Industrial project with extended reality HT2025

Individual Project Portfolio 1

Student Name: Eric Marquez

Role in the project:

Level 1 connectivity, data pipeline implementation and project leader.

Sprint 1:

User Story 1 – PLC & Node-red setup

Description:

As a PLC/OPC developer, I want to connect Siemens PLCs to Node-RED via OPC UA so that machine data becomes available for streaming.

Task 1 - Installing node-red and implementing OPC UA nodes.
Description:

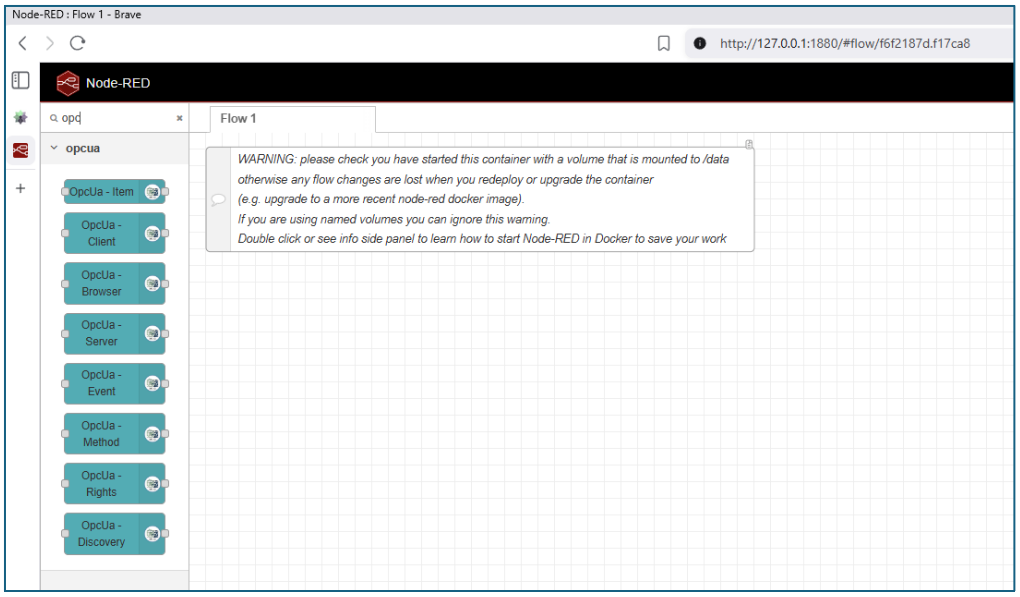
After reading the node-red general guide and the windows installation guide, node-red was installed on windows 11 and additional OPC-related nodes were installed as well.

At first, I conceived a containerized infrastructure in which node-red and later influx are deployed in dockers containers, to this end, node-red was also installed on a Linux-based container following the instructions here.

Challenge:

When using containers, it is advisable to understand how this work, specially how the network communication is implemented on docker with WSL2 (in case windows 11 is the host system), otherwise the node-red instance won't be able to "see" the OPC UA simulator or the actual PLC.

Regardless of the platform, the outcome is a node-red instance running and accessible trough http://127.0.0.1:1880/



WARNING: please check you have started this container with a volume that is mounted to /data otherwise any flow changes are lost when you redeploy or upgrade the container (e.g. upgrade to a more recent node-red docker image).
If you are using named volumes you can ignore this warning.
Double click or see info side panel to learn how to start Node-RED in Docker to save your work

Figure 1 - A first setup of node-red with OPC UA nodes installed

Task 2 - Connecting to PLC Siemens in station SIF-402

Description:

There are two ways to access data from the PLC controlling the stations: A direct way using OPC-UA and indirect way through the SIF API available on the server computer.

The immediate and fastest way to test communication with the PLC is through the API but first is necessary to read the documentation so we can know which endpoints are available, such documentation can be found in one of the manufacturer's manual: "*Manual de software SIFMES-400 V4 EN.pdf*", specifically on Section 3. SIFMES-400 API.

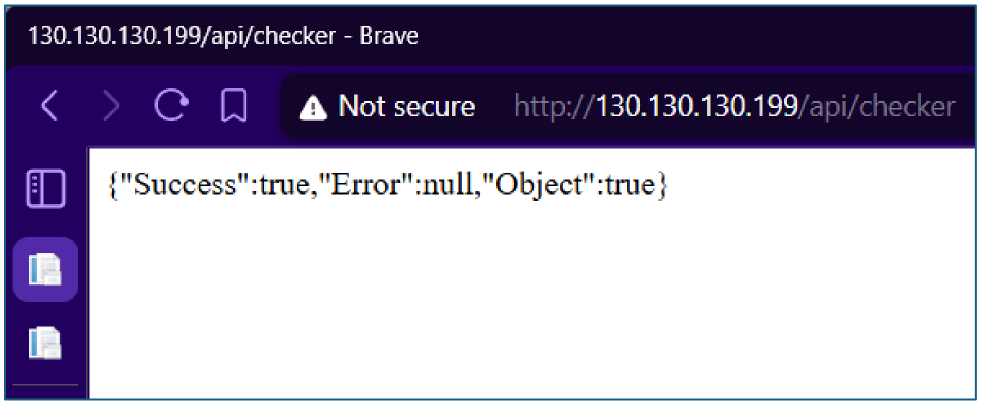
First API test:

A first test was made using the "checker" service that allows you to check if there is connection with the API and it is in operation. The URI of this web service (GET) is as follows:

130.130.130.199/api/checker

After the GET request, the expected result shown in a browser will be like this (as seen in Figure 2:

```
{"Success":true,"Error":null,"Object":null}
```

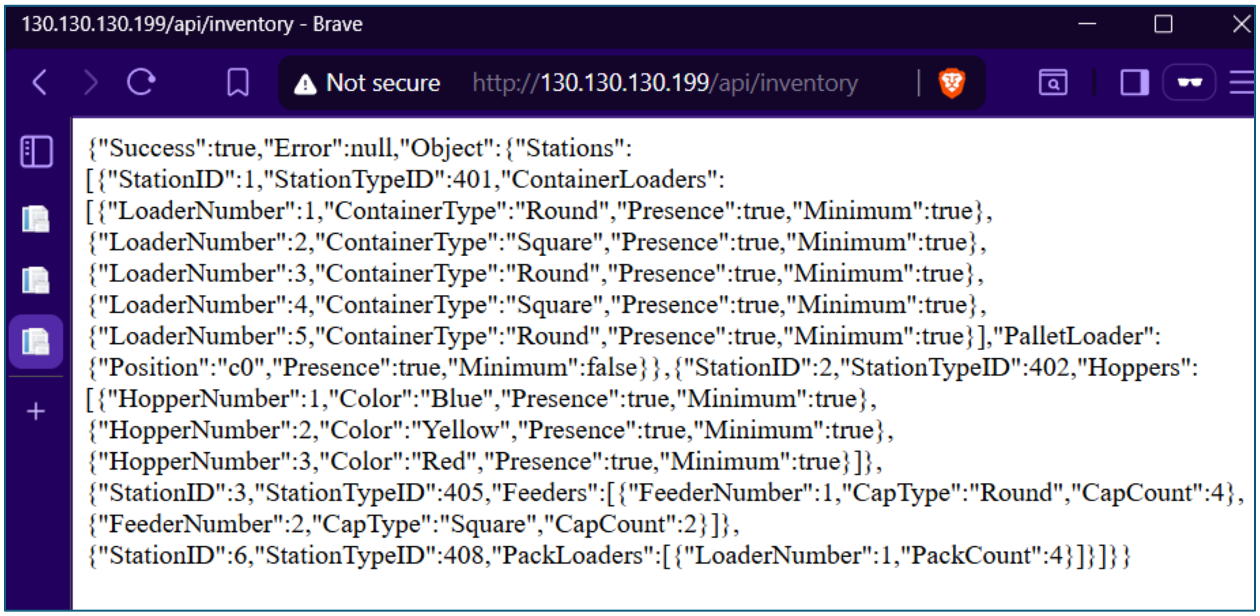

130.130.130.199/api/checker - Brave
Not secure
http://130.130.130.199/api/checker

```
{"Success":true,"Error":null,"Object":true}
```

Figure 2 – Response obtained from the GET request to the checker endpoint. Success: true confirms that the API is working

Additional API test:

A second test was done by calling the “*Inventory”* service, which allows rescuing the inventory data of the different stations of the SIF-400 system. The following information is collected: SIF-402 station: Presence indicator, minimum indicator, and type of color for each hopper, as seen in Figure 3.


130.130.130.199/api/inventory - Brave
Not secure
http://130.130.130.199/api/inventory

```
{"Success":true,"Error":null,"Object":{"Stations":
[{"StationID":1,"StationTypeID":401,"ContainerLoaders":
[{"LoaderNumber":1,"ContainerType":"Round","Presence":true,"Minimum":true},
{"LoaderNumber":2,"ContainerType":"Square","Presence":true,"Minimum":true},
{"LoaderNumber":3,"ContainerType":"Round","Presence":true,"Minimum":true},
{"LoaderNumber":4,"ContainerType":"Square","Presence":true,"Minimum":true},
{"LoaderNumber":5,"ContainerType":"Round","Presence":true,"Minimum":true}],"PalletLoader":
{"Position":"c0","Presence":true,"Minimum":false}},{"StationID":2,"StationTypeID":402,"Hoppers":
[{"HopperNumber":1,"Color":"Blue","Presence":true,"Minimum":true},
{"HopperNumber":2,"Color":"Yellow","Presence":true,"Minimum":true},
{"HopperNumber":3,"Color":"Red","Presence":true,"Minimum":true}]},
{"StationID":3,"StationTypeID":405,"Feeders":[{"FeederNumber":1,"CapType":"Round","CapCount":4},
{"FeederNumber":2,"CapType":"Square","CapCount":2}]},
{"StationID":6,"StationTypeID":408,"PackLoaders":[{"LoaderNumber":1,"PackCount":4}]}]}}
```

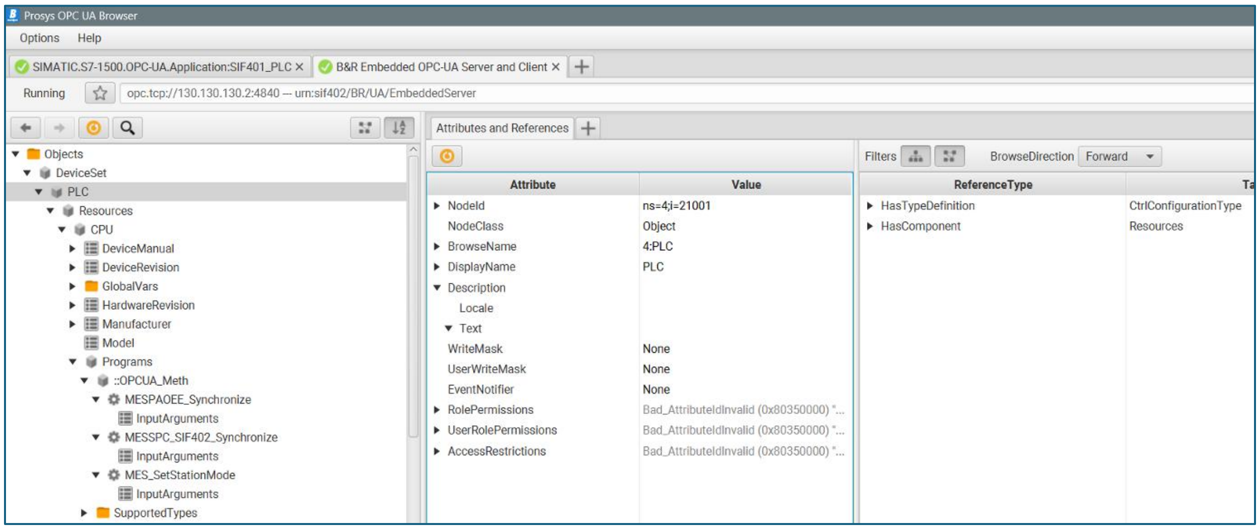
Figure 3 - Response obtained from the GET request to the inventory endpoint

Testing communication via OPC UA:

In my opinion, before attempting to connect to the PLC via node-red, it is advisable to use an OPC UA browser to validate the communication with the PLC and understand how the variables can be found and what is their full path. However, to understand how to use this tool and the information it provides, it is important to:

- Read section 1.2.(Connectivity) of the manufacturer manual to know the Ip address of each station.
- Read the excel file with the list of signals available on each machine and their description.

The access to the station SIF402 from OPCUA browser 130.130.130.2:4840 was tested as shown in Figure 4.


Prosys OPC UA Browser
Options
Help
SIMATIC.S7-1500.OPC-UA.Application:SIF401_PLC ×
B&R Embedded OPC-UA Server and Client ×
Attributes and References
Filters
BrowseDirection

Running

opc.tcp://130.130.130.2:4840 — urn:sif402/BR/UA/EmbeddedServer

```
▼ Objects
  ▼ DeviceSet
    ▼ PLC
      ▼ Resources
        ▼ CPU
          ▶ DeviceManual
          ▶ DeviceRevision
          ▶ GlobalVars
          ▶ HardwareRevision
          ▶ Manufacturer
            Model
          ▼ Programs
            ▼ ::OPCUA_Meth
              ▼ MESPAOEE_Synchronize
                  InputArguments
              ▼ MESSPC_SIF402_Synchronize
                  InputArguments
              ▼ MES_SetStationMode
                  InputArguments
            ▶ SupportedTypes
```

Attribute	Value
▶ NodeId	ns=4;i=21001
NodeClass	Object
▶ BrowseName	4:PLC
▶ DisplayName	PLC
▼ Description	
Locale	
▼ Text	
WriteMask	None
UserWriteMask	None
EventNotifier	None
▶ RolePermissions	Bad_AttributeIdInvalid (0x80350000) "...
▶ UserRolePermissions	Bad_AttributeIdInvalid (0x80350000) "...
▶ AccessRestrictions	Bad_AttributeIdInvalid (0x80350000) "...

