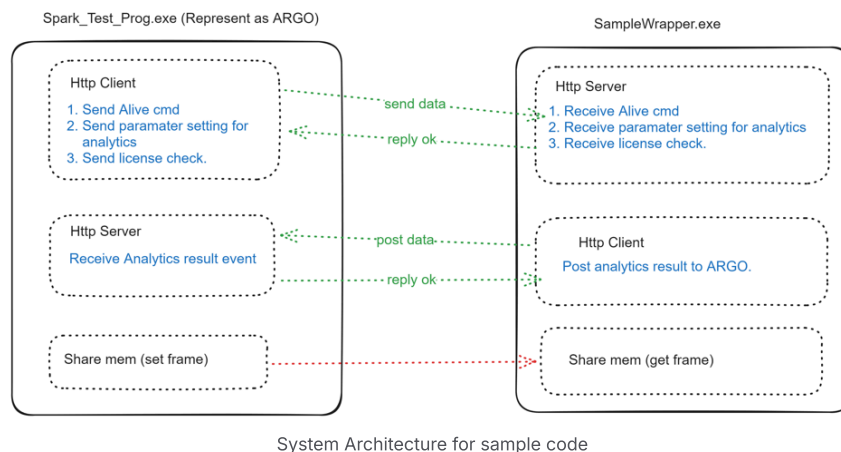


ARGO Generic AI Integration Sample Code Guide Version 1.2

1. System Architecture
 - 1.1 Spark_Test_Prog
 - 1.2 SampleWrapper
 2. ARGO AI Integration Specifications
 - 2.1. Communication APIs
 - 2.2. Suggested Detailed Settings (JSON)
 - 2.3. Shared Memory Settings
 - 2.4. Execution and Naming Conventions
 3. ARGO AI Integration UI on Client and Config Client
 - 3.1. Add Generic AI Detection on Spark AI Devices
 - 3.2. Analyticsstreamperimeter on Spark AI Device - Generic AI Detection
 4. License

1. System Architecture



System Architecture for sample code

1.1 Spark_Test_Prog

1.1.1 What is Spark_Test_Prog

Spark_Test_Prog is a program designed to simulate Argo behavior. It encompasses the following key features:

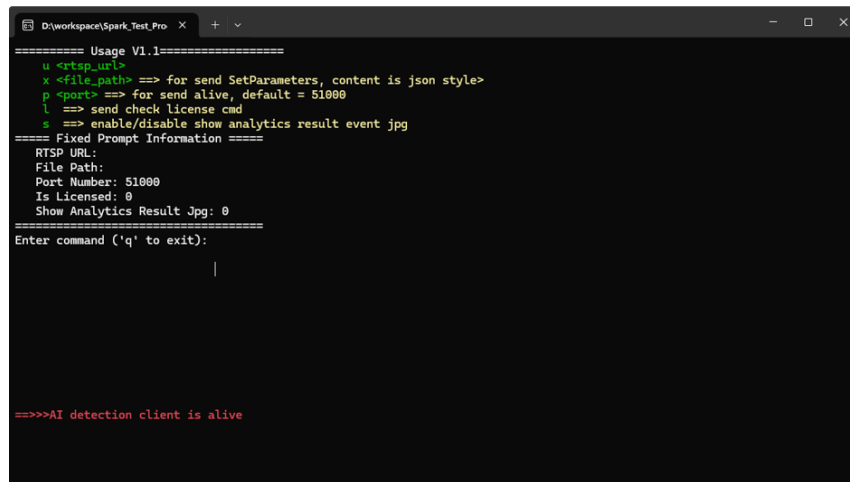
1. Capability to receive RTSP (Real-Time Streaming Protocol) streams
2. Functionality to display video frames
3. All corresponding functions necessary for integrating Generic AI

This program serves as a valuable tool for developers looking to create their own Generic AI Wrapper. It facilitates AI integration with Argo by providing a testing environment that mimics Argo's behavior and interactions.

