Setup Guide

Configuration Setup Guide Automation AIS-Plugin Executer v2.1

Updated: Date 2020-09-25

Addovation © 2020

Contents

1 Plugins development 3
2 System requirements 15
3 Compatibility matrix 16
4 Limitations 17
5 Contact 18

1 Plugins development

Plugins can easily be developed using AIS. The plugins can then run as a Windows Service or scheduled task. This chapter will describe how to develop such a plugin.

After AIS has been installed, open up Microsoft Visual Studio and select the following:

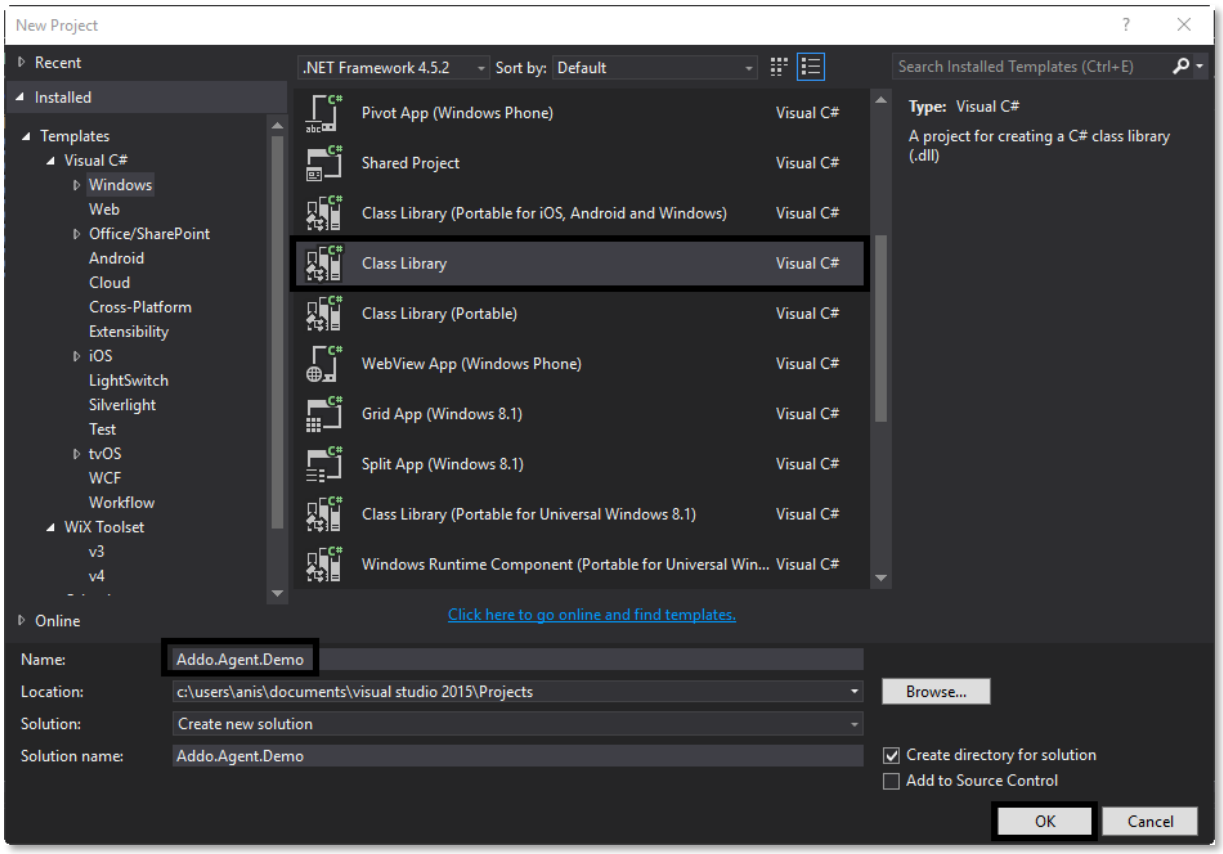

Figure 1 Create Class Library

Re-factor the naming to something more appropriate:

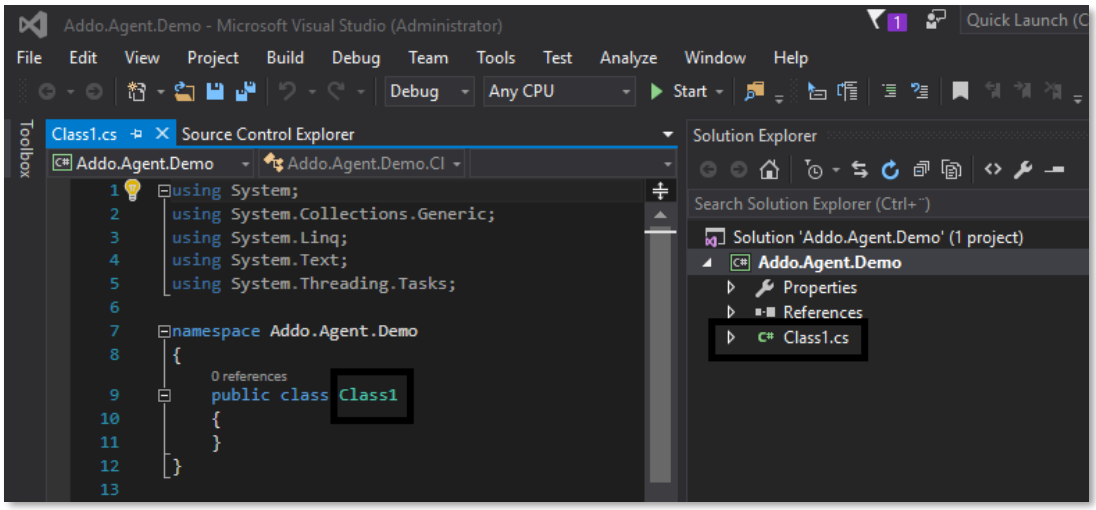

Figure 2 Re-factor Class Library

for an instance:

