EPICS ADS Device support

Reference Manual

|  |  |
| --- | --- |
| Revision: | 2.0 |
| Status: | Released |
| Repository: | N/A |
| Project: | I-GroundZero-ADS |
| Folder: | N/A |
| Document ID: | CSL-MAN-18-159258 |
| File: | EPICS\_ADS\_Reference\_manual.docx |
| Owner: | Andrej Manojlović |
| Last modification: | September 7, 2020 |
| Created: | April 13, 2016 |
| Release Date: | October 24, 2012 |

|  |  |  |
| --- | --- | --- |
| **Prepared by** | **Reviewed by** | **Approved by** |
| Andrej Manojlović, Dejan Kocjan |  |  |
|  |  |  |

Document History



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Date | Changed/reviewed | Section(s) | Modification |
| 0.1 | 2018-08-24 | amanojlovic | All | First version of the document. |
| 1.0 | 2019-02-20 | amanojlovic | All | Released document |
| 2.0 | 2020-07-10 | amanojlovic | All | Update to match ADS port driver v2.0.0 |

Confidentiality

This document is classified as a confidential document. As such, it or parts thereof must not be made accessible to anyone not listed in the Audience section, neither in electronic nor in any other form.

Scope

The scope of this document is limited to:

* Instructions for configuring and running the software.

Audience

The intended audience of this document is Cosylab personnel.

Table of Contents

[1. Overview 7](#_Toc45630492)

[1.1. Software 7](#_Toc45630493)

[1.1.1. EPICS Base 7](#_Toc45630494)

[1.1.2. asynDriver 7](#_Toc45630495)

[1.1.3. Beckhoff ADS client library 8](#_Toc45630496)

[1.1.4. Dynamic Asyn Port Driver 8](#_Toc45630497)

[1.1.5. ADS port driver 8](#_Toc45630498)

[1.1.6. ADS sample IOC 8](#_Toc45630499)

[1.2. Terms used 8](#_Toc45630500)

[2. Installation Guide 10](#_Toc45630501)

[2.1. Software 10](#_Toc45630502)

[2.1.1. Compile the ADS EPICS modules 10](#_Toc45630503)

[3. Getting Started 14](#_Toc45630504)

[3.1. ADS Sample IOC 14](#_Toc45630505)

[3.1.1. TwinCAT XAE 14](#_Toc45630506)

[3.1.2. Sample IOC 17](#_Toc45630507)

[3.2. Include ADS device support in a new IOC 20](#_Toc45630508)

[4. Reference Manual 25](#_Toc45630509)

[4.1. Database record configuration 25](#_Toc45630510)

[4.2. iocsh commands 27](#_Toc45630511)

[4.2.1. AdsSetLocalAMSNetId 27](#_Toc45630512)

[4.2.2. AdsOpen 27](#_Toc45630513)

[5. Troubleshooting 29](#_Toc45630514)

[5.1. IOC fails to connect with "read frame failed with error: Connection reset by peer" errors 29](#_Toc45630515)

[5.2. IOC fails to connect with "Info: connection closed by remote" messages 29](#_Toc45630516)

[5.3. IOC fails to connect with "Connect TCP socket failed with: 111" errors 30](#_Toc45630517)

[5.4. IOC fails to connect with "Connect TCP socket failed with 110; Adding ADS route failed" error 30](#_Toc45630518)

[5.5. IOC loses ADS connection with "read frame failed with error: Connection reset by peer" and "connection closed by remote" messages 31](#_Toc45630519)

[5.6. IOC fails to connect with "Connect TCP socket failed with 113; Adding ADS route failed" error 31](#_Toc45630520)

[5.7. IOC connects but prints "Port 0x2710 is out of range" and "No response pending" warnings 32](#_Toc45630521)

[Appendix A. Record support 33](#_Toc45630522)

[A.1. Supported EPICS record types 34](#_Toc45630523)

[A.2. Supported TwinCAT PLC data types 35](#_Toc45630524)

[A.3. Database record configuration examples 35](#_Toc45630525)

[A.4. Supported ADS port names 37](#_Toc45630526)

Figures

[Figure 1: Example ADS address information for PLC variable “TestPlan.reg\_int32\_counter” 9](#_Toc45630527)

[Figure 2: Example of add route dialog 15](#_Toc45630528)

[Figure 3: Build TwinCAT project in XAE 16](#_Toc45630529)

[Figure 4: Activate boot project in XAE 17](#_Toc45630530)

Tables

[Table 1: List of software, hardware and documentation items in the system 7](#_Toc45630531)

[Table 2: Supported EPICS records with available asyn interfaces 34](#_Toc45630532)

[Table 3: TwinCAT PLC data types supported by asyn interfaces 35](#_Toc45630533)

[Table 4: ADS port names 37](#_Toc45630534)

Typography

This document uses the following styles:

|  |  |
| --- | --- |
|  | A box like this contains important information. |

|  |  |
| --- | --- |
|  | A box like this would contain sidebar text. |

|  |  |  |
| --- | --- | --- |
|  | Warning!  A box like this provides information, which should not be disregarded! |  |

Glossary of Terms

|  |  |
| --- | --- |
| ADS | Automation Device Specification |
| AMS | Automation Message Specification |
| PLC | Programmable Logic Controller |
| EPICS | Experimental Physics and Industrial Control System |
| IOC | Input/Output Controller |
| CA | Channel Access |
| PV | Process Variable |
| OPI | Operator Interface |
| CSS | Control System Studio |
| BOY | Best OPI, Yet |

References

1. Beckhoff ADS introduction: <https://infosys.beckhoff.com/content/1033/tcadscommon/html/tcadscommon_intro.htm?id=898081192215463875>
2. Beckhoff AMS TCP packet structure: <https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_ads_intro/9007199370585355.html>
3. Beckhoff ADS client library: <https://github.com/Beckhoff/ADS>
4. EPICS: <https://epics.anl.gov/>
5. asynDriver: <https://epics.anl.gov/modules/soft/asyn/>
6. Dynamic Asyn Port Driver: <https://git.cosylab.com/groundzero/beckhoffads/DynamicAsynPortDriver>
7. ADS port driver: <https://git.cosylab.com/groundzero/beckhoffads/ads-epics>
8. ADS sample IOC: <https://git.cosylab.com/groundzero/beckhoffads/ads-ioc>
9. ADS project proposal document: <https://git.cosylab.com/groundzero/g0-mngmnt/blob/master/proposals/CSL-G0-001-INT-aman-ADS-PRP.docx>
10. ADS software architecture document: <https://git.cosylab.com/epics-ads/epics-ads-management/blob/master/docs/architecture/EPICS_ADS_Architecture.docx>
11. ADS test plan document: <https://git.cosylab.com/epics-ads/epics-ads-management/blob/master/docs/test_plan/EPICS_ADS_Test_plan.docx>
12. Beckhoff TwinCAT PLC data types: <https://infosys.beckhoff.com/english.php?content=../content/1033/tcplccontrol/html/tcplcctrl_plc_data_types_overview.htm&id>
13. Phoebus: <https://github.com/ControlSystemStudio/phoebus>

# Overview

The EPICS ADS device support library is intended to connect EPICS database records to ADS variables and registers on ADS capable devices, also called ADS servers.

ADS [1] is a client/server interface or a set of commands (further referred as *ADS protocol*) that is used for communication with ADS capable devices. The basic set of ADS commands is:

* Read/write ADS data.
* Add/remove ADS data notification (subscribe to value change on the device).
* Read ADS device state and information.

The ADS protocol is encapsulated within the AMS protocol [2], which is itself usually encapsulated within the TCP protocol.

The system consists of the components listed below. The hardware listed is intended only for demonstration purposes.

