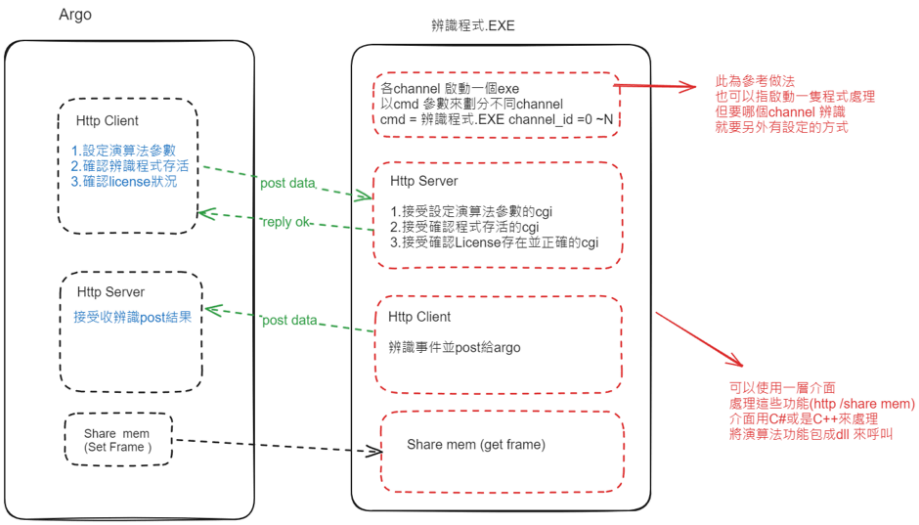
Argo 通用 AI 整合文件 version 1.2

一、實作架構:
SampleWrapper.exe 執行行為與內部處理流程說明
1. 啟動 HTTP 伺服器
2. 接收 SetParameters 請求
3. 建立 Shared Memory
4. 從 Shared Memory 讀取畫面並執行偵測
5. 將分析結果回傳至 ARGO
6. 持續運作與控制
7. 注意事項與實作建議
二、溝通API
1. 設定演算法參數 (Argo → 辨識.exe)
2. 程式通道Alive(Argo → 辨識.exe)
3. 事件參數(辨識.exe → Argo)
三、設定詳細內容(json)
1. 設定演算法參數 (支援最多10組ROI 和每個ROI 有自己的設定值)
2.辨識事件結果
四、Shared mem 設定

一、實作架構:

參考下圖說明



SampleWrapper.exe 執行行為與內部處理流程說明

本節說明 **SampleWrapper.exe**（執行分析的包裝程式）的執行時行為與內部處理步驟。
請依下列步驟了解此 wrapper 啟動後所執行的動作。

---

1. 啟動 HTTP 伺服器

啟動時，**SampleWrapper.exe** 會建立並啟動一個 HTTP 伺服器（埠號可透過命令列參數或設定檔指定）。
伺服器會提供多個控制與狀態的 API endpoint，例如：

- `/Alive`
- `/SetParameters`
- `/GetLicense`

並針對各項請求回傳適當的 HTTP 回應。

---

2. 接收 SetParameters 請求

當收到 `POST /SetParameters` 時，伺服器會解析 JSON 載荷（payload），並擷取以下資訊：

a. 支援的版本號（Supported version number）
b. 分析結果回傳的目標網址（`analytics_event_api_url`）
c. 影像的寬與高（`image_width`, `image_height`）
d. ROI 與其他偵測相關設定（例如 sensitivity、threshold、rois、jpg_compress 等）

---

3. 建立 Shared Memory

根據 `SetParameters` 傳來的資訊（影像大小、畫面尺寸等），
wrapper 會建立或開啟指定名稱的 shared-memory 段（named shared memory segment），
並設定內部緩衝區與狀態結構，用於讀取畫面標頭（如狀態旗標、timestamp、大小等）。

4. 從 Shared Memory 讀取畫面並執行偵測

wrapper 會持續輪詢（poll）或等待 shared memory 中的新畫面出現
（藉由狀態旗標判斷是否有新 frame）。

當偵測到新畫面可用時，會：

- 讀取 frame 位元資料（依照約定的 header 格式）
- 進行必要的轉換或解碼
- 將影像輸入偵測流程（例如 YOLO 模型、原生 DLL、或其他分析模組）

