AAComm API with MATLAB
Basic Setup

Application Note

www.agito-akribis.com
Member of Akribis Systems group

Revision History

Version	Description	Date
1.0	Initial Release	1 May 2023

Contact Information

Manufacturer	Agito Akribis Systems Ltd., Member of Akribis Systems Group
Address	6 Yad-Harutsim St., P.O.Box 7172, Kfar-Saba 4464103
Telephone	+972-9-8909797
Website	www.agito-akribis.com

Copyright Notice

Products Rights

AGDx, AGCx, AGMx, AGAx, AGIx, and AGLx are products designed by Agito Akribis Systems Ltd. in Israel. Sales of the products are licensed to Akribis Systems Pte Ltd. under intercompany license agreement.

Agito Akribis Systems Ltd. has full rights to distribute above products worldwide.

Disclaimer

This product documentation was accurate and reliable at the time of its release.

Agito Akribis Systems Ltd. reserves the right to change the specifications of the product described in this manual without notice at any time.

Trademarks

Agito PCSuite is a trademark of Agito Akribis Systems Ltd..

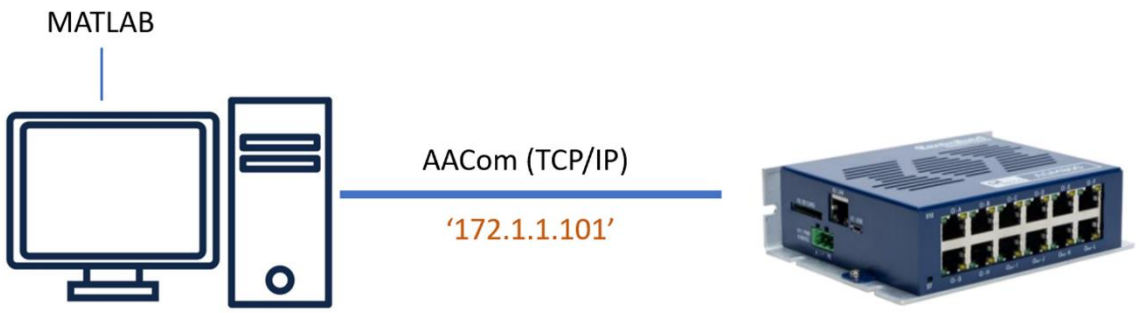
Warranty

This product is warranted to be free of defects in material and workmanship and conforms to the specifications listed in this manual, for a period of 12 months from the shipment date from factory.

Contents

About This Document

This document demonstrates how to use AAComm in MATLAB on Windows platforms. AAComm is a .NET Assembly that serves as an API to control Agito controller products. The support for AAComm from MATLAB enables the user to utilize the powerful mathematical tools in MATLAB with Agito controllers seamlessly. This document requires the user to have some basic understanding and experience with Agito controller products and AAComm API.



How to Add AAComm in MATLAB

The user can download the latest 'AAComm.DLL' file from:

https://www.nuget.org/packages/Agito.AAComm

Alternatively, the user can ask for the file from Agito Akribis Systems. AAComm supports .NET Framework 4.0, 4.6 and 4.8. The user needs to have the corresponding version of .NET framework installed on the PC before using AAComm.

Open MATLAB, add the folder that contains the 'AAComm.DLL' file into the MATLAB path and run the following command to add the AAComm assembly into the MATLAB base environment:

```
asm = NET.addAssembly(FilePath));
```

Where 'FilePath' is the full path of the 'AAComm.DLL' and 'asm' is the returned NET.assembly object containing the members of the AAComm assembly.

For more information on 'NET.addAssembly' command in MATLAB, refer here:

https://www.mathworks.com/help/matlab/ref/net.addassembly.html

Upon successful addition of AAComm in MATLAB, the user can view all the members including classes, interfaces, enums and structures of the AAComm assembly in MATLAB.

For example:

Typing '*asm*' will replay the following:

```
asm =

  NET.Assembly handle
  Package: NET

Properties for class NET.Assembly:

    AssemblyHandle
    Classes
    Structures
    Enums
    GenericTypes
    Interfaces
    Delegates
```

```
asm.Classes

ans =

    'AAComm.CommAPI'
    'AAComm.CommAPIWrapper'
    'AAComm.Shared.ConnectionDataDefaults'
    'AAComm.Shared.ConnectResultInterpreter'
    'AAComm.Shared.ProductTypes'
    'AAComm.Shared.TimeoutMs'
    'AAComm.Services.AACommEventArgs'
    'AAComm.Services.AACommMessage'
    'AAComm.Services.AllStatInterpreter'
    'AAComm.Services.ARPTools'
    'AAComm.Services.CanDescriptionBuilder'
    'AAComm.Services.COMTools'
    'AAComm.Services.ConnectionData'
    'AAComm.Services.ControllerErrorInterpreter'
    'AAComm.Services.ControllerIdentityContainer'
    'AAComm.Services.ControllerMessagesContainer'
    'AAComm.Services.TaskFunc'
    'AAComm.Services.GlobalVar'
    'AAComm.Services.HeaderParser'
    'AAComm.Services.LocaleProvider'
```

```
    'AAComm.Services.MessageData'
    'AAComm.Services.UserProgVarsAdapter'
    'AAComm.Services.XmlHandler'
    'AAComm.Extensions.AACommDownloadFPGA'
    'AAComm.Extensions.AACommDownloadFPGARemote'
    'AAComm.Extensions.AACommDownloadFW'
    'AAComm.Extensions.AACommDownloadUP'
    'AAComm.Extensions.AACommFwInfo'
    'AAComm.Extensions.AACommRemoteFpgaInfo'
    'AAComm.Services.ARPTools+LookupResult'
    'AAComm.Services.ARPTools+EthernetLookupResults'
    'AAComm.Services.ControllerMessagesContainer+MessageCategory'
    'AAComm.Services.XmlHandler+TextNodeData'
```

```
asm.Structures

ans =

    'AAComm.Shared.AllStatAxisData'
    'AAComm.Shared.AllStatData'
```

```
asm.Enums

ans =

    'AAComm.Shared.ChannelType'
    'AAComm.Shared.ConnectResult'
    'AAComm.Shared.MessageType'
    'AAComm.Extensions.AACommFwInfo+Targets'
```

```
asm.Structures

ans =

    'AAComm.Shared.AllStatAxisData'
    'AAComm.Shared.AllStatData'
```

How to Use AAComm in MATLAB

Once the AAComm assembly has been added, the user can use AAComm in MATLAB just like in the C# environment. In this example, an AGD200 controller is used to demonstrate basics of using AAComm. For detailed guide on using AAComm, please contact Agito-Akribis application support.

Creating an AAComm API Object

Run the following code to create a API object in MATLAB:

```
AGD200_api = AAComm.CommAPI;
```

An AGD200 object has been created. The user can open the object in the MATLAB base workspace to view the members of the controller object:

```
AGD200_api =


ans =


  CommAPI with properties:


        MinMsgDelayMsec: 0
        IsAACommServerRunning: 1
        IsConnected: 0
        IsQueueEmpty: 1
        CurrentConnectionData: []
        MaxNumOfAxes: 12
        NumberOfAxes: 12
        AACommServerVersion: [1x1 System.Version]
        IdentityContainer: [1x1 AAComm.Services.ControllerIdentityContainer]
        MessagesContainer: [1x1 AAComm.Services.ControllerMessagesContainer]
        UserProgAdapter: [1x1 AAComm.Services.UserProgVarsAdapter]
        DocsAppFolder: [1x1 System.String]
        MaxTheoreticalNumOfAxes: 12
```

Connecting to the Controller

The communication between the PC and the controller is handled by AAComm server. TCP/IP protocol is used to download/upload data between the PC and controller. To connect to the controller, the user needs to run the following line:

```
AGD200_api.Connect(connectionData_);
```

Where '*connectionData_*' contains the required information for the *Connect()* method:

```
connectionData_ =  AAComm.Services.ConnectionData;


connectionData_ =


ConnectionData with properties:
        Name: [1x1 System.String]
        ControllerType: 5
        CommChannelType: [1x1 AAComm.Shared.ChannelType]
        Rs232_PortName: [1x1 System.String]
        Rs232_BaudRate: 115200
        Rs232_Parity: [1x1 System.IO.Ports.Parity]
        Rs232_StopBit: [1x1 System.IO.Ports.StopBits]
        Rs232_DataBits: 8
        Rs232_CableType: 0
        CAN_BaudRate: 1000000
        CAN_Channel: 0
        CAN_SendAddress: 64
        CAN_ReceiveAddress: 65
        CAN_Delay: 6
        ET_Port: 50000
        ET_IP_1: 172
        ET_IP_2: 1
        ET_IP_3: 1
        ET_IP_4: 101
        ET_IP_Str: [1x1 System.String]
        IsUser1: 0
```

For example, notice the default IP address which appears on ET_IP_1, ET_IP_2, ET_IP_3, and ET_IP_4. Also, the '*ControllerType*' which is assigned to be 5 which is the number that represents the 2-axes controller AGD200.