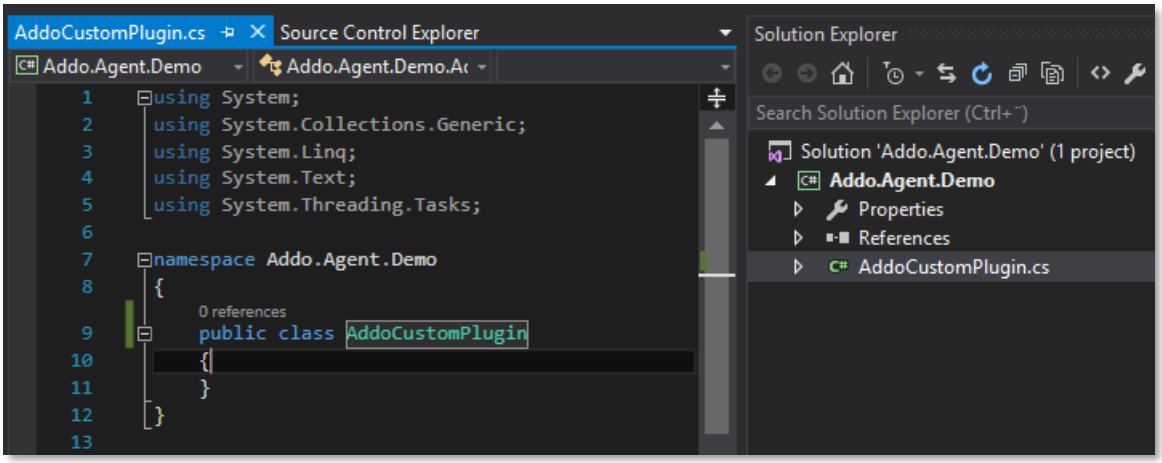

Figure 3 Example Class Library

Next step is to point to AIS references and enable debugging for development.

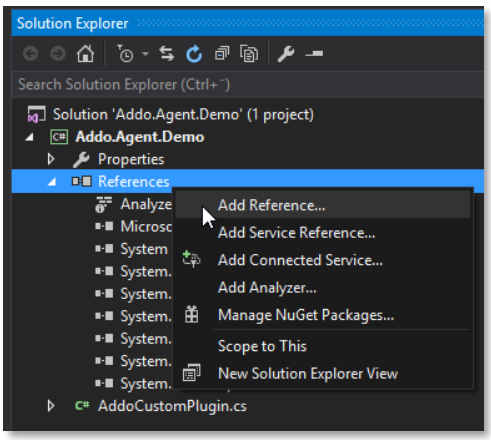

Figure 4 References

Select browse and add the following references:

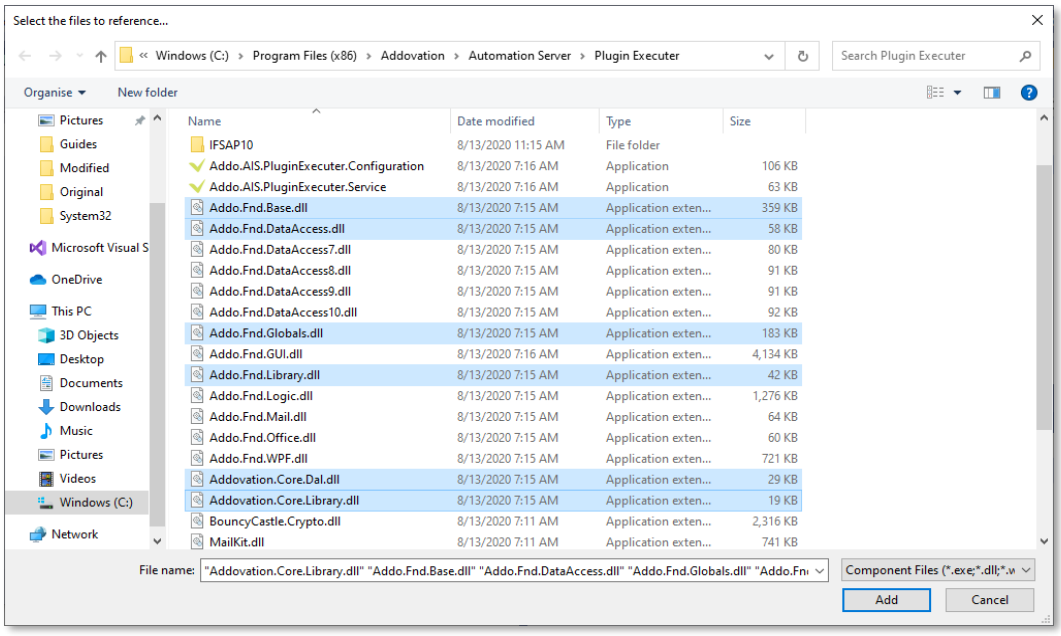

Figure 5 Add References

Press OK when done:

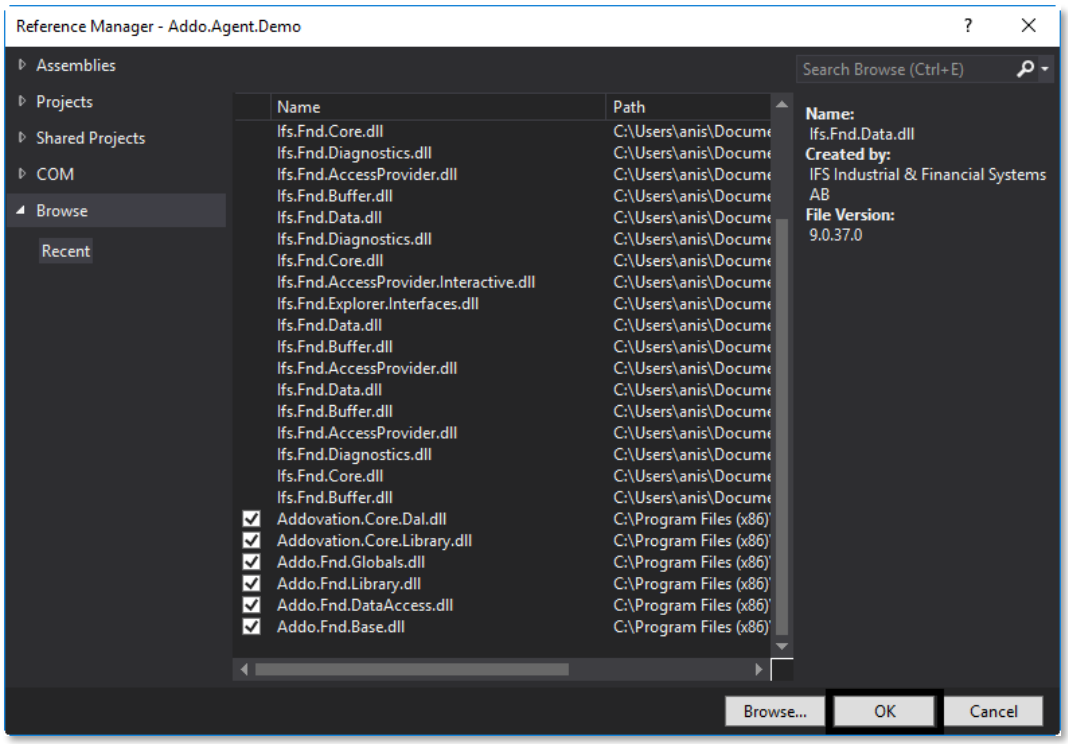

Figure 6 – Add reference

In order to run as plugin, we need to implement an interface called IProcess:

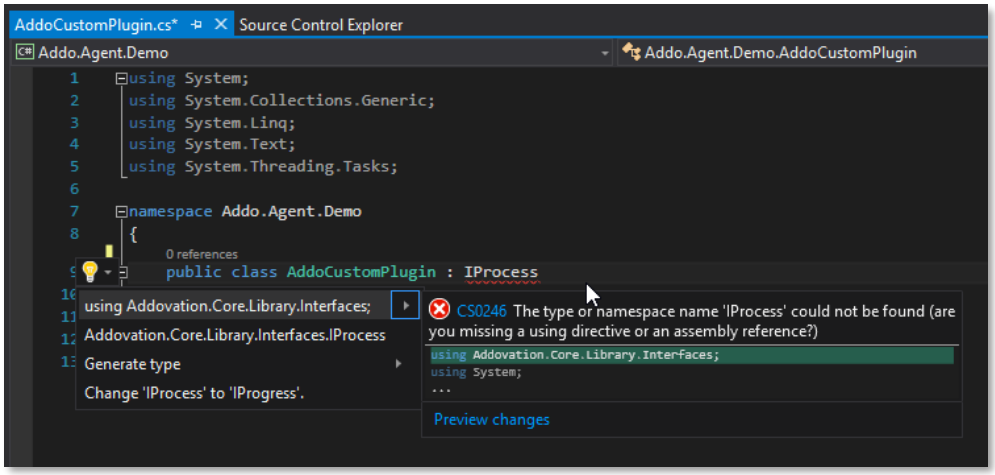

Figure 7 Implement IProcess

Add the reference and also choose to implement it:

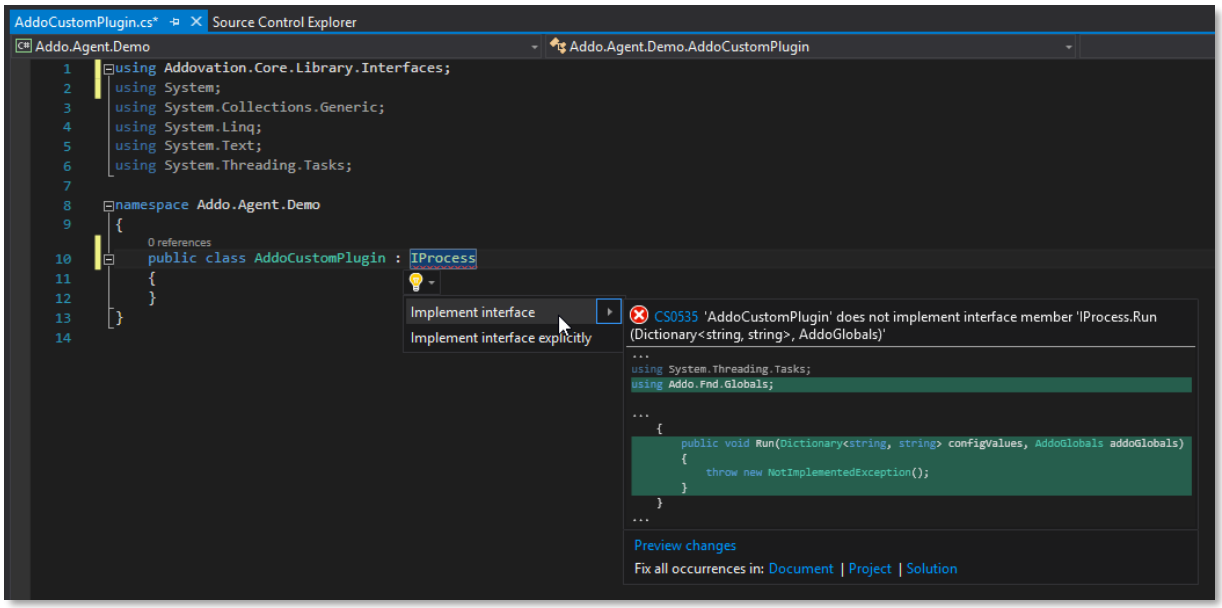

Figure 8 Implement interface

```
namespace Addo.Agent.Demo
{
    0 references
    public class AddoCustomPlugin : IProcess
    {
        0 references
        public void Run(Dictionary<string, string> configValues, AddoGlobals addoGlobals)
        {
            throw new NotImplementedException();
        }
    }
}
```

Figure 9 Implement interface

Change the implementation of Run and also add the process description like following:

```
namespace Addo.Agent.Demo
{
    [ProcessDescription("Some description of our plugin")]
    0 references
    public class AddoCustomPlugin : IProcess
    {
        0 references
        public void Run(Dictionary<string, string> configValues, AddoGlobals addoGlobals)
        {
            try
            {
                Console.WriteLine("Hello Addovation!");
            }
            catch(Exception ex)
            {
                AddoDebug.LogEvent(ex.Message + "\n\n" + ex.StackTrace, configValues["LOG_FOLDER"], EventLogEntryType.Information);
            }
        }
    }
}
```

Figure 10 process description

Now we have a simple console app that only writes to debug console for testing purposes, next step is to debug this. Start the Addovation.Configuration.Client.exe (found in start menu or in AddoServices folder). Connect to some IFS instance and select folders for source, log, outgoing but most important – select the folder in where your DLL is being set at the moment, for instance:

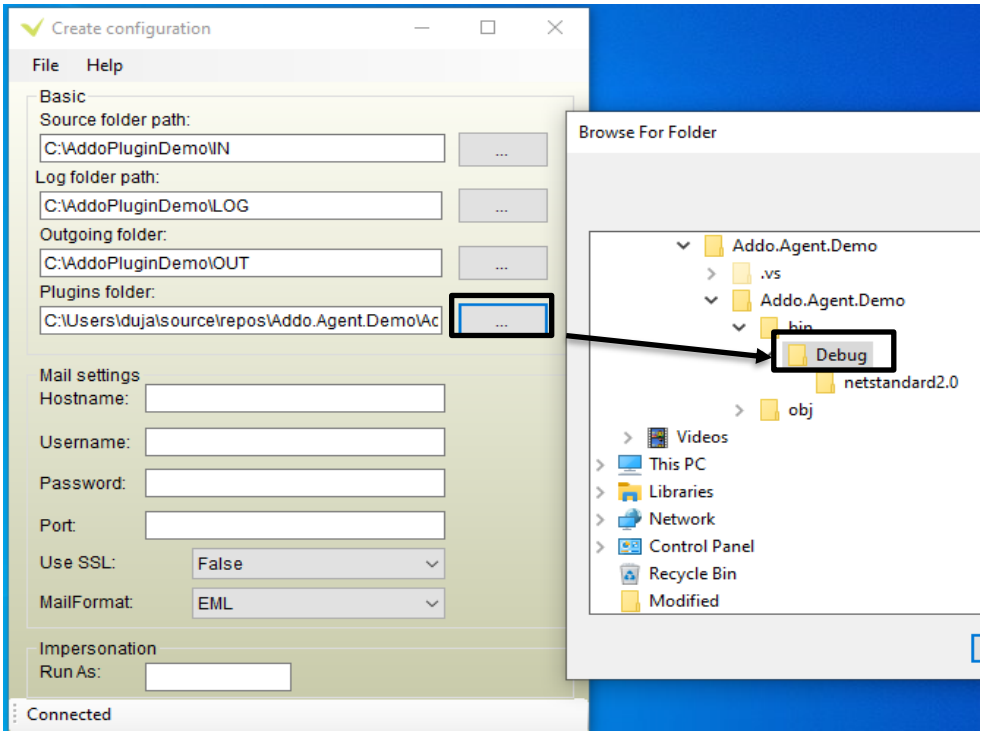

Figure 11

Save the configuration by selecting Save As:

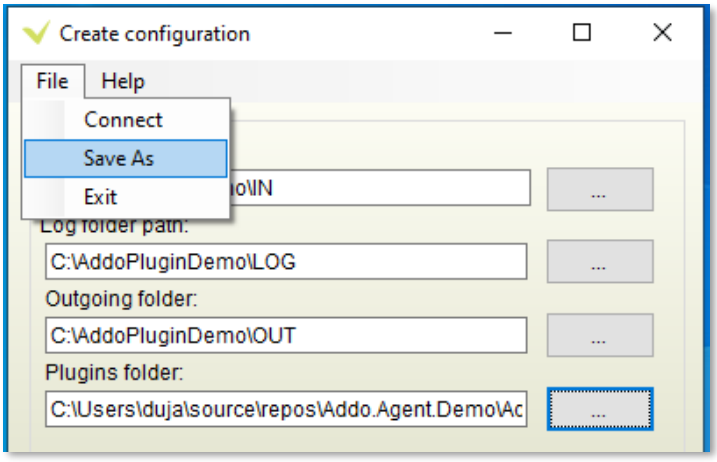

Figure 12 Save as

Save it into the AddoServices folder (Administrator permission might be required):