BrowseDirection: Forward

ReferenceType	Ta
▶ HasTypeDefinition	CtrlConfigurationType
▶ HasComponent	Resources

Figure 4 - A glance of the OPC UA browser information available for station SIF402

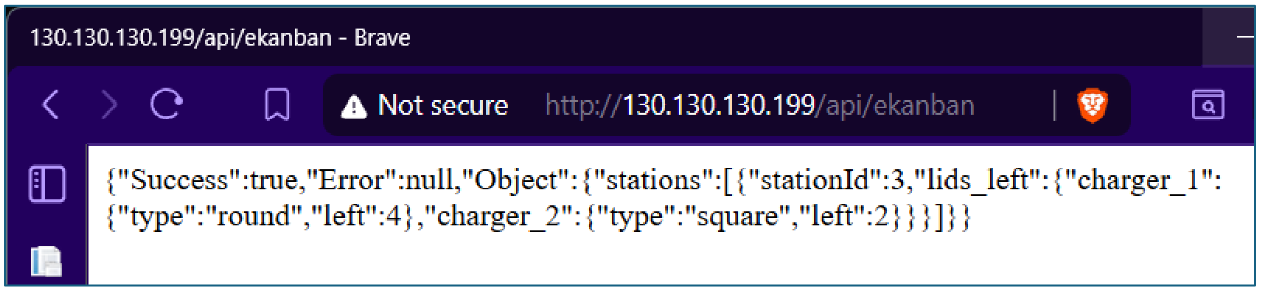
Challenge:

The OPC UA browser software can be confusing at the beginning and the information we are looking for might not be easy to find, the first thing to understand here is the structure used for the presentation of the objects tree.

Task 3 - Connecting to PLC Siemens in station SIF-405

The same test was applied to station 5, in this case via API with the service "*ekanban*", which allows to receive the number of caps at the SIF-405 capping station. Both the number of caps per feeder and the type of each of the feeders placed are received.

The URI of this web service (GET) is: 130.130.130.199/api/ekanban. And the response is shown in


130.130.130.199/api/ekanban - Brave
Not secure
http://130.130.130.199/api/ekanban

```
{"Success":true,"Error":null,"Object":{"stations":[{"stationId":3,"lids_left":{"charger_1":
{"type":"round","left":4},"charger_2":{"type":"square","left":2}}}]}}
```

Figure 5 - Response of the ekanban endpoint for station SIF405

Task 4 - Connecting to PLC Siemens in station SIF-405 via OPC UA in node-red

Description:

To access the PLC data available trough the OPC UA server running on each PLC, we must use the OPC UA client node in node-red, in this case the "*RunningTime"* signal was selected arbitrarily to run this test, and all the metadata required was extracted from the OPC UA browser.

Description of the test:

- An OPC UA Item node was used and its properties configured in this way:
  - Item: ns=6;s=::AsGlobalPV:MESPAOEE_udiRunningTime
  - Type: UInt32
  - Name: MESPAOEE_udiRunningTime
- An OPC UA Client node with:
  - Endpoint: opc.tcp://130.130.130.2:4840/UA/EmbeddedServer
  - Security Policy: None
  - Security Mode: None
  - User: Anonymous
  - Action: READ
  - Certificate: None
- The topology implemented is shown in Figure 6.

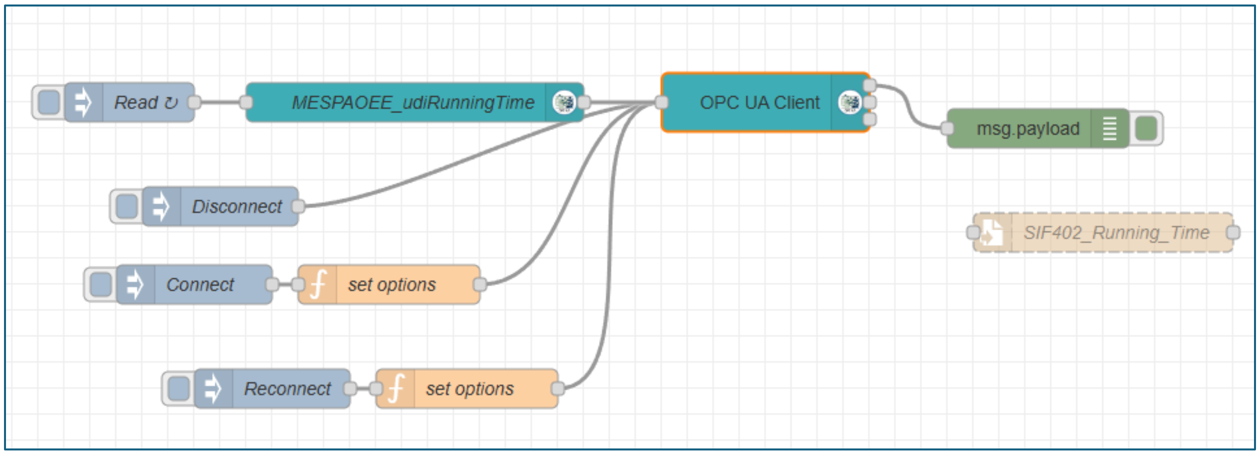


Figure 6 - Node-red flow implemented to access a PLC signal via OPC UA

- Later, the test was scaled up to process three signals, as seen in Figure 7.

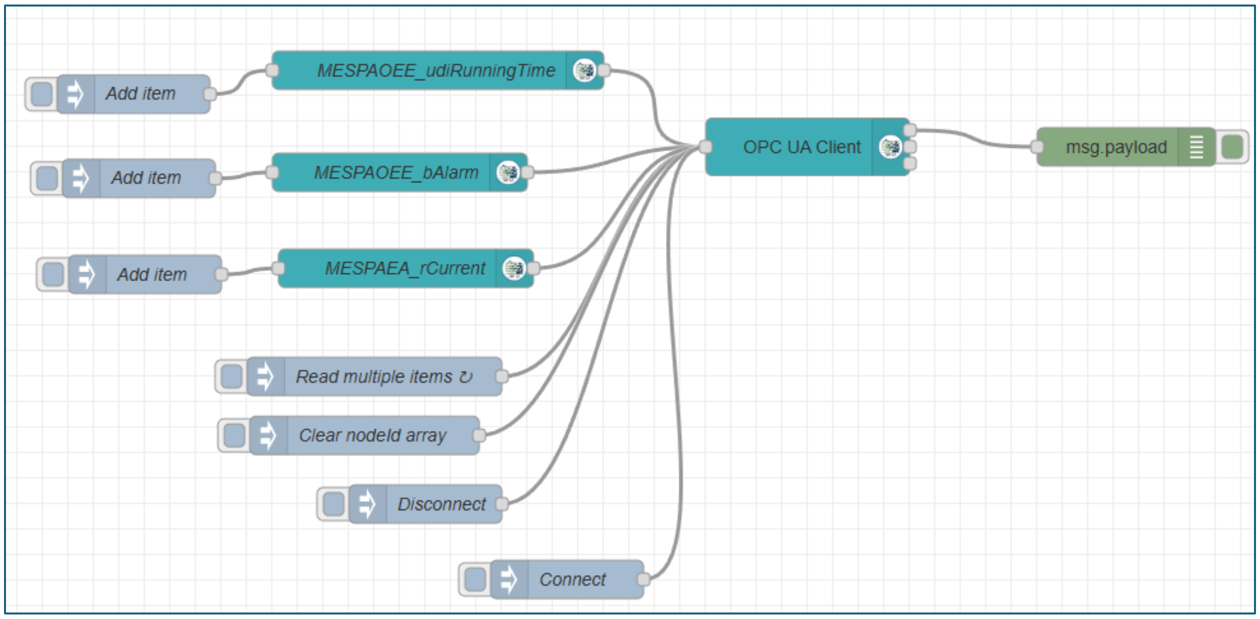


Figure 7 - Node-red flow implemented to access three PLC signal via OPC UA

Challenges:

Using node-red for the first time might be overwhelming, therefore I advise checking these videos from node-red academy which are perfect for understanding the basics as well as the intermediate features.

Coding in node-red is done in JavaScript, in principle is not required to know the language, especially if using assistance from generative AI agents, however understanding programming logic is definitively a skill that comes handy here to process and transform the raw signals into a structured format that can later be processed by the Unity application.

Sprints 2 and 3:

User Story 2 – Backend API Setup

Description:

As a data/API developer, I want to expose PLC values through a REST endpoint so that Unity can request data easily.

Task 1 -To implement node-red HTTP In/Out nodes.

Description:

To broadcast the information coming from Level 1, a REST API was implemented, in node-red this is done through the HTTP in and out nodes. In this case, for a mockup test, a flow variable "blue_lastSignal" was defined.

Important:

Since the data flowing through a flow, can change periodically, to make a value available trough an API endpoint, first is necessary to “retain a snapshot” of such value, this can be done through the “flow.set” function.

Description of the test:

- A function node (*Store blue latest value*) is implemented to retain the value of the variable:

  -
    ```
    flow.set("blue_lastSignal", msg.payload);
    return msg;
    ```
- Another function node (*Return latest value*) is defined to get this value and prepare it for HTTP transmission:

  -
    ```
    const data = flow.get("blue_lastSignal");
    if (!data) {
    ```

```
msg.payload = { error: "No data available yet" };
msg.statusCode = 503;
return msg;
}

msg.payload = data;
msg.headers = { "Content-Type": "application/json" };
return msg;
```

- The (*Return latest value*) function node must be placed in between the In and Out http nodes as shown in Figure 8.
- The HTTP In node must be configured with:
    - Method: GET
    - A definition for the endpoint such as: URL = /mes/blue_latest
- The HTTP Output node does not need any configuration.

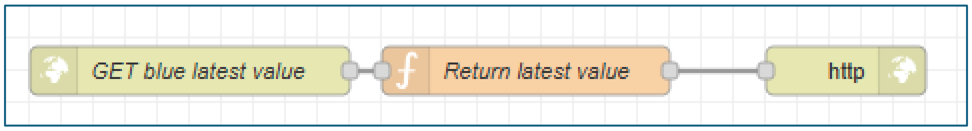
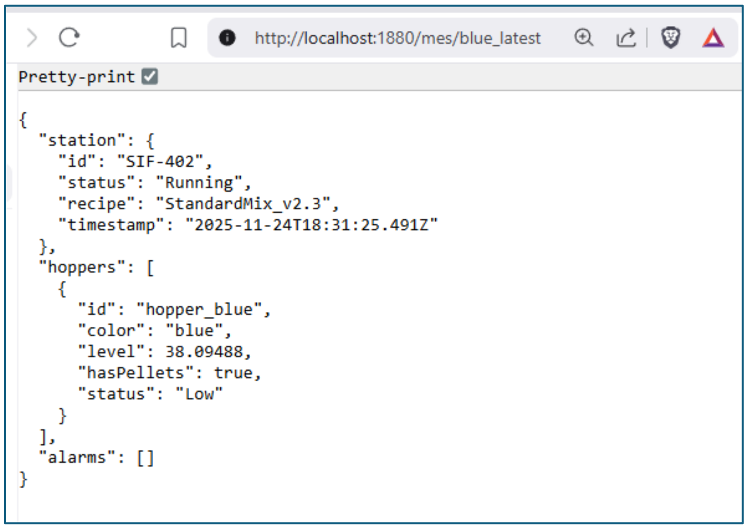


Figure 8 - Implementation of an API REST endpoint in node-red

To confirm that the API endpoint is working, the following URL must be entered on a browser: http://localhost:1880/mes/blue_latest and the expected answer would look similar to the Figure 9.



```
{
  "station": {
    "id": "SIF-402",
    "status": "Running",
    "recipe": "StandardMix_v2.3",
    "timestamp": "2025-11-24T18:31:25.491Z"
  },
  "hoppers": [
    {
      "id": "hopper_blue",
      "color": "blue",
      "level": 38.09488,
      "hasPellets": true,
      "status": "Low"
    }
  ],
  "alarms": []
}
```

Figure 9 - Response of the API endpoint implemented on node-red

Challenges:

Using node-red HTTP nodes to implement API REST is relatively simple, however the previously mentioned videos help to understand how to do it correctly. Also is advisable to have a basic understanding of how API REST endpoints work.

User Story 3 – To get data from level 1 into Unity logic part

Description:

As a data/API developer, I want to verify access to the PLC values through the node-red REST endpoint from Unity side.

Task 1 - To get familiarized with C# and Unity.

Description:

I learned C/C++ programming during my bachelor and used C# on my last job, so that part was pretty much straightforward.

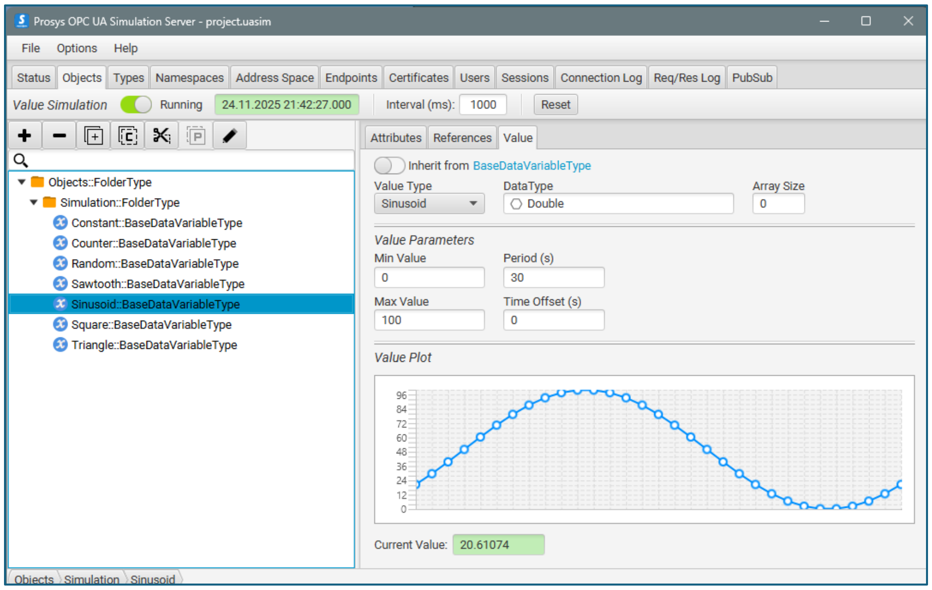
Unity, however, presented a mayor challenge and to ease this learning curve my teammate Hamza elaborated a step-by-step guide on how to deploy a mockup application and Game-scene on Unity. Thanks to this, I was able to build a basic script for getting data from the API and show it on the scene.

Task 2 - To deploy simulated data through the node-red API.

Description:

Before proceeding to implement the logic in Unity to get access to node-red data, it seemed logical to create simulated signals so the development and testing could be done without requiring to be connected to the PLC network. To this end, the following were done:

- The OPC UA simulator server was installed. Three arbitrary signals were selected with values ranging from 0 to 100: Random, Triangle and Sinusoid (see Figure 10).
- These signals were processed in other function nodes to make them look like the mockup data expected for the Unity application, as seen in Figure 11.