Table 1: List of software, hardware and documentation items in the system

|  |  |
| --- | --- |
| Software | * EPICS Base [4] * asynDriver [5] * Beckhoff ADS client library [3] * Dynamic Asyn Port Driver [6] * ADS port driver [7] * ADS sample IOC [8] |
| Documentation | * Project proposal [9] * Software architecture [10] * Test plan [11] |

## Software

This chapter briefly describes the software that constitutes the EPICS ADS device support.

### EPICS Base

The *EPICS Base* (third party software) provides the tools, libraries and headers needed to compile and run the ADS device support library, as well as the ADS sample IOC.

Refer to the official website [4] for more information.

### asynDriver

The *asynDriver* (third party software) is a general-purpose library for interfacing device specific code to low level drivers. It is used as a base by the Dynamic Asyn Port Driver and the ADS port driver.

Refer to the official website [5] for more information.

### Beckhoff ADS client library

*Beckhoff ADS client library* implements the AMS/ADS protocol on top of TCP/IP. The library is used by the ADS port driver.

Refer to the official website [3] for more information.

### Dynamic Asyn Port Driver

The Dynamic Asyn Port Driver is a library that simplifies integration of devices with lots of I/O signals of different types, which are handled in a similar way. The ADS port driver extends this library to keep track of which ADS addresses map to which database records in the IOC.

### ADS port driver

The ADS port driver implements the layer between the database records and the Beckhoff ADS client library. It exchanges data between input/output records and corresponding ADS variables.

### ADS sample IOC

The ADS sample IOC provides an example IOC configuration (a reference implementation) that uses the ADS port driver.

The IOC includesdatabase used with Test Plan [11] simulator PLC program, running in TwinCAT XAE. It demonstrates how different EPICS record types map to the PLC data types.

## Terms used

The term register is used to describe an address on the ADS service, which corresponds to a specific I/O terminal's input or output channel raw value or a PLC program variable. For example, a variable can have an address specification such as this: Port: 300, IGrp: 0x0x8502000, IOffs: 0x8103E800, Len: 4 [Figure 1].

The term variable or symbolic name is used to describe a named data block in the memory of the program running on the ADS server. Each variable has a specified size and type (e.g. INT, UINT, FLOAT, array of BOOL, etc.). An example variable name can be MAIN.input\_temperature, where the first part MAIN represents the name of the running program (in this case the PLC main program) and the second part input\_temperature the name of the variable used in that PLC program. Depending on the PLC program, variables can either be linked to physical I/O terminals (e.g. to a specific 0..10 V analog input channel connected to a temperature sensor), or they can be logical – derived from I/O channels (e.g. converted raw input value from temperature sensor to degrees Celsius), or representing some internal state (e.g. program running cycle counter).

Note that with ADS, both registers and variables are finally accessed in the same way: using the port, group, offset and length specifiers. The difference is that for variables, the symbolic name is first dereferenced into an ADS group and offset specifiers (further referenced in this document as ***handle***).

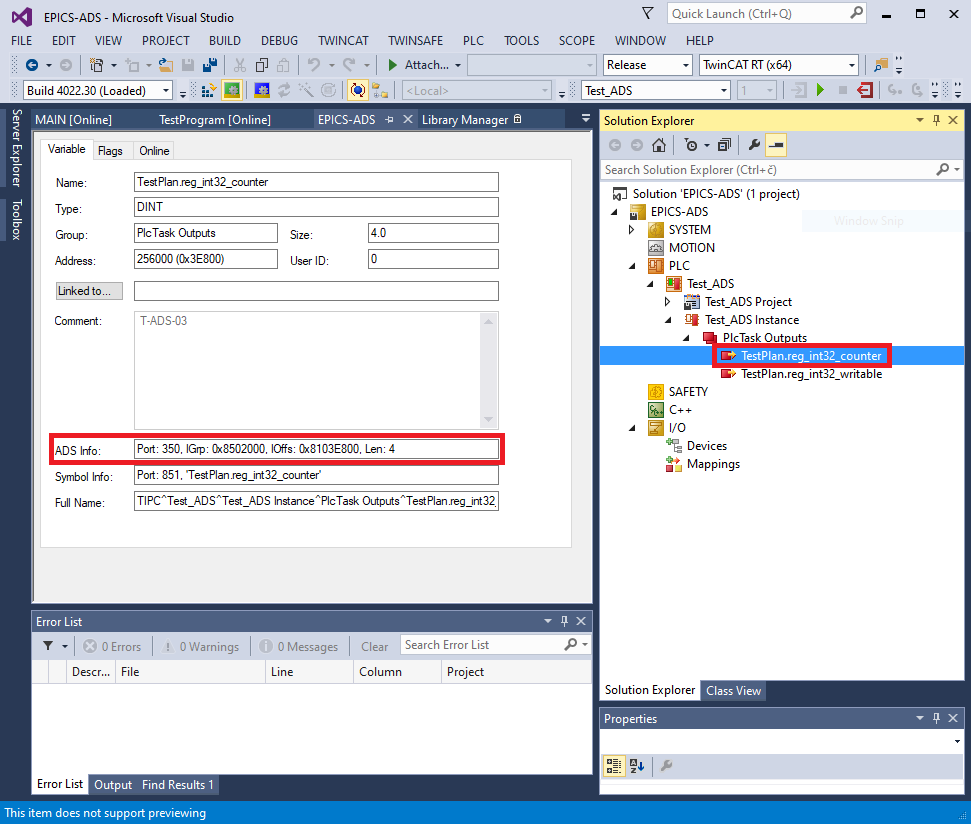


Figure 1: Example ADS address information for PLC variable “TestPlan.reg\_int32\_counter”

# Installation Guide

This chapter describes how to compile the ADS device support EPICS modules.

## Software

### Compile the ADS EPICS modules

The ADS device support software consists of modules in Git repositories: the Dynamic Asyn Port Driver [6] and the ADS port driver [7].

**Pre-requisite software:**

* EPICS base: R3.15
* asynDriver: R4-39
* Boost: The Boost C++ libraries
* gcc: GNU Compiler Collection
* g++: GCC for C++
* git: version control system

#### Dynamic Asyn Port Driver

Download the latest stable version from Gitlab [6]:

$ git clone git@git.cosylab.com:epics-ads/DynamicAsynPortDriver.git

Cloning into 'DynamicAsynPortDriver'...

remote: Enumerating objects: 284, done.

remote: Counting objects: 100% (284/284), done.

remote: Compressing objects: 100% (121/121), done.

remote: Total 284 (delta 191), reused 251 (delta 159)

Receiving objects: 100% (284/284), 81.87 KiB | 0 bytes/s, done.

Resolving deltas: 100% (191/191), done.

The provided configure/RELEASE file assumes that the module’s parent directory contains a RELEASE file with the required module paths (i.e. ASYN and EPICS\_BASE).

# RELEASE - Location of external support modules

#

# Required external module paths:

# ASYN - asynDriver

# EPICS\_BASE - EPICS base

# External modules are expected to be defined in a RELEASE file located

# in $(TOP)'s parent directory.

-include $(TOP)/../RELEASE

Edit the configure/RELEASE file and set the correct path to the EPICS\_BASE and ASYN modules:

$ vim DynamicAsynPortDriver/configure/RELEASE

# RELEASE - Location of external support modules

...

## EPICS Modules

**ASYN=/opt/epics/modules/asyn**

# If using the sequencer, point SNCSEQ at its top directory:

#SNCSEQ = $(MODULES)/seq-ver

# EPICS\_BASE should appear last so earlier modules can override stuff:

**EPICS\_BASE = /opt/epics/base**

# Set RULES here if you want to use build rules from somewhere

# other than EPICS\_BASE:

...

