milestone

Guide

Installed Integration Insights

Prepared by:
Katrine Bollmann, Business Operations Specialist,
Technology Partnerships and Open Platform

Table of Contents

1. Introduction 3

2. How-to guide 4
2.1 Plugin integrations 4
2.2 Standalone Integrations (aka Component Integrations) 7
2.3 MIP Driver Framework 9
2.4 Protocol Integrations 10
2.5 AI Bridge Integrations 12

3. Summary 13

1. Introduction

As a Milestone Technology Partner (or Developer), you need to enable the MIP SDK features related to the Installed Integration Insights initiative that allows us to gather datapoints about your integrations.

To do so, you must inform the Milestone Technology Partner Program Team of your “Manufacturer Name” and the “Integration Name” that is used in the development process, and the source code of your integration.

All integration methods provided by the MIP provide the means to apply your “Manufacturer Name” and “Integration Name” in the code.

During the enrolment and verification processes, you will be asked to provide the “Manufacturer Name” and “Integration Name” specified in your source code.

Please refer to the sections below for the different integration methods. If you have any questions, please feel free to ask in the Milestone Developer Forum.

2. How-to guide

2.1 Plugin integrations

MIP Plugin Integrations have a C# Source Code file called “MIP_Plugin_Name_Definition.CS” where “MIP_Plugin_Name_” is replaced by your plugin name.

In that file, there is a class deriving from PluginDefinition:

```
public class MIPPluginDefinition : PluginDefinition
```

And with a region called “Identification Properties”:

```
#region Identification Properties
```

This is where various properties are read for the plugin to identify itself. One of the properties is for the plugin to provide its unique Id:

```
public override Guid Id
        {
            get
            {
                return IntegrationId; // IntegrationId is a static Guid
            }
        }
```

> *Please make sure to generate a Guid that is unique for this integration and hardcode it in the plugin.*

Another is the “Manufacturer Name”:

```
public override string Manufacturer
        {
            get
            {
                return "Your Company name";
            }
        }
```

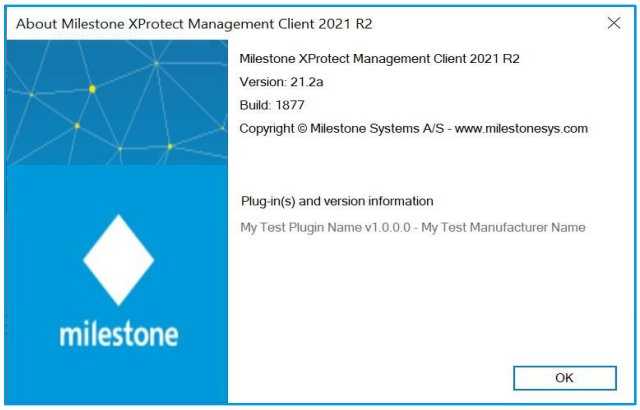
Please provide us with the exact “Manufacturer Name” and “Integration Name” which you have specified in your code. Spellings are important and are case-sensitive. The best practice could be copy-pasting it or taking a screenshot from that function.

Furthermore, in the same section, there is a property called Name in which “Integration Name” is returned.

