TwinCAT PLC to Unity

COMMUNICATION RESEARCH

Heesewijk,Rik R.F.J.G. van,
Aarts,Luuk L.H.K.

4-20-2023

Table of Content

Introduction

In this file you will find the done research on the best communication protocol for between Unity and TwinCAT and how to use and install our final choice. We will use the DOT framework for our research and we will answer a few main questions we had to answer, in order to set up the communication between our Digital-twin (Unity) and our PLC (TwinCAT).

Questions

Main questions

- Which communication protocol is best to use for the communication between Unity and TwinCAT?
- How do you implement the protocol?

Sub questions

- Can Unity and TwinCAT communicate with each other?
- Is the protocol easy to implement?

Why do we need to conduct this research?

We need to find out how to communicate between the digital twin and PLC that will control the robot arm. We are all new to working with a PLC and a digital twin, so we will have to find a way to make a connection between those two systems. To prove we are using the best protocol available and to find out how to work with that protocol, we will do some research. In this document we provide you with the information we have collected on this subject.

How do we research the protocols?

To implement the DOT framework in the correct way, this is why we used the 'library method' to do research. After learning about the desired topics, we checked it, wrote it down and with our final solution, we tested it was indeed the wright choice. When is clear that we will indeed work with one final protocol, we will do further research and make a demo with explanation on how to implement the chosen protocol.

TwinCAT ADS protocol

TwinCAT ADS is a protocol written by the makers of TwinCAT (Beckhoff), this means that it is fully included within core system of TwinCAT. The protocol is also supported by the visualization software we are going to use, Unity. To answer the question if the communication is possible, yes, it is definitely possible. The only remaining question is then if it is easy to research, implement and is good for our use cases.

The group before us used TwinCAT ADS as a protocol for communication, this alone would not be enough for us to conclude that ADS is indeed a good and easy protocol. Some other sources and videos showed us that ADS is indeed very easy to use.

The requirements for this protocol are:

- An internet connection.
- Port 851 open.
- Software like Visual Studio and Unity that support the TwinCAT.ADS library.
- The AMS Net id of the PLC.

OPC UA client SDK protocol

We found out that the protocol OPC UA client SDK is a second good protocol, also widely used in the industry. The problem is that finding good documentation and examples on how to implement this code was much harder to find. We did find that the possibility's where almost the same and the OPC UA client SDK protocol was mostly written as a general data transporter, while the ADS is just used for TwinCAT. We had a hard Time finding concrete examples and facts on OPC UA client SDK, so we figured this protocol was not worth so much trouble if we already had a good and more than sufficient solution.

Implementing The TwinCAT ADS protocol

Installing the adsClient library in Visual Studio

- Tools
- NuGet Package Manager
- Manage NuGet Package for Solutions
- Browse
- beckhoff.ads (not the server one but the Beckhoff.TwinCAT.Ads one)

Example program in Visual Studio

```
using (AdsClient myClient = new AdsClient())
{
    myClient.Connect(AmsNetId.Local, 851);
    if (myClient.IsConnected)
    {
        Console.WriteLine("ADS Client has been initialised!");
    }
    else
    {
        Console.WriteLine("Failed to connect ADS client");
    }

    var testBool = myClient.CreateVariableHandle("MAIN.testBool");
    var testInt = myClient.CreateVariableHandle("MAIN.testInt");

    //read variable
    var responseBool = myClient.ReadAny(testBool, typeof(bool));
    var responseInt = myClient.ReadAny(testInt, typeof(int));
    var responseString = myClient.ReadValue("MAIN.testString", typeof(string));

    Console.WriteLine("The variables where:");
    Console.WriteLine(responseBool);
    Console.WriteLine(responseInt);
    Console.WriteLine(responseString);

    //write variable
    myClient.WriteAny(testBool, false);
    myClient.WriteValue("MAIN.testInt", 15);
    myClient.WriteValue("MAIN.testString", "Hello world");

    responseBool = myClient.ReadAny(testBool, typeof(bool));
    responseInt = myClient.ReadAny(testInt, typeof(int));
    responseString = myClient.ReadValue("MAIN.testString", typeof(string));

    Console.WriteLine("The variables are now changed to:");
    Console.WriteLine(responseBool);
    Console.WriteLine(responseInt);
    Console.WriteLine(responseString);

    myClient.Disconnect();
    Console.WriteLine("Disconnected!");
}
```

the first thing we do is we create a client. We do this by using (AdsClient “clientname” = new AdsClient()). In this namespace we can then write all of our client code as show above. We then use myClient.Connect to connect the client to the plc. In this case it is my local laptop we are connecting to so we use AmsNetId.Local and we use the default port of 851. Port 851 is used in twincat 3. Port 850 is used in twincat 2, but since we use twincat 3, we will probably only use port 851.

