Beckhoff ADS
using c++ on Linux and Windows

April 2022

Contents

1 Introduction

This is a guide for using Beckhoff ADS to communicate with TwinCAT devices using c++. This guide will show you how to implement it on Linux and Windows.

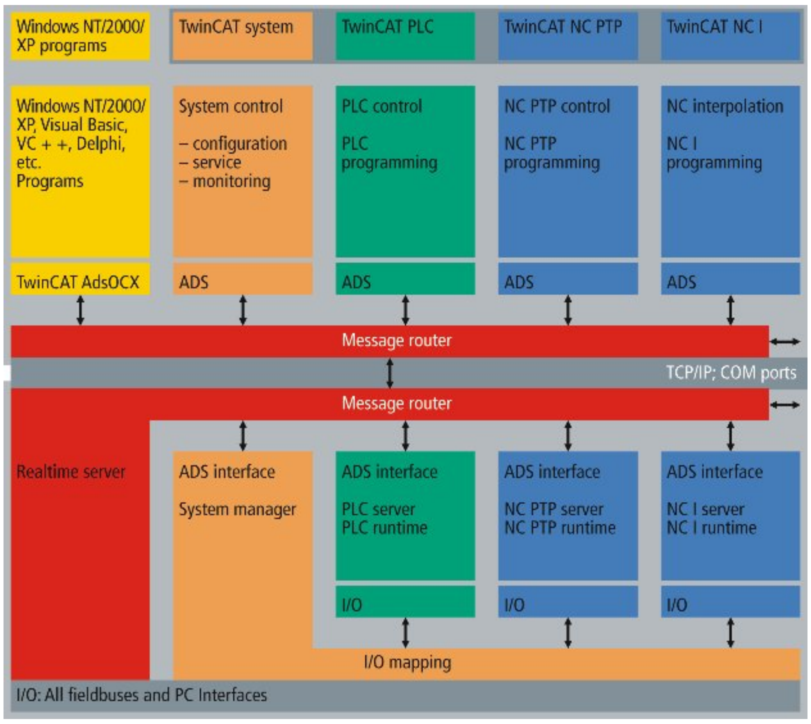

Windows NT/2000/ XP programs
TwinCAT system
TwinCAT PLC
TwinCAT NC PTP
TwinCAT NC I
Windows NT/2000/ XP, Visual Basic, VC + +, Delphi, etc. Programs
System control – configuration – service – monitoring
PLC control PLC programming
NC PTP control NC PTP programming
NC interpolation NC I programming
TwinCAT AdsOCX
ADS
ADS
ADS
ADS
Message router
TCP/IP; COM ports
Message router
Realtime server
ADS interface System manager
ADS interface PLC server PLC runtime
ADS interface NC PTP server NC PTP runtime
ADS interface NC I server NC I runtime
I/O
I/O
I/O
I/O mapping
I/O: All fieldbuses and PC Interfaces

Figure 1: ADS interface

2 Why use ADS in C++?

Experiments were conducted to determine the fastest way to communicate with TwinCAT. The initial solution used UDP to exchange data. We then tried using pyads which is a python library using the C API provided by TcAdsDll.dll . the results were better than UDP. Then we implemented the c++ solution using Beckhoff ADS library, the results were better than pyads.

3 Beckhoff ADS on Linux

In this section we will explain the basic way to configure ADS for c++ on Linux and ROS. It will also show you the basics way to communicate with your TwinCAT project, for more informations please refer to the official git : https://github.com/Beckhoff/ADS .

If you have already the compiled library and want to implement it in the simplest way in a ROS program, please refer to section 3.2. But for more information, section 3.1 will show you how to compile the library from the official gitHub.

3.1 Get the library from GitHub

Please follow this step-by-step tutorial to initialize your project:

- clone initial repository: git clone https://github.com/Beckhoff/ADS.git
- move to the repository : cd ADS
- configure meson build : meson build
- build the library : ninja -C build

On a Linux project you can now use the library by adding the following code:

```
#include "AdsLib.h"
#include "AdsVariable.h"
#include "AdsNotificationOOI.h"
```

Compile and run the project by following these commands:

- meson example/build example
- ninja -C example/build
- ./example/build/example

If you want to use ADS on a ROS program by compiling the library follow these steps after doing the previous ones.

- copy the built library libAdsLib.a (located in the build/ folder) to the AdsLib/ folder.
- copy in your ROS package the AdsLib/ folder.