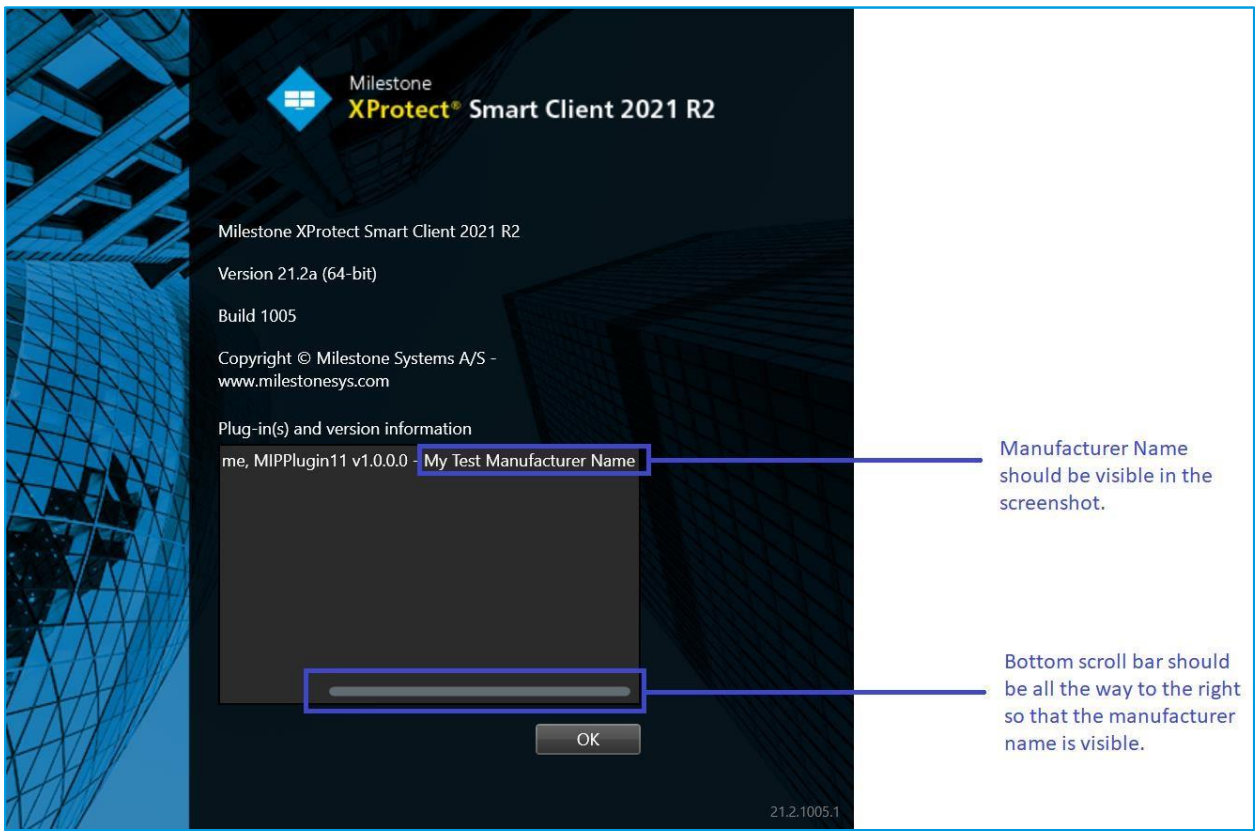
```
        /// <summary>
        /// Define name of top level Tree node - e.g. A product name
        /// </summary>

        public override string Name
        {
            get { return "This is the Integration Name!"; }
        }
```

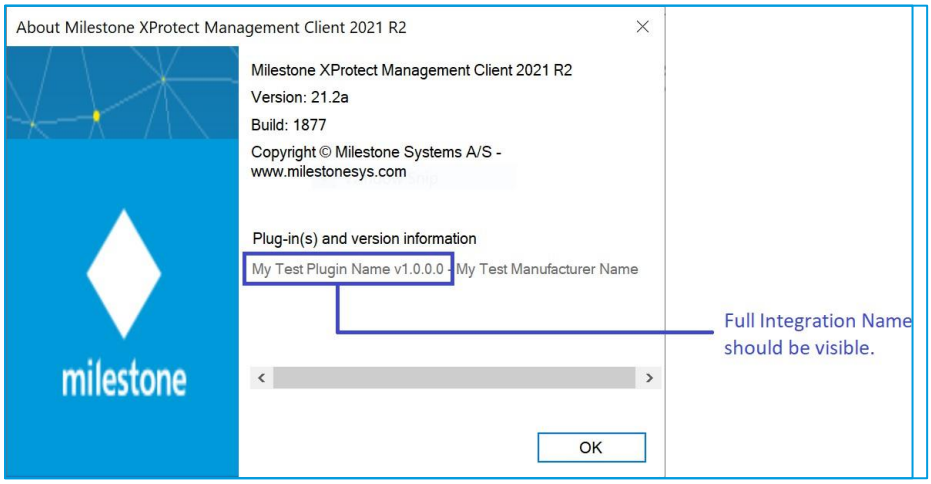
As part of the verification process, we also need a screenshot of the Management Client About Box and Smart Client About Box when your integration is loaded. Both Manufacturer Name and Integration Name should be visible in the screenshot.

