Symphony Server.

Every camera connected to the Symphony Server has its own AI Tracker service (`Trackerapp.exe`). The video from the camera is decompressed and passed into AI Tracker filter (`Tracker.ax`), which contains the analytic and rule engines. The analytic engine processes the video and produces XML metadata that is passed to the rules engine. The rules engine processes the metadata and creates alarms in the Alarm database table.

When a user configures an analytic or rule in the Symphony Server configuration interface, the Symphony Server saves the configurations in the database. The XML strings are used to initialize the analytic and rule engines when the AI Tracker service starts. The following is an example of the XML:

```
<State><DisplayText>% has been changed</DisplayText><DisplayPercentage>1</DisplayPercentage></
State>
```

In addition to the XML strings, there are some properties used to describe the analytic to the Symphony Server. The following fields are of special note:

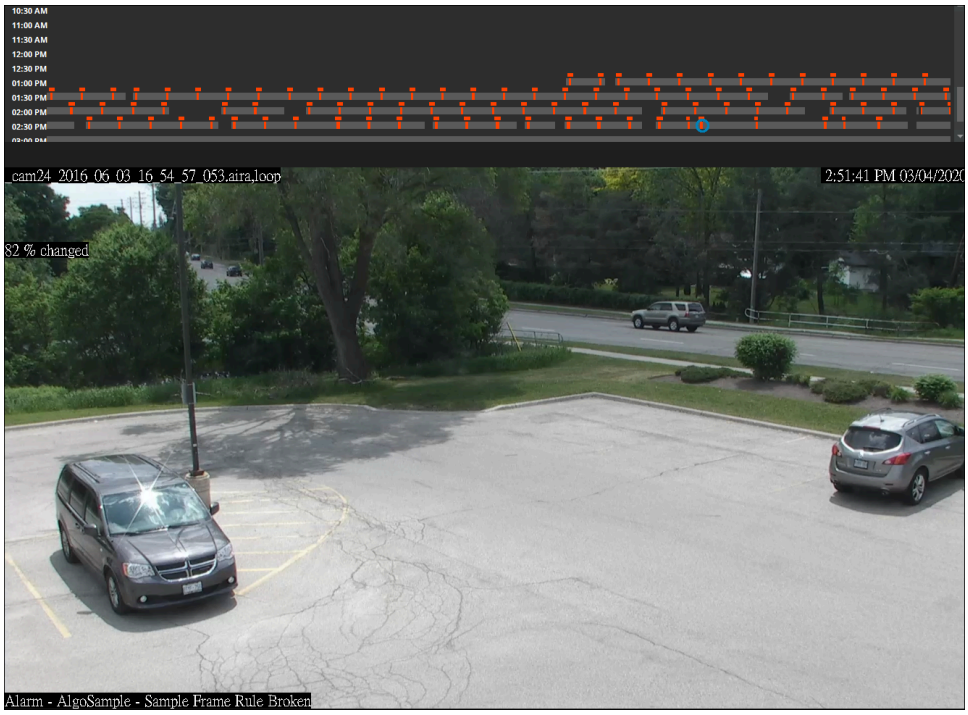
- Name: the name of the analytic displayed in the Symphony Server configuration interface
- Description: the description of the analytic displayed in Symphony Server configuration interface
- Configurable: whether the analytic has Web interface for configuration
- ConfigurableForRules: whether the analytic has a Web interface for rule configuration
- SearchSupported: flag to enable/disable searching by the metadata created by the analytic

There are two projects in the Symphony SDK that provide a sample video analytic that you can integrate with the Symphony Server:

- `AlgoSample`: the video analytic and rule engine project
- `ConfigureWeb_AlgoSample`: the video analytic and rule configuration project

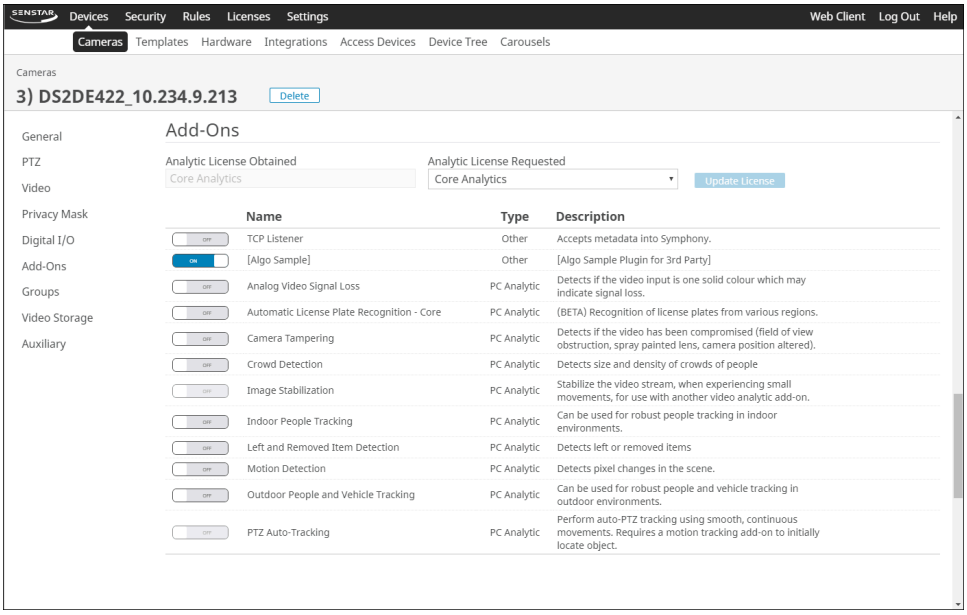
If you change the names of the sample projects, the name of the configuration project must start with `ConfigureWeb_` for the Symphony Server to recognize the project.

