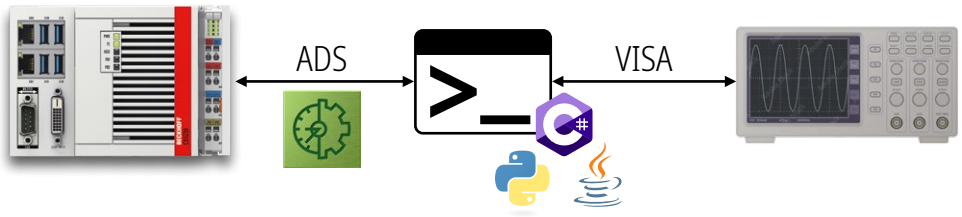
BECKHOFF

Application Note

Keywords
TwinCAT
.NET
ADS
VISA
IVI
SCPI
GPIB
IEEE 488

Controlling test and measurement equipment via ADS and TwinCAT



Integrating test and measurement (T&M) technology into existing systems is a challenge, as these devices are not usually compatible with the controllers' real-time bus systems and use different communication protocols. In addition, some hardware interfaces are used that are not standard practice in automation technology. The development of a software interface that enables communication between the various system components is therefore necessary to allow integration. Programming languages and tools that are widely used in the T&M environment, such as MATLAB® or LabVIEW™, can be integrated into TwinCAT via software packages. You can also use measurement devices without the other advantages of these tools. Using a PC as an automation system offers the advantage of an extensive range of hardware interfaces that are made available ex-factory or can be easily extended.

The information in this document describes how measurement device communication can be achieved from a PC-based controller based on TwinCAT. A minimal ADS server program is created to execute code from third-party libraries in user mode. This enables devices to be controlled from TwinCAT via the VISA protocol.

▶ www.beckhoff.com/measurement-technology

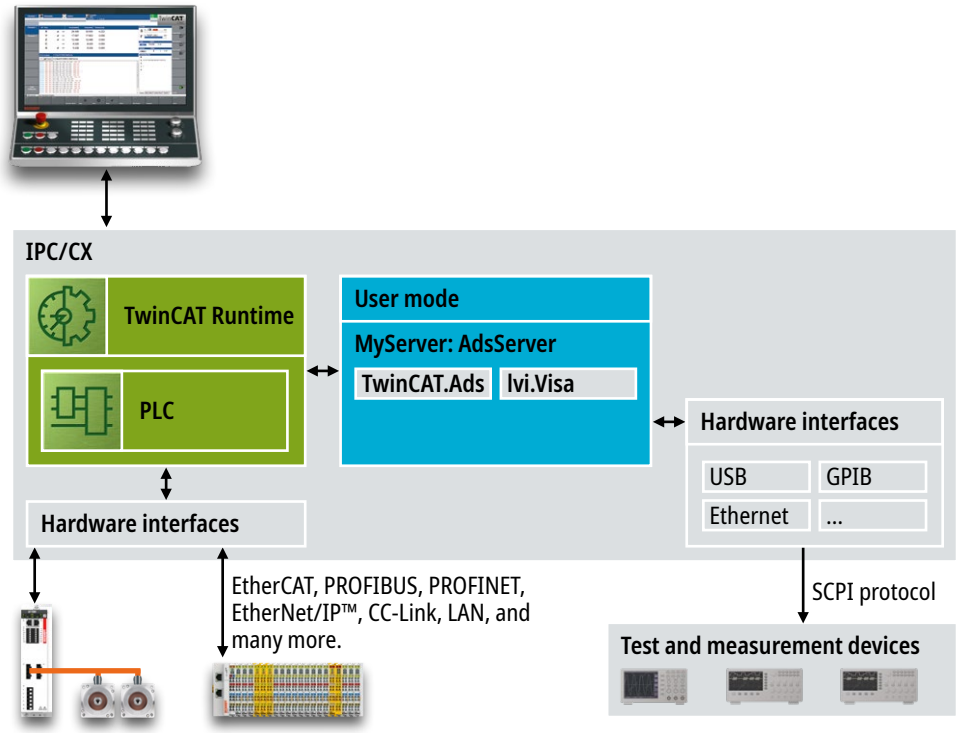


Figure 1: Schematic diagram of the system components

The following sections provide an overview of relevant specifications and industry standards that form the basis for communication between the controller and measurement devices.

ADS – Automation Device Specification

ADS is a communication protocol that was developed by Beckhoff in the 1990s for the interaction between different devices and software components within a TwinCAT automation system. The protocol can be used without additional licensing, implemented in various programming languages, and applied independently of the platform.
If it is necessary to communicate with another PC or device, the ADS protocol is used on top of TCP/IP. The devices are addressed by their net ID and services are differentiated by port numbers. The relevant data is differentiated on the basis of the index group and the offset.

