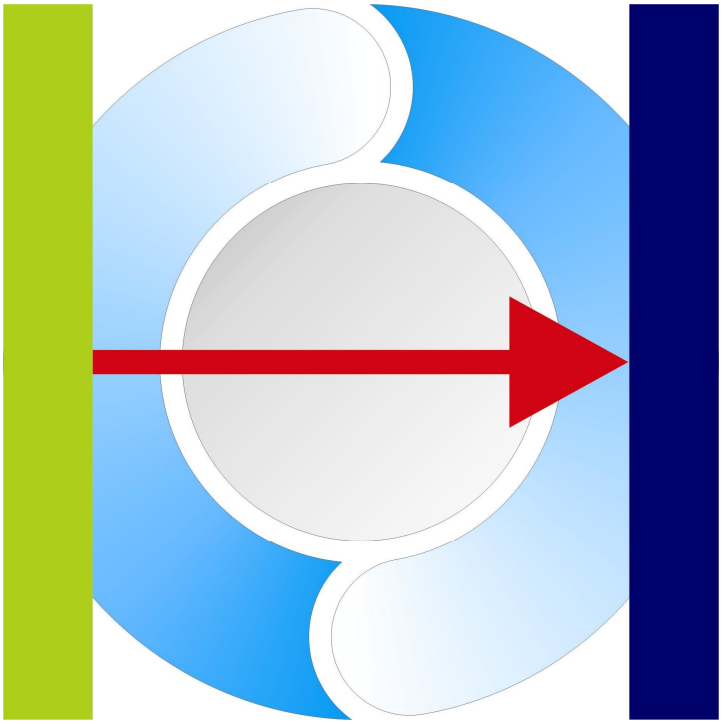


X-GO Flex Logic Control Documentation (Cluster 64 Bit)

Date: Sept,10.2020

X-GO Flex Logic Control Documentation

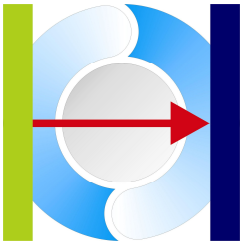
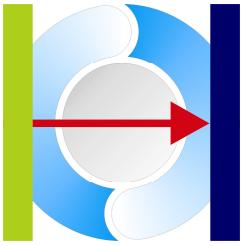