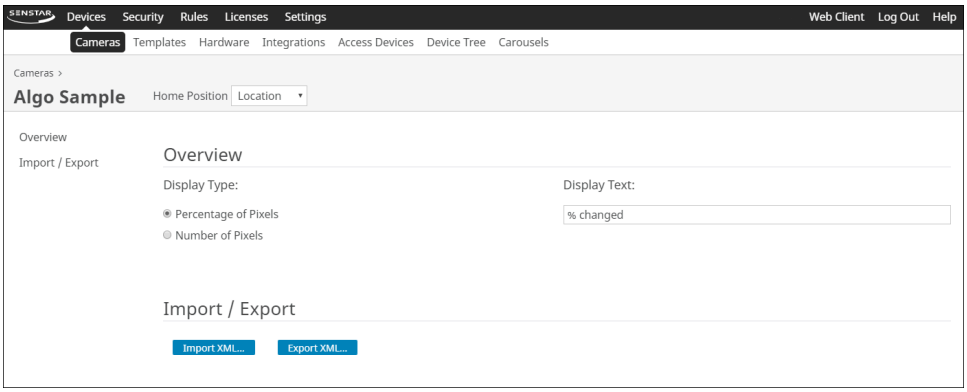
The `AlgoSample` video analytic counts the number of frames that change and displays either the count or the percentage as a decoration.



1. Download the newest Symphony SDK.
2. On the computer that hosts the Symphony Server, stop all of the Symphony services (`killall 9`).
3. Copy the `AlgoSample.*` and `ConfigureWeb_AlgoSample.*` files from the `SDK\bin\algos` folder to the `...\Senstar\Symphony SDK\bin\algos\` folder on the computer that hosts the Symphony Server.
4. On the computer that hosts the Symphony Server, start all of the Symphony services (`killall 5`).

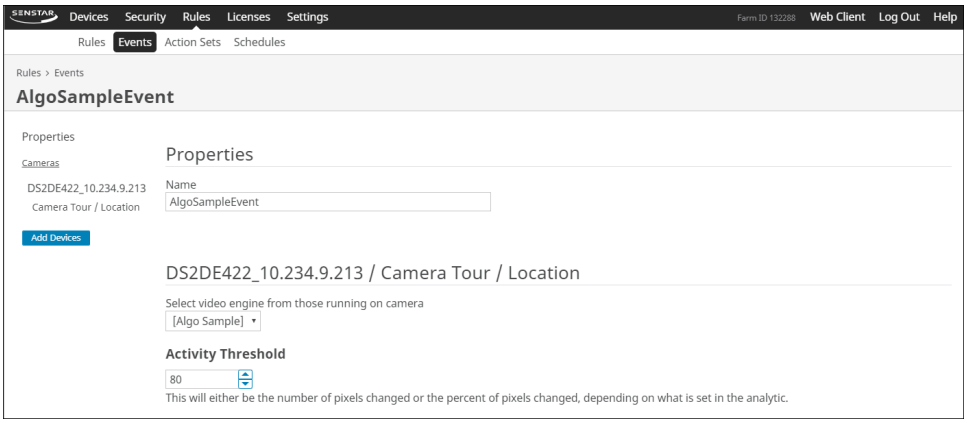
5. In the Symphony Server configuration interface, configure the a camera to use the `AlgoSample` project as a video analytic.





6. Test the `AlgoSample` video analytic.
7. In the Symphony Server configuration interface, create an event for the `AlgoSample` video analytic that triggers a rule.

   The `AlgoSample` video analytic should trigger an alarm when the number or percentage goes above a threshold that you set in the sample code.



8. Modify and compile the `AlgoSample` and `ConfigureWeb_AlgoSample` projects to integrate the third-party video analytic.

   You must compile the `AlgoSample` and `ConfigureWeb_AlgoSample` projects in Release mode for the `AlgoSample` video analytic to work with the Symphony Server.
9. Repeat steps 2 to 5 for the new `AlgoSample` and `ConfigureWeb_AlgoSample` projects.

Legal information

This material is for informational purposes only. Senstar makes no warranties, express, implied or statutory, as to the information in this document.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Senstar Corporation

Senstar may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Senstar, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Senstar and the Senstar logo are registered trademarks of Senstar Corporation.

All other trademarks are the property of their respective owners.