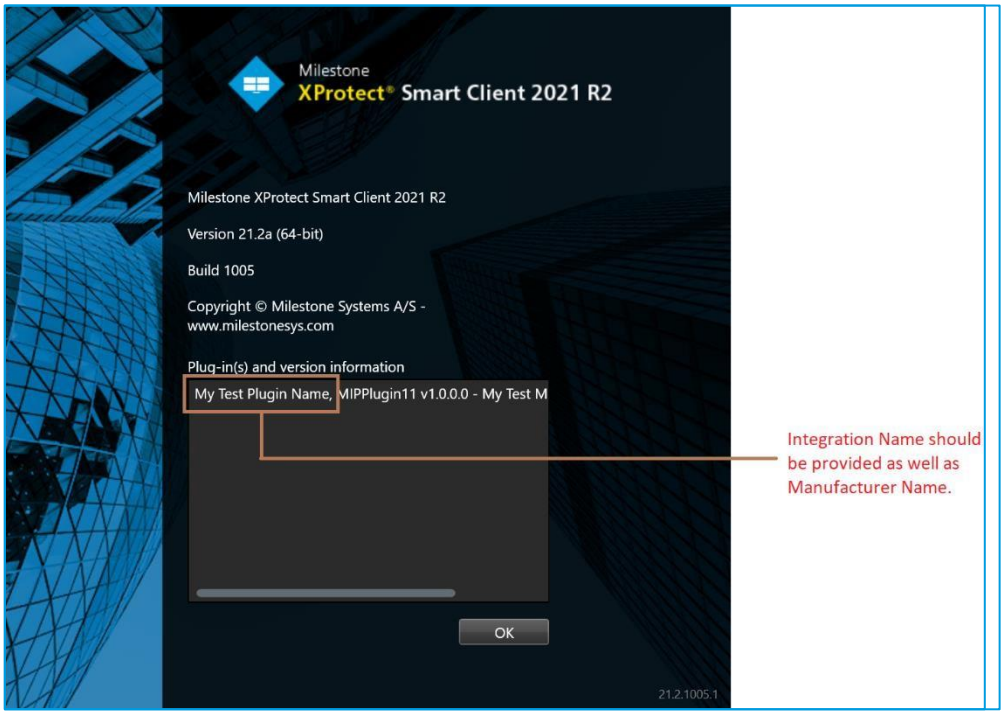


The screenshot of the Management Client About Box should display the plugin loaded. Additionally, a screenshot of the Smart Client About Box is required.



Two screenshots of Management Client About Box should be provided. One like the image above, where "Manufacturer Name" is visible. Another one, like the image below, where the integration name is visible:





2.2 Standalone Integrations (aka Component Integrations)

Standalone applications taking advantage of Components provided by XProtect also can provide the Id, the "Manufacturer Name", the Version and the "Integration Name" of the integration to MIP SDK.

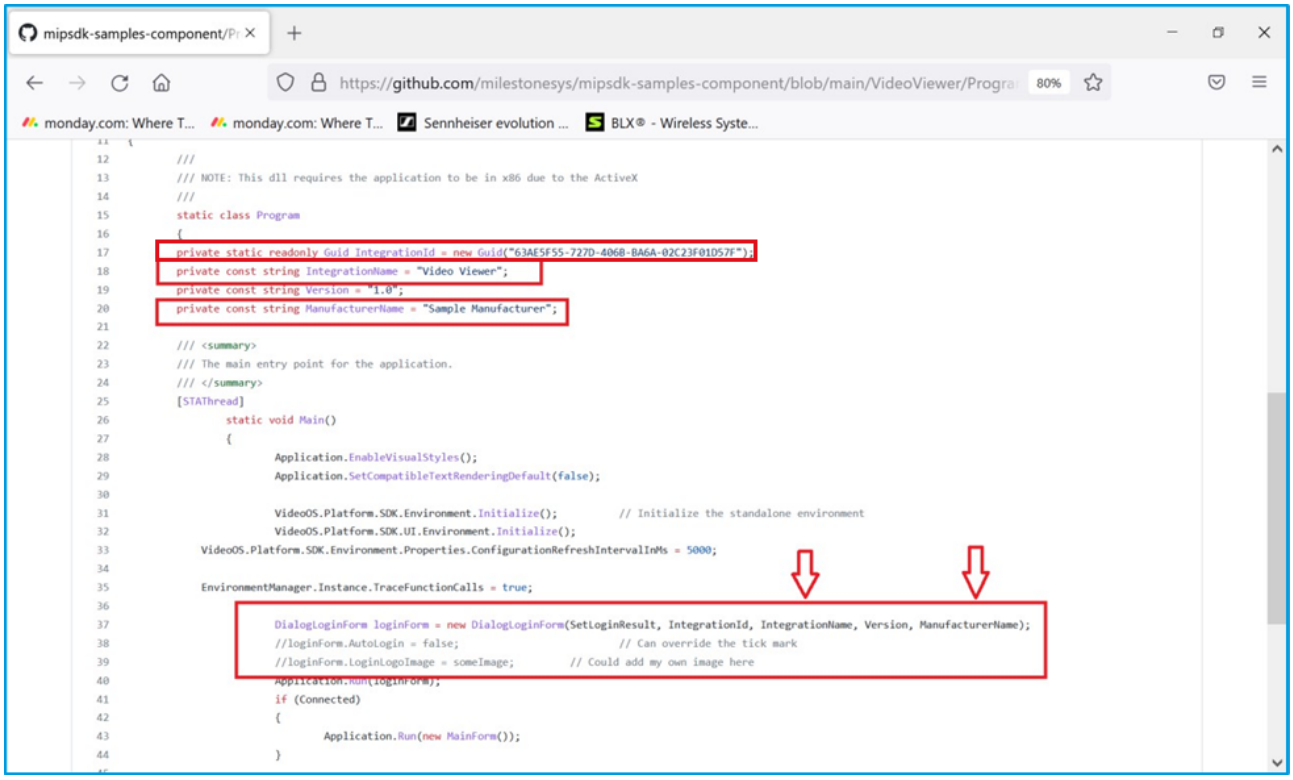
All of these can be provided as parameters to either the DialogLoginForm constructor or the Environment.Login method.