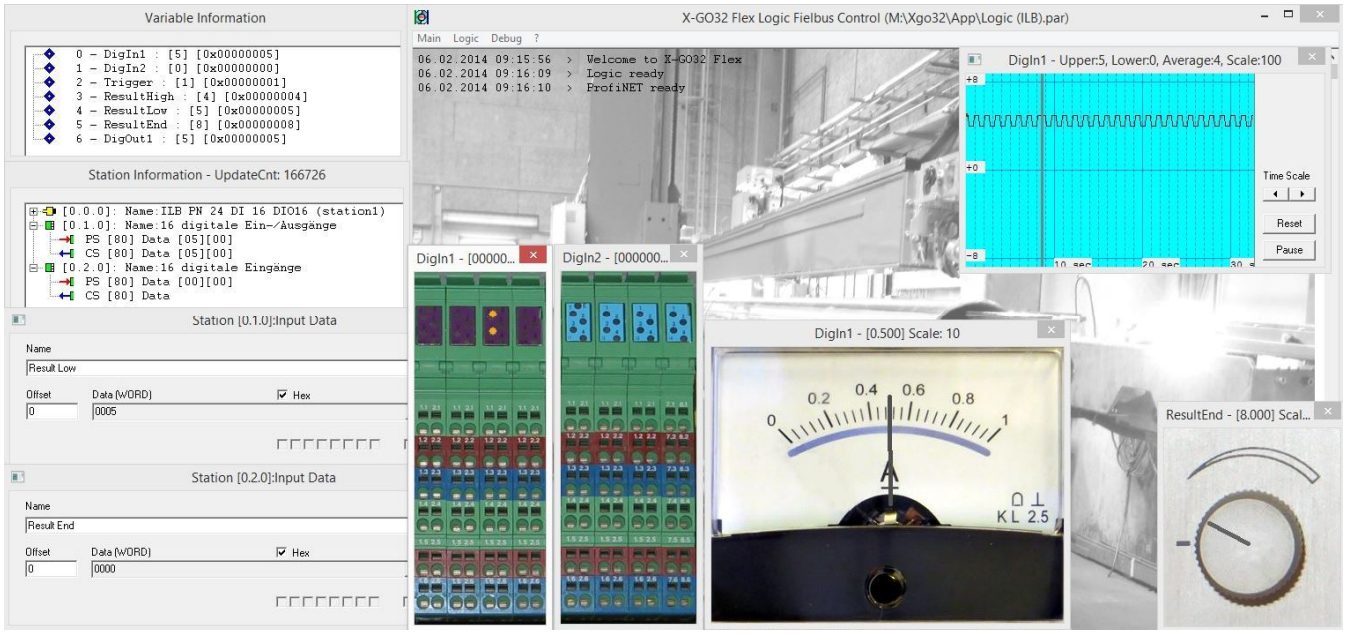
Table of content



X-GO Flex Logic Control Documentation

1 Introduction

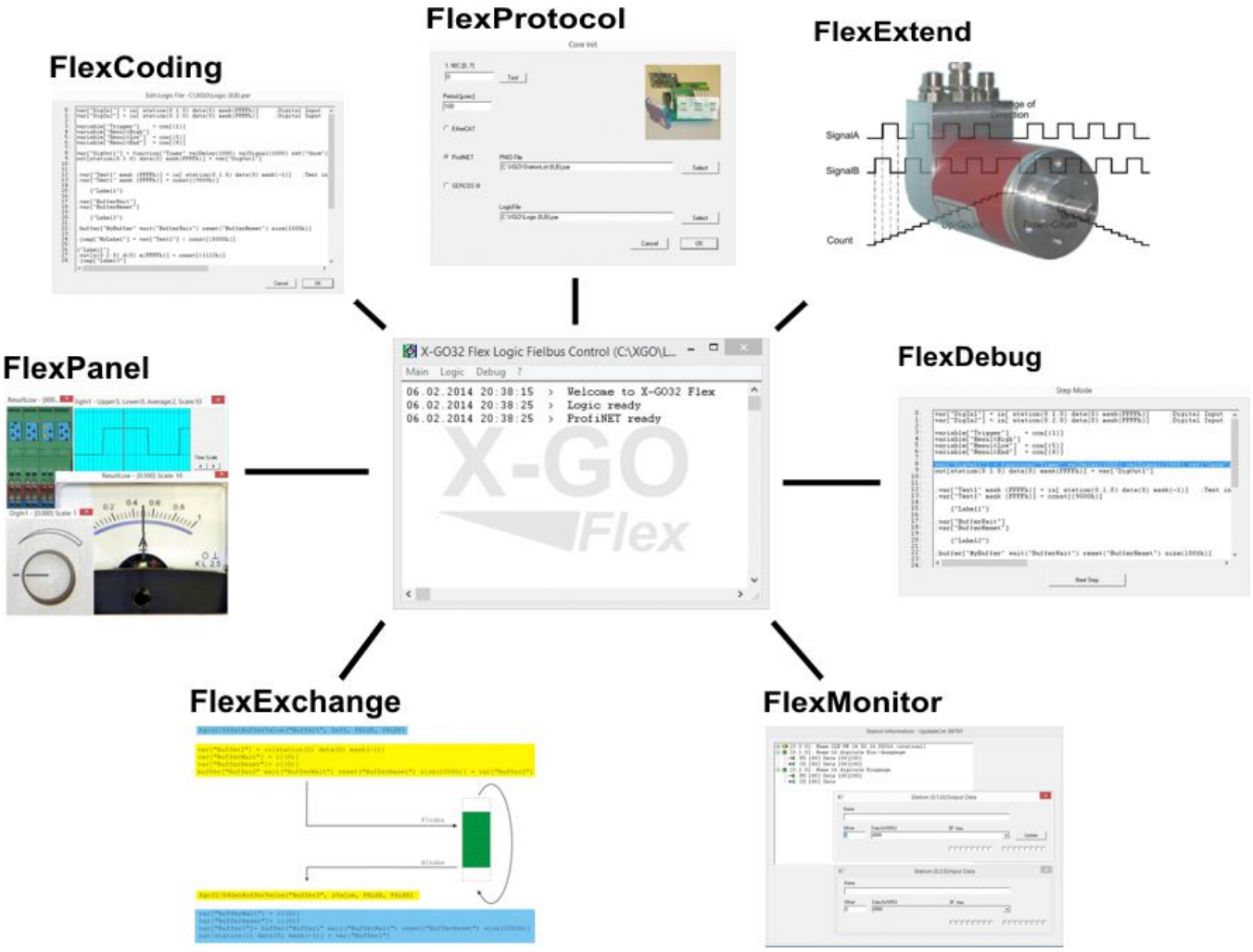
The software X-GO Flex for Windows allows the control of fieldbus devices (EtherCAT, ProfiNET, SERCOS III and Ethernet/IP) under realtime condition, without programming. All stations can be controlled logically with a PLC programming language. On this occasion, different language elements are provided for evaluation of constants and station values, for timer, counter and conditional jumps. Additional function modules may be added to the system. Moreover, the input is checked for logical errors with a PLC program checker. X-GO Flex enables the complete administration of all fieldbus stations. The station monitor shows the current input and output conditions. Data exchange between PLC system and outside applications can be easily done with the X-GO Software Development Kit (SDK). The ability of offering a photorealistic desktop screen, simply designed by parameter files, allows a new dimension for control applications.

