SINUMERIK SDK
SINUMERIK connected Development

SINUMERIK SDK für .NET

Getestet? Du willst es?

Lizenzmodell Angebot

Book - Das gesamte Handbuch als eBook

Development Guides

Development Guide Häufige Fragen

Download

Das SINUMERIK .NET SDK kommt mit einer **Testlizenz die je Anwendungsstart 30 Minuten uneingeschränkt zur Entwicklung** verwendet werden kann. Sollte diese Einschränkung ihre Evaluationsmöglichkeiten einschränken, besteht die Möglichkeit eine **alternative Testlizenz** bei uns **kostenlos** zu beantragen. Fragen Sie einfach unseren Support (via support@traeger.de) oder lassen Sie sich gleich direkt von uns beraten und offene Fragen durch unsere Entwickler klären!

SINUMERIK .NET SDK – Evaluationspaket[1)]
Download ZIP Archiv von SinumerikNet.Advanced (Version: 1.1.0.1 – 2022-04-08)
Download NuGet Paket von SinumerikNet.Advanced (Version: 1.1.0.1 – 2022-04-08)

Versionshistorie - Die Liste der Verbesserungen pro Version

SINUMERIK LINK

Development Guide

Beispiel Code: Überwachung einer Achse

- C#
- VB

```
namespace Position
{
    using System;
    using System.Threading;

    using Sinumerik.Advanced;

    public class Program
    {
        /// <summary>
        /// This sample demonstrates how to implement an app which monitors the position.
        /// </summary>
        public static void Main()
        {
            // The following setup connects to a Sinumerik SolutionLine (Sl)
            // Just replace "sl" with "pl" to connect to a Sinumerik PowerLine instead.
            using (var client = new SinumerikClient("s840d.sl://192.168.0.80")) {
                client.Connect();

                while (true) {
                    var position = client.ReadValue("/Channel/MachineAxis/measPos1[u1, 1]");
                    Console.WriteLine($"Current Position of Axis 1 is {position} mm");

                    Thread.Sleep(1000);
                }
            }
        }
    }
}
```

```
Imports System
Imports System.Threading

Imports Sinumerik.Advanced

Namespace App
    Public Class Program
        Public Shared Sub Main()
            Dim device = New SinumerikDevice("192.168.0.80")

            Using connection = device.CreateConnection()
                connection.Open()

                While True
                    Dim position = connection.ReadDouble("/Channel/MachineAxis/measPos1[u1,
1]")
                    Console.WriteLine("Current Position of Axis 1 is {0} mm", position)

                    Thread.Sleep(1000)
                End While
            End Using
        End Sub
    End Class
End Namespace
```

[1] Mit Ihrem „License Code“ wird das Paket zur produktiven Vollversion.

SINUMERIK SDK
SINUMERIK connected Development

Development Einführung

Getestet? Du willst es?

Lizenzmodell Angebot

SPS-Adressierung

Operanden

Das Framework unterstützt die Adressierung von Eingängen (engl. Input), Peripherieeingängen (engl. Periphery Input), Ausgängen (engl. Output), Peripherieausgängen (engl. Periphery Output), Merkern (engl. Flag), Datenbausteinen (engl. Data Block), Zählern (engl. Counter) und Zeitgebern (engl. Timer). Auf welchen der eben genannten Komponenten zugegriffen werden soll wird über den Operandenanteil (engl. Operand) der SPS Datenadresse (kurz SPS Adresse) beschrieben. Die folgende Tabelle beschreibt die dafür gültigen Kürzel die als Operand bezeichnet werden:

Name	Abkürzung (Siemens, DE)	Abkürzung (IEC)
Eingang	E	I
Peripherieeingang	PE	PI
Ausgang	A	Q
Peripherieausgang	PA	PQ
Merker	M	M
Datenbaustein	DB	DB
Zähler	Z	C
Zeitgeber	T	T

Auf den Operanden folgt, im Falle eines Datenbausteins, die Nummer des Datenbausteins. Darauf folgt ein Punkt und wiederholt das Kürzel für den Operanden für Datenbausteine.

Die Backus-Naur-Form eines Operanden ist dabei wie folgt festgelegt:

- `<Datenbaustein-Nummer> ::= 0-65535`
- `<Siemens-Operand> ::= E | PE | A | PA | M | DB<Datenbaustein-Nummer>.DB | Z | T`
- `<IEC-Operand> ::= I | PI | Q | PQ | M | DB<Datenbaustein-Nummer>.DB | C | T`

Offsets

Der eigentliche Adressierungsteil der SPS Adresse (engl. PLC Address) kommt nach dem Operanden und unterteilt sich in den Datentypbezeichner (im Rohdatenformat) und dem Offset für das zu adressierende Byte sowie gegebenfalls das zu adressierende Bit (getrennt durch einen Punkt). Die dabei unterstützten Datentypbezeichner und ihre gültigen Kürzel (für den Rohdatentypen) beschreibt die folgende Tabelle:

Datentyp	Kürzel	Bits	Wertebereich	Beschreibung	Array
BOOL	X	1	0 bis 1	ein einzelnes Bit zur Darstellung von wahr (1) oder falsch (0)	ja
BYTE	B	8	0 bis 255	Vorzeichenlose 8-Bit Ganzzahl	ja
WORD	W	16	0 bis 65.535	Vorzeichenlose 16-Bit Ganzzahl (Word)	ja
DWORD	D	32	0 bis 2^{32} -1	Vorzeichenlose 32-Bit Ganzzahl (Double Word)	ja
CHAR	B	8	A+00 bis A+ff	Vorzeichenloses 8-Bit Zeichen im ASCII-Code	ja
INT	W	16	-32.768 bis 32.767	Vorzeichenbehaftete 16-Bit Ganzzahl	ja
DINT	D	32	-2^{31} bis 2^{31}-1	Vorzeichenbehaftete 32-Bit Ganzzahl	ja
REAL	D	32	+-1.5e-45 bis +-3.4e38	32-Bit Gleitkommazahl mit einfacher Genauigkeit (gemäß IEEE754)	ja

Datentyp	Kürzel	Bits	Wertebereich	Beschreibung	Array
S5TIME	W	16	00.00:00:00.100 bis 00.02:46:30.000	binär codierte Dezimalzahl (BCD), die eine Zeitspanne repräsentiert	
TIME	D	32	00.00:00:00.000 bis 24.20:31:23.647	Vorzeichenbehaftete 16-Bit Ganzzahl, die eine Zeitspanne in Millisekunden repräsentiert	
TIME_OF_DAY	D	32	00.00:00:00.000 bis 00.23:59:59.999	Vorzeichenlose 16-Bit Ganzzahl, die eine Zeitspanne in Millisekunden repräsentiert	
DATE	W	16	01.01.1990 bis 31.12.2168	Vorzeichenlose 16-Bit Ganzzahl, die ein Datum in Tagen repräsentiert	
DATE_AND_TIME	D	64	00:00:00.000 01.01.1990 bis 23:59:59.999 31.12.2089	binär codierte Dezimalzahl (BCD), die ein Datum mit Uhrzeit repräsentiert	
S7STRING	B	-	A+00 bis A+ff	eine im ASCII-Code codierte Zeichenkette aus maximal 254 Bytes	

Rohdatentypen

Dabei stehen die Kürzel für:

- X = Bit, für die Adressierung eines einzelnen Bits *(bei der Adressierung eines Bits ist das X optional)*
- B = Byte, für die Adressierung von 8 Bits
- W = Word (= *Wort*), für die Adressierung von 16 Bits
- D = Double Word (= *Doppelwort*, kurz DWord), für die Adressierung von 32 Bits

Werden Operandenteil und die eben genannten Datentypbezeichner zusammengesetzt, ergibt sich die folgende Backus-Naur-Form:
`<Operand mit Datentypbezeichner> ::= (<Siemens-Operand> | <IEC-Operand>) [ X ] | B | W | D`

Beispiele

Fügt man der PLC Address die Offset-Informationen hinzu, ist die Adresse vollständig und kann wie folgt aussehen:

Beispiel	Datentyp	PLC Address (Siemens, DE)	PLC Address (IEC)
Erstes Bit im ersten Byte des Eingangs	BOOL	E 1.0	I 1.0
Siebtes Bit im ersten Byte des Ausgangs	BOOL	A 1.7	Q 1.7
Erstes Bit im zehnten Byte des Merkers	BOOL	M 10.1	M 10.1
Nulltes Bit im ersten Byte im Datenbaustein mit der Nummer 1	BOOL	DB1.DBX 1.0	DB1.DBX 1.0
Erstes Bytes im Eingang	BYTE	EB 1	IB 1
Zehntes Byte im Ausgang	BYTE	AB 10	QB 10
Hundertes Byte im Merker	BYTE	MB 100	MB 100
Nulltes Byte im Peripherieeingang	BYTE	PEB 0	PIB 0
Erstes Byte im Peripherieausgang	BYTE	PAB 1	PQB 1
Erstes Byte im Datenbaustein mit der Nummer 2	BYTE	DB2.DBB 1	DB2.DBB 1
Nulltes Double Word im Peripherieeingang	DWORD	PED 0	PID 0

Mit der Backus-Naur-Form für den Byte und Bit Offset ergibt sich die vollständige PLC Address wie folgt:

- `<Byte Offset> ::= 0-65535`

- `<Bit Offset> ::= 0-7`
- `<PLC Address> ::= <Operand mit Datentypbezeichner> <Byte Offset> [. <Bit Offset> ]`

Die Verbindung zur SPS

Open

Das passiert beim Aufruf von 'Open':

1. Es wird geprüft, ob ein **Endpunkt festgelegt** wurde (EndPoint Eigenschaft).
2. Die Verbindung ändert ihren Status (**PlcDeviceConnection.State** Eigenschaft) auf den Wert ***Opening***.
3. Die Verbindung **prüft ihre Konfiguration** auf Gültigkeit und Schlüssigkeit.
4. Anschließend bereitet sich die Verbindung auf die erste Kommunikation mit der SPS vor.
5. Die Verbindung ändert ihren Status (**PlcDeviceConnection.State** Eigenschaft) auf den Wert ***Opened***.