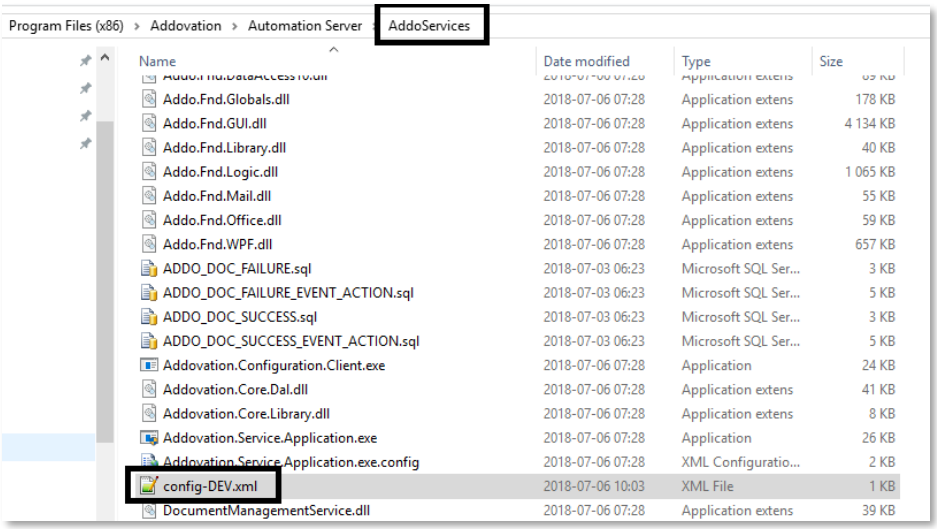

Figure 13 Config-Dev

Now, in order to debug in an easy way – we can start our class library by pointing to Addovation.Service.Application.exe

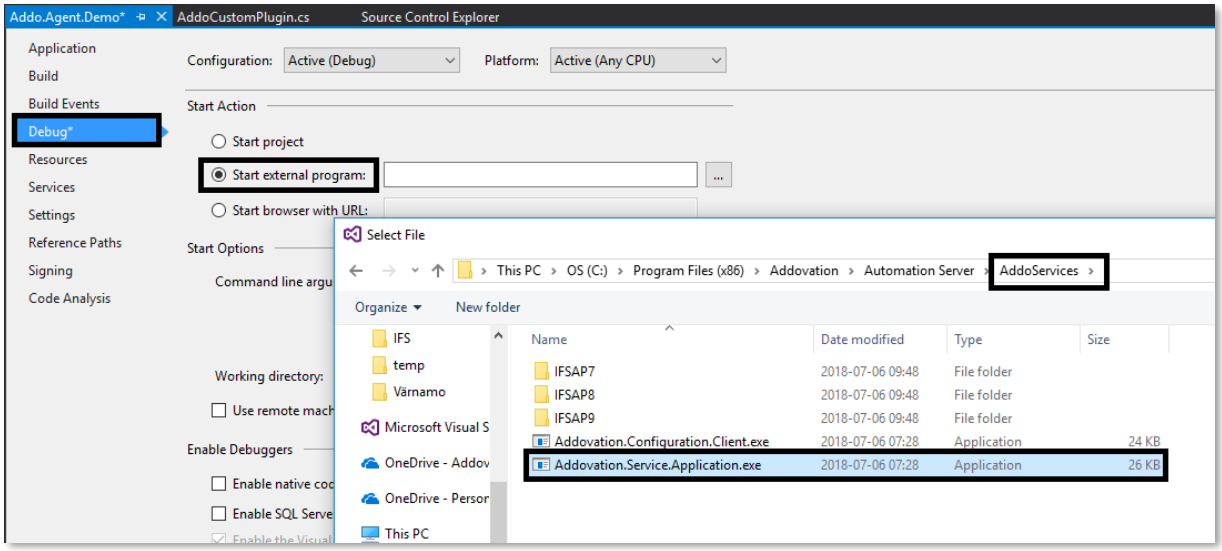

Figure 14 Start external program

Save everything, set a break point and run the library, if everything works as expected it should hit the breakpoint in the following way:

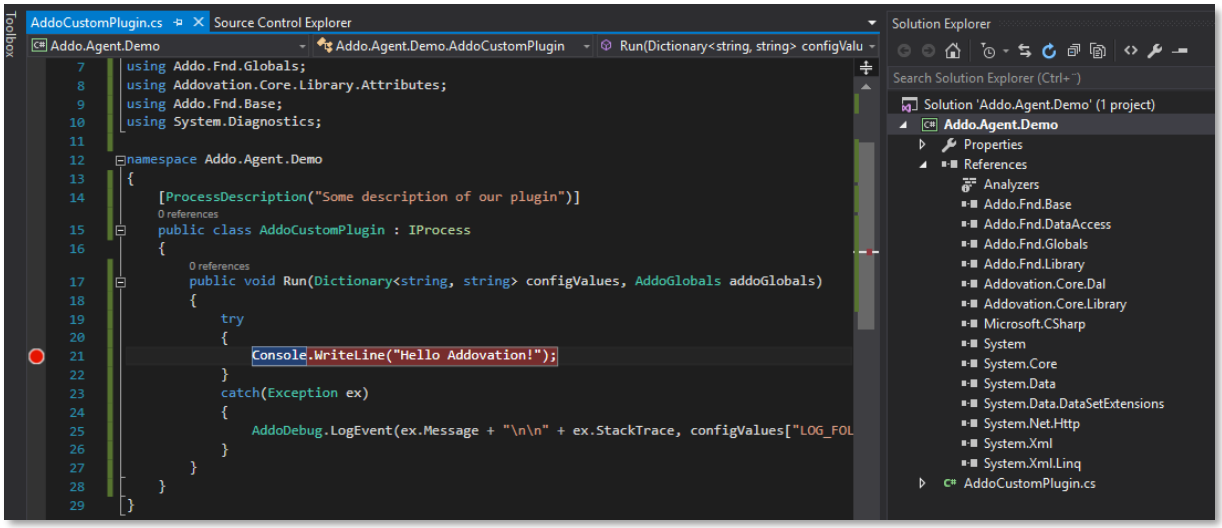

Figure 15 Break point

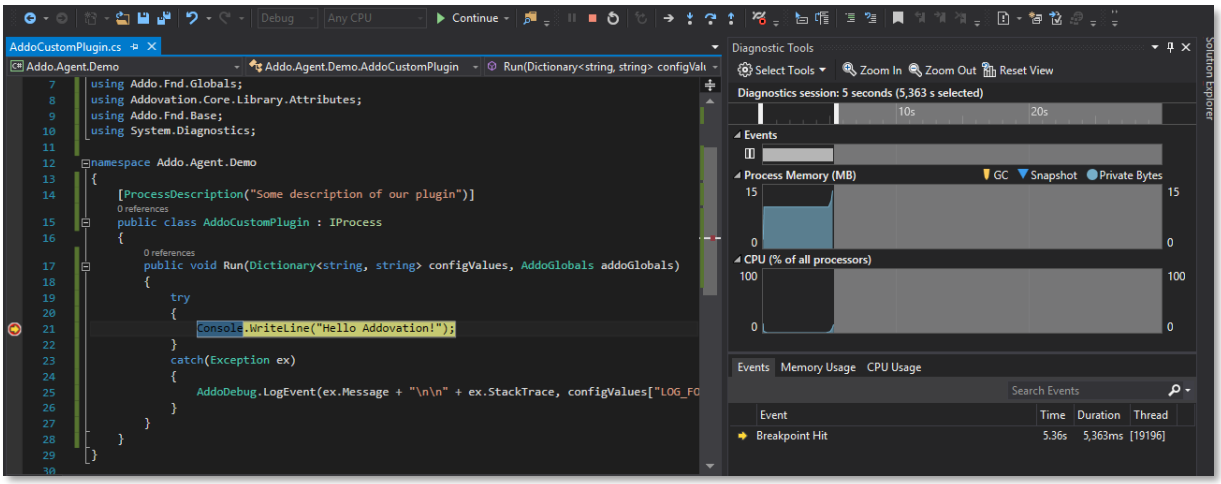

Figure 16

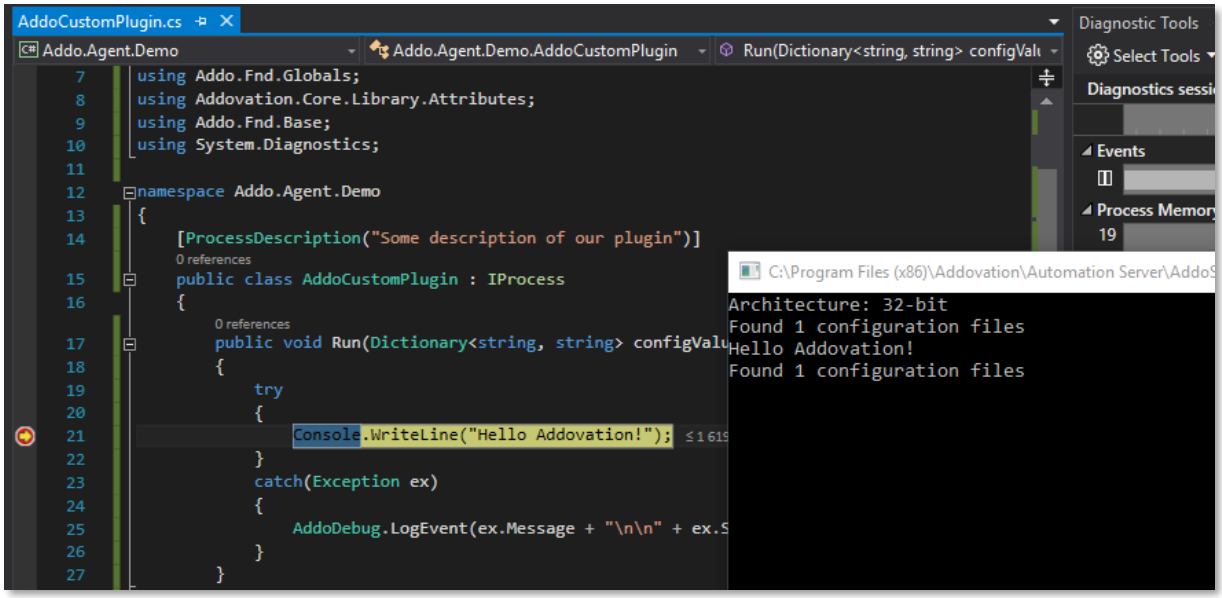

Figure 17

The AddoServices will on default run on 5 seconds intervals that can be overridden in the constructor. Next step is to configure a login towards IFS Applications using the credentials defined in the configuration XML file. So stop debugging and re-write the code to the following in order to login and do a simple query and parse the result:

```
0 references
public void Run(Dictionary<string, string> configValues, AddoGlobals addoGlobals)
{
    try
    {
        using (addoGlobals._gDbAccess)
        {
            //Try to login
            if (!Dal.TryLogon(configValues, ref addoGlobals, 180000))
                return;

            using (var result = addoGlobals._gDbAccess.RunSQLQuery("SELECT * FROM &AO.ABC_CLASS"))
            {
                foreach (DataRow row in result.Rows)
                    Console.WriteLine(row[0] + " " + row[1]);
            }
        }
    }
    catch (Exception ex)
    {
        AddoDebug.LogEvent(ex.Message + "\n\n" + ex.StackTrace, configValues["LOG_FOLDER"], EventLogEntryType.Information);
    }
}
```