In your c++ program, include these libraries.

```
#include "../AdsLib/AdsLib.h"
#include "../AdsLib/AdsVariable.h"
#include "../AdsLib/AdsNotificationOOI.h"
```

In the CMakeLists.txt of the package make the links to the library using this code.

```
FIND_LIBRARY(AdsLib_LIBRARIES AdsLib AdsLib/)
target_link_libraries(project_name ${AdsLib_LIBRARIES}
```

You should now be able to compile with catkin.

If you get this error : "undefined reference to symbol 'pthread_create@@GLIBC_2.2.5'" add the following lines to the CMakeLists.txt .

```
set(THREADS_PREFER_PTHREAD_FLAG ON)
find_package(Threads REQUIRED)
target_link_libraries(project_name Threads::Threads)
```

3.2 Use compiled library for ROS

In this section we will explain the basic way to use ADS on a ROS program using the compiled library. Please note that the library was compiled in March 2022 and may not be up to date, if this is the case, please refer to section 3.1 which will explain how to compile it using the official github. We recommend compiling the library yourself, but some people have had problems with this (especially when using virtual machines).

- Download the folder "AdsLib" on the github page.
- copy it in your ROS package.
- Include these libraries in your c++ program.

```
#include "../AdsLib/AdsLib.h"
#include "../AdsLib/AdsVariable.h"
#include "../AdsLib/AdsNotificationOOI.h"
```

In the CMakeLists.txt of the package make the links to the library using this code.

```
FIND_LIBRARY(AdsLib_LIBRARIES AdsLib AdsLib/)
target_link_libraries(project_name ${AdsLib_LIBRARIES}
```

You should now be able to compile with catkin.

If you get this error : "undefined reference to symbol 'pthread_create@@GLIBC_2.2.5'" add the following lines to the CMakeLists.txt .

```
set(THREADS_PREFER_PTHREAD_FLAG ON)
find_package(Threads REQUIRED)
target_link_libraries(project_name Threads::Threads)
```

3.3 How to use the library

This section will show you how to use the library and exchange data with TwinCAT. Please note that the following examples are from my experience with my ROS program for the mobile platform and you will probably have to adapt it to your specific program. You can see the complete program on the github page (especially the ADS_Communication.cpp file of the ads_control package).

Also note that I will only explain what I have already experienced and used from the library. So I strongly recommend you to have a look at the "example/example.cpp" file of the ADS git https://github.com/Beckhoff/ADS which lists all the updated functions you can use.

First you need to create the route to the device, define "remoteNetId" and "remoteIpV4" according to your project (TwinCAT).

```
static const AmsNetId remoteNetId { 169, 254, 220, 1, 1, 1 };
static const char remoteIpV4[] = "169.254.220.1";
AdsDevice route {remoteIpV4, remoteNetId, AMSPORT_R0_PLC_TC3};;
```

You can initialise a simple variable linked to a TwinCAT variable. You must adapt the type to the TwinCAT type (for example : double : LReal , int16_t : INT, ...). You must also specify the name of the variable to be reached. TwinCAT type size : https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_plc_intro/2529399691.html

```
AdsVariable<double> cppLReal {route, "GVL.cppLReal"};
```

You can now read and write the variable as if it were a simple c++ variable. The link with the TwinCAT variable will be made automatically.

```
std::cout <<" ADS read " << cppLReal << '\n'; //read
cppLReal = 100; //write
```

However, we recommend that you group the variables to be exchanged in an array (max 500 variables per array) as this will considerably reduce the time needed to transmit the data. An exchange will take on average 1ms, if you send all your data in an array it will only take 1 ms instead of number of variables * 1ms.

```
AdsVariable<std::array<double, 3>> vel_robot{ route, "ControlGVL.vel_robot" };
std::cout <<" ADS read " << (double)vel_robot[0] << '\n'; //read
vel_robot = {0.0 , 0.0, 0.0}; //write
```

In the mobile platform project, we used an array to transmit data and AdsNotification to receive data. The ADS notification will automatically inform the ROS program if the value of the TwinCAT variable has changed using a callback function. This allows us to receive data at a higher frequency (possibly up to the frequency of the TwinCAT program) and also to save an array to exchange (in our case to increase the frequency of the ROS program).

First of all, you will have to create the attribute of the notification. You will have to indicate the size of the data to be received (in this example we receive 6 doubles).

ADSTRANS_SERVERCYCLE allows you to receive data cyclically, you can use ADSTRANS_SERVERONCHA if you want to receive data only when it is changed. We recommend not to change the other two data which will be suitable for most projects (otherwise refer to the definition in "standalone/AdsDef.h").