Prosys OPC UA Simulation Server - project.uasim

Value Simulation: Running 24.11.2025 21:42:27.000 Interval (ms): 1000

- Objects::FolderType
  - Simulation::FolderType
    - Constant::BaseDataVariableType
    - Counter::BaseDataVariableType
    - Random::BaseDataVariableType
    - Sawtooth::BaseDataVariableType
    - Sinusoid::BaseDataVariableType
    - Square::BaseDataVariableType
    - Triangle::BaseDataVariableType

Inherit from BaseDataVariableType

Value Type: Sinusoid
DataType: Double
Array Size: 0

Value Parameters
Min Value: 0
Period (s): 30
Max Value: 100
Time Offset (s): 0

Current Value: 20.61074

Figure 10 - Example of a sinusoid simulated signal in OPC UA simulator

The code implemented in the “change” node *ChangeForUnity*, which takes a single simulated signal and converts it into the data expected on the Unity scene prepared by my teammate is below:

```
{
    "station": {
        "id": "SIF-402",
        "status": "Running",
        "recipe": "StandardMix_v2.3",
        "timestamp": $now()
    },
    "hoppers": [
        {
            "id": "hopper_blue",
            "color": "blue",
            "level": payload.read.value.value,    /* real live value */
            "hasPellets": payload.read.value.value > 20,
            "status": payload.read.value.value > 50 ? "OK" : "Low"
        }
    ],
    "alarms": payload.read.value.value < 20 ?
        [
            {
                "id": "alarm_low",
                "severity": "warning",
                "message": "Hopper level low",
                "timestamp": $now()
            }
        ]
    : []
}
```

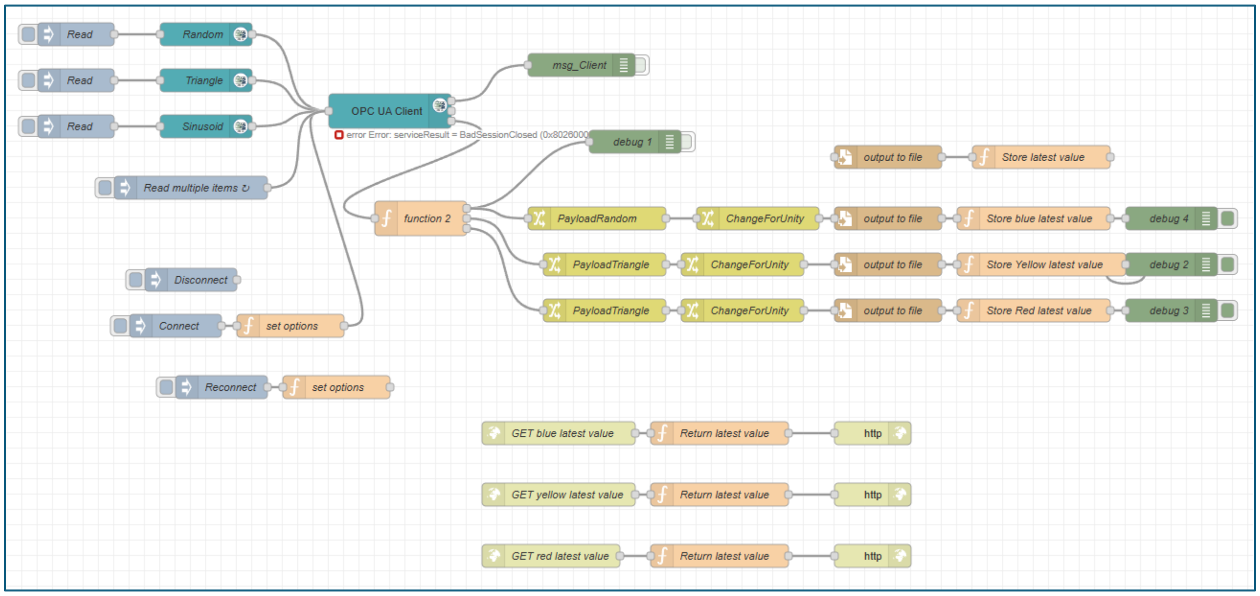
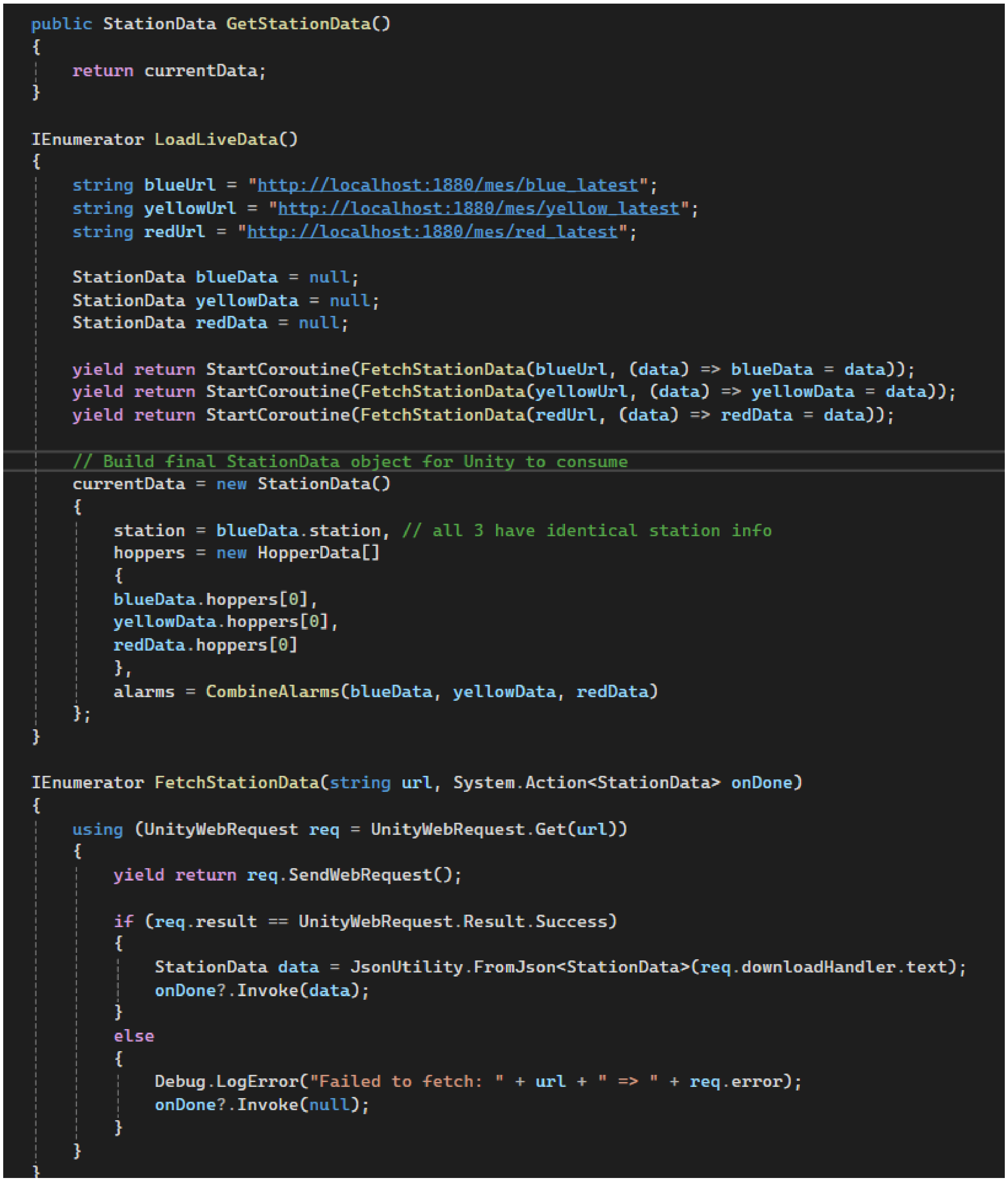


Figure 11 - Node-red flow implemented to provide three simulated signals through a node-red API REST, the code shown previously is implemented in each of the ChangeForUnity nodes.

Task 3 - To verify access to the node-red data from Unity.

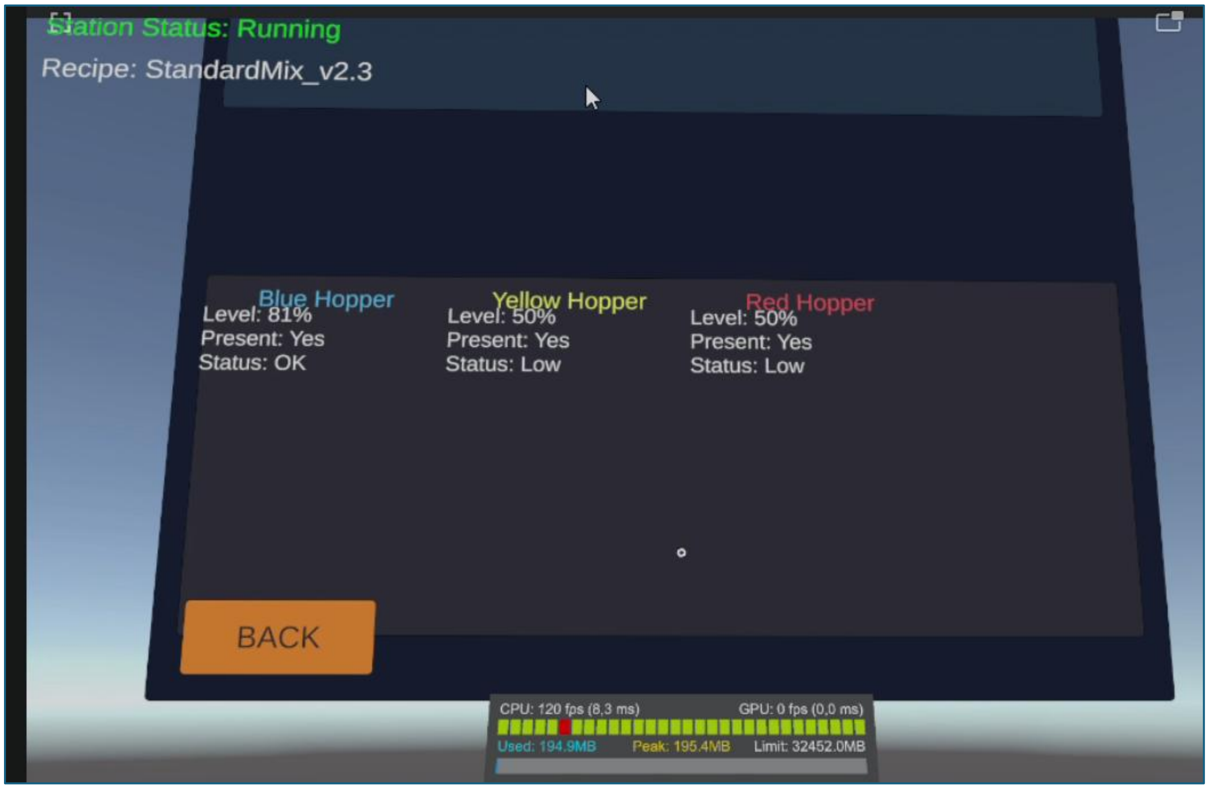
Description:

A panel was implemented on Unit with Text (TMP) associated with a C# script that reads data from the API endpoints, a section of the code is shown in Figure 12, and the results can be seen in Figure 13.



```
public StationData GetStationData()
{
    return currentData;
}

IEnumerator LoadLiveData()
{
    string blueUrl = "http://localhost:1880/mes/blue_latest";
    string yellowUrl = "http://localhost:1880/mes/yellow_latest";
    string redUrl = "http://localhost:1880/mes/red_latest";

    StationData blueData = null;
    StationData yellowData = null;
    StationData redData = null;

    yield return StartCoroutine(FetchStationData(blueUrl, (data) => blueData = data));
    yield return StartCoroutine(FetchStationData(yellowUrl, (data) => yellowData = data));
    yield return StartCoroutine(FetchStationData(redUrl, (data) => redData = data));

    // Build final StationData object for Unity to consume
    currentData = new StationData()
    {
        station = blueData.station, // all 3 have identical station info
        hoppers = new HopperData[]
        {
        blueData.hoppers[0],
        yellowData.hoppers[0],
        redData.hoppers[0]
        },
        alarms = CombineAlarms(blueData, yellowData, redData)
    };
}

IEnumerator FetchStationData(string url, System.Action<StationData> onDone)
{
    using (UnityWebRequest req = UnityWebRequest.Get(url))
    {
        yield return req.SendWebRequest();

        if (req.result == UnityWebRequest.Result.Success)
        {
            StationData data = JsonUtility.FromJson<StationData>(req.downloadHandler.text);
            onDone?.Invoke(data);
        }
        else
        {
            Debug.LogError("Failed to fetch: " + url + " => " + req.error);
            onDone?.Invoke(null);
        }
    }
}
```

Figure 12- A section of C# code implemented to fetch data from node-red API


Station Status: Running
Recipe: StandardMix_v2.3

Blue Hopper
Level: 81%
Present: Yes
Status: OK

Yellow Hopper
Level: 50%
Present: Yes
Status: Low

Red Hopper
Level: 50%
Present: Yes
Status: Low

Figure 13 - Screen capture of the panel running on the Unity scene showing simulated data from node-red

Task 4 - To implement access to node-red real PLC data from Unity.

Description:

A final change was made so we could display actual PLC data in Unity when connected to the PLC Wi-Fi in the laboratory, to this end we replaced the simulated signals with actual ones, as can be seen in Figure 14.

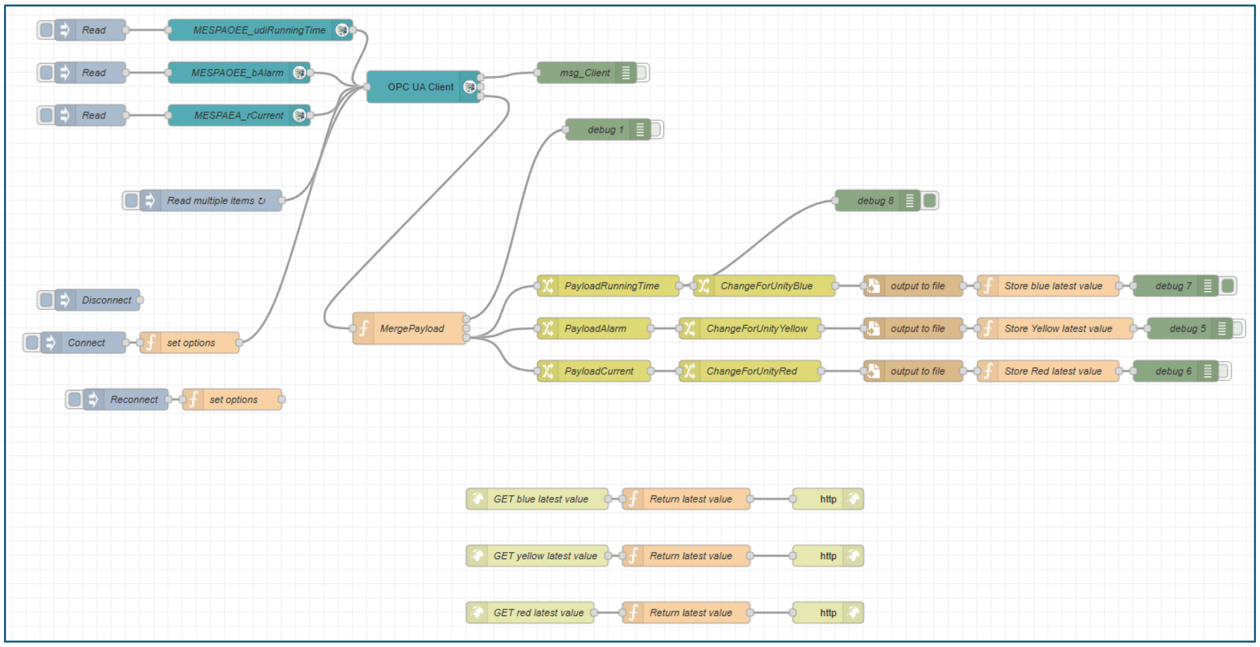


Figure 14 - Node-red flow implemented to provide three real PLC signals via node-red REST API

Challenges:

Using Unity for the first time is definitively a daunting endeavor so the tutorials provided in the course page were helpful, however the most straightforward approach in my opinion is to be mentored by someone with experience or who has already played around with it like in my case my teammate as mentioned before. Some useful videos, although outdated, can be found on this YouTube channel.