Basic AAComm Commands

```
>> res = AGD200_api.Connect(connectionData_)


res  =


Success
```

Next, a message object needs to be instantiated as follow:

```
msg = AAComm.Services.AACommMessage('apos');  % Read the postion of A axis.
```

Where *'apos'* i.e., the argument is the keyword we would like to read / write.

Another example for writing to a variable:

```
msg = AAComm.Services.AACommMessage('asetposition=0');  % Write a postion to A axis.
```

Next, we'll have to send the msg to the controller, and wait for it to replay with an 'OK' or with the value we wanted to read.

The *'SendReceive()'* method is a blocking method that sends the required message to the controller and waits for his response.

```
result = AGD200_api.SendReceive(msg);
```

The result contains the following data:

```
result =
        AACommEventArgs with properties:

            IsError: 0
            IsBoot: 0
            MsgReceived: [1x1 System.String]
            MsgSent: [1x1 System.String]
```

In our example, in *'result.MsgReceived'* we'll get the position of axis A, according to the keyword we've sent – 'apos'.

Now we can review a full example that sets up a controller, initiates a PTP motion, recording some preset keywords, and uploads the raw data into MATLAB.

```
% Local folder containing the .dll file.

>> dll_file_path = 'C:\Users\User\User\....\Documents\MATLAB\AAComm.dll';

% Import the dotNet assembly into MATLAB environment.

>> asm = NET.addAssembly(dll_file_path);

% Instantiate an API object.

>> AGD200_api = AAComm.CommAPI;

% Instantiate an connectionData object

>> connectionData = AAComm.Services.ConnectionData;

% Change the controller type to AGD200 (number 5)

>> connectionData.ControllerType = 5;

% Optional, test the controller type number with keyword name

>> AAComm.Shared.ProductTypes.GetProductIdByName('AGD200orAGC300')

ans =

     5

% This assignment should work as well

connectionData.ControllerType = AAComm.Shared.ProductTypes.GetProductIdByName('AGD200orAGC300')

% Connect

>> Res = AGD200_api.Connect(connectionData);

% Connection test

>> res = AGD200_api.IsConnected();

% Create a message object

msg = AAComm.Services.AACommMessage('AMotionMode = 1')

result = AGD200_api.SendReceive(msg);
```

To simplify the following commends we'll define a function as follows.

```
function [ Sent, Received ] = do( api, string )

msg = AAComm.Services.AACommMessage(string);
res = api.SendReceive(msg);
Sent = res.MsgSent;
Received = res.MsgReceived;

end
```

Now, for every action we'd like to do, we can simply call *do(api, string)* with the API object we defined and the relevant string, for example: *[s, r] = do (AGD200_api, 'ASetPosition = 0');*

Back to our example, we'll need to set the motion mode, begin a motion, start a recording, stop the motion, and upload the recorded data.

```
[s, r] = do ( AGD200_api, 'ASetPosition=0' );
% PTP motion mode
[s, r] = do ( AGD200_api, 'AMotionMode=1' );
% Set absolute target position
[s, r] = do ( AGD200_api, 'AAbsTrgt=100000' );
% Set speed
[s, r] = do ( AGD200_api, 'ASpeed=100000' );
% Motor On
[s, r] = do ( AGD200_api, 'AMotorOn=1' );
% Set recording trigger
[s, r] = do ( AGD200_api, 'ARecTrigTyp = 7' );
% Start Motion
[s, r] = do ( AGD200_api, 'ABegin' );
% Start the recording
[s, r] = do ( AGD200_api, 'ARecStart' );
[s, r] = do ( AGD200_api, 'ARecTrigForce' );
% Check the status of the recording
[s, r_] = do ( AGD200_api, 'ARecStat' );
```

- *Note: here we can handle r_ (the received message) to monitor the status of the recording.*

```
% After the recording has finished, we can upload the results
[ s, uploaded_data ] = do ( AGD200_api, 'ARecUpload' );
% After the recording has finished, we can upload the results
AGD200_api.Disconnect();
```

Finally, we can review the uploaded data that has been saved in *uploaded_data.*

Naturally, the data is saved into some string format, to reformat the data to a generic MATLAB numbers array, we can use the following functions:

```
Data_chars = char(Data);
Data_num = str2num(Data_char);
```

Now we have an array of raw data that needs to be parsed.

Good Luck!

- Note, in the scope of this application note, not all the relevant keywords have been reviewed, for more information, please refer to Agito's Keywords Manual