1.1.2 Primary Use Case

Developers can utilize Spark_Test_Prog to assist in the development and testing of Generic AI Wrappers intended for integration with Argo. This allows for a more streamlined development process and helps ensure compatibility before final implementation.

1.1.3 Interface



```
===== Usage V1.1=====
u <rtsp_url> ==> for send SetParameters, content is json style>
x <file_path> ==> for send alive, default = 51000
p <port> ==> for send alive, default = 51000
l ==> send check license cmd
s ==> enable/disable show analytics result event jpg
===== Fixed Prompt Information =====
RTSP URL:
File Path:
Port Number: 51000
Is Licensed: 0
Show Analytics Result Jpg: 0
=====
Enter command ('q' to exit):

|

====>>>AI detection client is alive
```

1.1.4 Spark_Test_Prog Command Instructions

RTSP URL Setting

Command: `u <rtsp_url>` Example: `u rtsp://admin:admin@192.168.1.219/stream1`

Sets the RTSP URL. After setting, it will automatically connect, decode, and send to shared memory. **Currently supports decoding up to 1920x1080.**

SetParameters JSON File

Command: `x <file_path>` Example: `x D:\param.json`

Sets the JSON content for SetParameters, used to configure the recognition algorithm.

Port Number Setting

Command: `p <port_num>` Example: `p 51000`

Sets the port for the corresponding **SampleWrapper** to activate. In Argo, each channel corresponds to a **SampleWrapper** for executing recognition. Current setting: Port number suffix is the channel number.

Example: To activate recognition for the 7th channel, set to 51007. The **SampleWrapper** will also be launched using the corresponding port.

License Check (optional function)

Command: `l`

Sends a command to **SampleWrapper** to check if the license exists.

Toggle Analytics Result Event Image Display

Command: `s`

Toggles the display of JPG image files in the analytics result event on or off.

Note: In Argo, each channel will open a corresponding **SampleWrapper** to perform recognition.

1.1.5 Spark_Test_Prog Operation Guide

Setup Procedure

1. Set the port number
2. Configure the RTSP URL

3. Set the JSON file for SetParameter

Important Notes

- It's crucial to set the channel number and port number before configuring the RTSP URL.
- After startup, the program will send an "Alive" message to **SampleWrapper** every 30 seconds to confirm its existence.
- All setup messages and results will be displayed in the lower half of the window.

Operation Flow

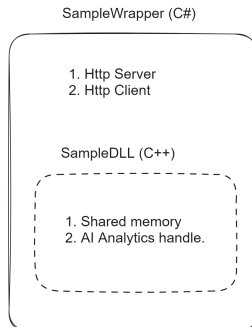
1. Start Spark_Test_Prog
2. Use the **p** command to set the port number
3. Use the **u** command to set the RTSP URL
4. Use the **x** command to set the SetParameter JSON file
5. Monitor the lower half of the window for setup messages and results

Continuous Operation

- The program will automatically send "Alive" messages every 30 seconds
- Use other commands as needed (e.g., **l** for license check, **s** to toggle analytics result display)
- Continue monitoring the lower half of the window for ongoing messages and results

Remember to refer to the command instructions for specific syntax of each command.

1.2 SampleWrapper



1.2.1 Overview

SampleWrapper is a program implemented in C# that serves as an interface between ARGO and AI recognition functionalities.

1.2.2 Architecture

1. C# Layer:
 - Implements HTTP client and HTTP server functionalities
 - Handles communication with ARGO
2. C++ DLL Layer:
 - Manages Shared Memory
 - Contains AI recognition-related functionalities

1.2.3 Key Features

1. HTTP Client/Server: Implemented in the C# layer for communication.

2. Shared Memory: Implemented in the C++ DLL layer for efficient data sharing.
3. AI Recognition: Should be implemented in the C++ DLL layer to utilize image data from Shared Memory.

1.2.4 Data Flow

1. Image data is received through Shared Memory.
2. AI recognition is performed on this data in the C++ DLL layer.
3. Recognition results are returned to the C# layer.
4. Results are sent to ARGO via the HTTP server.