Knowing how to code in C# is not strictly required, specially with AI help, but of course is necessary to understand the logic we want to implement and the interaction between the game objects in Unity and the code behind.

Important: Before attempting deployment to Hololens 2

To access the node-red API endpoint:

We ran node-red on a computer and for this to be accessible by another device in the network like the Hololens2 it is necessary to either turn the firewall off (recommended only for initial testing) or to create an inbound firewall rule for TCP port 1880.

Ensure Node-RED is listening on all interfaces

In Node-RED settings.js, verify this line is enabled:

uiHost: "0.0.0.0",

If it is set to 127.0.0.1 or "localhost", it will sometimes respond once due to loopback quirks but refuse external connections afterward.

Self-Assessment for IPP 1

To conclude this first version of the portfolio, I consider that my contribution has been solid. I have established the foundation and the pipeline for acquiring process data, provided support to my teammates, suggested the structure for the project Git repository, and fulfilled the responsibilities of the project leader. As a team, we have successfully overcome several challenges and implemented a basic pipeline that shows real data in a Unity application deployed on the Hololens 2.

Here concludes the first part of the portfolio.

Individual Project Portfolio 2

Student Name: Eric Marquez

Role in the project:

Level 1 connectivity, data pipeline implementation and project leader.

Sprints 4 and 5:

User Story - To reformat the data from SIF 402

Description:

The previous sprint served as a test for extracting, processing and displaying data from SIF402, however after running some tests in Unity we think this data must be reformatted to make a single API call.

Task - To reformat data from SIF402

Description:

To make a single API call from Unity, we grouped together in a single JSON: operational signals from SIF402: instantaneous electrical current consumption, station running time and alarm status. Together with hopper's status signals, this is, for each hopper we have its presences status (true/false) and minimum level (true/false).

The node-red flow implemented is shown in Figure 15.

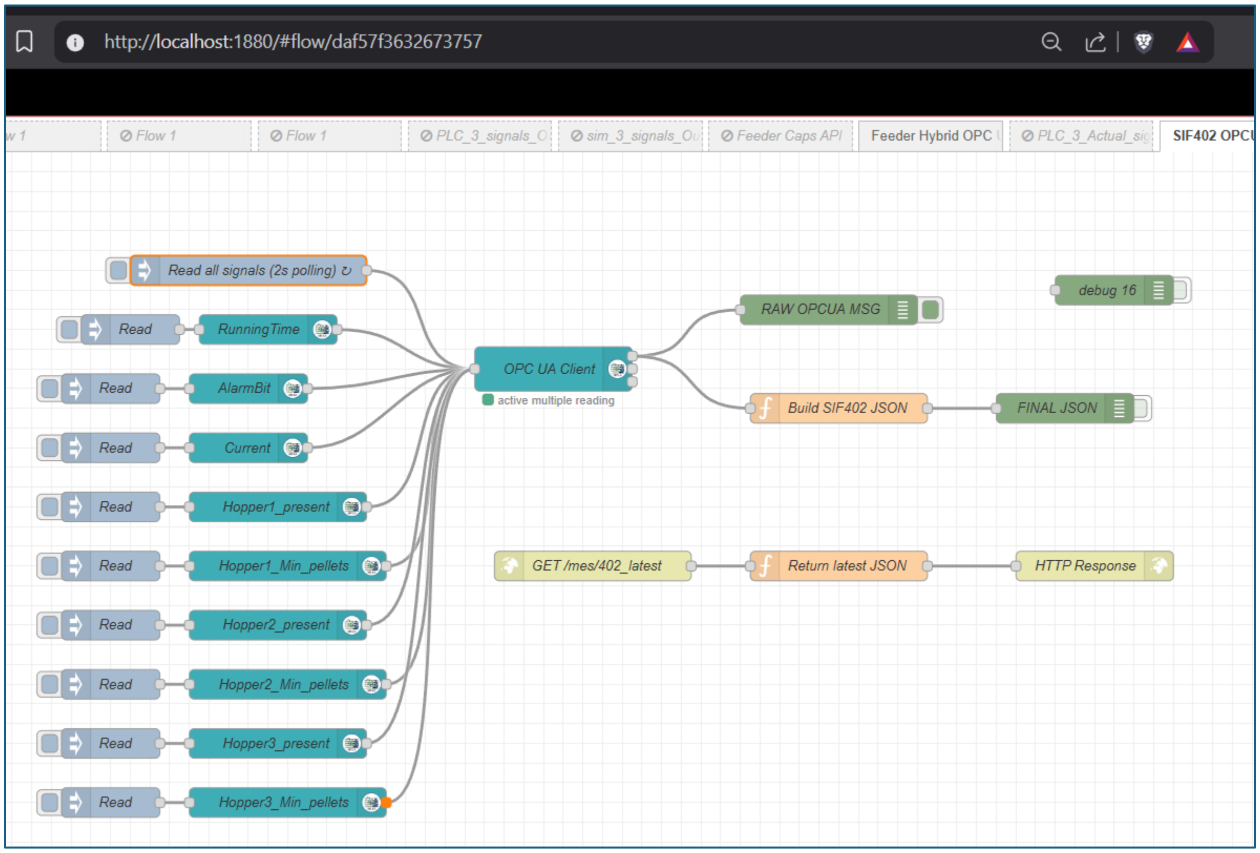
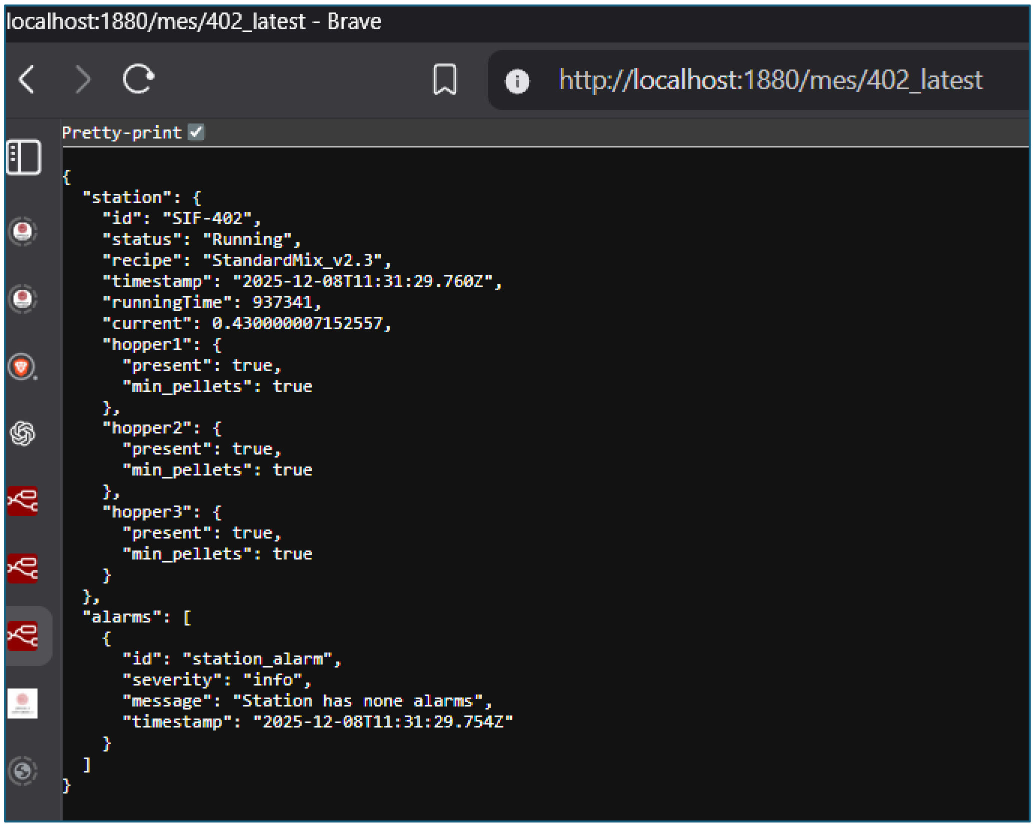


Figure 15 - node-red flow implemented to get together operational and hopper's data from SIF402

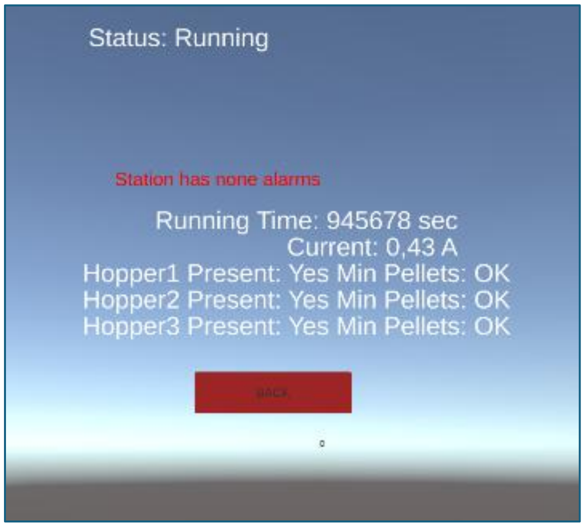
The API end point output is displayed in Figure 16.


localhost:1880/mes/402_latest - Brave
http://localhost:1880/mes/402_latest
Pretty-print ✓

```
{
  "station": {
    "id": "SIF-402",
    "status": "Running",
    "recipe": "StandardMix_v2.3",
    "timestamp": "2025-12-08T11:31:29.760Z",
    "runningTime": 937341,
    "current": 0.430000007152557,
    "hopper1": {
      "present": true,
      "min_pellets": true
    },
    "hopper2": {
      "present": true,
      "min_pellets": true
    },
    "hopper3": {
      "present": true,
      "min_pellets": true
    }
  },
  "alarms": [
    {
      "id": "station_alarm",
      "severity": "info",
      "message": "Station has none alarms",
      "timestamp": "2025-12-08T11:31:29.754Z"
    }
  ]
}
```

Figure 16 - API REST endpoint for SIF402 data

Finally, Figure 17 shows how the data looks in Unity, once the corresponding C# scripts are updated accordingly.



```
Status: Running

Station has none alarms

Running Time: 945678 sec
Current: 0,43 A
Hopper1 Present: Yes Min Pellets: OK
Hopper2 Present: Yes Min Pellets: OK
Hopper3 Present: Yes Min Pellets: OK
```

Figure 17 - Unity test app to show live data from SIF402

The JavaScript code in the “function node” that implements the packed JSON output is below:

```
// Extract actual OPC UA browseName
let browseName = msg.topic?.browseName;
let value = msg.payload;

// Load last stored state
let state = flow.get("402_state") || {
    station: {
        id: "SIF-402",
        status: "Unknown",
        recipe: "StandardMix_v2.3",
        timestamp: "",
        runningTime: 0,
        current: 0,
        hopper1: {
            present: true,
            minPellets: true
        },
        hopper2: {
            present: true,
            minPellets: true
        },
        hopper3: {
            present: true,
            minPellets: true
        }
    },
    alarms: []
};

// DEBUG print browseName
node.warn("Received browseName: " + browseName);

// Process based on REAL browseName values
switch (browseName) {

    case "RunningTime":
        state.station.runningTime = value;
        break;

    case "AlarmBit":
        if (value === true) {
            state.station.status = "Alarm";
            state.alarms = [
                {
                    id: "station_alarm",
                    severity: "warning",
                    message: "Station is in Alarm !",
                    timestamp: new Date().toISOString()
                }
            ];
        } else {
            state.station.status = "Running";
            state.alarms = [
                {
                    id: "station_alarm",
                    severity: "info",
                    message: "Station has none alarms",
                    timestamp: new Date().toISOString()
                }
            ];
        }
        break;

    case "Current":
        state.station.current = value;
        break;

    case "Hopper1_present":
        state.station.hopper1.present = value;
        break;

    case "Hopper1_Min_pellets":
        state.station.hopper1.minPellets = value;
        break;

    case "Hopper2_present":
        state.station.hopper2.present = value;
        break;

    case "Hopper2_Min_pellets":
        state.station.hopper2.minPellets = value;
        break;

    case "Hopper3_present":
        state.station.hopper3.present = value;
        break;

    case "Hopper3_Min_pellets":
        state.station.hopper3.minPellets = value;
        break;
}

// Update timestamp
state.station.timestamp = new Date().toISOString();

// Save back into flow context
flow.set("402_state", state);

// Output for debugging
msg.payload = state;
return msg;
```

User Story – To prepare data from SIF 405 so it can be displayed on unity

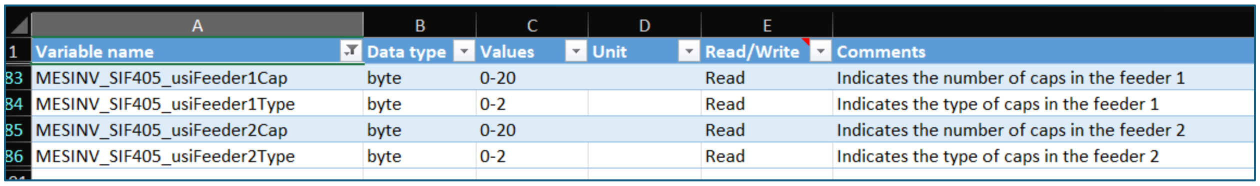
Description:

Once we have finished with SIF 402, we must work on identifying, collecting and displaying operational data from SIF405

Task - To identify data of interest in SIF 405

Description:

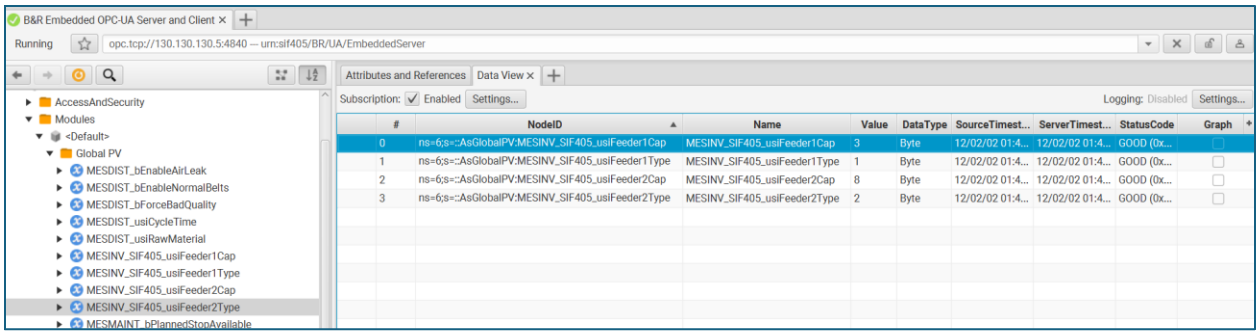
One more time, we analyze the excel file with the list of signals available on each machine and their description, in search for variables related to station SIF 405, the results are in Figure 18.