Set the target installation location in configure/CONFIG\_SITE (optional step):

$ vim DynamicAsynPortDriver/configure/CONFIG\_SITE

# CONFIG\_SITE

...

# To install files into a location other than $(TOP) define

# INSTALL\_LOCATION here.

**INSTALL\_LOCATION=/opt/epics/modules/DynamicAsynPortDriver**

...

Finally, compile the module:

$ cd DynamicAsynPortDriver/

$ make

make -C ./configure install

make[1]: Entering directory `/home/epics-dev/DynamicAsynPortDriver/configure'

perl -CSD /opt/epics/base/bin/linux-x86\_64/makeMakefile.pl O.linux-x86\_64 ../..

mkdir O.Common

make -C O.linux-x86\_64 -f ../Makefile TOP=../.. \

T\_A=linux-x86\_64 install

... <output removed for brevity>

make -C ./iocBoot install

make[1]: Entering directory `/home/epics-dev/DynamicAsynPortDriver/iocBoot'

make -C ./iocreasonTest install

make[2]: Entering directory `/home/epics-dev/DynamicAsynPortDriver/iocBoot/iocreasonTest'

perl -CSD /opt/epics/base/bin/linux-x86\_64/convertRelease.pl -t /opt/epics/modules/DynamicAsynPortDriver envPaths

make[2]: Leaving directory `/home/epics-dev/DynamicAsynPortDriver/iocBoot/iocreasonTest'

make[1]: Leaving directory `/home/epics-dev/DynamicAsynPortDriver/iocBoot'

#### ADS port driver

The procedure is similar to the one described for the Dynamic Asyn Port Driver above. First download the latest stable version from Gitlab [7]:

$ git clone git@git.cosylab.com:epics-ads/ads.git

Cloning into 'ads'...

remote: Enumerating objects: 371, done.

remote: Counting objects: 100% (371/371), done.

remote: Compressing objects: 100% (216/216), done.

remote: Total 371 (delta 203), reused 287 (delta 137)

Receiving objects: 100% (371/371), 618.27 KiB | 0 bytes/s, done.

Resolving deltas: 100% (203/203), done.

The provided configure/RELEASE file assumes that the module’s parent directory contains a RELEASE file with the required module paths (i.e. ASYN, DYNAMICASYNPORTDRIVER and EPICS\_BASE).

# RELEASE - Location of external support modules

#

# Required external module paths:

# ASYN - asynDriver

# DYNAMICASYNPORTDRIVER - DynamicAsynPortDriver

# EPICS\_BASE - EPICS base

# External modules are expected to be defined in a RELEASE file located

# in $(TOP)'s parent directory.

-include $(TOP)/../RELEASE

Set the target installation location in configure/CONFIG\_SITE (optional step):

$ vim ads/configure/CONFIG\_SITE

# CONFIG\_SITE

...

# To install files into a location other than $(TOP) define

# INSTALL\_LOCATION here.

**INSTALL\_LOCATION=/opt/epics/modules/ads**

...

Compile the module:

$ cd ads/

$ make

make -C ./configure install

make[1]: Entering directory `/home/epics-dev/ads/configure'

perl -CSD /opt/epics/base/bin/linux-x86\_64/makeMakefile.pl O.linux-x86\_64 ../..

mkdir O.Common

make -C O.linux-x86\_64 -f ../Makefile TOP=../.. \

T\_A=linux-x86\_64 install

... <output removed for brevity>

make -C O.linux-x86\_64 -f ../Makefile TOP=../../.. \

T\_A=linux-x86\_64 install

make[3]: Entering directory `/home/epics-dev/ads/adsApp/Db/O.linux-x86\_64'

make[3]: Nothing to be done for `install'.

make[3]: Leaving directory `/home/epics-dev/ads/adsApp/Db/O.linux-x86\_64'

make[2]: Leaving directory `/home/epics-dev/ads/adsApp/Db'

make[1]: Leaving directory `/home/epics-dev/ads/adsApp'

The *make* step also compiles the included Beckhoff ADS client library, which is required by the ADS port driver.

# Getting Started

Section 3.1 describes how to compile and run the ADS sample IOC, which is provided as a reference implementation with the ADS device support library. Section 3.2 describes how to include the ADS device support in a new IOC.

## ADS Sample IOC

The ADS sample IOC is provided as a reference IOC implementation with the ADS device support library. To successfully run the IOC, a connection with the PLC program running in TwinCAT XAE must be setup.

### TwinCAT XAE

The PLC program can be found in the TC3\_Project directory in the ads-sample-ioc git repository [8].

#### Configure ADS route

First open the project in TwinCAT XAE, click SYSTEM -> Routes, then click the Add button. Fill in route parameters [Figure 2]:

* **Route name**: Descriptive name of the new route.
* **AmsNetId**: AMS net ID of the IOC that will connect to the PLC program.
* **Address info**: IP address of the IOC that will connect to the PLC program.
* **Host name/IP address**: select *IP address* to match the above *Address info* specification.
* **Remote route**: set to None.

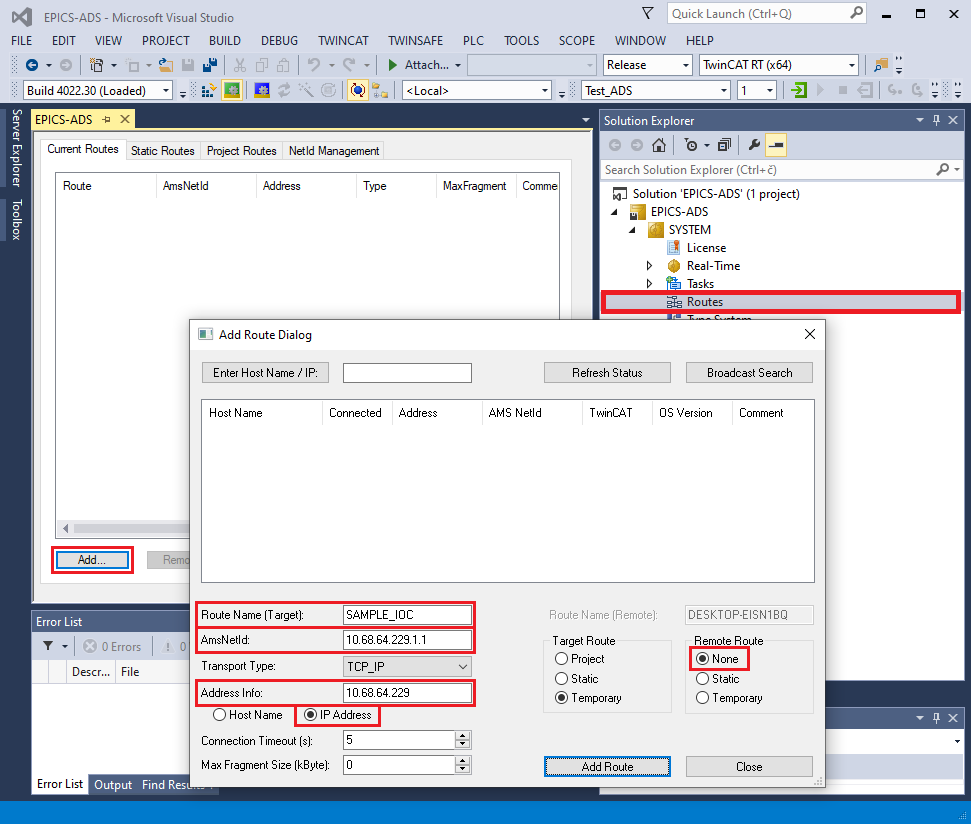


Figure 2: Example of add route dialog

#### Build PLC project

Next, right click on Test\_ADS Project , then click Build to build the project [Figure 3].