1.2.5 Usage by ARGO

- ARGO launches a separate instance of SampleWrapper for each channel.
- Each instance is started with different parameters.

Example: To start AI recognition for Channel 7, ARGO would execute:

```
1 | SampleWrapper.exe port=51007
```

1.2.6 Development Guidelines

When developing AI recognition features:

1. Implement the core functionality in the C++ DLL layer.
2. Use Shared Memory to access image data for recognition.
3. Ensure results are properly communicated back to the C# layer for transmission to ARGO.

1.2.7 Runtime Flow

This section describes the runtime behavior and internal processing steps of `SampleWrapper.exe` (the wrapper that performs analytics). Follow these steps to understand what the wrapper does after it starts.

1. Start HTTP server
 - On startup, `SampleWrapper.exe` creates and starts an HTTP server (port configurable via CLI or config). The server exposes endpoints for control and status (for example, `/Alive`, `/SetParameters`, `/GetLicense`) and returns appropriate HTTP responses.
2. Receive `SetParameters`
 - When `POST /SetParameters` is received, the server parses the JSON payload and extracts:
 - a. Supported version number(s)
 - b. Post URL for analytics results (`analytics_event_api_url`)
 - c. Image width and height (`image_width` , `image_height`)
 - d. ROI and other detection-related settings (`sensitivity` , `threshold` , `rois` , `jpg_compress` , etc.)
3. Create shared memory
 - Using the information from `SetParameters` (image size, frame size), the wrapper creates or opens the named shared-memory segment that will be used to read frames. It sets up any internal buffers and state needed to read frame headers (status flags, timestamps, sizes).
4. Read frames from shared memory and run detection

- The wrapper continuously polls or waits on the shared-memory segment to find new frames (status flag indicating a fresh frame).
 - When a new frame is available, it reads the frame bytes (using the agreed header format), converts/decodes if necessary, and passes the frame into the detection pipeline (YOLO model, native DLL, or other analyzer).
5. Post analytics results to ARGO
- After inference, the wrapper formats the detection result into the agreed JSON schema (versioned) and POSTs it to the `analytics_event_api_url` provided in step 2 (ARGO's HTTP endpoint).
 - It expects an HTTP OK response and may retry or log errors on failure.

Continuous operation and control

- Steps 4 and 5 repeat continuously while the wrapper is running; the wrapper also concurrently accepts and responds to `Alive` and `GetLicense` requests at any time.
- `Alive` should return a simple status (OK + optional metadata). `GetLicense` should validate or return license status per the implementation.

Notes and implementation tips


- The shared-memory header (see `MMF_Data` in `CSharp/SampleDLL/dllmain.cpp`) contains useful fields (header/footer markers, status, `image_width` / `image_height`, `image_size`, `timestamp`) — follow the same layout for cross-language compatibility.
- Keep HTTP control logic separate from the frame read / detection loop (use separate threads or async tasks) so control endpoints remain responsive.
- Implement exponential backoff or retry logic when posting results to ARGO to handle transient network errors.

2. ARGO AI Integration Specifications

2.1. Communication APIs


2.1.1 Set Algorithm Parameters (ARGO → SampleWrapper.exe)

- Method: POST
- URL: `http://127.0.0.1:{port_num}/SetParameters`
- MIME: `application/json`
- HTTP Code: Only 200
- Content: See detailed explanation

 Note: The last digit of the port number represents the channel number. For example, 51007 for channel 7.


2.1.2. Program Channel Alive (ARGO → SampleWrapper.exe)

- Method: GET
- URL: `http://127.0.0.1:{port_num}/Alive`
- MIME: `plain/text`
- HTTP Code: Only 200
- Content: Empty string

 Note: The last digit of the port number represents the channel number. For example, 51007 for channel 7.


2.1.3. Program License Exist (ARGO → SampleWrapper.exe)

- Method: GET
- URL: http://127.0.0.1:{port_num}/GetLicense
- MIME: plain/text
- HTTP Code: Only 200
- Content: Empty string

 Note: The last digit of the port number represents the channel number. For example, 51007 for channel 7.