	A	B	C	D	E	
1	Variable name	Data type	Values	Unit	Read/Write	Comments
83	MESINV_SIF405_usiFeeder1Cap	byte	0-20		Read	Indicates the number of caps in the feeder 1
84	MESINV_SIF405_usiFeeder1Type	byte	0-2		Read	Indicates the type of caps in the feeder 1
85	MESINV_SIF405_usiFeeder2Cap	byte	0-20		Read	Indicates the number of caps in the feeder 2
86	MESINV_SIF405_usiFeeder2Type	byte	0-2		Read	Indicates the type of caps in the feeder 2

Figure 18 - List of variables filtered for station 405

These are the variables that contain the number and type of caps on each feeder, the next step is to find their OPC UA address through the OPC UA Browser utility, this is done by connecting to the SIF405 PLC with the Ip address: 130.130.130.5:4840 as seen in Figure 19.



B&R Embedded OPC-UA Server and Client

Running   opc.tcp://130.130.130.5:4840 — urn:sif405/BR/UA/EmbeddedServer

```
▸ AccessAndSecurity
▾ Modules
  ▾ <Default>
    ▾ Global PV
      ▸ MESDIST_bEnableAirLeak
      ▸ MESDIST_bEnableNormalBelts
      ▸ MESDIST_bForceBadQuality
      ▸ MESDIST_usiCycleTime
      ▸ MESDIST_usiRawMaterial
      ▸ MESINV_SIF405_usiFeeder1Cap
      ▸ MESINV_SIF405_usiFeeder1Type
      ▸ MESINV_SIF405_usiFeeder2Cap
      ▸ MESINV_SIF405_usiFeeder2Type
      ▸ MESMAINT_bPlannedStopAvailable
```

Attributes and References | Data View

Subscription: ✓ Enabled   Logging: Disabled

#	NodeID	Name	Value	DataType	SourceTimest...	ServerTimest...	StatusCode	Graph
0	ns=6;s=::AsGlobalPV:MESINV_SIF405_usiFeeder1Cap	MESINV_SIF405_usiFeeder1Cap	3	Byte	12/02/02 01:4...	12/02/02 01:4...	GOOD (0x...	☐
1	ns=6;s=::AsGlobalPV:MESINV_SIF405_usiFeeder1Type	MESINV_SIF405_usiFeeder1Type	1	Byte	12/02/02 01:4...	12/02/02 01:4...	GOOD (0x...	☐
2	ns=6;s=::AsGlobalPV:MESINV_SIF405_usiFeeder2Cap	MESINV_SIF405_usiFeeder2Cap	8	Byte	12/02/02 01:4...	12/02/02 01:4...	GOOD (0x...	☐
3	ns=6;s=::AsGlobalPV:MESINV_SIF405_usiFeeder2Type	MESINV_SIF405_usiFeeder2Type	2	Byte	12/02/02 01:4...	12/02/02 01:4...	GOOD (0x...	☐

Figure 19 - OPC UA Browser, showing the variables of the feeders in station SIF 405

Task - To create a node-red flow that captures the SIF405 data to serve it via API REST

Description:

Now we must implement a node-red flow to extract the data from SIF405 and make it available through an API REST endpoint in JSON format, so Unity scripts can take it from there. The flow is shown in Figure 20.

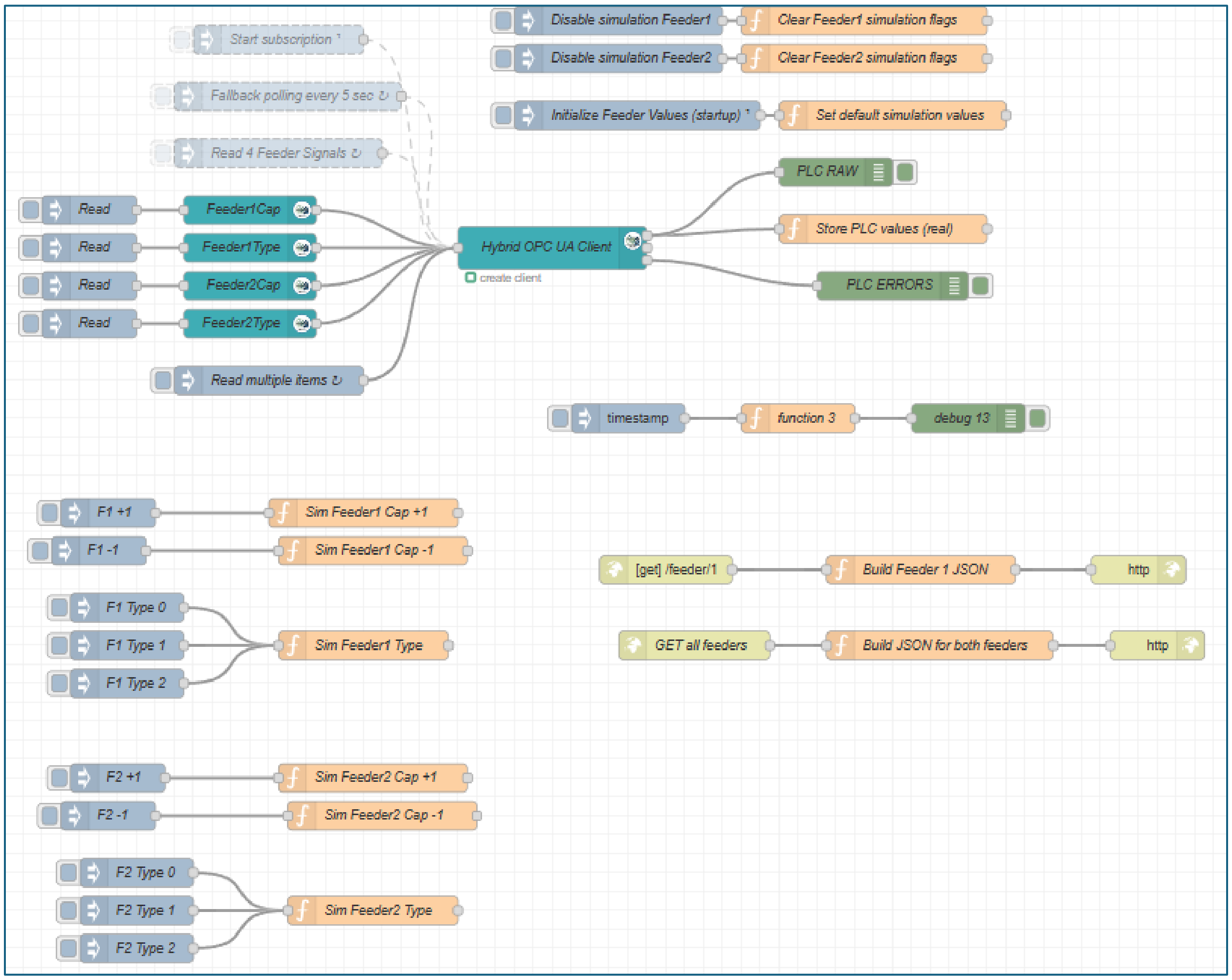
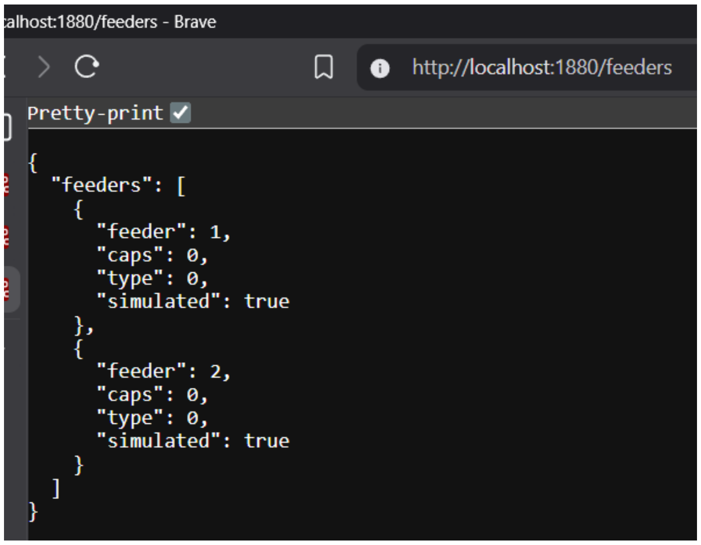


Figure 20 - node-red flow to extract SIF405 caps feeders' data. The upper section reads the actual data, while the bottom diagram is for simulation purposes.

The resulting API endpoint is seen in Figure 21.



```
{
  "feeders": [
    {
      "feeder": 1,
      "caps": 0,
      "type": 0,
      "simulated": true
    },
    {
      "feeder": 2,
      "caps": 0,
      "type": 0,
      "simulated": true
    }
  ]
}
```

Figure 21 - API REST endpoint for SIF 405 feeders' data

Figure 22 was captured from footage from Hololens 2, running the application displaying the current round cap count of 10 and the square cap count of 4.

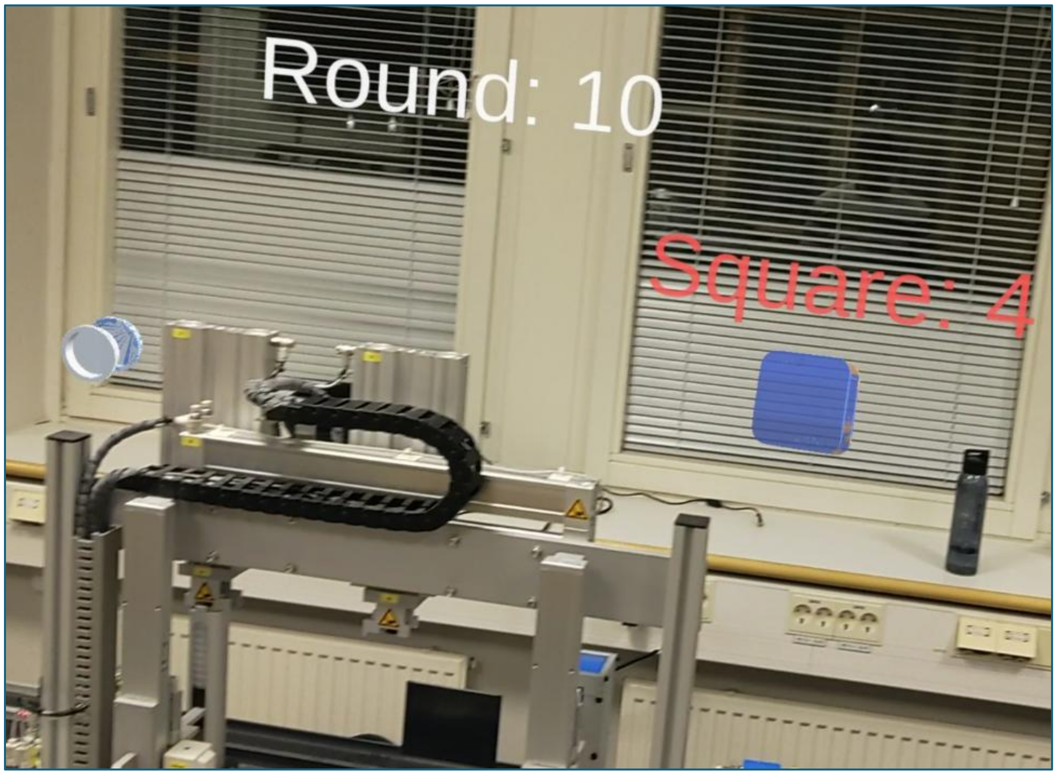


Figure 22 - Capture of footage from the HoloLens 2, running the application displaying the current round cap count of 10 and the square cap count of 4.

Task - To show the same operational data from SIF402 in SIF 405

Description:

The product owner requested to visualize in the Hololens Unity application, the same operational data we already displayed for SIF402, this is: instantaneous electrical current consumption, station running time and alarm status.

Since the OPC UA addresses for these signals are the same in any PLC, the easiest approach was to replicate the same node-red flow implemented for SIF402, remove the hoppers' signals and change the Ip address to 130.130.130.5:4840, the resulting flow can be seen in Figure 23.

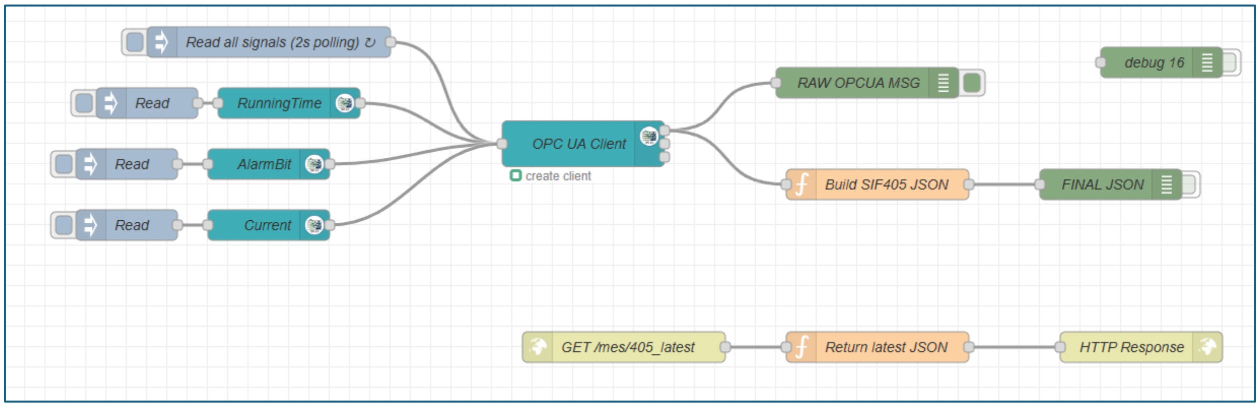


Figure 23 - node-red flow implemented to get operational data from SIF405, like Figure 15.