The "Manufacturer Name" in the scope of component integrations is the last parameter passed to either of these, and the "Integration Name" is the third parameter.

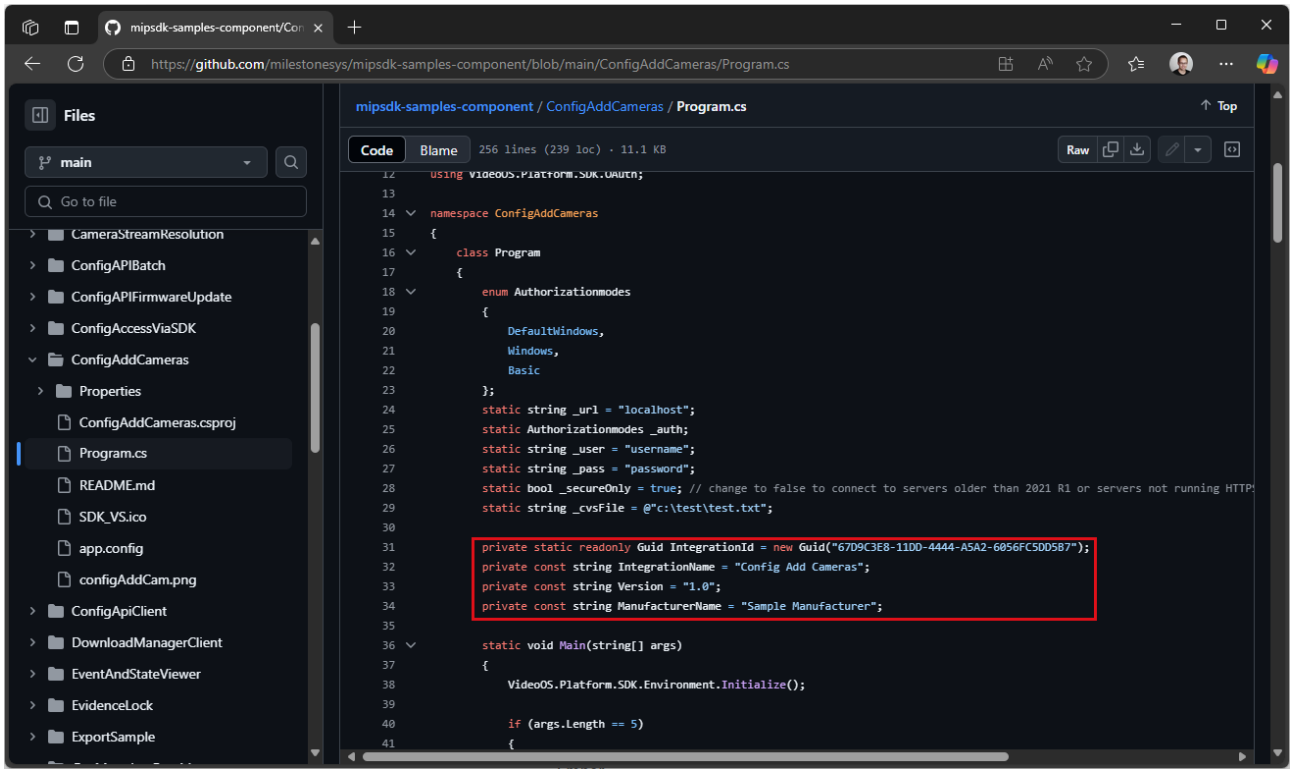
> *Please make sure once and for all to generate a Guid that is unique for this integration and hardcode it in the integration code.*