2.1.4. Event Parameters (SampleWrapper.exe → ARGO)

- Method: POST
- URL: <http://127.0.0.1:{portnum on SetParameters json}>/PostAnalyticsResult
- MIME: application/json
- HTTP Code: Only 200
- Content: See detailed explanation

 Note: The last digit of the port number represents the channel number. For example, 51007 for channel 7.

2.2. Suggested Detailed Settings (JSON)

2.2.1. Set Algorithm Parameters

```
1 {
2   "version": "1.2",
3
4   "analytics_event_api_url": "http://127.0.0.1:9901/PostAnalyticsResult",
5   "image_width": 1280,
6   "image_height": 720,
7   "jpg_compress": 20,
8   "rois": [
9     {
10      "sensitivity": 50,
11      "threshold": 50,
12      "rects": [
13        {
14          "x": 100, "y": 100,
15          "x": 200, "y": 200,
16          "x": 300, "y": 300,
17          "x": 400, "y": 400
18        },
19        {
20          "x": 500, "y": 500,
21          "x": 600, "y": 600,
22          "x": 700, "y": 700,
23          "x": 800, "y": 800
24        }
25      ]
26    }
27  ]
28 }
```

★The supported settings allow up to 10 ROIs, with each ROI containing a maximum of 10 points.

2.2.2 Parameter Description

2.2.2.1 Top-level Parameters

Name	Type	Example	Description
------	------	---------	-------------

version	string	"1.2"	Configuration file version, used for versioning and compatibility checks
analytics_event_api_url	string (URL)	"http://127.0.0.1:9901/PostAnalyticsResult"	The HTTP API endpoint where analytics results will be posted
image_width	int	1280	Image width in pixels
image_height	int	720	Image height in pixels
jpg_compression	int (0-100)	20	JPEG compression rate. Lower values mean higher compression and lower quality; higher values mean better quality but larger file size

2.2.2.2 rois Array

The **rois** array defines multiple Regions of Interest (ROI). Each ROI includes detection parameters and the coordinates of its rectangle area.

ROI Object

Name	Type	Example	Description
sensitivity	int (0-100)	50	Detection sensitivity. Higher values mean more sensitive detection

<code>threshold</code>	int (0–100)	<code>50</code>	Detection threshold. Lower values mean events are triggered more easily
<code>rects</code>	array	4 coordinate points	Defines the rectangle boundary of the ROI. Four corner points are required

2.2.2.3 `rects` Parameter

`rects` defines a rectangle using four coordinate points (`x` , `y`).

It is recommended to follow a fixed order of points:

Top-left → **Top-right** → **Bottom-right** → **Bottom-left**

Point Coordinates

Name	Type	Example	Description
<code>x</code>	int	<code>100</code>	Horizontal coordinate (pixels), measured from the left edge of the image
<code>y</code>	int	<code>100</code>	Vertical coordinate (pixels), measured from the top edge of the image

Example

```
1 "rects": [  
2   {"x": 100, "y": 100}, // Top-left  
3   {"x": 200, "y": 100}, // Top-right  
4   {"x": 200, "y": 200}, // Bottom-right  
5   {"x": 100, "y": 200} // Bottom-left  
6 ]
```

2.2.3 Recognition Event Results

```
1 {  
2   "version": "1.2",  
3   "port_num": 1,
```

```
4  "keyframe": "/9j/4AAQSkZJR...",
5  "timestamp": 1500321576000,
6  "rois_rects": [
7    [
8      {"x": 0, "y": 0},
9      {"x": 10, "y": 0},
10     {"x": 10, "y": 10},
11     {"x": 0, "y": 10}
12   ],
13   [
14     {"x": 50, "y": 50},
15     {"x": 60, "y": 50},
16     {"x": 60, "y": 60},
17     {"x": 50, "y": 60}
18   ]
19 ]
20 }
```

- ⚠ For deep integration, please define your own algorithm parameter settings and provide documentation.
- “keyframe” is in JPG format and then encoded into a Base64 JPG string.
- “version” is used to define the version number of the transmitted JSON.