Figure 24 was captured from footage from Hololens 2, running the application displaying some of the values for both stations, in this case for example SIF402 (to the right) is on Alarm, while SIF405 (to the left) is on and has round cap count of 10 inside the respective feeder.

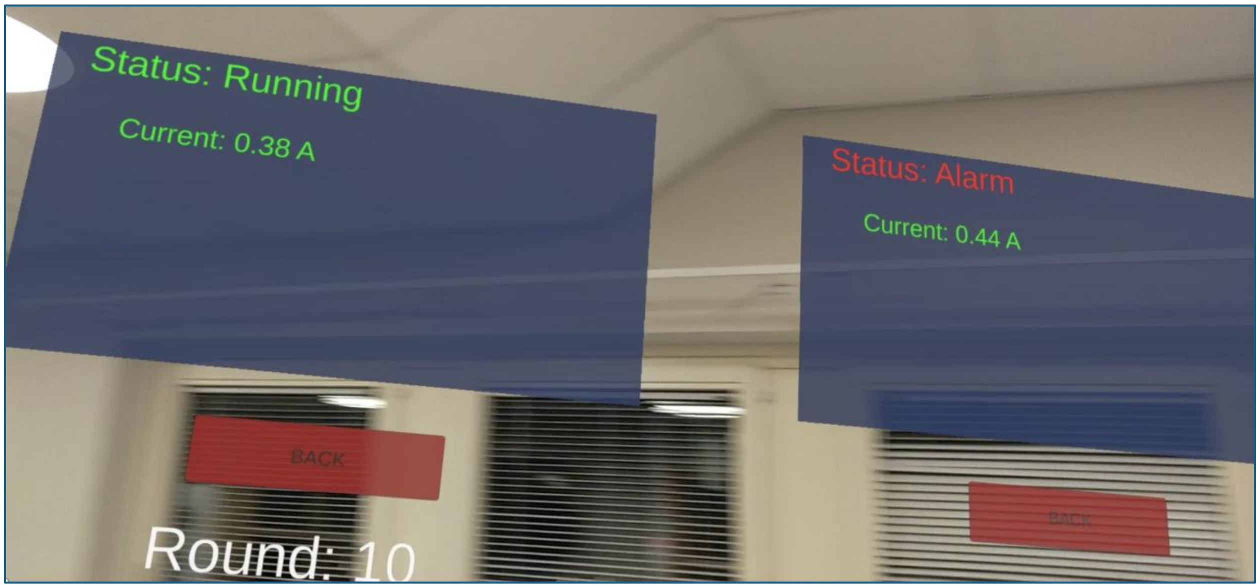


Figure 24 – Capture of footage from Hololens 2, running the application displaying some of the values for both stations, SIF402 (to the right) is on Alarm, while SIF405 (to the left) is on and has round cap count of 10 inside the respective feeder.

The JavaScript code in the "function node" that implements the packed JSON output is below:

```
// Extract actual OPC UA browseName
let browseName = msg.topic?.browseName;
let value = msg.payload;

// Load last stored state
let state = flow.get("405_state") || {
    station: {
        id: "SIF-405",
        status: "Unknown",
        recipe: "StandardMix_v2.3",
        timestamp: "",
        runningTime: 0,
        current: 0
    },
    alarms: []
};

// DEBUG print browseName
node.warn("Received browseName: " + browseName);

// Process based on REAL browseName values
```

```
switch (browseName) {

    case "RunningTime":
        state.station.runningTime = value;
        break;

    case "AlarmBit":
        if (value === true) {
            state.station.status = "Alarm";
            state.alarms = [
                {
                    id: "station_alarm",
                    severity: "warning",
                    message: "Station is in Alarm !",
                    timestamp: new Date().toISOString()
                }
            ];
        } else {
            state.station.status = "Running";
            state.alarms = [
                {
                    id: "station_alarm",
                    severity: "info",
                    message: "Station has none alarms",
                    timestamp: new Date().toISOString()
                }
            ];
        }
        break;

    case "Current":
        state.station.current = value;
        break;

}

// Update timestamp
state.station.timestamp = new Date().toISOString();

// Save back into flow context
flow.set("405_state", state);

// Output for debugging
msg.payload = state;
return msg;
```

Version control

Unity version control:

For the team to work in parallel on different features of the project, a GitHub repository was created:

https://github.com/Sallamhamza/unity-hololens-industrial-data.git,

However, to properly backup the files from Unity project and avoid pushing to the repo unnecessary and large files, there is a stablished procedure summarized in this video.

Node-red version control:

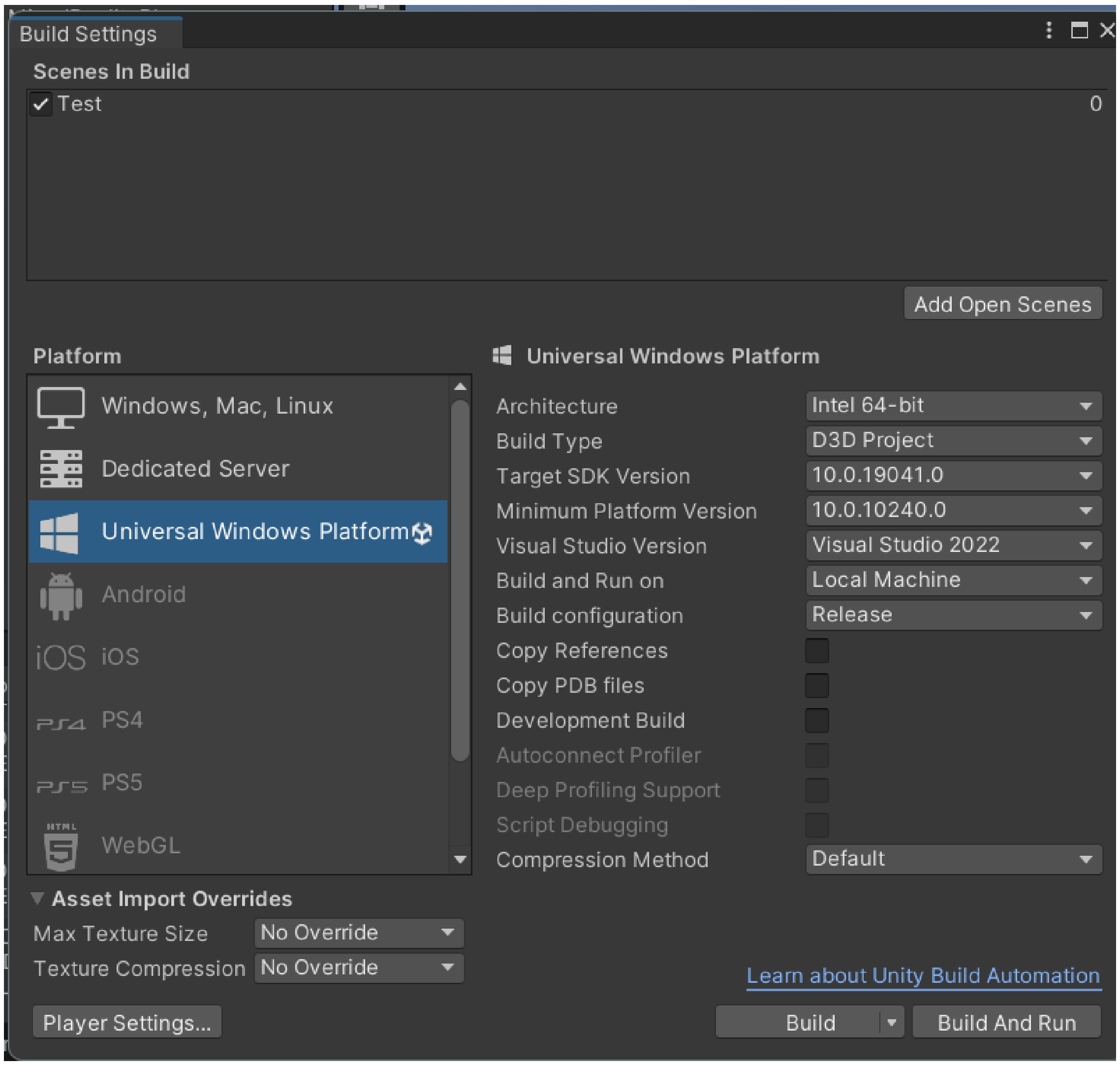
By default, node-red does not come ready to work for version control, to this end, a feature called “Projects Mode” must be enabled, which is explained here.

Steps required to deploy the Unity app into Hololens 2

Description:

Below are the steps that we as a team have put together to successfully deploy the Unity app in our specific use case.

File -> Build settings



Build Settings

Scenes In Build

✓ Test 0

Platform: Universal Windows Platform

Architecture	Intel 64-bit
Build Type	D3D Project
Target SDK Version	10.0.19041.0
Minimum Platform Version	10.0.10240.0
Visual Studio Version	Visual Studio 2022
Build and Run on	Local Machine
Build configuration	Release
Copy References	
Copy PDB files	
Development Build	
Autoconnect Profiler	
Deep Profiling Support	
Script Debugging	
Compression Method	Default

Asset Import Overrides

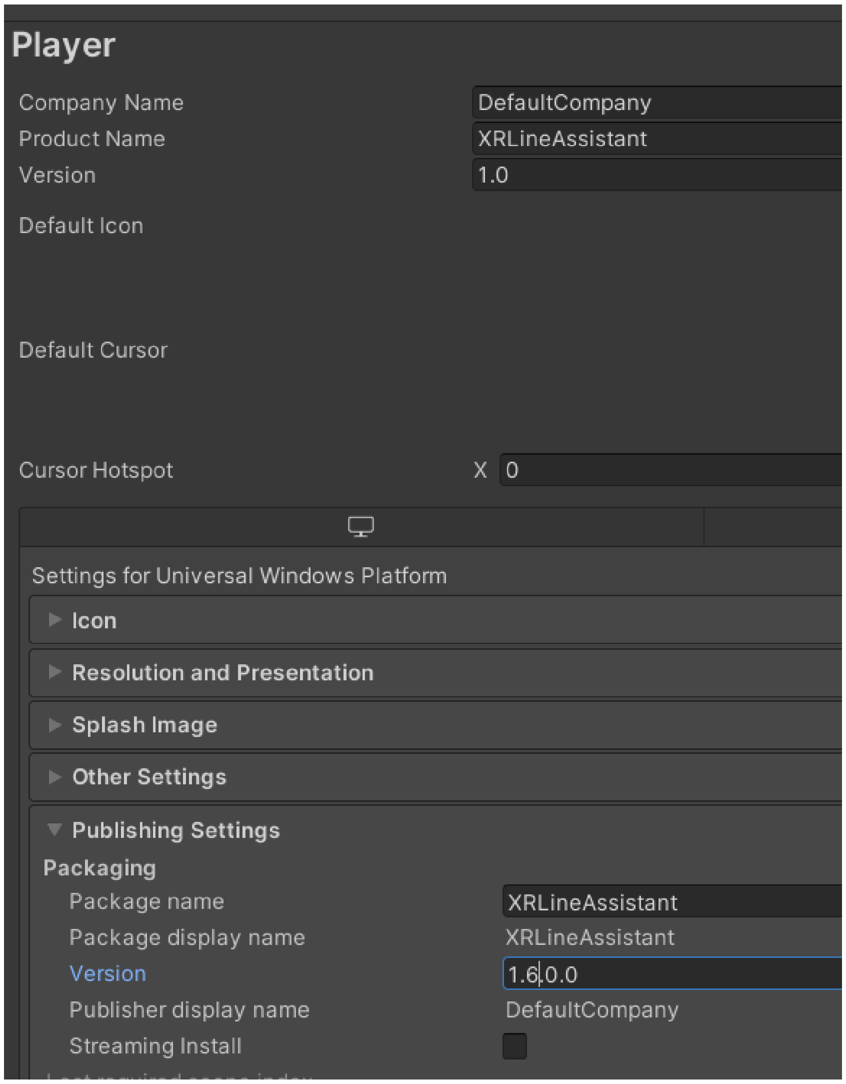
Max Texture Size	No Override
Texture Compression	No Override

Make sure the Right scene is checked (upper left) otherwise click on "Add Open Scenes" button

SDK version must match what we have installed on our PC

Visual Studio must be 22 and must be installed of course on our PC

Click on Player settings and change the version to the next one ex: 1.6.0 -> 1.7.0:



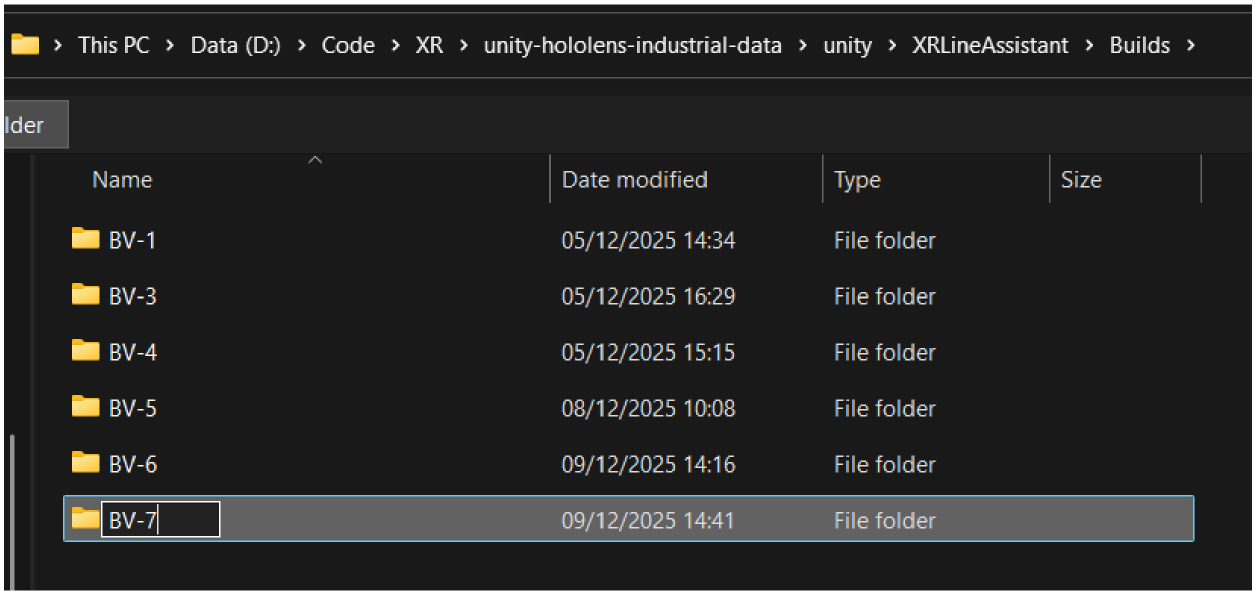
Player

Company Name: DefaultCompany
Product Name: XRLineAssistant
Version: 1.0
Default Icon
Default Cursor
Cursor Hotspot: X 0

Settings for Universal Windows Platform

Packaging
Package name: XRLineAssistant
Package display name: XRLineAssistant
Version: 1.6.0.0
Publisher display name: DefaultCompany
Streaming Install: ☐

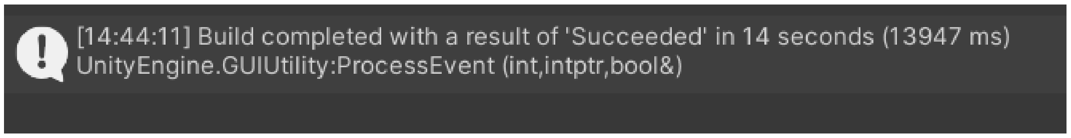
Create a new folder just for this release:


› This PC › Data (D:) › Code › XR › unity-hololens-industrial-data › unity › XRLineAssistant › Builds ›
lder

Name	Date modified	Type	Size
BV-1	05/12/2025 14:34	File folder	
BV-3	05/12/2025 16:29	File folder	
BV-4	05/12/2025 15:15	File folder	
BV-5	08/12/2025 10:08	File folder	
BV-6	09/12/2025 14:16	File folder	
BV-7	09/12/2025 14:41	File folder	

Then you go back to the Build Settings screen and click on the Build... button

If all goes well:



```
[14:44:11] Build completed with a result of 'Succeeded' in 14 seconds (13947 ms)
UnityEngine.GUIUtility:ProcessEvent (int,intptr,bool&)
```

Then we go to the folder in which the file was created.