Data traffic can be read with the TF6010 TwinCAT 3 ADS Monitor.

▶ infosys.beckhoff.com

SCPI – Standard Commands for Programmable Instruments

The SCPI protocol involves standardized commands (command set) in ASCII format, which are used to control measurement devices. For compatible devices with *IDN?, for example, the serial number can be queried and *RST resets the device. Manufacturers can also add their own commands based on this syntax. The protocol can be transmitted via various physical means, including USB and GPIB.

VISA – Virtual Instrument Software Architecture

VISA is an API (application programming interface, software interface) that enables communication between a computer (or the programs running on it) and measurement devices via various hardware interfaces, such as GPIB, USB, or Ethernet. A VISA application sends/receives the SCPI protocol, among other things.

This industry standard is implemented by various companies in the industry. As shared components of the IVI Foundation are used, they can run in parallel on the same system. The manufacturers differ mainly in terms of the additional software they offer. Examples of VISA implementations are:

Keysight	IO Libraries Suite
National Instruments™	NI-VISA
Rohde & Schwarz	R&S®VISA
Tektronix	TekVISA

IVI – Interchangeable Virtual Instruments

The IVI Foundation is an open consortium founded to promote specifications for programming test and measurement devices. The concept aims to facilitate the exchange of measurement devices in automated test and measurement systems through standardized driver architectures. They offer drivers that group classes of instruments by function (digital multimeter, oscilloscope, function generator, etc.) and specify the shared components (Ivi.Visa) for VISA implementations by manufacturers. The IVI maintains aspects such as the SCPI command set and VISA.

Links: The SCPI Standard | IVI Foundation

First steps with VISA

Before starting the implementation of your own service, it can be advantageous to familiarize yourself with the basic functions of VISA communication. The graphical utilities of the device manufacturers provide a quick introduction. National Instruments™ offers VISA Interactive Control. The NI™ software package is very comprehensive and contains components needed to use LabVIEW™. This contrasts with the VISA Tools from Rohde & Schwarz, which have a comparably small footprint. The Connection Expert from Keysight, which is included in the IO Libraries Suite along with other useful tools, is used below. The software selection shown is not exhaustive and serves only as a representative overview. The basic range of functions of the manufacturers presented is largely comparable.

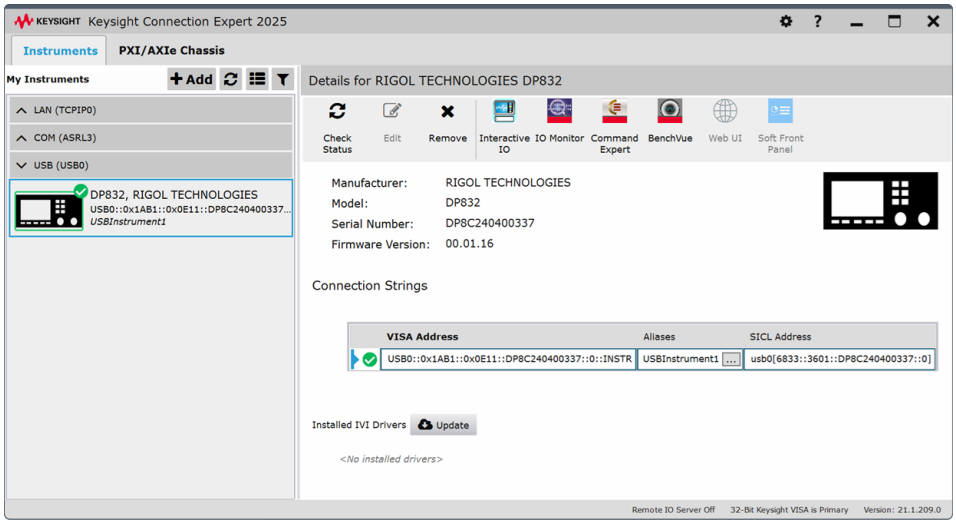


Figure 1:
List of connected devices in Connection Expert

Once a power supply has been connected via the USB interface, it is detected by the automatic scanning process. Serial devices may first have to be configured manually via [+ Add]. The VISA address of the device is displayed and this can then be used to establish communication with the device. The structure of the address differs according to the interface type and contains mandatory and optional elements. The USB address shown here can be broken down into its components as follows:

USB0 :: 0x1AB1 :: 0x0E11 :: DP8C240400337 :: 0 :: INSTR

- USB card
- Manufacturer ID
- Product ID
- Device type
- Interface number
- Serial number of the device