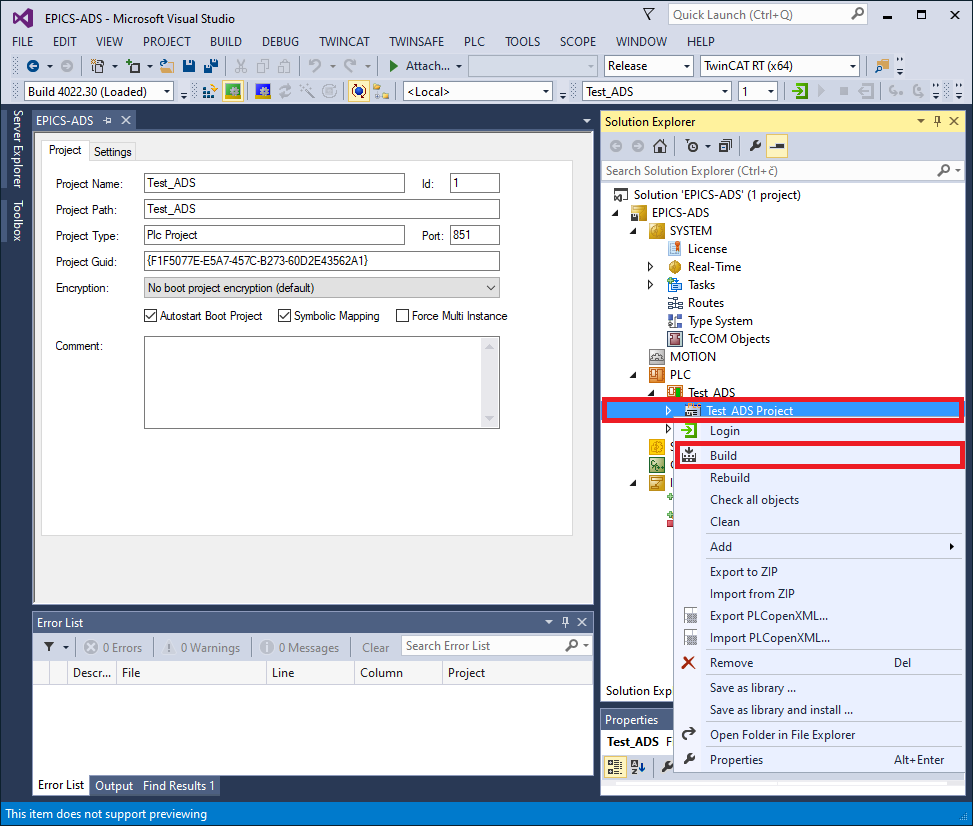


Figure 3: Build TwinCAT project in XAE

#### Auto-start and activate PLC project

Assuming that the program compiled without errors, click on Test\_ADS and make sure that Autostart Boot Project is checked. Then right click on Test\_ADS again and click Activate Boot Project.

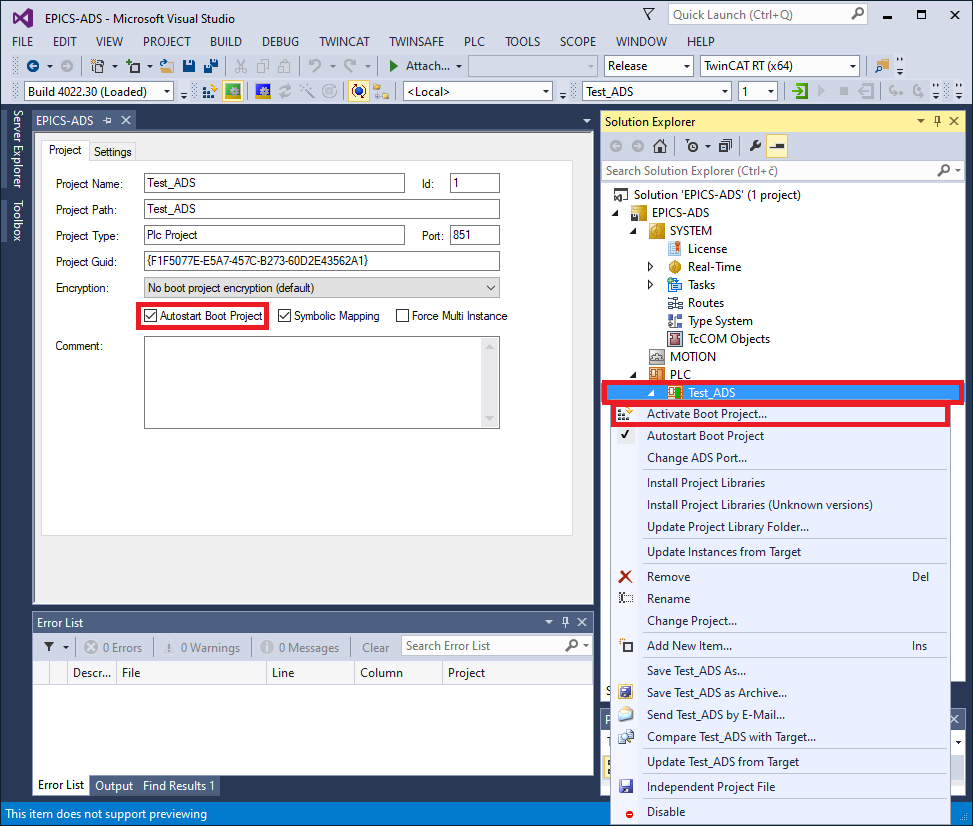


Figure 4: Activate boot project in XAE

#### Activate configuration

The last step is to activate the configuration by clicking TWINCAT -> Activate Configuration. Make sure that the PLC is in Run mode before connecting to the PLC with the IOC.

### Sample IOC

#### Compile the sample IOC

First make sure that the Dynamic Asyn Port Driver [2.1.1.1] and ADS port driver [2.1.1.2] are both compiled. The same software pre-requisites apply as for the above software.

Next, download the latest stable version from Gitlab [8].

$ git clone git@git.cosylab.com:epics-ads/ads-sample-ioc.git

Cloning into 'ads-sample-ioc'...

remote: Enumerating objects: 525, done.

remote: Counting objects: 100% (525/525), done.

remote: Compressing objects: 100% (298/298), done.

remote: Total 525 (delta 231), reused 486 (delta 193)

Receiving objects: 100% (525/525), 3.30 MiB | 0 bytes/s, done.

Resolving deltas: 100% (231/231), done.

The provided configure/RELEASE file assumes that the module's parent directory contains a RELEASE file with th erequired module paths (i.e. ASYN, ADS and EPICS\_BASE).