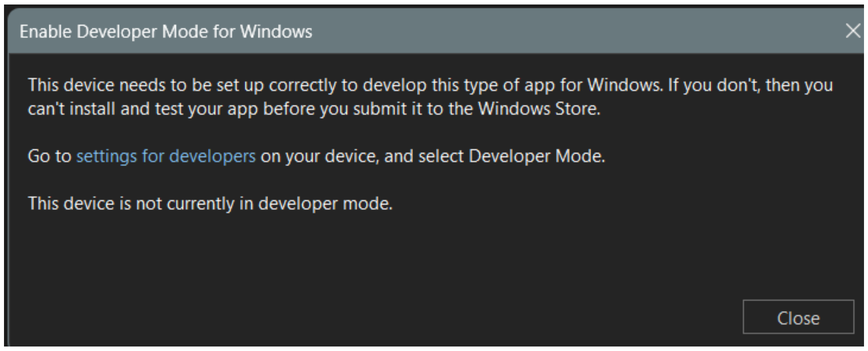
Ex: D:\Code\XR\unity-hololens-industrial-data\unity\XRLineAssistant\Builds\BV-7

There we find the project file: XRLineAssistant.sln

Right click on it, Open with -> Visual Studio 2022

In Visual Studio

You could see this:



Enable Developer Mode for Windows

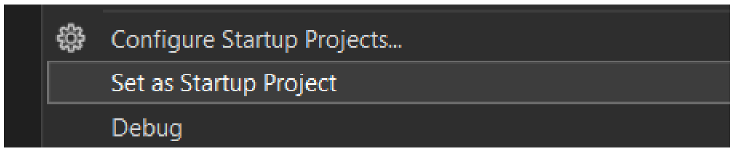
This device needs to be set up correctly to develop this type of app for Windows. If you don't, then you can't install and test your app before you submit it to the Windows Store.

Go to settings for developers on your device, and select Developer Mode.

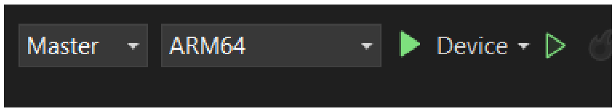
This device is not currently in developer mode.

Click Close

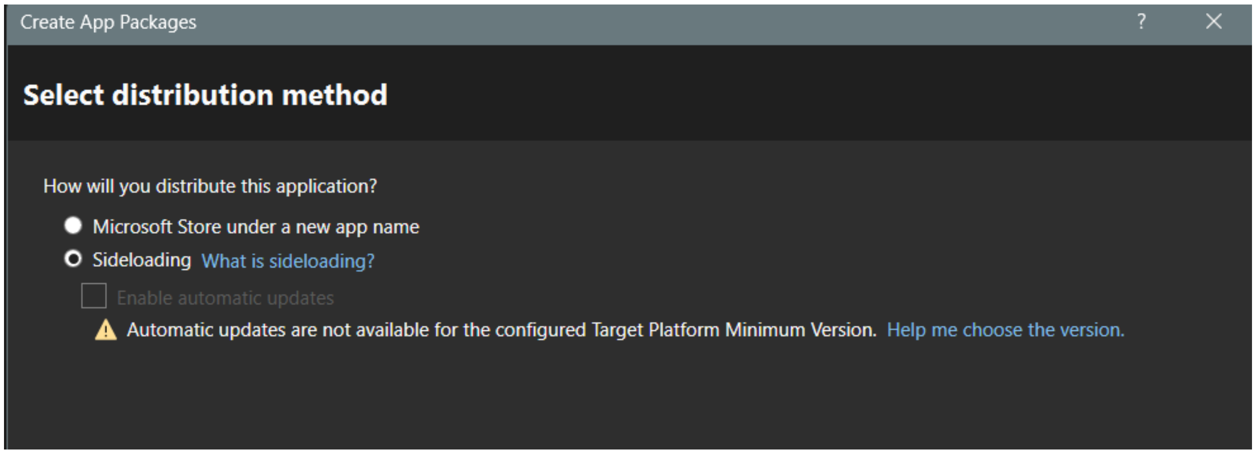
Right click on the XRLineAssistant file on the Solution Explorer



On the top of the screen put it like this: (if we are deploying into the lenses via USB cable)



Project ->Publish -> Create App packages...



Create App Packages

Select distribution method

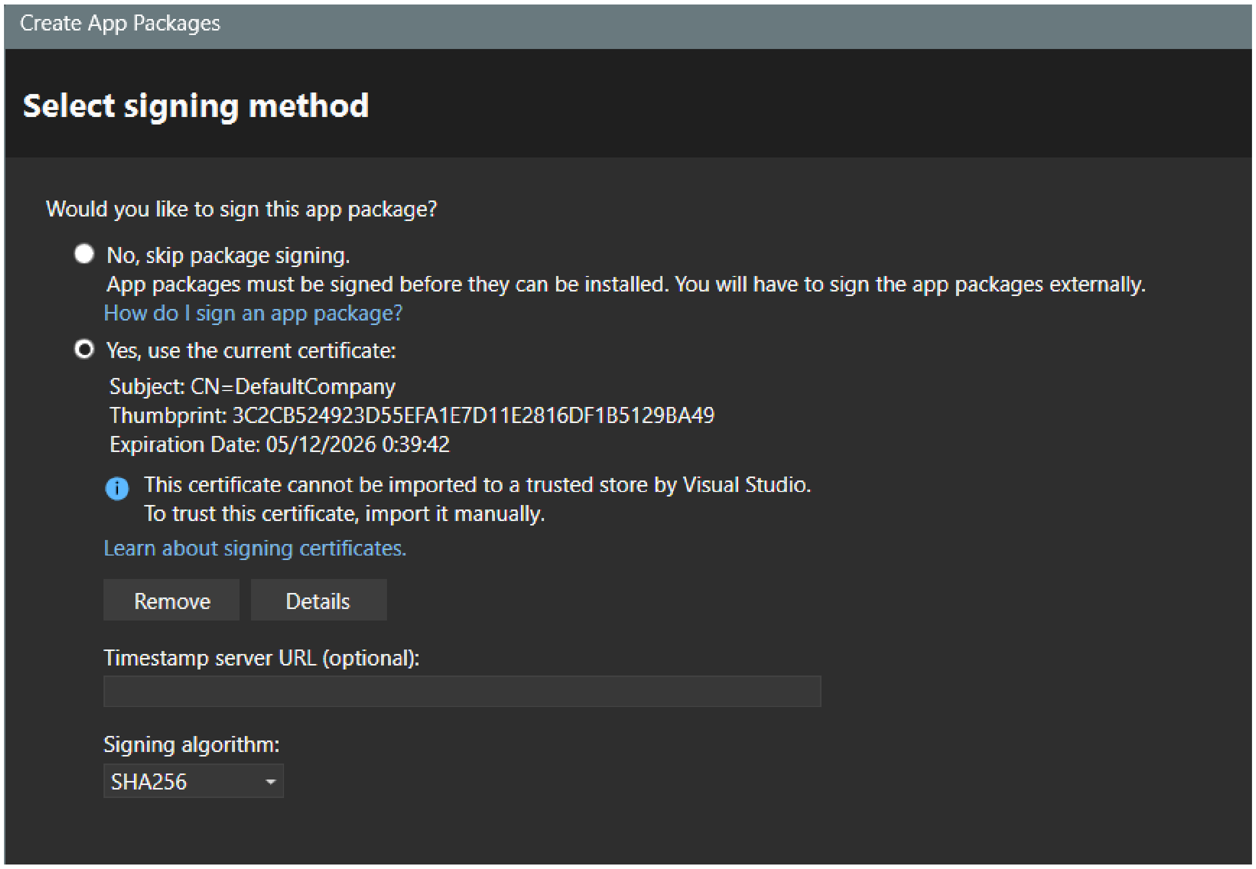
How will you distribute this application?

- Microsoft Store under a new app name
- Sideloading What is sideloading?

☐ Enable automatic updates

Automatic updates are not available for the configured Target Platform Minimum Version. Help me choose the version.

Next..



Create App Packages

Select signing method

Would you like to sign this app package?

No, skip package signing.
App packages must be signed before they can be installed. You will have to sign the app packages externally.
How do I sign an app package?

[✓ Yes, use the current certificate:]
Subject: CN=DefaultCompany
Thumbprint: 3C2CB524923D55EFA1E7D11E2816DF1B5129BA49
Expiration Date: 05/12/2026 0:39:42

This certificate cannot be imported to a trusted store by Visual Studio.
To trust this certificate, import it manually.

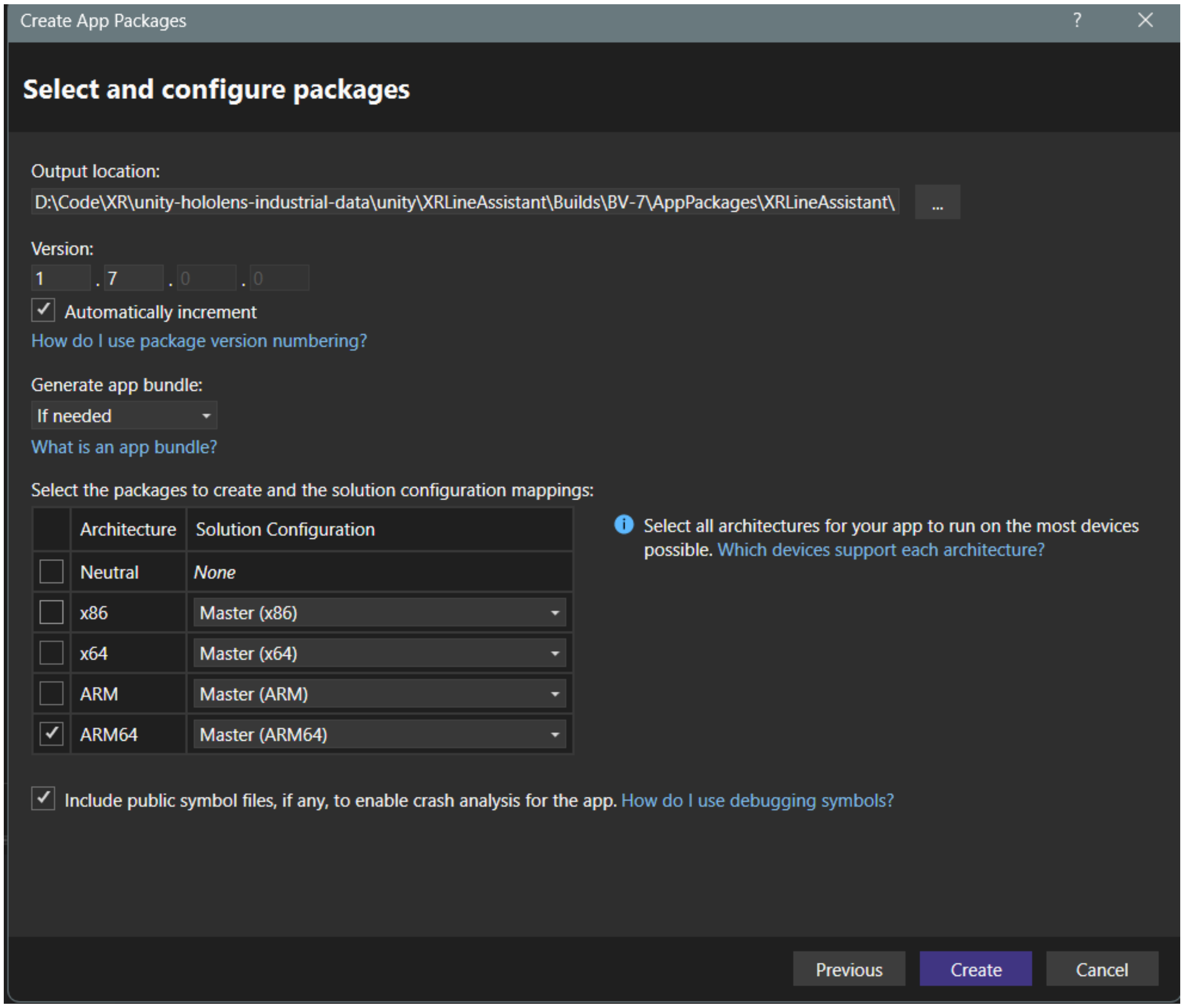
Learn about signing certificates.

Timestamp server URL (optional):

Signing algorithm: SHA256

Next:

Make sure ARM64 is the only one checked and click on Create



Create App Packages

Select and configure packages

Output location:
D:\Code\XR\unity-hololens-industrial-data\unity\XRLineAssistant\Builds\BV-7\AppPackages\XRLineAssistant\

Version:
1 . 7 . 0 . 0

[✓] Automatically increment

How do I use package version numbering?