Figure 18

The plugin will now try and login, execute a query and show the result set. Hit Start again in Visual Studio and if everything being done correct you should be able to see something like:

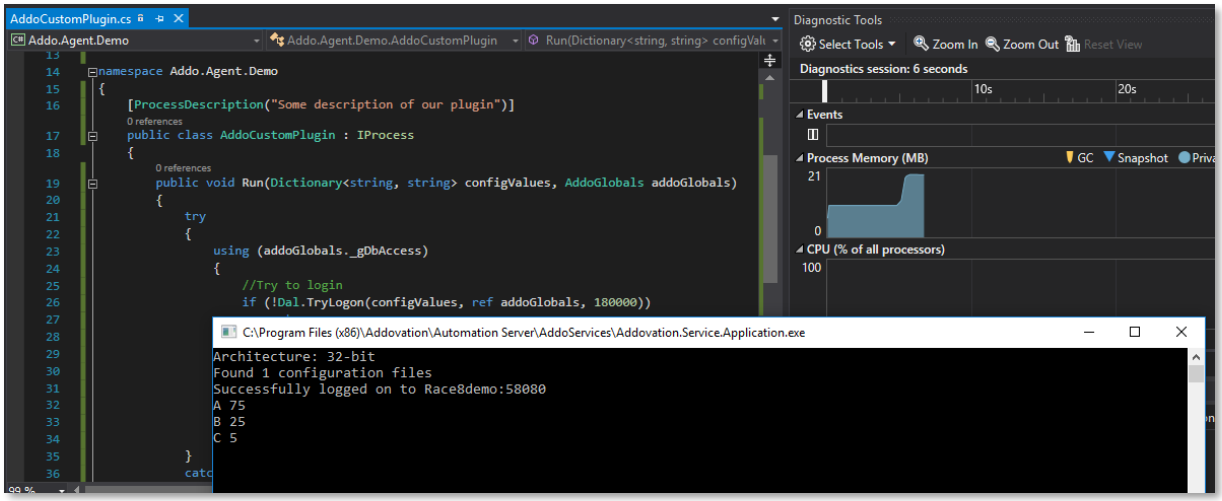

Figure 19

Congratulations, you are now ready to develop plugins of any kind with endless possibilities.

When deploying the plugin on another server, simply compile it in release mode and copy the DLL file to some folder (remember to preserve source files in SVN though). Re-configure the XML and make sure that the PLUGINS points to that folder:

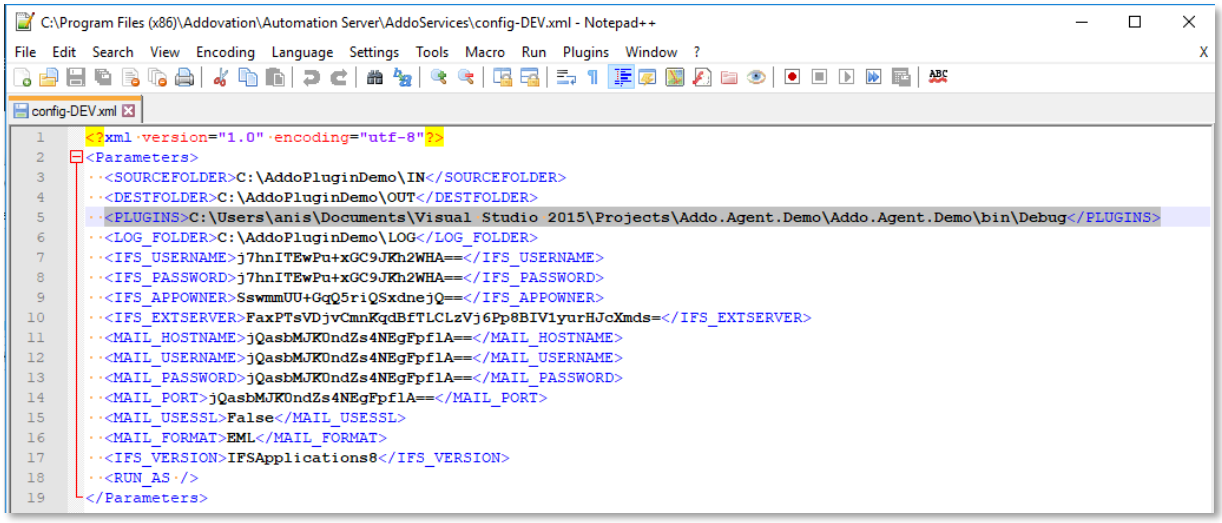

Figure 20 Config file

If starting AddoServices as a windows service it will automatically try and run the DLL now every 5 seconds by default. If the previous instance is not done, it will not start another instance. So if you have long running operations – only 1 active instance will be executed at any given time.