# RELEASE - Location of external support modules

#

# Required external module paths:

# ASYN - asynDriver

# ADS - ADS port driver

# EPICS\_BASE - EPICS base

# External modules are expected to be defined in a RELEASE file located

# in $(TOP)'s parent directory.

-include $(TOP)/../RELEASE

Compile the sample IOC:

$ cd ads-sample-ioc/

$ make

make -C ./configure install

make[1]: Entering directory `/home/epics-dev/ads-sample-ioc/configure'

perl -CSD /opt/epics/base/bin/linux-x86\_64/makeMakefile.pl O.linux-x86\_64 ../..

mkdir O.Common

make -C O.linux-x86\_64 -f ../Makefile TOP=../.. \

T\_A=linux-x86\_64 install

... <output removed for brevity>

make -C ./iocBoot install

make[1]: Entering directory `/home/epics-dev/ads-sample-ioc/iocBoot'

make -C ./iocads-sample install

make[2]: Entering directory `/home/epics-dev/ads-sample-ioc/iocBoot/iocads-sample'

perl -CSD /opt/epics/base/bin/linux-x86\_64/convertRelease.pl -t /home/epics-dev/ads-sample-ioc envPaths

make[2]: Leaving directory `/home/epics-dev/ads-sample-ioc/iocBoot/iocads-sample'

make[1]: Leaving directory `/home/epics-dev/ads-sample-ioc/iocBoot'

#### st.cmd configuration

Edit st.cmd:

1. Change the local AMS net ID. This can be the same as the local IP address, with an additional “.1.1” suffix, e.g. 10.68.64.229.1.1). This local (IOC) AMS ID and IP address must be added as a route as described in section 3.1.1.
2. Set the connection parameters for the host running the *Test plan project* PLC program:
   * **IP\_TEST\_PLAN**: IP address
   * **AMS\_ID\_TEST\_PLAN**: AMS ID

**$ vim iocBoot/iocads-sample/st.cmd**

#!../../bin/linux-x86\_64/ads-sample

## Register all support components

dbLoadDatabase "../../dbd/ads-sample.dbd"

ads\_sample\_registerRecordDeviceDriver pdbbase

# Set local AMS net ID

**AdsSetLocalAMSNetID("10.68.64.229.1.1")**

# Test Plan PLC program connection parameters

epicsEnvSet("PREFIX\_TEST\_PLAN", "TEST-PLAN-01")

epicsEnvSet("PORT\_TEST\_PLAN", "ads-test-plan")

**epicsEnvSet("IP\_TEST\_PLAN", "10.68.6.45")**

**epicsEnvSet("AMS\_ID\_TEST\_PLAN", "10.68.79.25.1.1")**

# Load record instances for Test Plan PLC program

dbLoadRecords("../../db/test\_plan\_simulator.db","P=$(PREFIX\_TEST\_PLAN), PORT=$(PORT\_TEST\_PLAN)")

# Open ADS port

AdsOpen("$(PORT\_TEST\_PLAN)", "$(IP\_TEST\_PLAN)", "$(AMS\_ID\_TEST\_PLAN)")

# Second instance of test Plan PLC program connection parameters

#epicsEnvSet("PREFIX\_TEST\_PLAN", "TEST-PLAN-02")

#epicsEnvSet("PORT\_TEST\_PLAN", "ads-test-plan2")

#epicsEnvSet("IP\_TEST\_PLAN", "10.68.6.46")

#epicsEnvSet("AMS\_ID\_TEST\_PLAN", "10.68.79.26.1.1")

# Load record instances for Test Plan PLC program

#dbLoadRecords("../../db/test\_plan\_simulator.db","P=$(PREFIX\_TEST\_PLAN), PORT=$(PORT\_TEST\_PLAN)")

# Open ADS port

#AdsOpen("$(PORT\_TEST\_PLAN)", "$(IP\_TEST\_PLAN)", "$(AMS\_ID\_TEST\_PLAN)")

# Enable asyn trace output for errors and warnings

asynSetTraceMask("$(PORT\_TEST\_PLAN)", 0, 0x21)

# Alternatively, output everything

#asynSetTraceMask("$(PORT\_TEST\_PLAN)", 0, 0xff)

iocInit

# End of IOC initialization

#### Run the sample IOC

Now that the simulated PLC is running in the TwinCAT XAE and the st.cmd is configured, the IOC can be started:

$ cd iocBoot/iocads-sample/

$ ../../bin/linux-x86\_64-debug/ads-sample st.cmd

#!../../bin/linux-x86\_64/ads-sample

## Register all support components

dbLoadDatabase "../../dbd/ads-sample.dbd"

ads\_sample\_registerRecordDeviceDriver pdbbase

# Set local AMS net ID

AdsSetLocalAMSNetID("10.68.64.229.1.1")

# Test Plan PLC program connection parameters

epicsEnvSet("PREFIX\_TEST\_PLAN", "TEST-PLAN-01")

epicsEnvSet("PORT\_TEST\_PLAN", "ads-test-plan")

epicsEnvSet("IP\_TEST\_PLAN", "10.68.6.45")

epicsEnvSet("AMS\_ID\_TEST\_PLAN", "10.68.79.25.1.1")

# Load record instances for Test Plan PLC program

dbLoadRecords("../../db/test\_plan\_simulator.db","P=TEST-PLAN-01, PORT=ads-test-plan")

# Open ADS port

AdsOpen("ads-test-plan", "10.68.6.45", "10.68.79.25.1.1")

# Enable asyn trace output for errors and warnings

asynSetTraceMask("ads-test-plan", 0, 0x21)

# Alternatively, output everything

#asynSetTraceMask("$(PORT\_TEST\_PLAN)", 0, 0xff)

iocInit

Starting iocInit

############################################################################

## EPICS R7.0.2.2

## EPICS Base built Feb 11 2020

############################################################################

2020/07/10 13:53:37.555 [WARNING] ADSPortDriver.cpp:169 connect(): [ads-test-plan] IOC not initialized

2020-07-10T13:53:38+0200 Info: Connected to 10.68.6.45

2020/07/10 13:53:38.085 [WARNING] ADSPortDriver.cpp:193 connect(): [ads-test-plan] connected to ADS device (IP: 10.68.6.45)

2020/07/10 13:53:38.422 [WARNING] ADSPortDriver.cpp:203 connect(): [ads-test-plan] resolved 22 ADS read variable names

2020/07/10 13:53:38.766 [WARNING] ADSPortDriver.cpp:213 connect(): [ads-test-plan] resolved 22 ADS write variable names

2020/07/10 13:53:38.766 [WARNING] ADSPortDriver.cpp:221 connect(): [ads-test-plan] initialized sum-read request buffers

iocRun: All initialization complete

# End of IOC initialization

epics>

Refer to the troubleshooting section [5] if the IOC fails to connect to the ADS devices.

|  |  |
| --- | --- |
|  | The ads-sample-ioc repository contains GUI screens for the simulated PLC (gui/bob/test\_plan.bob) which can be open in Phoebus [13]. |

## Include ADS device support in a new IOC

This example shows how to create a new IOC with the ADS device support included. For this example, create a new IOC using makeBaseApp.pl:

**$ mkdir test-ioc**

**$ cd test-ioc/**

**$ makeBaseApp.pl -t ioc test**

**$ makeBaseApp.pl -i -t ioc test**

Using target architecture linux-x86\_64 (only one available)

The following applications are available:

test

What application should the IOC(s) boot?

The default uses the IOC's name, even if not listed above.

Application name? **test**

Which should produce a directory structure similar to this:

**$ ls -R**

.:

configure iocBoot Makefile testApp

./configure:

CONFIG CONFIG\_SITE Makefile RELEASE RULES RULES\_DIRS RULES.ioc RULES\_TOP

./iocBoot:

ioctest Makefile

./iocBoot/ioctest:

Makefile st.cmd

./testApp:

Db Makefile src

./testApp/Db:

Makefile

./testApp/src:

Makefile testMain.cpp

Now add the required ADS EPICS modules to the configure/RELEASE file: ASYN and ADS. It is not necessary to set the DYNAMICASYNPORTDRIVER module path, since it is linked with the ADS port driver library. Also, because the IOC was created using makeBaseApp.pl, the EPICS\_BASE should already be pointing to the correct location:

**$ vim configure/RELEASE**

# RELEASE - Location of external support modules

...

# Variables and paths to dependent modules:

#MODULES = /path/to/modules

#MYMODULE = $(MODULES)/my-module

**ASYN = /opt/epics/modules/asyn**

**ADS = /opt/epics/modules/ads**

# If using the sequencer, point SNCSEQ at its top directory:

#SNCSEQ = $(MODULES)/seq-ver

# EPICS\_BASE should appear last so earlier modules can override stuff:

**EPICS\_BASE = /opt/epics/base**

# Set RULES here if you want to use build rules from somewhere

# other than EPICS\_BASE:

...