2.2.4 Parameter Description

2.2.4.1 Top-level Parameters

Name	Type	Example	Description
version	string	"1.2"	Data format version for compatibility tracking
port_num	int	1	Identifier for the video input port or channel
keyframe	string (Base64)	"/9j/4AAQSkZJR..."	Keyframe image encoded in Base64, usually in JPEG format
timestamp	int (epoch-based)	1500321576000	Timestamp of the frame. Unit can be milliseconds or microseconds depending on the system

2.2.4.2 rois_rects Array

The `rois_rects` parameter is an array of multiple rectangular ROIs. Each ROI is defined by **four corner points**.

ROI Rectangle Example

```
1 [
2   {"x": 0, "y": 0},
3   {"x": 10, "y": 0},
4   {"x": 10, "y": 10},
5   {"x": 0, "y": 10}
6 ]
7
```

This defines a 10×10 pixel square ROI.

Point Coordinates

Name	Type	Example	Description
x	int	50	Horizontal coordinate in pixels, relative to the left edge of the image
y	int	50	Vertical coordinate in pixels, relative to the top edge of the image

Note:

- Each ROI must have exactly **four points**, ordered consistently (Top-left → Top-right → Bottom-right → Bottom-left).
- Inconsistent ordering may result in incorrectly shaped polygons.

2.3. Shared Memory Settings

Using Windows shared memory, currently supporting a maximum resolution of 1920x1080.

```
1 const int MMF_DATA_HEADER = 0x1234;
2 const int MMF_DATA_FOOTER = 0x4321;
3 struct MMF_Data_Generic
4 {
5     __int64 header = MMF_DATA_HEADER;
6     //video status : 0=no use , 1=new frame, 2=detection got frame
7     int image_status = 0;
8     //resolution
9     int image_width = 0;
10    int image_height = 0;
11    //video size
12    int image_size = 0;
13    //timestamp in Windows FileTime style
14    uint64_t timestamp = 0;
15    //video data
16    unsigned char* image_data[1920 * 1080 * 3];
17    __int64 footer = MMF_DATA_FOOTER;
18 };
```

- `image_status` : 0 = no use, 1 = new frame, 2 = detection got frame

- **timestamp** : in Windows FileTime style
- Shared mem Name Rule : ChannelFrame_%d
%d is PortNum

2.4. Execution and Naming Conventions

Executable Naming Flexibility

SampleWrapper.exe can be renamed to any desired name. ARGO supports different program names for integration. However, the parameter format remains fixed and must be adhered to.

Parameter Format

The parameter format is standardized and must be maintained regardless of the executable name:

```
1 [executable_name].exe port=[port_number]
```

Example

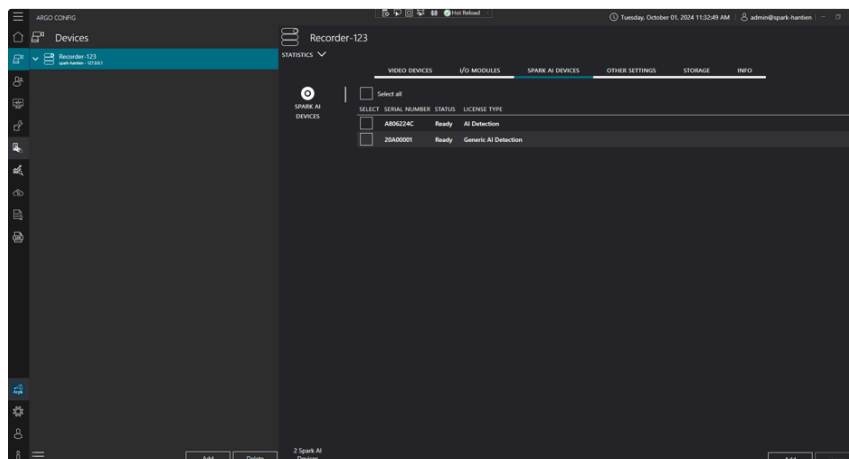
```
1 yourprogramname.exe port=51007
```

Important Notes

1. While the executable name is flexible, ensure that ARGO is configured to call the correct filename.
2. The port number format (51000 + channel_number) must be strictly followed.
3. Always use the "port=" parameter as shown in the example.
4. Ensure that your program correctly interprets and uses the provided port number for communication with ARGO.
5. The executable must be a **signed application** to ensure it runs correctly and is not flagged as malicious by Windows or antivirus software.