If you want to run the plugin on scheduled intervals, one can simply setup a bat file or scheduled task using the following configuration, for instance

.bat file configuration:

```
start cmd.exe /k ""c:\Program Files (x86)\Automation
Server\AddoServices\Addovation.Service.Application.exe" 0
config="c:\Addovation\AIS Plugins\IFS8TEST\Addo.Agent.Demo\config-DEV.xml"
console"
```

Scheduled task

Create a new scheduled task, select start a program:

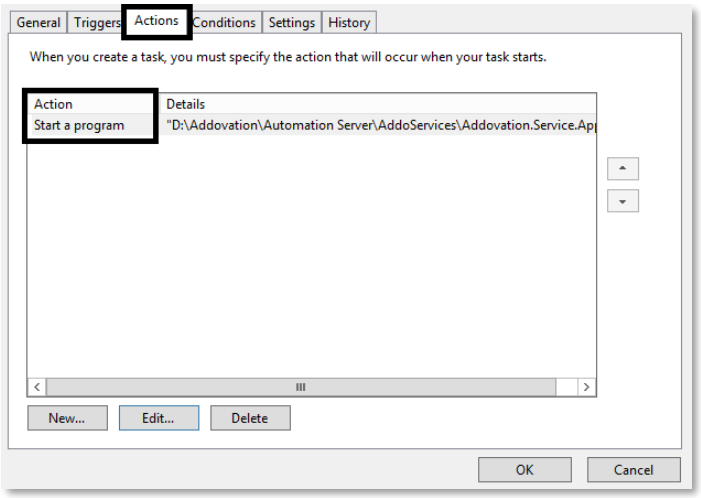

Figure 21 Actions

and point program / script to Addovation.Service.Application.exe

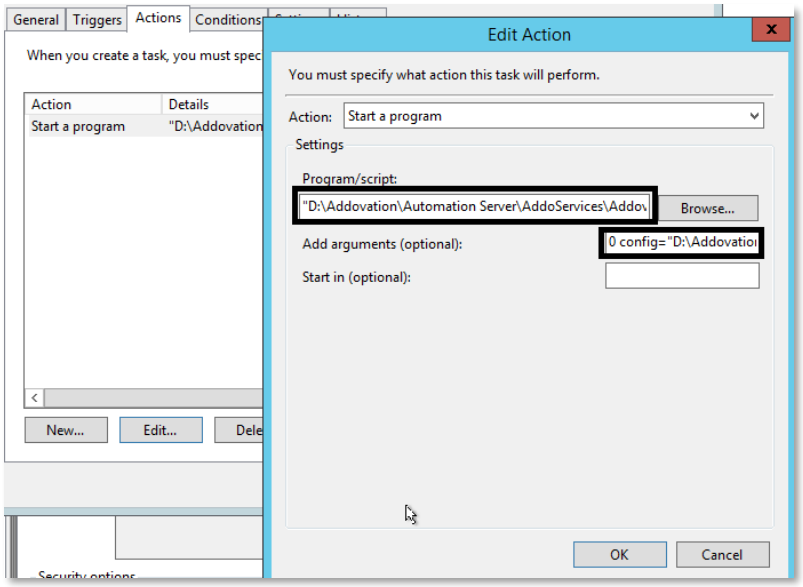

Figure 22 Point program

and arguments as: `0 config="c:\Addovation\AIS Plugins\IFS8TEST\Addo.Agent.Demo\config-DEV.xml" console`

The 0 indicates to only run service once for every execution, and not repeatedly. The config points to what configuration XML file to use and last indicate to run in console mode and not as a service.

2 System requirements

The table below describes the system requirements for client machines.

System Requirements for plugins development	
Operating System	Windows 10 (32-bit or 64-bit) and above Windows Server 2019 (32-bit or 64-bit) and above
Memory	4 GB RAM or more
Visual Studio	2017 version 15.8.7 or above
CPU	Intel/AMD (Dual core or more recommended)
.NET Framework	Full 4.6.2 or newer
Addovation AIS	Major version 2

*) Other Microsoft Windows and Visual Studio versions might work but is unsupported.

3 Compatibility matrix

Addovation Plugins Executer is compatible with/requires the following operating systems, frameworks and applications:

SUPPORTS / REQUIRES	VERSION / BITNESS	PLUGIN EXECUTER
WINDOWS SERVER 2012	32 and 64	✗
WINDOWS SERVER 2016	32 and 64	✓
WINDOWS SERVER 2019	32 and 64	✓
WINDOWS 7	32 and 64	✗
WINDOWS 8	32 and 64	✗
WINDOWS 10	32 and 64	✓
IFS APPLICATIONS	7	✗
IFS APPLICATIONS	8	✗
IFS APPLICATIONS	9 Upd 15	✓
IFS APPLICATIONS	10 Upd 9	✓

4 Limitations

Non-standard configurations in terms of authentication towards IFS Applications is not supported. Also customizations that might affect standard API:s being used is not supported, for full list of used queries and API:s please contact Addovation if in doubt.

Installation of software requires writing to registry in local machine hive.

Addovation does not install any special views or packages in the database.

5 Contact

For further inquiries, please contact your nearest Addovation office:

NORWAY – DRAMMEN

Addovation, Strandveien 11, 3015 Drammen, Norway

+47 32 22 38 88

sales@addovation.com

SWEDEN – JÖNKÖPING

Munksjötornet, Vaggerydsgatan 1, 553 30 Jönköping, Sweden

+46 (0)765510202

sales@addovation.com

SWEDEN – GOTHENBURG

Marieholmsgatan 42, Box 13120, 402 52 Göteborg, Sweden

+46 707 330 399

sales@addovation.com

SWEDEN – LULEÅ

Storgatan 7, 972 38 Luleå, Sweden

+46 73 650 48 40

sales@addovation.com

SRI LANKA – PANNIPITIYA

Addovation (Pvt) Ltd, Level 5, 370, Highlevel Road, Pannipitiya, Sri Lanka

+46 73 650 48 40

sales@addovation.com

SWEDEN – STOCKHOLM

Rålambsvägen 17, 112 59 Luleå, Sweden

+46 76 517 31 04

sales@addovation.com

For existing customers, please use http://support.addovation.com