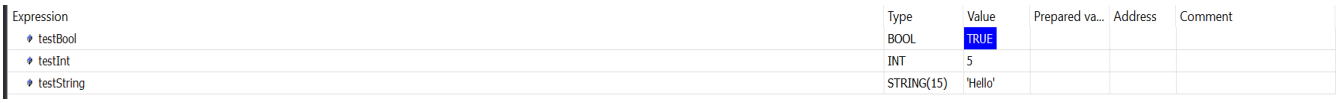
```
var responseBool = myClient.ReadAny(testBool, typeof(bool));
var responseInt = myClient.ReadAny(testInt, typeof(int));
var responseString = myClient.ReadValue("MAIN.testString", typeof(string));
```

ReadAny reads the value of the current variable it then places it in the var. the first parameter is used for where the variable can be found. The string for example can be found in MAIN and the name of the string is testString (MAIN.testString). The second parameter is used to tell the client what type of variable it is that it is going to read. For using bool and int we can use ReadAny. For strings we have to use readValue because otherwise we do not know how long the string is we want to read. By using Readvalue we avoid this issue.

```
myClient.WriteAny(testBool, true);
myClient.WriteValue("MAIN.testInt", 5);
myClient.WriteValue("MAIN.testString", "Hello");
```

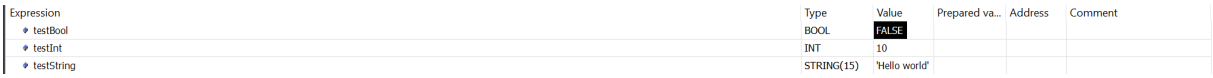
For writing values we use WriteAny or WriteValue. WriteAny can be used for bools. WirteValue is used for writing the other variables. It will again first need the location of the variable and then the type it is going to write.

The Results of the example project:



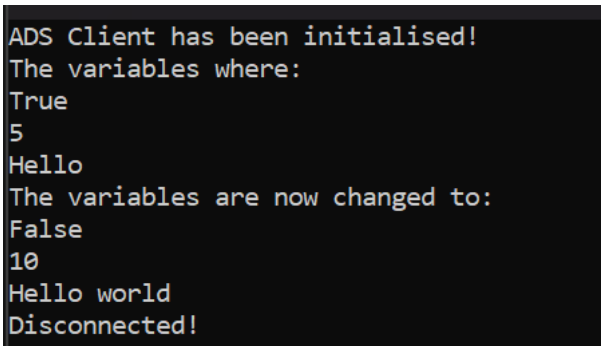
Expression	Type	Value	Prepared va...	Address	Comment
testBool	BOOL	TRUE			
testInt	INT	5			
testString	STRING(15)	'Hello'			

Figure 1 : The variables before ADS contact



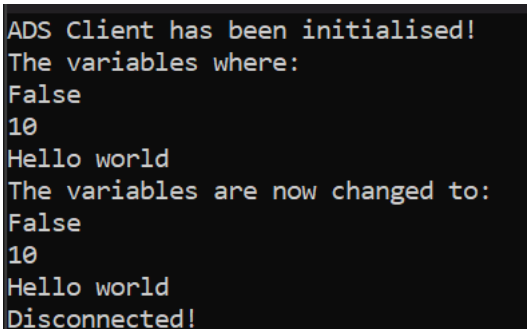
Expression	Type	Value	Prepared va...	Address	Comment
testBool	BOOL	FALSE			
testInt	INT	10			
testString	STRING(15)	'Hello world'			

Figure 2 : The variable after ADS contact



```
ADS Client has been initialised!
The variables where:
True
5
Hello
The variables are now changed to:
False
10
Hello world
Disconnected!
```

The Visual Studio terminal displays the actions and variables correctly.



```
ADS Client has been initialised!
The variables where:
False
10
Hello world
The variables are now changed to:
False
10
Hello world
Disconnected!
```

If we send a ADS contact request but send the values it already has, the values stay the same and it will display it correctly.