Generate app bundle:
If needed

What is an app bundle?

Select the packages to create and the solution configuration mappings:

	Architecture	Solution Configuration
☐	Neutral	*None*
☐	x86	Master (x86)
☐	x64	Master (x64)
☐	ARM	Master (ARM)
✓	ARM64	Master (ARM64)

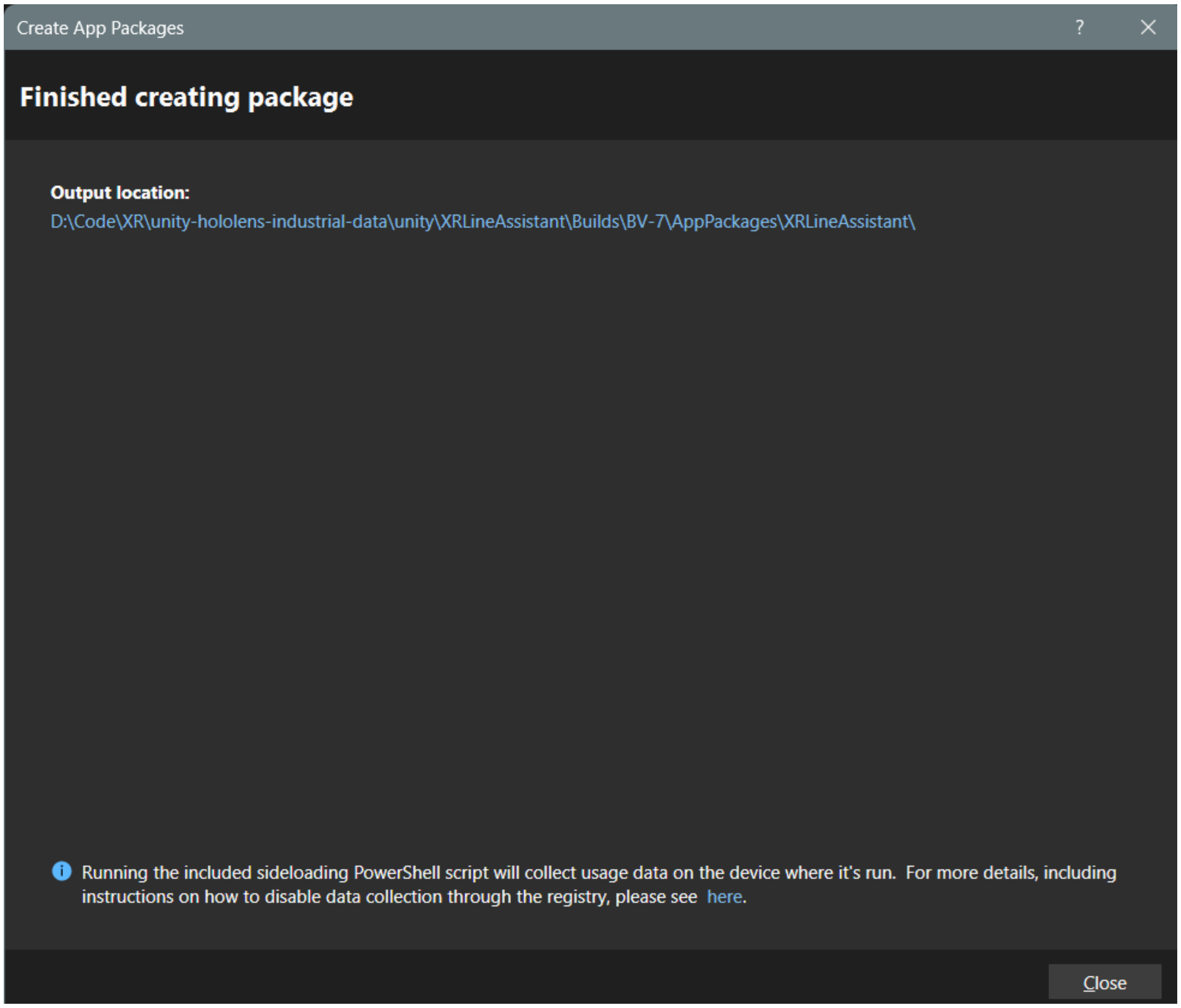
Select all architectures for your app to run on the most devices possible. Which devices support each architecture?

[✓] Include public symbol files, if any, to enable crash analysis for the app. How do I use debugging symbols?

Now we wait...

The only indicator that this is working (building the package) is this icon animated:

Then

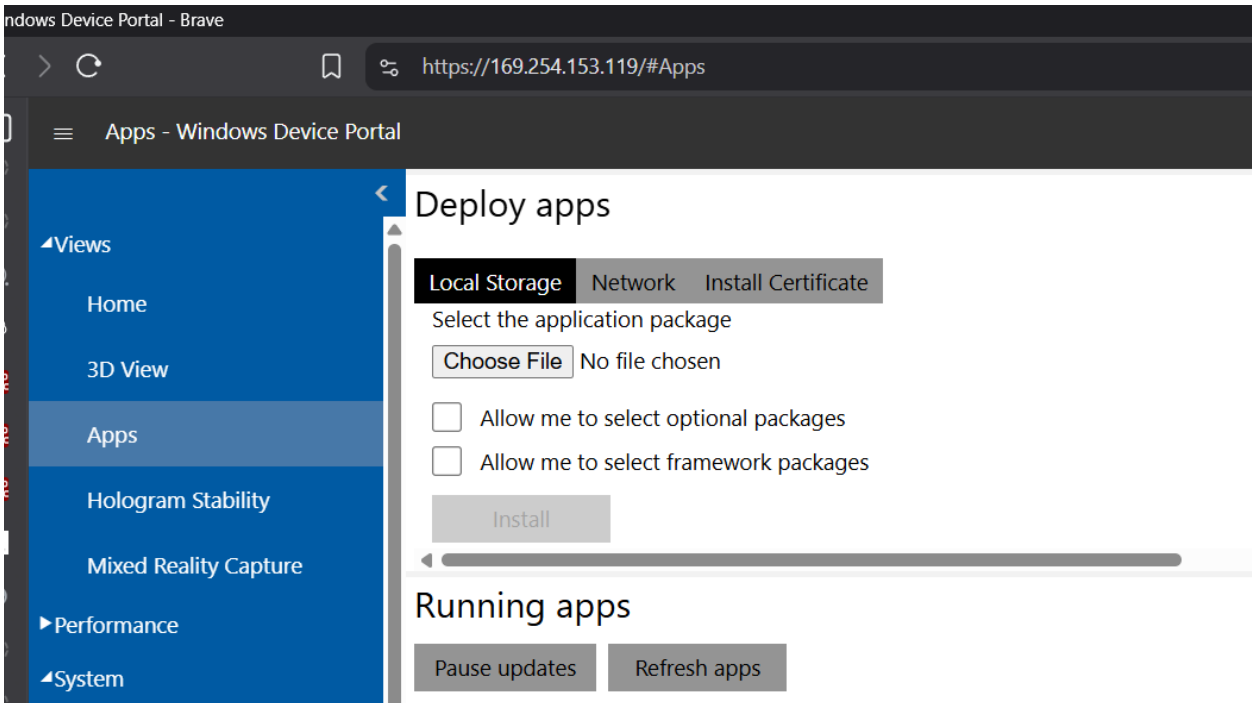


Finished creating package

Output location:
D:\Code\XR\unity-hololens-industrial-data\unity\XRLineAssistant\Builds\BV-7\AppPackages\XRLineAssistant\

Running the included sideloading PowerShell script will collect usage data on the device where it's run. For more details, including instructions on how to disable data collection through the registry, please see here.

Now. Form Windows Device portal (must know the IP address of the lenses for this)



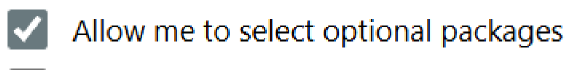
Click on Choose File

Search in the folder you just created for the .appx

"D:\Code\XR\unity-hololens-industrial-data\unity\XRLineAssistant\Builds\BV-7\AppPackages\XRLineAssistant\XRLineAssistant_1.7.0.0_ARM64_Master_Test\XRLineAssistant_1.7.0.0_ARM64_Master.appx"

Ex: XRLineAssistant_1.7.0.0_ARM64_Master.appx

Then check on

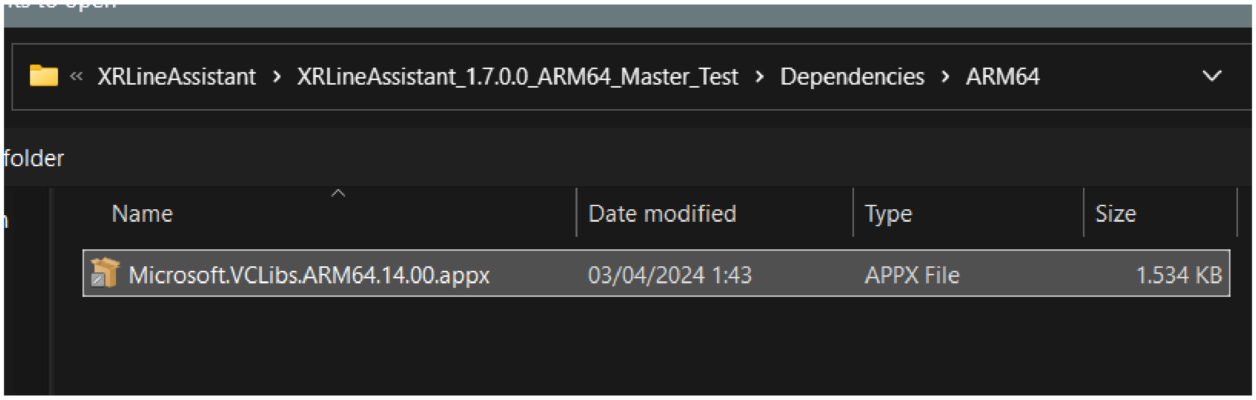


✓ Allow me to select optional packages

And Next:

Again, Choose File

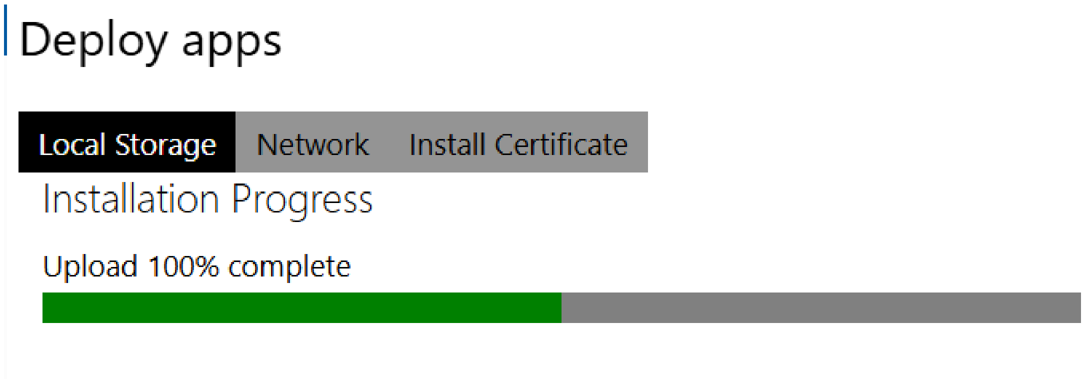
Here select:


XRLineAssistant › XRLineAssistant_1.7.0.0_ARM64_Master_Test › Dependencies › ARM64

folder

Name	Date modified	Type	Size
Microsoft.VCLibs.ARM64.14.00.appx	03/04/2024 1:43	APPX File	1.534 KB

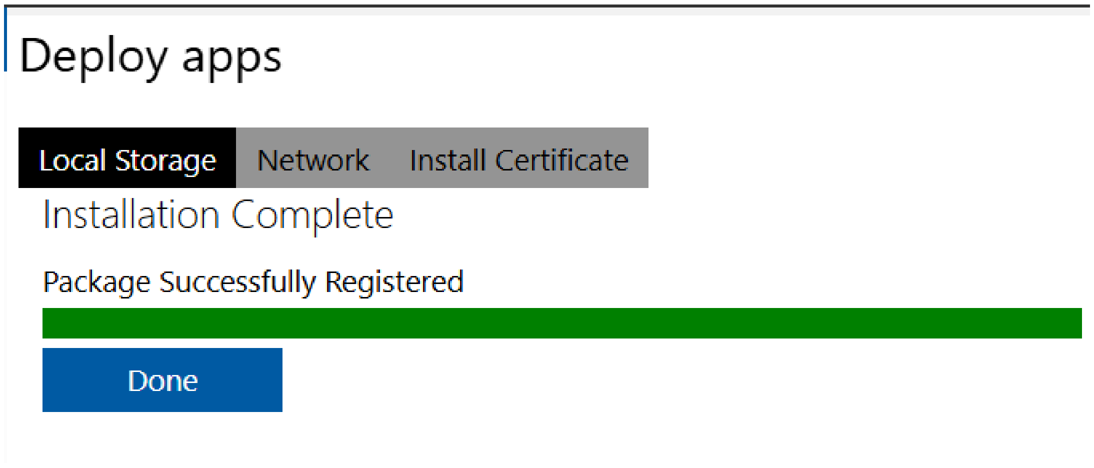
Click on Install


Deploy apps

Installation Progress

Upload 100% complete

When is Done


Deploy apps

Installation Complete

Package Successfully Registered

Self-Assessment for IPP part 2

To conclude this second version of the portfolio, I consider that my contribution has been solid. I have established and improved the foundation and the pipeline for acquiring process data, provided support to my teammates, suggested the structure for the project Git repository, participated in the tests for communication, debugging and deployment on Hololens2, and fulfilled the responsibilities of the project leader. As a team, we have successfully overcome several challenges and implemented a pipeline that shows real data in a Unity application deployed on the Hololens 2.

Peer feedback from the group

Peer	**Feedback**	**My Response**
Leon	Great ownership of node red part and pushing the group forward	Thanks.
Hamza	Organized and a problem solver who always makes sure everything works properly. He was responsible for the project's backend data access and communication of real-time data and he contributed to Unity and UI development	Thanks.
Jorge	Key enabler of the interface between the system and the XR deployment with extensive experience with PLCs and industrial equipment. His command of APIs and PLCs proved instrumental to the success of the project	Thanks.

My assessment of the team

Peer	**My Feedback**
Leon	Pushed and led the image recognition feature of the application and contributed to keeping the project implementation simple but functional.
Hamza	Hard worker, built from cero the Unity app, explained to the team the details on how the

	components interact with the scripts and how to deploy to the lenses. Tied together all the pieces into the application.
Jorge	Good overview of the tasks, keeping track of missing details, presentation slides and playing a key role organizing and hosting the podcast