Then, in the notification initialization, specify the address of the function to call when the notification is received.

```
const AdsNotificationAttrib attrib = { sizeof(double)* 6,
                                        ADSTRANS_SERVERCYCLE, 0, { 10000 } };
AdsNotification notification{ route, "ControlGVL.robot_odom",
                               attrib, &odometryCallback, 0xBEEFDEAD };
```

Create your callback function with the following parameters and use reinterpret_cast to get the data. You can now use the variable as described above.

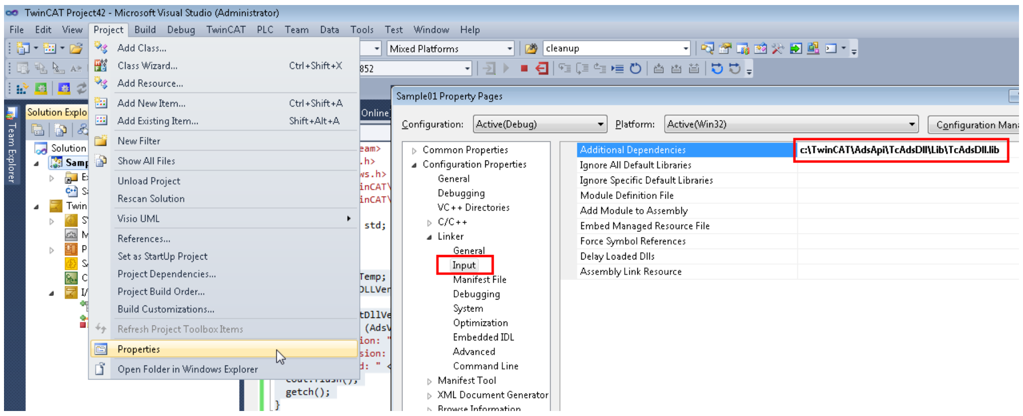
```
void odometryCallback(const AmsAddr* pAddr,
                      const AdsNotificationHeader* pNotification, uint32_t hUser){
        const double* data = reinterpret_cast<const double*>(pNotification + 1);
        std::cout << " ADS read "  << (double)data[0];}
```

4 Beckhoff ADS on Windows

In this section we will explain the basic way to read and write variables from a TwinCAT project. for more informations please refer to the Beckhoff website:
https://infosys.beckhoff.com/english.php?content=../content/1033/tcinfosys3/11291871243.html
First you have to include the following libraries to your program.

```
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsDef.h"
#include "C:\TwinCAT\AdsApi\TcAdsDll\Include\TcAdsApi.h"
```

You also need to add the path to the TcsAdsDll usually in the folder C:\TwinCAT\AdsApi\TcAdsDll.lib To add in visual code select Project/Properties/Configuration Properties/Linker/Input and add in additional dependencies the library.



Additional Dependencies: c:\TwinCAT\AdsApi\TcAdsDll\Lib\TcAdsDll.lib
Ignore All Default Libraries
Ignore Specific Default Libraries
Module Definition File
Add Module to Assembly
Embed Managed Resource File
Force Symbol References
Delay Loaded Dlls
Assembly Link Resource

Figure 2: Visual code include TcAdsDll

In the main function (or another function) you must first open communication with the ADS router. You will have to adapt the remote Net ID to your project (TwinCAT).

```
long        nErr, nPort;
AmsAddr     Addr;
PAmsAddr    pAddr = &Addr;

// Open communication port on the ADS router
nPort = AdsPortOpen();
nErr = AdsGetLocalAddress(pAddr);
if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';
pAddr->port = 851;
pAddr->netId = { 169, 254, 220, 1, 1, 1 };
```

Get the handle of a TwinCAT variable using the following code, replace varLRealName by the TwinCAT variable name.

```
ULONG      varLReal;
char       varLRealName[] = { "GVL.cppLReal" };

// Get the handle of the PLC-variable
nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0,
                           sizeof(varLReal), &varLReal,
                           sizeof(varLRealName), varLRealName);
if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '\n';
```

You can release the variable when you have finished to use it.

```
nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_RELEASEHND, 0,
                       sizeof(varLReal), &varLReal);
if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';
```