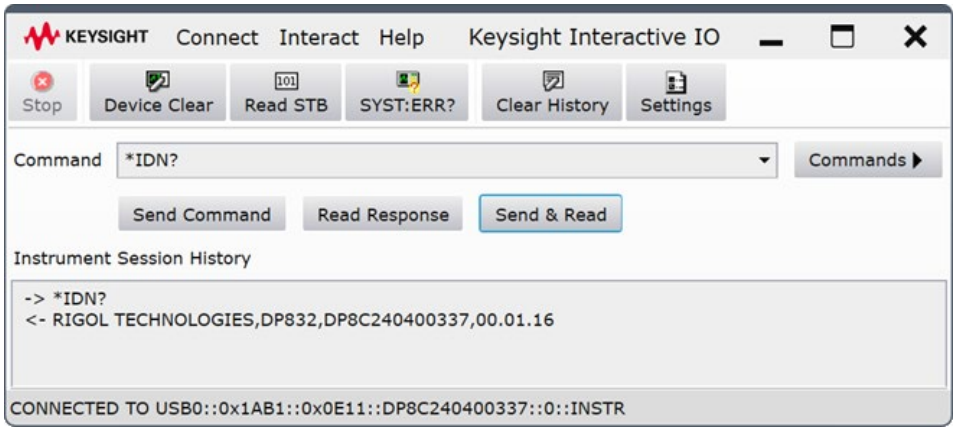


Figure 2:
Querying the serial number via Interactive IO

Examples of various interface descriptions can be found in the documentation of the VISA providers.[1] Two other tools that have been installed are Interactive IO and IO Monitor. The first can be used to send SCPI commands and extended commands implemented by the device manufacturer and to query the response from the device. The standardized ID query returns a response string with device data.

The second tool can be used to record the communication history with a device. It serves as an aid during development, but can also be used for troubleshooting in existing systems. If Interactive IO is opened after recording has started, a series of calls to the VISA-C library are listed.

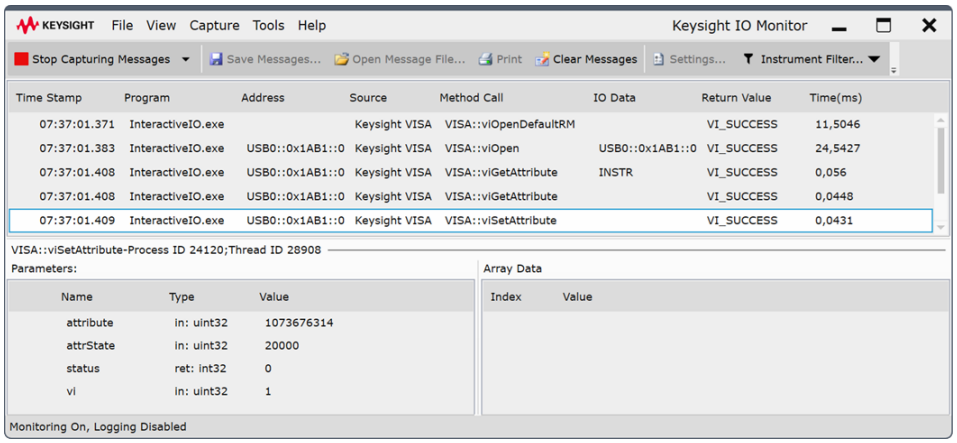


Figure 3:
Recording VISA communication using IO Monitor

The application created in this example will be based on VISA.NET, which is a wrapper for VISA-C. If a function is executed in C#, IO Monitor displays the equivalent call from the C library. The calls reveal that a communication channel (called a session) is opened with the device, attributes are read, and an attribute is also set. In .NET, these attributes are mapped by properties of the connection object. A more detailed description of the set attribute call is shown in the parameter field.

[1] https://www.ni.com/docs/en-US/bundle/ni-visa/page/visa-resource-syntax-and-examples.html

Handle 1 was assigned to the device (vi 1) and it executed the command without error (status 0). A 10-digit number refers to the attribute to which the value 20,000 was assigned. The code refers to VI_ATTR_TMO_VALUE, the timeout for this connection. This can be set under Settings in Interactive IO. The codes are defined for all VISA implementations and documented on the manufacturer's pages.[2] The operations performed so far can be mapped in C# as follows:

```
using Ivi.Visa;
using Ivi.Visa.FormattedIO;

// Suche nach allen angeschlossenen Geräten
IEnumerable<string> devices = GlobalResourceManager.Find();
foreach (var dev in devices)
    Console.WriteLine(dev);

// Erstelle eine textbasierte Verbindung zum Gerät
string VISA_ADDRESS = "USB0::0x1AB1::0x0E11::DP8C240400337::0::INSTR";
var session = GlobalResourceManager.Open(VISA_ADDRESS) as IMessageBasedSession;
var formattedIO = new MessageBasedFormattedIO(session);

// Lege die Timeout-Eigenschaft fest
session.TimeoutMilliseconds = 20_000;

// Sende das SCPI-Kommando um die ID abzufragen
formattedIO.WriteLine("*IDN?");
string response = formattedIO.ReadLine();
Console.WriteLine(response);

// Freigeben der Ressource
session.Dispose();
```