---

5. 將分析結果回傳至 ARGO

完成推論後，wrapper 會依照定義好的 JSON schema（具版本號）
格式化偵測結果，並透過 HTTP `POST` 傳送到在第 2 步中提供的
`analytics_event_api_url`（即 ARGO 的 HTTP 接收端）。

程式會期望收到 HTTP OK 回應；若失敗則進行重試或記錄錯誤。

---

6. 持續運作與控制

步驟 4 與 5 會在 wrapper 執行期間持續重複進行。
同時，wrapper 也會持續接受並回應控制請求，例如：

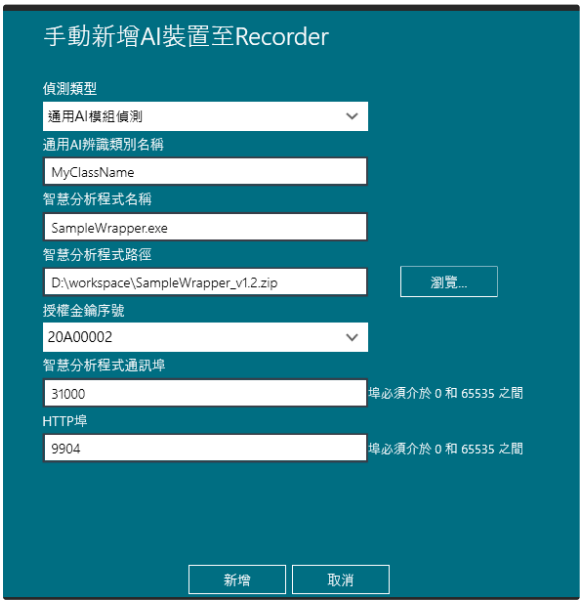
- `/Alive`：回傳當前狀態（OK 及可選擇性附加的系統資訊）
- `/GetLicense`：驗證或回傳授權狀態（依實作而定）

---

7. 注意事項與實作建議

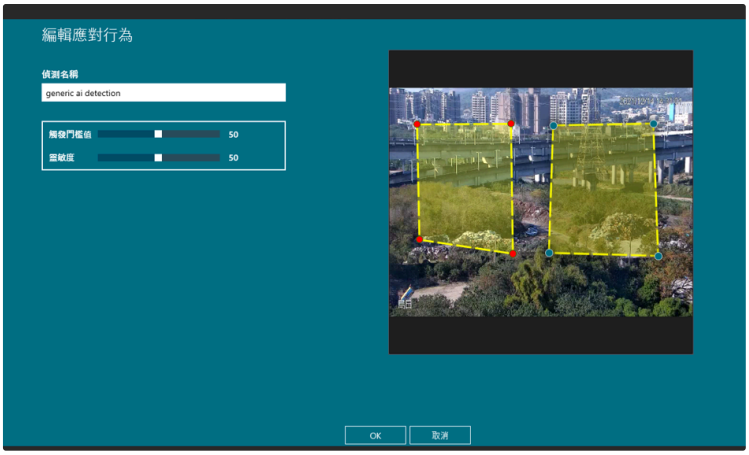
- **Shared memory 標頭結構**
  （請參考 `CSharp/SampleDLL/dllmain.cpp` 中的 `MMF_Data`）
  包含實用欄位：header/footer 標記、狀態、image_width / image_height、image_size、timestamp 等。
  請保持相同的記憶體結構以確保跨語言相容性。
- **HTTP 控制邏輯與影像偵測流程分開執行**
  請使用不同的 thread 或 async task，
  確保控制端點（例如 `/Alive`）在高負載下仍能即時回應。
- **實作重試與退避機制（exponential backoff）**
  當向 ARGO 回傳結果時若遇到網路暫時性錯誤，
  應實作重試與退避機制以確保穩定性。

Argo 可以上傳指定的辨識程式.exe ,經由Zip 檔上傳,並產生一組AI裝置(通用AI模組偵測)

