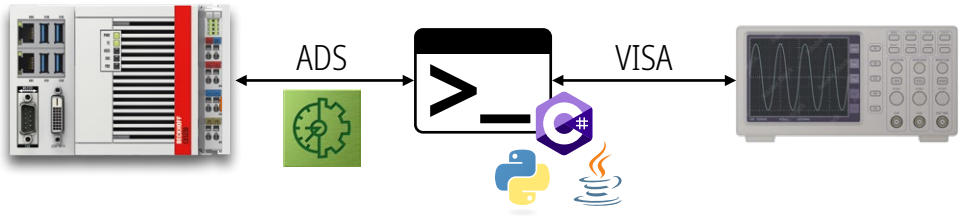
BECKHOFF

Application Note

Keywords
TwinCAT
.NET
ADS
VISA
IVI
SCPI
GPIB
IEEE 488

Controlling test and measurement equipment via ADS and TwinCAT



In cases where the functional scope of a system needs to be extended by including testing and measurement technology (T&M), or a complex laboratory setup needs to be automated (using automated test equipment/ATE), an interface between the two environments is required. 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 (as a "service") 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

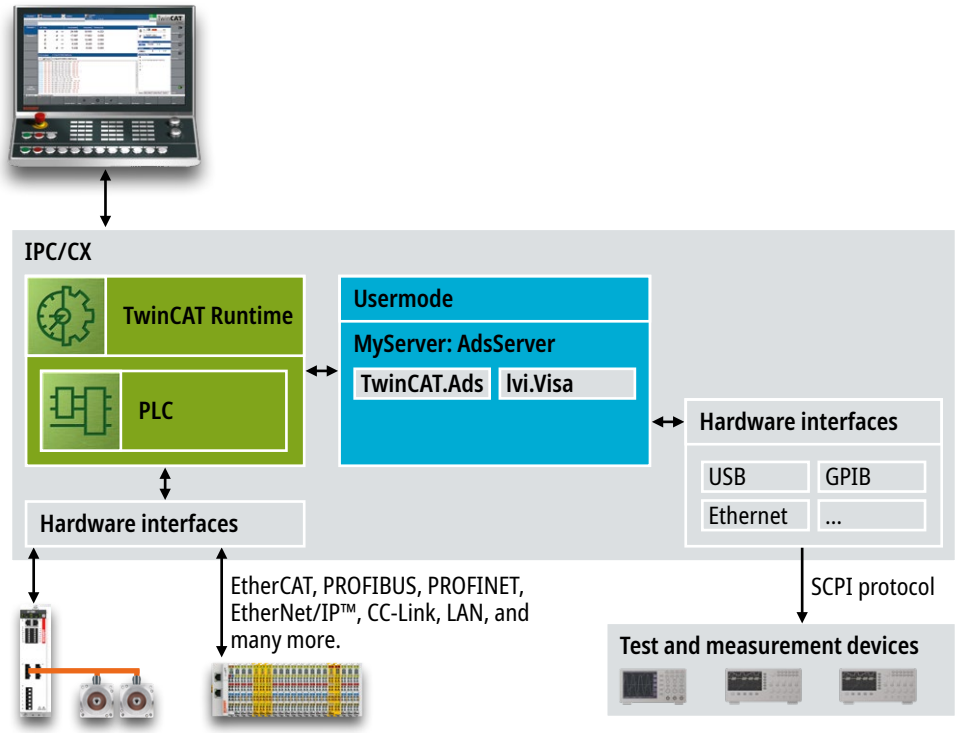


Figure 1: Schematic diagram of the system components

First of all, here are a few term definitions. These explain the difference between electrical transmission (physical communication layer such as RS232, LAN, or GPIB) and a protocol:

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 can be implemented 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 ADS Monitor.

▶ infosys.beckhoff.com

GPIB, IEEE 488

This is an electrical parallel interface for connecting devices. It is not defined which protocol (commands) are to be given over it. Special hardware (PC plug-in cards, USB converters) is used for the GPIB connection.

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 basic range of functions is similar, so it does not matter which software package is used for the application described here.