Connect

Das passiert beim Aufruf von 'Connect':

1. Es wird geprüft, ob die Verbindung bereits geöffent wurde (State Eigenschaft).
2. Die Verbindung ändert ihren Status (**PlcDeviceConnection.State** Eigenschaft) auf den Wert ***Connecting***.
3. Die Verbindung stellt eine für die SPS physikalische Verbindung her und belegt ab sofort eine der maximal möglichen Verbindungen zur SPS.
4. Anschließend werden noch Vorkehrungen für die **Überwachung der Verbindung** getroffen:
    1. „KeepAlive-Tracking“ zur Erkennung von Verbindungsabbrüchen
    2. ~~„Notification-Tracking“ zum Empfangen von Benachrichtigungen~~
5. Die Verbindung ändert ihren Status (**PlcDeviceConnection.State** Eigenschaft) auf den Wert ***Connected***.

Close

Das passiert beim Aufruf von 'Close':

1. Die Verbindung ändert ihren Status (**PlcDeviceConnection.State** Eigenschaft) auf den Wert ***Closing***.
2. Die Verbindung **gibt** alle **erworbenen Ressourcen** wieder **frei**
3. **Beendet** die **Überwachung der physikalischen Verbindung**
4. Eine gegebenfalls aufgebaute physikalische **Verbindung zur SPS wird beendet** und steht somit wieder anderen Teilnehmern zur Verfügung.
5. Der beim Connect erstellte **Socket wird geschlossen und verworfen**.
6. Die Verbindung ändert ihren Status (**PlcDeviceConnection.State** Eigenschaft) auf den Wert ***Closed***.

BreakDetection

Die „BreakDetection“-**Abbrucherkennung** bezeichnet den Mechanismus, der für die Erkennung von Verbindungsabbrüchen zuständig ist. Hierbei kommt das KeepAlive Verfahren zum Einsatz, um so einen **Timeout der Verbindung zur SPS** zu **erkennen**. Kommt es zum Timeout, so versucht das Framework

automatisch wieder eine Verbindung zur SPS herzustellen. Während beim KeepAlive in regelmäßigen Abständen KeepAlive-Nachrichten zur SPS gesendet werden, um so die Verbindung „zu testen“ und „aufrecht zu erhalten“, wird bei zu langen Antwortzeiten (= Timeout erreicht?) auf eine KeepAlive-Nachricht angenommen, dass die Verbindung unterbrochen ist. Ist das der Fall, wird in immer größeren Abständen eine weitere KeepAlive-Nachricht gesendet. Bleiben auch diese unbeantwortet, wird von einer abgebrochenen Verbindung ausgegangen und der zuvor beschriebene Mechanismus zur Wiederaufnahme der Verbindung tritt in Kraft. Aktiviert wird die Abbrucherkennung, welche standardmäßig nicht aktiviert ist, über die **PlcDeviceConnection.UseBreakDetection** Eigenschaft.

Verbindungsparameter

Damit eine Verbindung zur SPS aufgebaut werden kann, müssen die richtigen Parameter festgelegt werden. **Generell** wird die **IP Adresse der SPS** (**IPDeviceEndPoint.Adress** Eigenschaft) **benötigt**, unter der sie zu erreichen ist. Die dabei vom **SiemensDevice** erwartete Endpunkt-Instanz (engl. Endpoint) liefert der Verbindung alle primär nötigen Informationen über die SPS. Anstelle der statischen IP Adresse kann auch der DNS Name der SPS verwendet werden, so könnte anstelle von „192.168.0.80“ auch „plc_man_drill_001“ verwendet werden. Zusätzlich zur IP Adresse der SPS benötigt der Endpunkt die **Nummer des Racks** (die Position der Montageschiene) und die **Nummer des Slots** der CPU. Werden keine Rack beziehungsweise Slot Nummer beim Endpunkt konfiguriert, versucht das Framework **automatisch die passenden Nummern** festzulegen. Welche Nummer im Einzelfall manuell angegeben werden muss hängt immer vom Aufbau (engl. setup) der Anlage / des Schaltschranks ab. Die Rack Nummer (beginnend bei 0) legt die Position der Montageschiene, auf der die SPS angebracht wurde, im Aufbau fest. Die Slot Nummer (beginned bei 1) legt die Position der CPU im Setup der Module der SPS fest. Eine auf die erste Hutschiene montierte S7-300 hat somit die Rack Nummer 0. Entspricht die Anordnung der Module (auf der Montageschiene) dem Schema:

1. „Power Supply (SP)“ Modul
2. „CPU“ Modul
3. „Interface“ Modul (IM)
4. „Signal“ Modul 1 (SM)
5. „Signal“ Modul 2 (SM)
6. „Signal“ Modul 3 (SM)
7. ...

Dann gilt für diese SPS die Slot Nummer 2, weil an dieser Position die CPU positioniert wurde.

SINUMERIK SDK
SINUMERIK connected Development

Development Guide & Tutorial

Getestet? Du willst es?

Lizenzmodell Angebot

C# ▼ |VB|

Der App Frame

TODO

Inhaltsverzeichnis