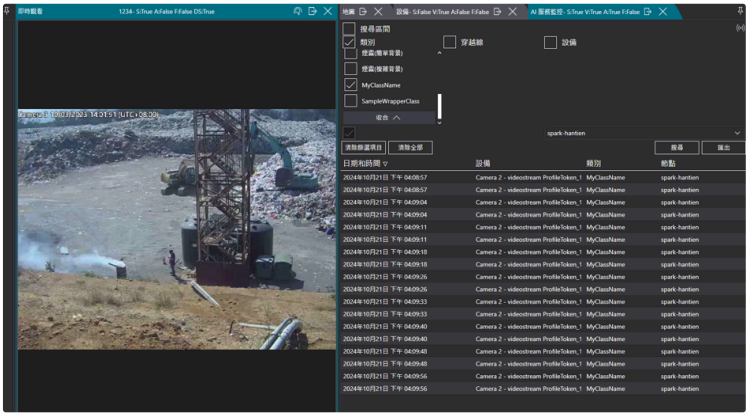


Argo 會綁定某隻攝影機的stream 來執行 ”辨識程式.exe”

透過UI的設定後，將設定結果傳給”辨識程式.exe”進行辨識



最後辨識的結果會顯示在Client 的AI服務監控裡面



搜尋歷史資料時可以從類別中去尋找設定的通用辨識AI類別名稱

二、溝通API

1. 設定演算法參數 (Argo → 辨識.exe)

方法　　POST
網址　　**http :// 127.0.0.1:智慧分析程式通訊埠/SetParameters**
MIME　 application/json
內容　　http code = 只有200
content = 詳說明

2. 程式通道Alive(Argo → 辨識.exe)

方法　　GET
網址　　**http :// 127.0.0.1:智慧分析程式通訊埠/Alive**
MIME　 plain/text
內容　　http code = 只有200
content = 空字串

3. 事件參數(辨識.exe → Argo)

方法　　POST
網址　　**http :// 127.0.0.1:Http_port/PostAnalyticsResult**
MIME　 application/json
內容　　http code = 只有200
content = 詳說明

三、設定詳細內容(json)

1. 設定演算法參數 (支援最多10組ROI 和每個ROI 有自己的設定值)

```
{
    “version”:”1.2”,                                    //溝通json版號
    "analytics_event_api_url":"http://127.0.0.1:port/PostAnalyticsResult", //設定spark server接收事件的URL
    "image_width": 1920,                                //影像大小
    "image_height":1080,                                //影像大小
    "rois":[
          { "sensitivity": 50,
            "threshold":  50,
             "rects":  [
                        {"x": 100 , "y":100},
                        {"x": 200 , "y":200},
                        {"x": 300 , "y":300},
                        {"x": 400 , "y":400}
            ]},
            {"sensitivity": 50,
             "threshold": 50,
            "rects":
            [
             {"x": 500 , "y":500},
             {"x": 600 , "y":600},
             {"x": 700 , "y":700},
             {"x": 800 , "y":800}
             ]}
             ]
    }
```

★ 支援設定的ROI最多10組, 每組ROI內最多10個點

2.辨識事件結果

```
{
      “version“:”1.2”,                                  //溝通json版號
      "port_num":  51000,                               // port num
      "keyframe":   /9j/4AAQSkZJR...,                   //JPG image (base64)
      "timestamp":  15003215760000,                     //timestamp field in share mem (Windows Filetime timestamp)
      "rois_rects": [                                   //ROI發報框位置 (可內含多個框)
            [ {"x": 0, "y": 0},    {"x": 10, "y": 0},     {"x": 10, "y": 10},    {"x": 0, "y": 10} ],
            [{"x": 50, "y": 50},    {"x": 60, "y": 50},    {"x": 60, "y": 60},    {"x": 50, "y": 60}]
      ]
}
```

深度整合方請自行定義設定演算法參數的內容並提供文件

四、Shared mem 設定

使用window shared mem ,目前resolution最大支援 **1920x1080** 的資料

設定如下

```
const int MMF_DATA_HEADER = 0x1234;
const int MMF_DATA_FOOTER = 0x4321;
```

```
struct MMF_Data_Generic
{
	__int64 header = MMF_DATA_HEADER;
	//video status : 0=no use , 1=new frame, 2=detection got frame
	int image_status = 0;
	//resolution
	int image_width = 0;
	int image_height = 0;
	//video size
	int image_size = 0;
	//timestamp in Windows FileTime style
	uint64_t  timestamp = 0;
	//video data
	unsigned char* image_data[1920 * 1080 * 3];
	__int64 footer = MMF_DATA_FOOTER;
};
```

- Shared mem Name Rule : ChannelFrame_%d
  %d 是**智慧分析程式通訊埠**