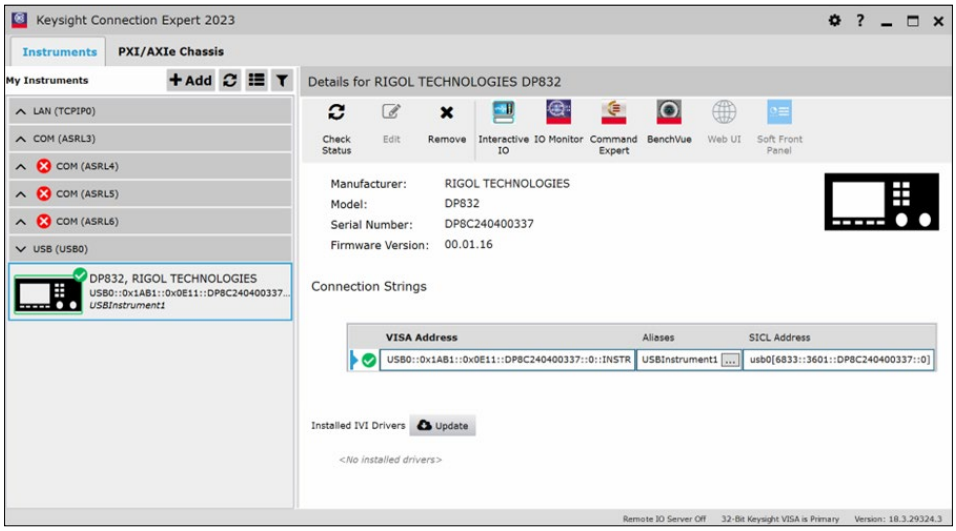


Figure 1:
List of connected devices in Connection Expert

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

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.

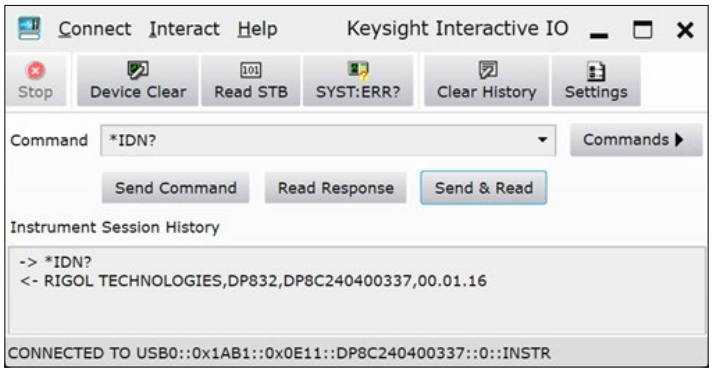


Figure 2:
Querying the serial number via Interactive IO

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.

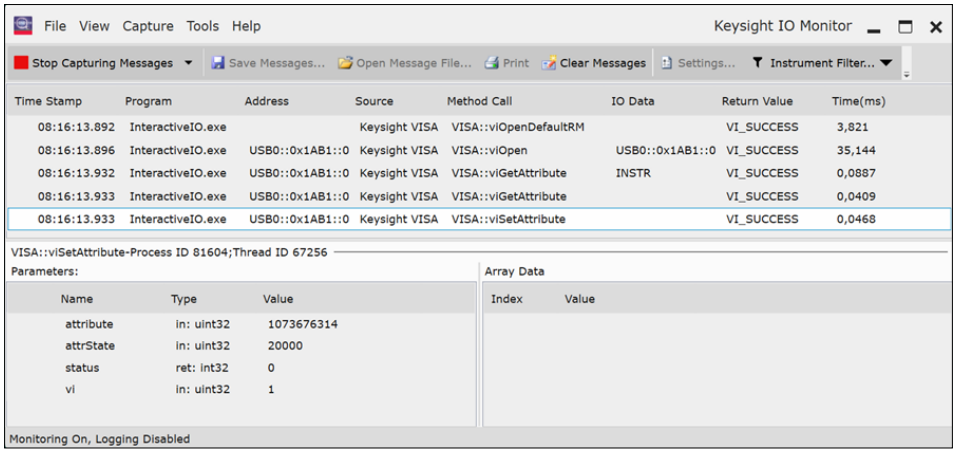


Figure 3:
Recording VISA communication using IO Monitor

The software presented is written in C# for the .NET Framework. The C-API is therefore not used directly, but VISA.NET is, and serves as a wrapper for VISA-C. 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. 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