Edit the IOC's src/Makefile and include asynDriver and ADS port driver's DBD and library files. It is not necessary to include DBD and library files from the Dynamic Asyn Port Driver module.

$ vim testApp/src/Makefile

TOP=../..

include $(TOP)/configure/CONFIG

#----------------------------------------

# ADD MACRO DEFINITIONS AFTER THIS LINE

#=============================

#=============================

# Build the IOC application

PROD\_IOC = test

# test.dbd will be created and installed

DBD += test.dbd

# test.dbd will be made up from these files:

test\_DBD += base.dbd

# Include dbd files from all support applications:

#test\_DBD += xxx.dbd

**test\_DBD += asyn.dbd ads.dbd**

# Add all the support libraries needed by this IOC

#test\_LIBS += xxx

**test\_LIBS += asyn ads**

# test\_registerRecordDeviceDriver.cpp derives from test.dbd

test\_SRCS += test\_registerRecordDeviceDriver.cpp

# Build the main IOC entry point on workstation OSs.

test\_SRCS\_DEFAULT += testMain.cpp

test\_SRCS\_vxWorks += -nil-

# Add support from base/src/vxWorks if needed

#test\_OBJS\_vxWorks += $(EPICS\_BASE\_BIN)/vxComLibrary

# Finally link to the EPICS Base libraries

test\_LIBS += $(EPICS\_BASE\_IOC\_LIBS)

#===========================

include $(TOP)/configure/RULES

#----------------------------------------

# ADD RULES AFTER THIS LINE

Compile the IOC:

$ make

make -C ./configure install

make[1]: Entering directory `/home/epics-dev/test-ioc/configure'

perl -CSD /opt/epics/base/bin/linux-x86\_64/makeMakefile.pl O.linux-x86\_64 ../..

mkdir O.Common

make -C O.linux-x86\_64 -f ../Makefile TOP=../.. \

T\_A=linux-x86\_64 install

... <output removed for brevity>

make -C ./iocBoot install

make[1]: Entering directory `/home/epics-dev/test-ioc/iocBoot'

make -C ./ioctest install

make[2]: Entering directory `/home/epics-dev/test-ioc/iocBoot/ioctest'

perl -CSD /opt/epics/base/bin/linux-x86\_64/convertRelease.pl -t /home/epics-dev/test-ioc envPaths

make[2]: Leaving directory `/home/epics-dev/test-ioc/iocBoot/ioctest'

make[1]: Leaving directory `/home/epics-dev/test-ioc/iocBoot'

Assuming the compilation step was successful, you can run the IOC and check if ADS port driver commands are available in the IOC shell:

$ cd iocBoot/ioctest/

$ ../../bin/linux-x86\_64-debug/test st.cmd

#!../../bin/linux-x86\_64/test

## You may have to change test to something else

## everywhere it appears in this file

< envPaths

epicsEnvSet("IOC","ioctest")

epicsEnvSet("TOP","/home/epics-dev/test-ioc")

epicsEnvSet("ASYN","/opt/epics/modules/asyn")

epicsEnvSet("ADS","/opt/epics/modules/ads")

epicsEnvSet("EPICS\_BASE","/opt/epics/base")

cd "/home/epics-dev/test-ioc"

## Register all support components

dbLoadDatabase "dbd/test.dbd"

test\_registerRecordDeviceDriver pdbbase

## Load record instances

#dbLoadRecords("db/xxx.db","user=andrej")

cd "/home/epics-dev/test-ioc/iocBoot/ioctest"

iocInit

Starting iocInit

############################################################################

## EPICS R3.15.5

## EPICS Base built Aug 31 2018

############################################################################

iocRun: All initialization complete

## Start any sequence programs

#seq sncxxx,"user=andrej"

Run help command and verify that the ADS port driver commands are available, i.e. AdsOpen and AdsSetLocalAMDSNetID:

epics> help

Type 'help <command>' to see the arguments of <command>.

# AdsFindIOIntrRecord **AdsOpen**

**AdsSetLocalAMSNetID** ClockTime\_Report

ClockTime\_Shutdown asDumpHash asInit asSetFilename

asSetSubstitutions ascar asdbdump asphag

aspmem asprules aspuag astac

asynAutoConnect asynEnable asynInterposeEosConfig

asynInterposeFlushConfig asynOctetConnect

asynOctetDisconnect asynOctetFlush asynOctetGetInputEos

... <output removed for brevity>

You now have a functional IOC that can use the ADS protocol to communicate with ADS capable devices. What is missing is a functional database with record configuration for the ADS device that you are connecting to the IOC, and an open ADS connection between them.

Refer to the ADS sample IOC sources [8] to see an example how the EPICS database is configured and how to open an ADS client connection from the IOC to an ADS device. Refer to section 4 for record configuration and IOC command description.

# Reference Manual

This section describes how to configure database records and the commands available in the IOC shell.

## Database record configuration

Database records are configured as follows:

* DTYP field specifies the asyn interface (e.g. *asynInt32*, *asynFloat64*, *asynInt8ArrayIn*) and is set according to the record type. Some records allow different asyn interfaces to be set. For more information see asynDriver distribution's dbd directory, devAsyn\*.dbd (e.g. devAsynInt32.dbd) files. See [A.1] for a list of supported asyn interfaces.
* INP and OUT fields uses the asyn drvParams optional parameter (see [5], section Generic Device Support for EPICS records) to specify the ADS register/variable that the record is connecting to, number of elements and the operation to be performed (read or write). For example: INP=@asyn(plc-01 0 0) USINT R P=300 G=0x14002 O=0x349 or OUT=@asyn(plc-02 0 0) REAL[10] W P=IO V=Main.SetCalibration.
* SCAN field determines when the record is processed and, depending on the record type, how their value is handled:
  + Output records with SCAN=Passive/periodic write to their corresponding ADS variable immediately when they are processed.
  + Input records with SCAN=Passive/periodic have their VAL field populated with the latest value from their ADS variable's corresponding sum-read data buffers when processed.
  + Input records with SCAN=I/O Intr have their VAL field populated immediately after the port driver detects their corresponding ADS variable's value in the sum-read databa buffer has changed.

|  |  |  |
| --- | --- | --- |
|  | ***In current ADS port driver version, a large number of simultaneous write requests can saturate the ADS connection and cause the system to become unresponsive and cause records to time out.*** |  |

The format used to specify the ADS address/variable in the INP/OUT fields is: DATA\_TYPE[NELEM] OPERATION TARGET

Where:

* DATA\_TYPE part specifies one of the supported PLC data types, e.g. USINT, LREAL, STRING, etc. See [A.2] for a list of supported PLC data types.
* NELEM (optional) is used to specify number of elements for array access, as well as to specify the length of STRING PLC variables. The default value for the parameter is 1.
* OPERATION specifies if the PLC variable is read (R) or written (W).
* TARGET specifies the ADS register or variable:
  + For register access:
    - P= ADS port in string or numerical format, e.g. P=IO, P=300 or P=0x12c. See section A.4 for supported port names.
    - G= ADS group, e.g. G=0x14002 or G=81922.
    - O= ADS offset, e.g. O=0x349 or O=841.
  + For variable access:
    - P: ADS port in string or numerical format. The same parameter constraints apply as for register access.
    - V: ADS variable name in string format, e.g. V=Main.temperature.