3. ARGO AI Integration UI on Client and Config Client

3.1. Add Generic AI Detection on Spark AI Devices



Users can integrate AI into your Generic AI Device on the Spark AI Devices Page by clicking the Add button

SELECT SERIAL NUMBER STATUS LICENSE TYPE

Add AI device to recorder manually

Detection Type
Generic AI Detection

Generic AI Detection Class Name
MyObjClassName

Analytics application name
SampleWrapper.exe

Analytics application Path
D:\workspace\SampleWrapper_v1.2.zip Browse...

License serial
20A00001

Analytics application Port
31002 The port number must be between 0 and 65535

HTTP Port
9904 The port number must be between 0 and 65535

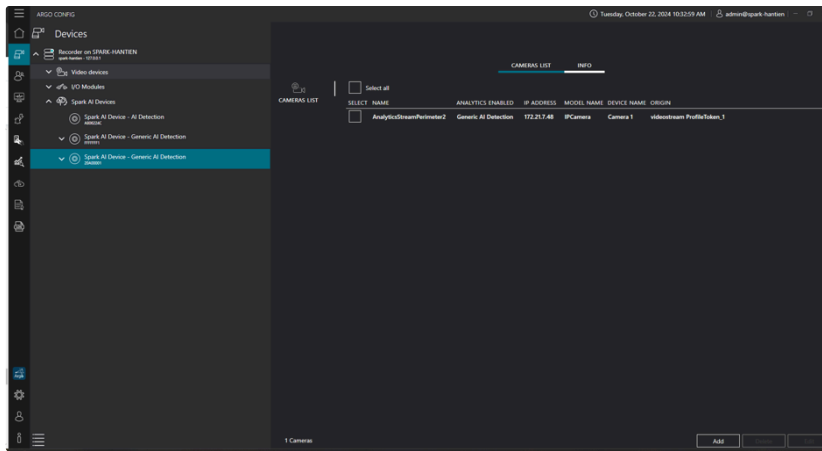
Add Cancel

Steps to Manually Add a Generic AI Device

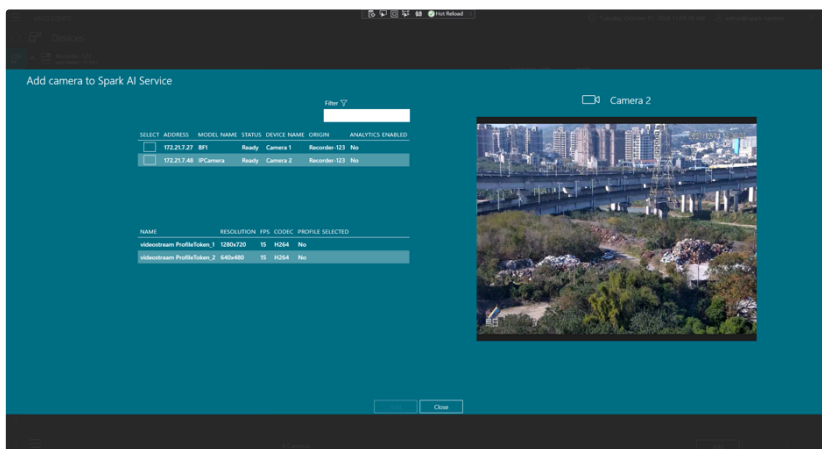
1. Select **Detection Type** as **Generic AI Detection**.
2. Enter the Class name detected by your recognition program in the **Generic AI Detection Class Name** field.
3. Enter the name of your recognition program (e.g., *SampleWrapper.exe*) in the **Analytics Application Name** field.
4. Use the **Browse** button to select the recognition program's ZIP file for upload.
5. Choose the **License Serial Number** to use.
 - This License Serial Number corresponds to the **Generic AI Detection License** for Argo.
 - If needed, please contact customer support for assistance.
6. Enter the **port** used by the recognition program for Argo in the **Analytics Application Port** field.
7. Enter the **HTTP port** used by the recognition program to push data to Argo in the **HTTP Port** field.
8. Click Add Button to add Generic AI Device.