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

all VISA implementations and documented on the manufacturer's pages.[2] The operations performed so far can be mapped in C# as follows:

```
// Search for all connected devices
IEnumerable<string> devices = GlobalResourceManager.Find();

// Create a text-based connection to the device
string VISA_ADDRESS =
"USB0::0x1AB1::0x0E11::DP8C240400337::0::INSTR";
var session = GlobalResourceManager.Open(VISA_ADDRESS) as IMes-
sageBasedSession;
var formattedIO = new MessageBasedFormattedIO(session);

// Set timeout property
session.TimeoutMilliseconds = 20_000;

// SeSend the SCPI command to query the ID
formattedIO.WriteLine("*IDN?");
string idnResponse = formattedIO.ReadLine();

// Release the resource
session.Dispose();
```

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

In addition to the documentation, the device manufacturers often offer application examples as downloads. These cover many typical standard applications and make it easier to get started with the VISA .NET API. At present, there are no VISA implementations for modern .NET. If the VISA libraries are to be used out of the box, the .NET framework must therefore be used.

In IVI specification VPP-4.3.6, it was announced that the VISA.NET shared components will be available for .NET 6+ from version 7.3.[3] To work in .NET 6+ now, the installed VISA versions can be found via the class Ivi.Visa.ConflictManager. These can be loaded from %windir%\assembly (or %windir%\Microsoft.NET\assembly for .NET Framework 4+) using the assembly class. The reason for this is that modern .NET applications no longer use the global assembly cache (GAC). Alternatively, you can also write your own wrapper around VISA-C.

[2] https://helpfiles.keysight.com/IO_Libraries_Suite/English/IOLS_2024_U1_Windows/VISA/Content/visa/VISA%20Attributes%20Table.html
[3] https://www.ivifoundation.org/specifications/default.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[4].

```
public class MyServer : AdsServer
{
    public MyServer() : base(25_000, "MyServer") { }

    protected override Task<ResultReadWriteBytes> OnRead-
WriteAsync(
        AmsAddress sender, uint invokeId,
        uint indexGroup, uint indexOffset,
        int readLength, <byte> writeData,
        CancellationToken cancel)
    {
        ResultReadWriteBytes result;
        switch (indexGroup)
        {
            case 1:
                result = ResultReadWriteBytes.CreateSuc-
cess(Memory<byte>.Empty);
                break;

            default:
                result = ResultReadWriteBytes.CreateError(
                AdsErrorCode.DeviceServiceNotSupported);
                break;
        }
        return Task.FromResult(result);
    }
}
```

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

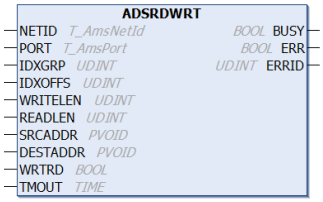
All software components required for the ADS aspect of the application are included in the Beckhoff.TwinCAT.Ads NuGet package[5]. Beckhoff reserves port numbers from 25000–25999 for specially created server applications in order to avoid overlapping with existing functions[6]. Code example 2 defines an ADS server that can be started locally on port 25000. The read/write method was overwritten so that it would return a success for a request for index group 1 and a NotSupported error for other requests (ADS return code 1793). This request can be made from the PLC program using the ADSRDWRT block from the Tc2_System library.

[4] https://infosys.beckhoff.com/index.php?content=../content/1031/tc3_ads.net/9407515403.html&id=
[5] https://www.nuget.org/packages/Beckhoff.TwinCAT.Ads/
[6] https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_grundlagen/116159883.html&id=

```
var
    AdsReadWrite : ADSRDWRT;
    Payload : usint;
    Response : usint;
end_var

AdsReadWrite(
    NETID:= '',
    PORT:= 25_000,
    IDXGRP:= 1,
    IDXOFFS:= 0,
    WRITELEN:= sizeof(Payload),
    READLEN:= sizeof(Response),
    SRCADDR:= adr(Payload),
    DESTADDR:= adr(Response),
    WRTRD:= true);
```

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