Example register address specifiers:

* Write an array of 5 UINT (16-bit unsigned int) values to PLC register with ADS index group 0x14002 and ADS index offset 0x349:

UINT[5] W P=PLC\_TC3 G=0x14002 O=0x349

* Write a REAL (32-bit float) value to ADS index group 0x14002 and ADS index offset 0x400. The lack of [nelem] implies a scalar type:

REAL W P=PLC\_TC3 G=0x14002 O=0x400

* Read 10 BYTE (8-bit int) values from ADS index group 0x14002 and ADS index offset 0x450:

BYTE[10] R P=PLC\_TC3 G=0x14002 O=0x400

Example variable name specifiers:

* Write an array of 5 UINT (16-bit unsigned int) values to PLC variable named *Main.Waveform*:

UINT[5] W P=PLC\_TC3 V=Main.Waveform

* Write a REAL (32-bit float) value to PLC variable named *Main.CorrectionFactor*. The lack of [nelem] implies a scalar type:

REAL W P=PLC\_TC3 V=Main.CorrectionFactor

* Read 10 BYTE (8-bit int) values from PLC variable named *Main.Values*:

BYTE[10] R P=PLC\_TC3 V=Main.Values

Example database record configuration:

record(ai, "BECKHOFF:ANALOG\_IN:01") {

field(DESC, "Read first 16bit analog input")

field(DTYP, "asynFloat64")

field(INP, "@asyn(beckhoff 0 0) INT R P=IO G=0x13002 O=0x0")

}

Refer to section [A.3] for more examples on how to configure database records.

## iocsh commands

This chapter describes the iocsh commands that are provided by the ADS device support software.

### AdsSetLocalAMSNetId

**Description:**

Set the local AMS net ID of the IOC. This is used in ADS communication to represent the identity of the IOC – the ADS client. This command must be called before any calls to AdsOpen [4.2.2], i.e. before any ADS connections are open.

**Interface:**

AdsSetLocalAMSNetId(ams\_net\_id)

**Parameters:**

* ams\_net\_id: AMS net ID that will be set locally on the IOC. E.g. "192.168.20.10.1.5".

**Example:**

# Set the local AMS net ID to "192.168.20.10.1.5".

AmsSetLocalAMSNetId("192.168.20.10.1.5")

### AdsOpen

**Description:**

Configure a new ADS connection. This command must be called before corresponding database records are loaded, i.e. before dbLoadRecord is called.

**Interface:**

AdsOpen(port\_name, ip\_addr, ams\_net\_id, sum\_buffer\_nelem, ads\_timeout, no\_auto\_connect, priority)

**Parameters:**

* port\_name: The port name that is registered with asynManager and is used in the INP/OUT address specifications for the records.
* ip\_addr: IP address of the remote ADS device.
* ams\_net\_id: AMS net ID of the remote ADS device.
* sum\_buffer\_nelem (optional)***:*** The maximum number of PVs that sum-read request and data buffers can contain. Defaults to 500.
* ads\_timeout (optional): The current version of the ADS device support (v2.0.0) does not implement *ADS function timeout* feature. ADS client library uses the default value of 5000 ms.
* no\_auto\_connect (optional): Disables auto-connect functionality for the port instance if set to a non-zero value.
* priority (optional): Thread priority for the port instance. If not specified or is set to zero, epicsThreadPriority is used. See epicsThread.h found in EPICS base for thread priority values.

Example:

# Configure an ADS connection with the optional parameters not specified.

AdsOpen("plc-01", "10.5.0.115", "10.5.0.115.1.1")

# Configure an ADS connection with all parameters. Here ADS sum operation buffer PV limit is set to 250, ads timeout to 1 second, auto connect is disabled and the default thread priority is used.

AdsOpen("plc-02", "10.5.0.120", "10.5.0.120.1.15", 250, 1000, 1, 0)

# Troubleshooting

## IOC fails to connect with "read frame failed with error: Connection reset by peer" errors

During IOC boot, when trying to connect to an ADS device the IOC reports *Error: read frame failed with error: Connection reset by peer* errors.

**Example:**

2020-07-10T17:35:28+0200 Info: Connected to 10.68.6.45

2020/07/10 17:35:28.326 [WARNING] ADSPortDriver.cpp:193 connect(): [ads-test-plan] connected to ADS device (IP: 10.68.6.45)

2020-07-10T17:35:28+0200 **Error: read frame failed with error: Connection reset by peer**

2020-07-10T17:35:28+0200 **Info: connection closed by remote**

[WARNING] Connection.cpp:122 resolve\_variables(): could not resolve ADS variable 'TestPlan.types\_stringin'

2020/07/10 17:35:33.327 [ERROR] ADSPortDriver.cpp:199 connect(): [ads-test-plan] could not resolve ADS read variable names (1012): ADS device is not connected

2020-07-10T17:35:33+0200 **Error: write frame failed with error: 32**

2020/07/10 17:35:33.328 [ERROR] ADSPortDriver.cpp:735 ads\_comm\_task(): [ads-test-plan] error reading ADS device info (-1): <unknown error>

iocRun: All initialization complete

# End of IOC initialization

**Possible reason:**

The ADS device you are connecting to does not have a configured ADS route for the IOC client.

**Problem solution:**

Add an ADS route to the ADS device that the IOC is connecting to.

## IOC fails to connect with "Info: connection closed by remote" messages

During IOC boot, when trying to connect to an ADS device the IOC reports *Info: connection closed by remote* messages.

**Example:**

2020-07-10T17:40:52+0200 Info: Connected to 10.68.6.45

2020/07/10 17:40:52.771 [WARNING] ADSPortDriver.cpp:193 connect(): [ads-test-plan] connected to ADS device (IP: 10.68.6.45)

2020-07-10T17:40:52+0200 **Info: connection closed by remote**

[WARNING] Connection.cpp:122 resolve\_variables(): could not resolve ADS variable 'TestPlan.types\_stringin'

2020/07/10 17:40:57.777 [ERROR] ADSPortDriver.cpp:199 connect(): [ads-test-plan] could not resolve ADS read variable names (1012): ADS device is not connected

**Possible reason:**

The TwinCAT XAE ADS server is running in Configuration mode instead of Active mode.

**Problem solution:**

Make sure that the TwinCAT XAE is in Active (running) mode.

## IOC fails to connect with "Connect TCP socket failed with: 111" errors

During IOC boot, when trying to connet to an ADS device the IOC reports *Error: Connect TCP socket failed with: 111* errors.

**Example:**

2020-07-10T17:46:53+0200 **Error: Connect TCP socket failed with: 111**

[ERROR] Connection.cpp:42 connect(): could not add ADS rout (6): target port not found

2020/07/10 17:46:53.314 [ERROR] ADSPortDriver.cpp:190 connect(): [ads-test-plan] could not connect to ADS device (1012): ADS device is not connected

**Possible reason:**

The return code 111 of Connect TCP socket resolves into a ***connection refused*** error. The IP address that the IOC is connecting is reachable, but the ADS server is not listening on the IP port.

**Possible solution:**

Make sure that the IOC is connecting to a valid ADS device. If connecting to a TwinCAT XAE ADS device, make sure that it is in the Active (running) mode and that the firewall permits ADS traffic.

## IOC fails to connect with "Connect TCP socket failed with 110; Adding ADS route failed" error

During IOC boot, when trying to connect to an ADS device the IOC reports *Connect TCP socket failed with: 110* error.

**Example:**

2020-07-10T17:56:18+0200 **Error: Connect TCP socket failed with: 110**

[ERROR] Connection.cpp:42 connect(): could not add ADS rout (6): target port not found

2020/07/10 17:56:18.095 [ERROR] ADSPortDriver.cpp:190 connect(): [ads-test-plan] could not connect to ADS device (1012): ADS device is not connected

**Possible reason:**

The return code 110 of the *Connect TCP socket failed* error usually resolves into a connection timed out error.