Get the value of the variable

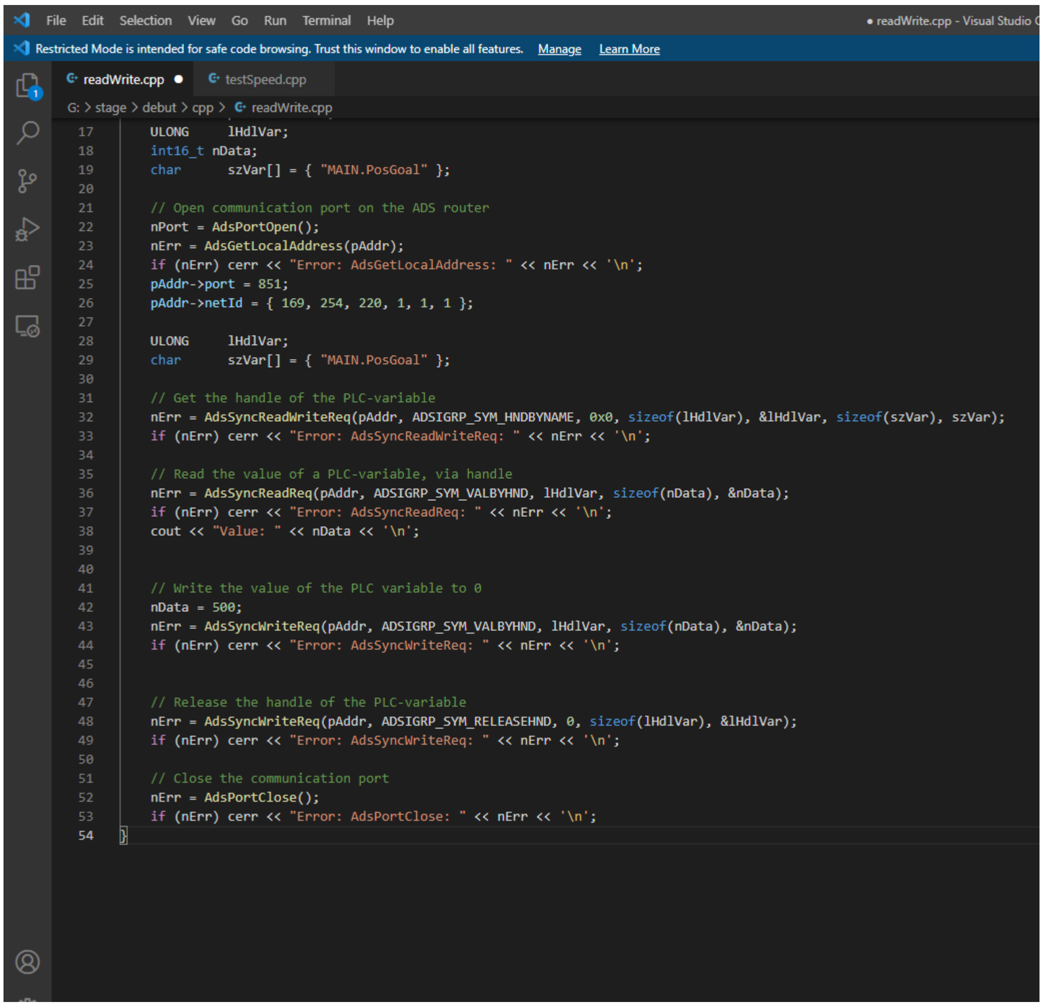
```
nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_RELEASEHND, 0,
                        sizeof(varLReal), &varLReal);
if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';
cout << "Value: " << varLReal << '\n';
```

Write the value of the variable

```
double data = 500;
nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND, varLReal,
                       sizeof(data), &data);
if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';
```

You can finally close the communication.

```
nErr = AdsPortClose();
if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
```


File Edit Selection View Go Run Terminal Help
● readWrite.cpp - Visual Studio
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
readWrite.cpp ●
testSpeed.cpp
G: > stage > debut > cpp > readWrite.cpp

```cpp
    ULONG     lHdlVar;
    int16_t nData;
    char      szVar[] = { "MAIN.PosGoal" };

    // Open communication port on the ADS router
    nPort = AdsPortOpen();
    nErr = AdsGetLocalAddress(pAddr);
    if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';
    pAddr->port = 851;
    pAddr->netId = { 169, 254, 220, 1, 1, 1 };

    ULONG     lHdlVar;
    char      szVar[] = { "MAIN.PosGoal" };

    // Get the handle of the PLC-variable
    nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof(lHdlVar), &lHdlVar, sizeof(szVar), szVar);
    if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '\n';

    // Read the value of a PLC-variable, via handle
    nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar, sizeof(nData), &nData);
    if (nErr) cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
    cout << "Value: " << nData << '\n';


    // Write the value of the PLC variable to 0
    nData = 500;
    nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar, sizeof(nData), &nData);
    if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';


    // Release the handle of the PLC-variable
    nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_RELEASEHND, 0, sizeof(lHdlVar), &lHdlVar);
    if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';

    // Close the communication port
    nErr = AdsPortClose();
    if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
}
```

Figure 3: simple c++ program