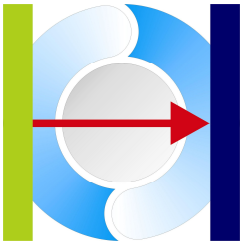


1.1 Features

- Flex Protocol — EtherCAT, ProfiNET, Sercos III and Ethernet/IP support
- Flex Monitor — Input / Output monitoring with position mask
- Flex Coding — adjustable PLC programming language syntax
- Flex Debug — interactive PLC debugging system with code interpretation
- Flex Panel — photorealistic, parameter controlled HMI
- Flex Extend — programming interface for PLC function extensions
- Flex Exchange — programming interface for exchanging data with outside applications

X-GO Flex Logic Control Documentation





1.2 Supported OS

- Windows 7
- Windows 8
- Windows 10

1.3 Supported Platforms

- INTEL Multi-Core (i3 / i5 / i7 / CoreDuo)
- AMD Multi-Core (x2 / x4)

1.4 Reference Hardware

X-GO was especially tested with following hardware platforms:

Mainboard:	DQ57TM
Processor:	INTEL I5
RAM:	4 GB
Graphics:	RADEON 9800 PRO

Mainboard:	ASROCK 785GM-S3
Processor:	ATHLON II X4 320
RAM:	4 GB
Graphics:	ATI Radeon HD 4200 (OnBoard)

Mainboard:	ASUS P5LD2-SE
Processor:	INTEL Core 2 Duo
RAM:	3 GB
Graphics:	ATI Radeon HD 4300

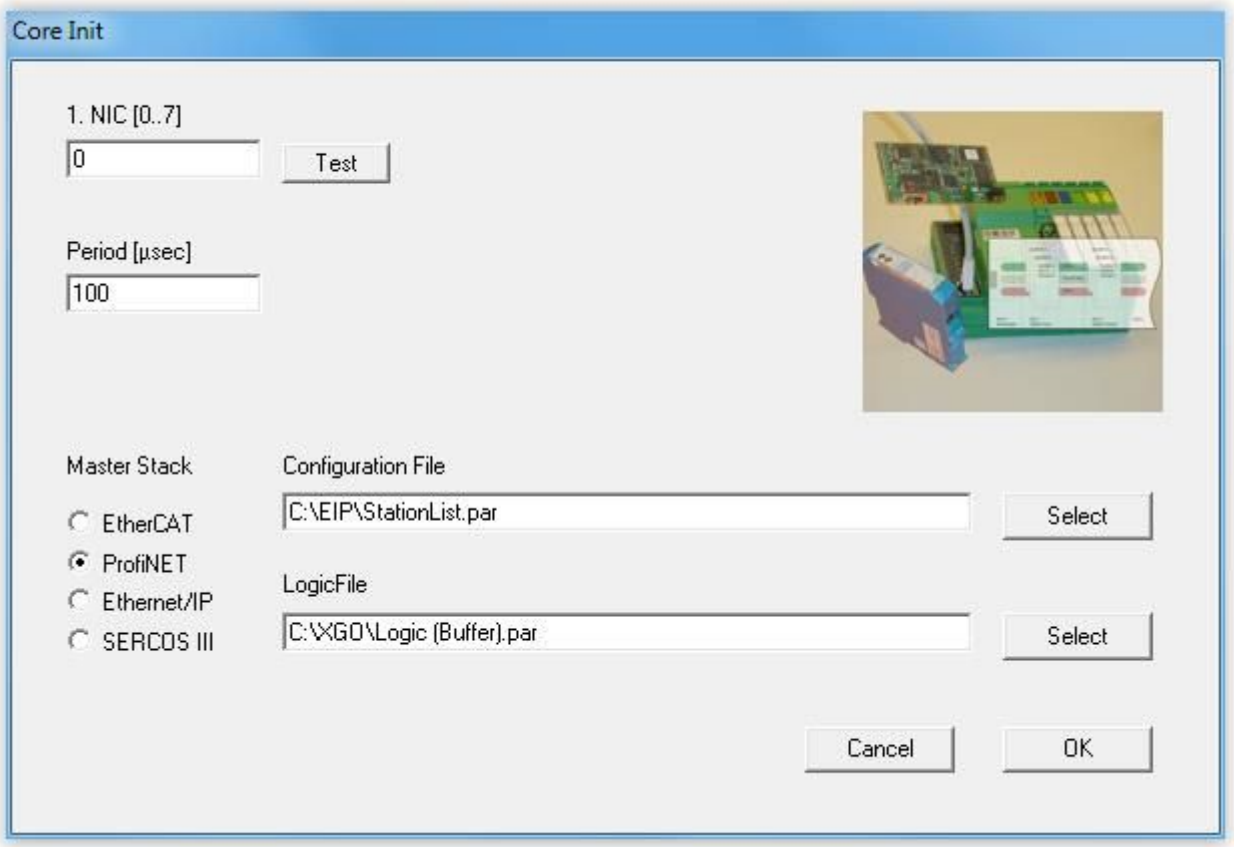
Mainboard:	Lenovo 44444XG
Processor:	INTEL Core i5 M480
RAM: 4 GB	
Graphics:	INTEL HD Graphics BR-1004 (OnBoard)