Sources

- The DOT Framework, explaining the DOT Framework, website: https://ictresearchmethods.nl/The_DOT_Framework.
- TwinCAT ADS, implementation and information, website: http://dronefactory.co.uk/?p=43.
- TwinCAT ADS, implementation video, video: https://www.youtube.com/watch?v=JQ1PdZGCyOE&ab_channel=Maxwell%27sProgrammingJourney
- TwinCAT ADS, implementation video, video: https://www.youtube.com/watch?v=JZChSdU2LMc&t=1370s&ab_channel=JakobSagatowski
- OPC UA client SDK, information on the protocol, website: https://opcfoundation.org/about/opc-technologies/opc-ua/
- OPC UA client SDK, information and implementation video, video: https://www.youtube.com/watch?v=KCW23eq4auw&t=940&ab_channel=IndustrialITandAutomation

Final result

For the final application we wanted to control the speed and the position of the roboticArm using ADS. To achieve this we made a roboticArmController class to control the robotic arm.

```
public RobotArmController(string targetAmsNetId)
{
    client = new AdsClient();

    try
    {
        client.Connect(targetAmsNetId, 851);
        Console.WriteLine($"connected to {targetAmsNetId}");
    }

    catch(Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    positionVeriableHandle = client.CreateVariableHandle(stringPositionVeriableHandle);
    speedVeriableHandle = client.CreateVariableHandle(stringspeedVeriableHandle);
}
```

I started with making a RobotArmController where we can pass in the Ip address of the plc we want to connect to. It will make a new client and tries to connect to the plc. If it cant connect it will throw an exception. If it can connect it will create the variable Handles of the variables we want to control.

```
public void MoveArmToPosition(uint position)
{
    if(GetCurrentArmLocation() ==  position)
    {
        Console.WriteLine("Robot is already at desired location");
        return;
    }

    Console.WriteLine($"Robot was at position: {client.ReadAny(positionVeriableHandle, typeof(int))}");

    client.WriteValue(stringPositionVeriableHandle, position);

    Console.WriteLine($"Robot Moved to position: {position}");
}
```

I then made a MoveArmToPosition function where we can control the arm based on a given position. This position will be calculated using the unity application. First we check if the robotic arm should be moved or if it is already at the location we want it to be. If the robot is already where we want it to be then it will Write to the console that the robot is already at the correct location and we return out of the application. If it is not yet at the correct position we will move it to the correct location. We do this by first displaying the current location. Then it will move to the location we want it to move to. We will then also print out the location it will move to.

```
public void ChangeArmSpeed(uint speed)
{
    if(GetCurrentArmSpeed() == speed)
    {
        Console.WriteLine("Robot is already at desired speed");
        return;
    }

    Console.WriteLine($"Robot had speed: {client.ReadAny(speedVeriableHandle, typeof(int))}");

    client.WriteValue(stringspeedVeriableHandle, speed);

    Console.WriteLine($"Robot now has speed: {speed}");
}
```

We also have a function to control the arm speed named ChangeArmSpeed. This function works in the same way as the MoveArmToPosition function works. The only difference is we use the speed variable here and not the position variable.

```
public void Disconnect()
{
    client.Disconnect();
}
```

```
public void Dispose()
{
    client.Dispose();
}
```

Then we have the Disconnect and the Dispose functions. The disconnect function disconnects the current plc form the client so another plc can be connected. The dispose function removes the client entirely so it should only be called at the end of the program.

```
private dynamic GetCurrentArmLocation()
{
    return client.ReadAny(positionVeriableHandle, typeof(int));
}
```

I then made a GetCurrentArmLocation function that returns a dynamic variable of the current arm location. This is needed to check the current location of the arm in the change arm location function.

```
2 references
private dynamic GetCurrentArmSpeed()
{
    return client.ReadAny(speedVeriableHandle, typeof(int));
}
```

I made a similar function to the GetCurrentArmLocation but this will return the current speed of the arm and not the location of the arm. This is used in the ChangeArmSpeed function.