**Problem solution:**

Make sure that the IOC is connecting to an existing ADS device. If connecting to a TwinCAT XAE ADS device, make sure that it is in the Active (running) mode and that the firewall permits ADS traffic.

## IOC loses ADS connection with "read frame failed with error: Connection reset by peer" and "connection closed by remote" messages

During IOC runtime, the ADS connection is suddenly lost, followed by Error: read frame failed with error: Connection reset by peer and Info: connection closed by remote messages.

**Example:**

2020-07-14T14:02:33+0200 **Error: read frame failed with error: Connection reset by peer**

2020-07-14T14:02:33+0200 **Info: connection closed by remote**

2020/07/14 14:02:38.768 [WARNING] ADSPortDriver.cpp:776 ads\_comm\_task(): [port-ads] sum-read failed due to no connection to ADS device

2020-07-14T14:02:38+0200 Error: write frame failed with error: 32

2020-07-14T14:02:38+0200 Error: write frame failed with error: 32

**Possible reason:**

Another ADS client from the same IP address and using the same AMS ID may have open a new connection to the ADS device.

**Possible solution:**

Make sure there is no other ADS client using the same AMS ID running on the same host.

## IOC fails to connect with "Connect TCP socket failed with 113; Adding ADS route failed" error

During IOC boot, when trying to connect to an ADS device the IOC first reports *Connect TCP socket failed with: 113*, followed by *Adding ADS route failed, did you specify valid address* error.

**Example:**

2018-09-07T11:17:52+0200 **Error: Connect TCP socket failed with: 113**

[ERROR] Connection.cpp:42 connect(): could not add ADS rout (6): target port not found

2020/07/10 17:56:18.095 [ERROR] ADSPortDriver.cpp:190 connect(): [ads-test-plan] could not connect to ADS device (1012): ADS device is not connected

**Possible reason:**

The return code 113 of the *Connect TCP socket failed* error usually resolves into a no route to host error. The IP address that the IOC is trying to connect to does not exist or is not reachable.

**Problem solution:**

Make sure that the ADS device IP address is correct and that it is reachable from the IOC.

## IOC connects but prints "Port 0x2710 is out of range" and "No response pending" warnings

During IOC runtime, you can see it connect to the PLC but periodically prints Port 0x2710 is out of range and No response pending warnings.

**Example:**

2020-07-10T18:02:03+0200 Warning: **Port 0x2710 is out of range**

2020-07-10T18:02:03+0200 Warning: **No response pending**

**Possible reason:**

One known reason is that on the ADS device, TwinCAT XAE is active.

**Possible solution:**

Either ignore the warnings or close TwinCAT XAE on the target ADS device. The PLC program should still continue to run normally.

1. Record support
   1. Supported EPICS record types

Table 2 lists EPICS records that are supported by the ADS device support and the asyn interfaces that are available for each record type.

Table 2: Supported EPICS records with available asyn interfaces

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ai/ao | bi/bo | longin/longout | stringin/stringout | mbbi/mbbo | mbbiDirect/mbboDirect | waveform |
| asynInt32 | **X** | **X** | **X** |  | **X** |  |  |
| asynFloat64 | **X** |  |  |  |  |  |  |
| asynUInt32Digital |  | **X** |  |  | **X** | **X** |  |
| asynOctet |  |  |  | **X** |  |  |  |
| asynInt8Array |  |  |  |  |  |  | **X** |
| asynInt16Array |  |  |  |  |  |  | **X** |
| asynInt32Array |  |  |  |  |  |  | **X** |
| asynFloat32Array |  |  |  |  |  |  | **X** |
| asynFloat64Array |  |  |  |  |  |  | **X** |

* 1. Supported TwinCAT PLC data types

Table 3 lists asyn interfaces and TwinCAT PLC data types [12] that the interfaces support.

Table 3: TwinCAT PLC data types supported by asyn interfaces

|  |  |
| --- | --- |
|  | Supported TwinCAT PLC data types |
| asynInt32 | BOOL, SINT, BYTE, USINT, INT, WORD, UINT, DINT |
| asynFloat64 | DWORD, UDINT, REAL, LREAL |
| asynUInt32Digital | BOOL, BYTE, USINT, WORD, UINT, DWORD, UDINT |
| asynOctet | STRING |
| asynInt8Array | BOOL, SINT, BYTE, USINT |
| asynInt16Array | INT, WORD, UINT |
| asynInt32Array | DINT, DWORD, UDINT |
| asynFloat32Array | REAL |
| asynFloat64Array | LREAL |

* 1. Database record configuration examples

This section contains examples of database record configuration.

*longin* record reads from a 16-bit integer ADS variable, addressed using port, group and offset:

record(longin, "$(P):reg\_int32\_counter") {

field(DTYP, "asynInt32")

field(INP, "@asyn($(PORT) 0 0) INT R P=350 G=0x8502000 O=0x8103E800")

}

*longout* record writes to a 32-bit integer ADS variable, addressed using port, group and offset.

record(longout, "$(P):reg\_int32\_writable") {

field(DTYP, "asynInt32")

field(OUT, "@asyn($(PORT) 0 0) DINT W P=350 G=0x8502000 O=0x8103E802")

}

*stringin* record reads from a string ADS variable of length 9. The variable is addressed using port and its symbolic name. Due to SCAN=I/O Intr, the ADS device support will update the record immediately when it detects that the addressed ADS variable has changed.

record(stringin, "$(P):types\_stringin") {

field(DTYP, "asynOctetRead")

field(SCAN, "I/O Intr")

field(INP, "@asyn($(PORT) 0 0) STRING[9] R P=350 V=TestPlan.types\_stringin")

}

*mbbiDirect* record reads from a 16-bit unsigned integer ADS variable, which is addressed by port and symbolic name.

record(mbbiDirect, "$(P):types\_mbbiDirect") {

field(DTYP, "asynUInt32Digital")

field(INP, "@asynMask($(PORT) 0 0xffff 0) UINT R P=PLC\_TC3 V=TestPlan.types\_mbbiDirect")

}

*waveform* record reads 10 elements of 32-bit unsigned integer ADS variables, addressed by port and symbolic name. Due to SCAN=I/O Intr, the ADS device support will update the record immediately when it detects that the addressed ADS variable has changed.

record(waveform, "$(P):types\_wf\_ulong\_in") {

field(DTYP, "asynInt32ArrayIn")

field(FTVL, "ULONG")

field(NELM, "10")

field(SCAN, "I/O Intr")

field(INP, "@asyn($(PORT) 0 0) UDINT R P=PLC\_TC3 V=TestPlan.types\_wf\_ulong\_in")

}

*waveform* record writes 10 elements of 32-bit floating point ADS variables, addressed by port and symbolic name.

record(waveform, "$(P):types\_wf\_float\_out") {

field(DTYP, "asynFloat32ArrayOut")

field(FTVL, "FLOAT")

field(PREC, "2")

field(NELM, "10")

field(INP, "@asyn($(PORT) 0 0) REAL W P=PLC\_TC3 V=TestPlan.types\_wf\_float\_out")

}

* 1. Supported ADS port names

Table 4 lists the ADS port names that can be specified by name in the record's INP/OUT fields (the P= parameter).

Table 4: ADS port names

|  |  |
| --- | --- |
| Port name | Value |
| LOGGER | 100 |
| RTIME | 200 |
| TRACE | 290 |
| IO | 300 |
| SPS | 400 |
| NC | 500 |
| ISG | 550 |
| PCS | 600 |
| PLC | 801 |
| PLC\_RTS1 | 801 |
| PLC\_RTS2 | 811 |
| PLC\_RTS3 | 821 |
| PLC\_RTS4 | 831 |
| PLC\_TC3 | 851 |