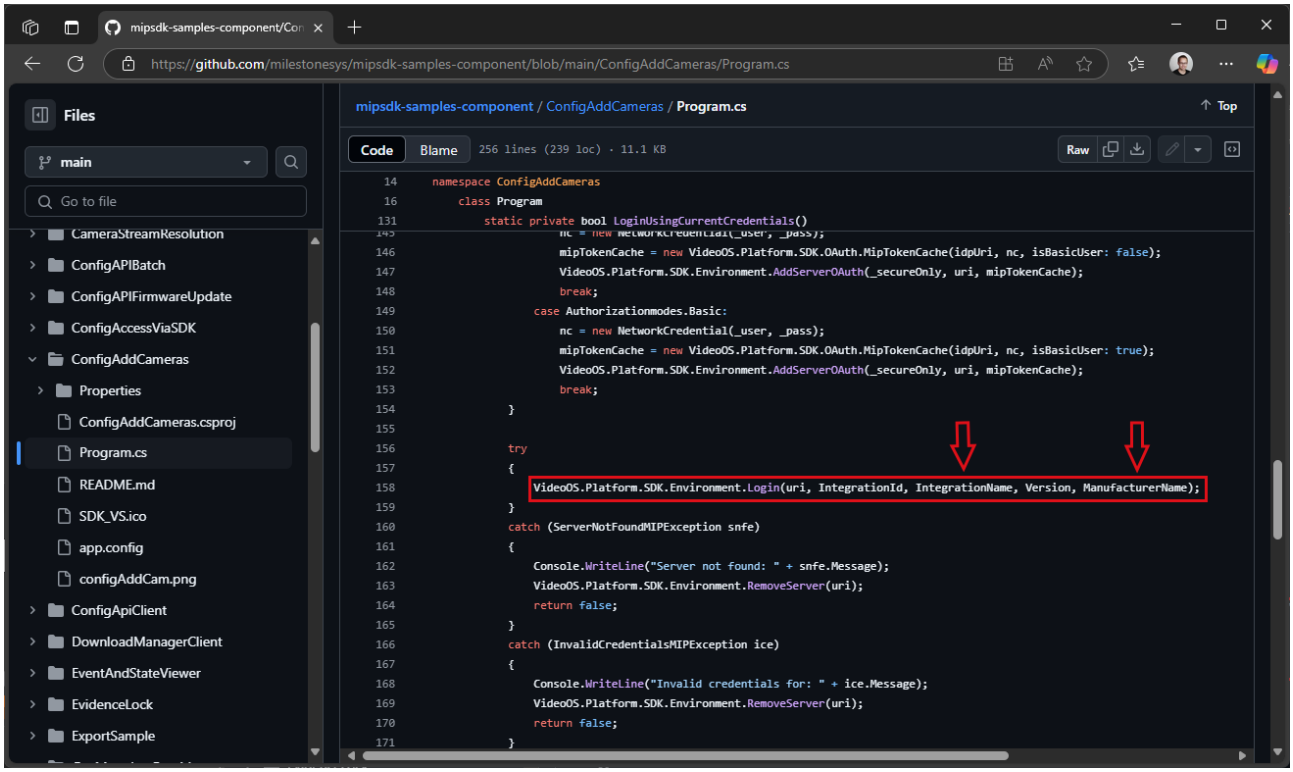
See the various examples in the following link: https://github.com/milestonesys/mipsdk-samples-component/

For example, for the VideoViewer, see below in Program.cs source code file:



Another example is the ConfigAddCameras, from which you can see two snippets of code from Program.cs below:



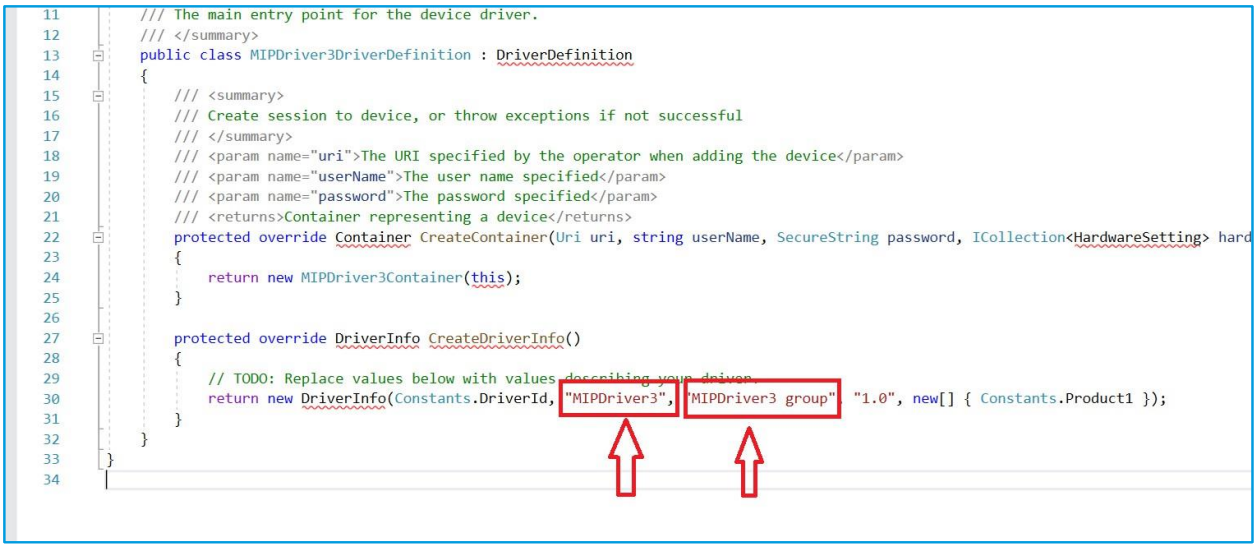


Please provide Milestone with the exact “Manufacturer Name” and “Integration Name” specified in your source code. Please observe that it is case-sensitive, so the best practice could be copy-pasting it.

2.3 MIP Driver Framework

For the case of MIP Driver Framework, Manufacturer Name and Integration Name is not defined with such name but instead XProtect considers the MIP Driver Group Name as Manufacturer Name and Driver Name as Integration Name.

The GroupName property is located on DriverInfo. When the call to the function CreateDriverInfo is made, the third parameter of the constructor specifies the MIP Driver Group Name, which XProtect considers the Manufacturer Name. This happens in the source code file ending in DriverDefinition.cs.



```
11      /// The main entry point for the device driver.
12      /// </summary>
13      public class MIPDriver3DriverDefinition : DriverDefinition
14      {
15          /// <summary>
16          /// Create session to device, or throw exceptions if not successful
17          /// </summary>
18          /// <param name="uri">The URI specified by the operator when adding the device</param>
19          /// <param name="userName">The user name specified</param>
20          /// <param name="password">The password specified</param>
21          /// <returns>Container representing a device</returns>
22          protected override Container CreateContainer(Uri uri, string userName, SecureString password, ICollection<HardwareSetting> hard
23          {
24              return new MIPDriver3Container(this);
25          }
26
27          protected override DriverInfo CreateDriverInfo()
28          {
29              // TODO: Replace values below with values describing your driver.
30              return new DriverInfo(Constants.DriverId, "MIPDriver3", "MIPDriver3 group", "1.0", new[] { Constants.Product1 });
31          }
32      }
33  }
34
```

2.4 Protocol Integrations

At the heart of any protocol integration, before using any of the SOAP services, your integration should first login to XProtect and perform authentication and get the authentication tokens necessary for later stages.

Please see the information below:
https://doc.developer.milestonesys.com/mipsdk/index.html?base=reference%2Fprotocols%2Fprotocol_authenticate.html&tree=tree_3.html

Under the section “Register your protocol integration application”, the parameters that are to be passed to ServerCommandWrapper for ServerCommandServiceOAuth.svc and ServerCommandService.svc are mentioned.