3.2. Analyticsstreamperimeter on Spark AI Device - Generic AI Detection

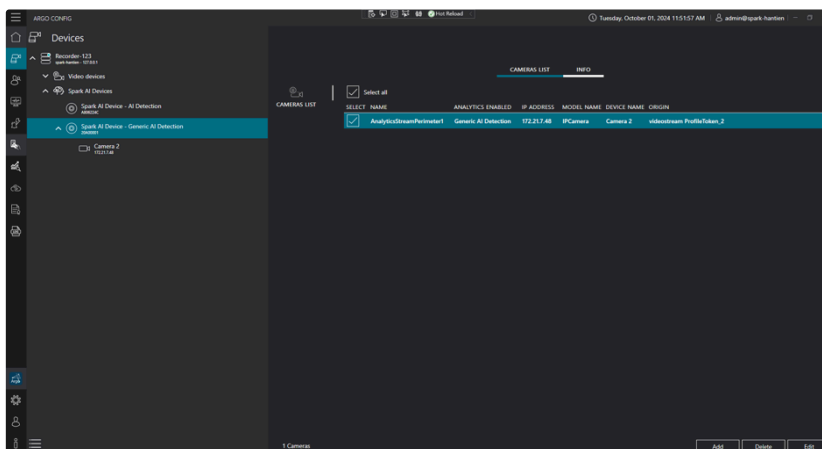
1. Select Spark AI Device- Generic AI Detection , you will see Add button on the right side.



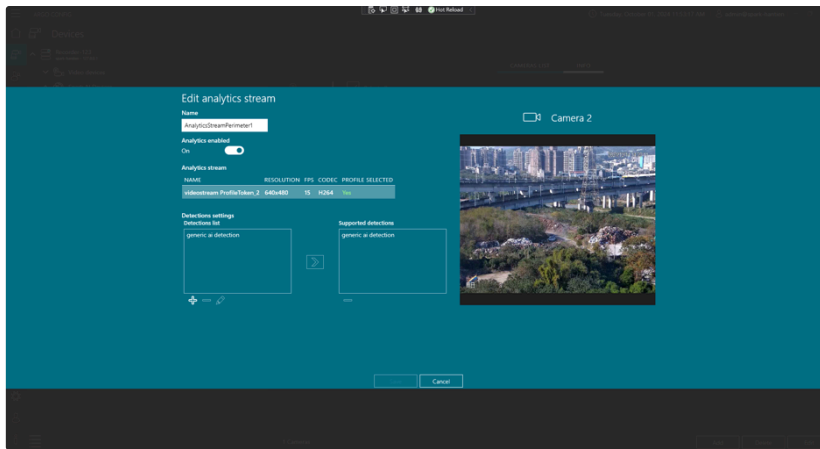
2. Click Add button to Add Analytiscisstreamperimeter