Code example 1: Minimal ADS server by overwriting the read/write indication

In order to execute the code, the packages IviFoundation.Visa (shared components) and, in this case, KeysightTechnologies.Visa (manufacturer implementation) are required. Both are available via NuGet and can be added to the project manually or via the command line.

dotnet add package IviFoundation.Visa
dotnet add package KeysightTechnologies.Visa

> In order to develop an application for .NET 6+ according to the procedure described, the manufacturer implementation used must support VISA.NET v7.4 or a later version. Detailed information is available on the respective NuGet page.

[2] https://helpfiles.keysight.com/IO_Libraries_Suite/English/IOLS_2024_U1_Windows/VISA/Content/visa/VISA%20Attributes%20Table.html

Implementation of the ADS server

In order to be able to access third-party libraries such as VISA from the PLC, a virtual function server should be implemented that communicates via ADS on the control side. The program is executed in user mode and is therefore not real-time capable (in contrast to TwinCAT runtime). The .NET API and descriptions for programming languages outside this framework can be found in Infosys[3].

```
using TwinCAT.Ads;
using TwinCAT.Ads.Server;

// Startet den Server auf Port 25000 dem Namen "MyServer"
var s = new MyServer(25_000, "MyServer");
s.ConnectServer();
Console.ReadLine();
s.Dispose();

public class MyServer(ushort port, string name) : AdsServer(port, name)
{
    protected override Task<ResultReadWriteBytes> OnReadWriteAsync(
        AmsAddress sender, uint invokeId, uint indexGroup,
        uint indexOffset, int readLength,
        ReadOnlyMemory<byte> writeData,
        CancellationToken cancel)
    {
        // IDXGRP 1 meldet einen Erfolg und gibt 1 zurück
        // Andere Index-Gruppen führen zum ADS Fehler 1793
        var notSupported = AdsErrorCode.DeviceServiceNotSupported;

        ResultReadWriteBytes result = indexGroup switch
        {
            1 => ResultReadWriteBytes.CreateSuccess(new Memory<byte>([1])),
            _ => ResultReadWriteBytes.CreateError(notSupported)
        };
        return Task.FromResult(result);
    }
}
```

Code example 2:
Minimal ADS server by overwriting the read/write indication

All software components required for the ADS aspect of the application are included in the Beckhoff.TwinCAT.Ads NuGet package[4].

dotnet add package Beckhoff.TwinCAT.Ads

Beckhoff reserves port numbers from 25000–25999 for specially created server applications in order to avoid overlapping with existing functions[5]. In code example 2, an ADS server with the name “MyServer” is started on port 25000. The read/write method has been overwritten so that it returns a one to the client for a request for index group 1. All other requests result in a NotSupported error (ADS return code 1793). The AdsServer class contains many other virtual methods that can be overwritten in this way. The resources are released again when the program is closed. This request can be made from the PLC program using the ADSRDWRT function block from the Tc2_System library.

[3] https://infosys.beckhoff.com/index.php?content=../content/1031/tc3_ads.net/9407515403.html&id=
[4] https://www.nuget.org/packages/Beckhoff.TwinCAT.Ads/
[5] https://infosys.beckhoff.com/index.php?content=../content/1031/tc3_grundlagen/116159883.html&id=

```
PROGRAM MAIN
VAR
      AdsRdWrt     : ADSRDWRT;
      IdxGroup     : UDINT;
      Payload      : USINT;
      Response     : USINT;
      Execute      : BOOL;
      ErrorId      : UDINT;
END_VAR

AdsRdWrt (
      NETID        := '',
      PORT         := 25_000,
      IDXGRP       := IdxGroup,
      IDXOFFS      := 0,
      WRITELEN     := SIZEOF(Payload),
      READLEN      := SIZEOF(Response),
      SRCADDR      := ADR(Payload),
      DESTADDR     := ADR(Response),
      WRTRD        := Execute,
      ERRID        => ErrorId);
IF NOT AdsRdWrt.BUSY THEN
      Execute := FALSE;
END_IF
```

Code example 3:
Defining and calling the parameterized AdsReadWrite FB

The ADS server can now be equipped with the required functions.
The specific implementation details depend on the respective application.

Further links:

ADS examples from Beckhoff

VISA examples from Keysight

BECKHOFF

Would you like to find out more?

We are happy to help and look forward to hearing from you:

support@beckhoff.com

▶ www.beckhoff.com/measurement-technology

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 963-0
info@beckhoff.com
www.beckhoff.com