The ServerCommandWrapper for ServerCommandServiceOAuth.svc and ServerCommandService.svc includes an overloaded Login() method that takes care of the registration.
When registering your application, you provide the following information:

Parameter	Type	Description
instanceId	string	Identifier uniquely identifying the calling instance. Typically, each ID should refer to a specific machine running an integration.
integrationId	guid	Unique identifier representing the integration. Should be hardcoded in the integrating application.
integrationVersion	string	Version of the calling application.
integrationName	string	Name of the calling application.
manufacturerName	string	Name of the manufacturer of the calling application

All strings are max 256 characters.

Of course, the SOAP interface can be used in different scenarios and languages, but as one example, please see the following file, which is in C# and interfacing using SOAP.

https://github.com/milestonesys/mipsdk-samples-protocol/blob/480911f2ac2e58a5aa3d6754d95553d1639f1821/ServerCommandWrapper/BasicConnection.cs

```
/// <summary>
/// Login to the server, and register the integration if applicable
/// </summary>
/// <returns>Info of valid log in</returns>
public LoginInfo Login(Guid integrationId, string version, string integrationName)
{
        Login();
    try
    {
        Server.RegisterIntegration(_isOAuthServer ? "" : _loginInfo.Token, Constants.InstanceId, integrationId, version, integrationName, Constants.ManufacturerName);
    }
    catch
    {
        // on older systems this method does not exist so we will just silently ignore any exceptions - and with no retry. Consequences of failing are ignorable anyway
    }

    return LoginInfo;
}

/// <summary>
/// Logout from the server
/// </summary>
public void Logout()
{
    Server.Logout(_thisInstance, LoginInfo.Token);
```

The Manufacturer Name is simply the last parameter when making calls to RegisterIntegration. The Integration Name is the one before it.

```
7538    void RegisterIntegration(string token, string instanceId, System.Guid
        integrationId, string integrationVersion, string integrationName, string
        manufacturerName);
8087    public void RegisterIntegration(string token, string instanceId, System.Guid
        integrationId, string integrationVersion, string integrationName, string
        manufacturerName)
```

2.5 AI Bridge Integrations

Applications integrating using AI Bridge provide the Manufacturer Name, Integration Name etc. during the Application registration mutation as described in the Developers Reference Manual: ... #definition-**AppInput:**

AppInput

Details about App to register.

Input Field	Description
id - ID!	Unique ID of App to register. Must be a GUID, Constant for the App
name - String!	Name of App as to present itself in the VMS.
version - String	Version of the App.
manufacturer - ManufacturerInput	Manufacturer of the app.

ManufacturerInput

Manufacturer of apps.

Input Field	Description
name - String!	Name of the manufacturer.

Example from scripted mutation payload:

```
{
  id: "${APP_ID}"
  name: "${APP_NAME}"
  version: "${TAG}"
  description: "${APP_DESCRIPTION}"
  manufacturer: {
    name: "${MANUFACTURER_NAME}"
  }
```

Also see this sample that shows how a the Installed Integration Insights registration can be implemented: MIP-AIBridge-samples/apps/golang/connectivitysample at main · milestonesys/MIP-AIBridge-samples

3. Summary

As described in the previous sections, different integration methods have specific locations in the source code to identify the “Manufacturer Name” and “Integration Name” as well as the associated “integration Id”, which is basically the name of the company developing the integration and the name and an identifier of the integration developed.

In relation to Installed Integration Insights (III), Milestone Technology Partners must inform the Milestone Technology Partner Team of the “Manufacturer Name” and “Integration Name” used in their source code by sending an email to partner@milestone.dk.

The “Manufacturer Name” and “Integration Name” must be written to Milestone exactly as they are in the code (case-sensitive); therefore, we recommend copying/pasting them from the source code and also including a screenshot.

About Milestone Systems

Milestone Systems is a leading provider of data-driven video technology software in and beyond security that helps the world see how to ensure safety, protect assets, and increase business efficiency. Milestone enables an open platform community that drives collaboration and innovation in the development and use of network video technology, with reliable and scalable solutions that are proven in more than 500,000 customer sites worldwide. Founded in 1998, Milestone is a stand-alone company in the Canon Group.

www.milestonesystems.com