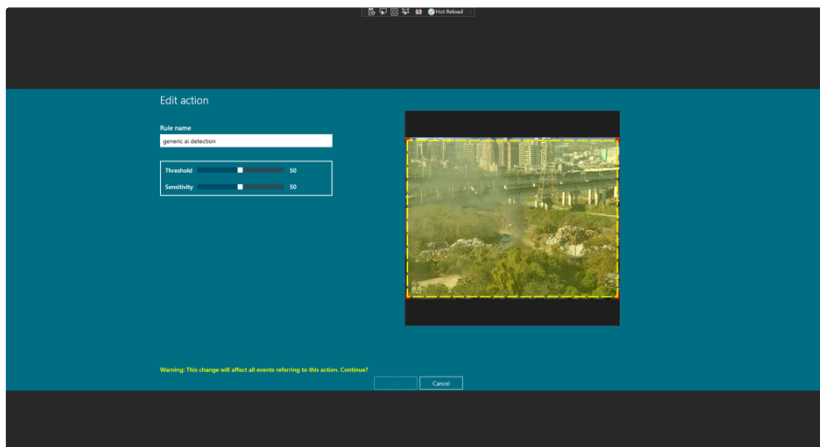
3. Users can select a camera device and stream for analysis . And Click add to create Analytiscisstreamperimeter on Generic AI Detecion Device.



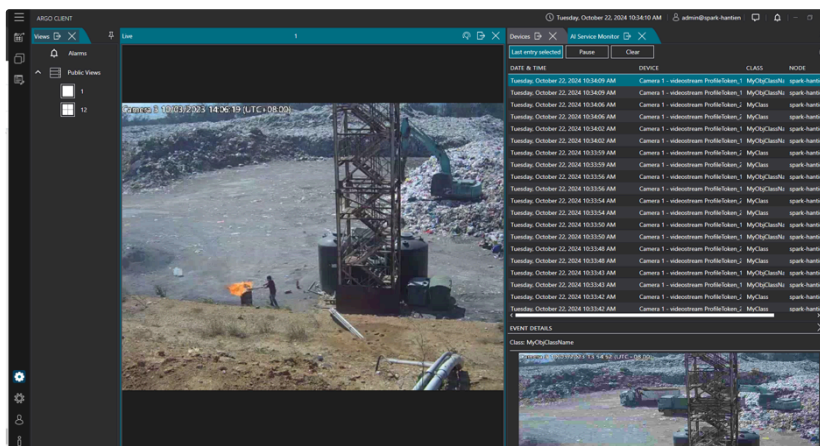
4. User can edit Analytiscisstreamperimeter by clicking Edit Button.



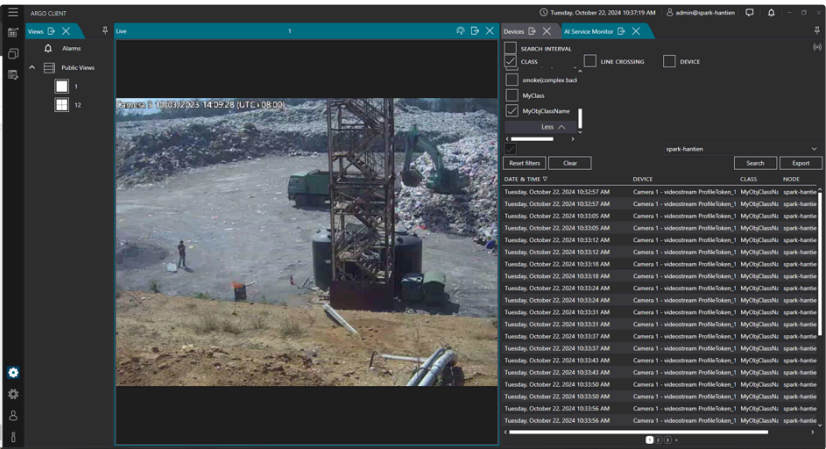
- Users can click the + button to create a new detection, the **pen** button to edit detections, the - button to delete detections, and the > button to use detections.



- Users can edit detection settings, such as threshold, sensitivity, and ROI region, on the Edit page.
- After these settings are configured, the analytics will start to run.
- Events pushed by the recognition program will be displayed in the **AI Service Monitor** on the client side



- The user can search using the name previously entered in the "**Generic AI Detection Class Name**" field.



4. License

The Generic AI Detection License serial number and channel count are provided by Taiwan Spark Technology Co.,Ltd . If you wish to use the above features, please contact the appropriate representative.