2 Flex Modules

2.1 FlexProtocol

Whether ProfiNET, EtherCAT, SERCOS III or Ethernet/IP, X-GO Flex based on the X-Realtime Engine for Windows controls the fieldbus protocols without complex programming. A multiple instantiation of X-GO Flex enables even the simultaneous operation of different protocols. All parameters are saved within the file XGO.PAR.

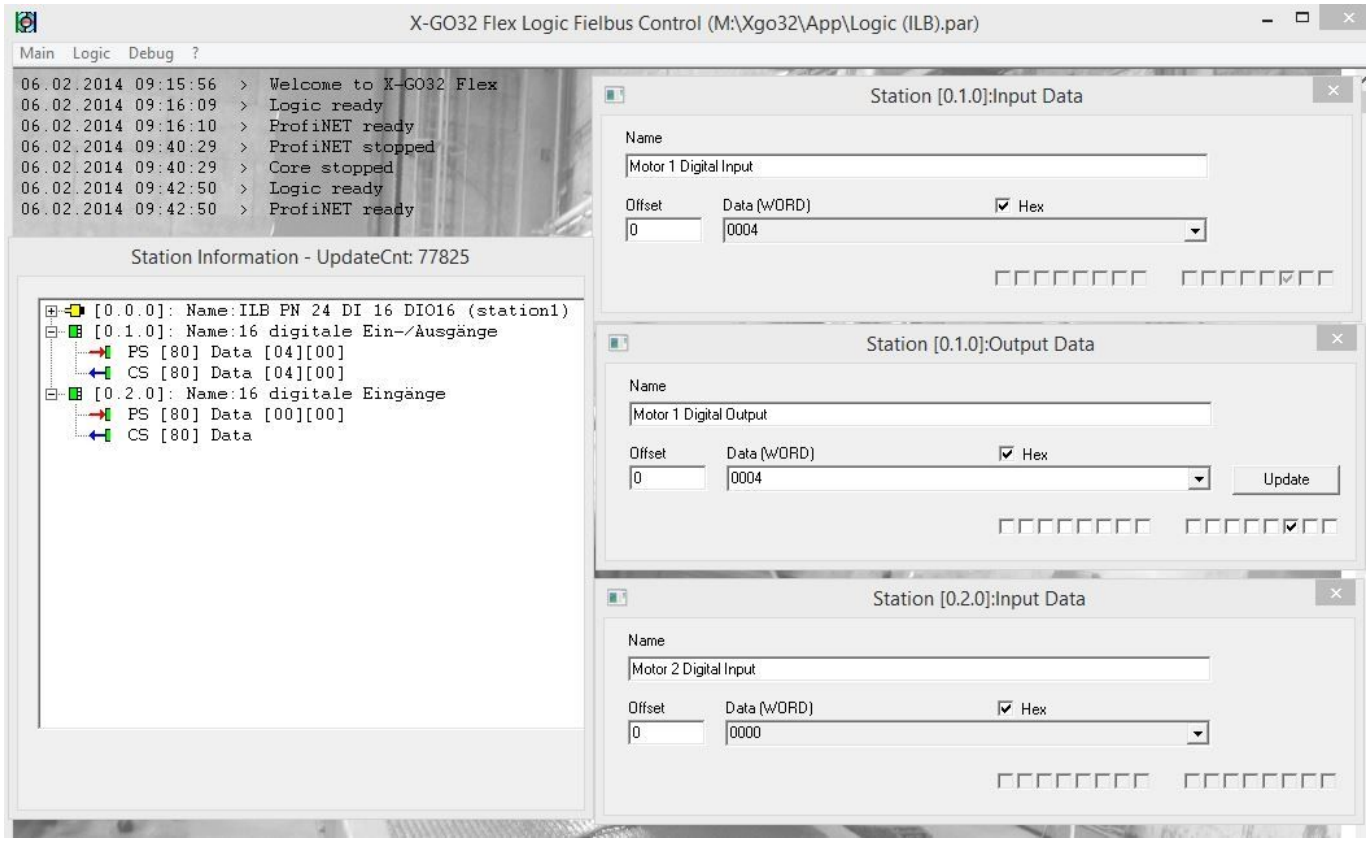
- EtherCAT
- ProfiNET
- Sercos III
- Ethernet/IP



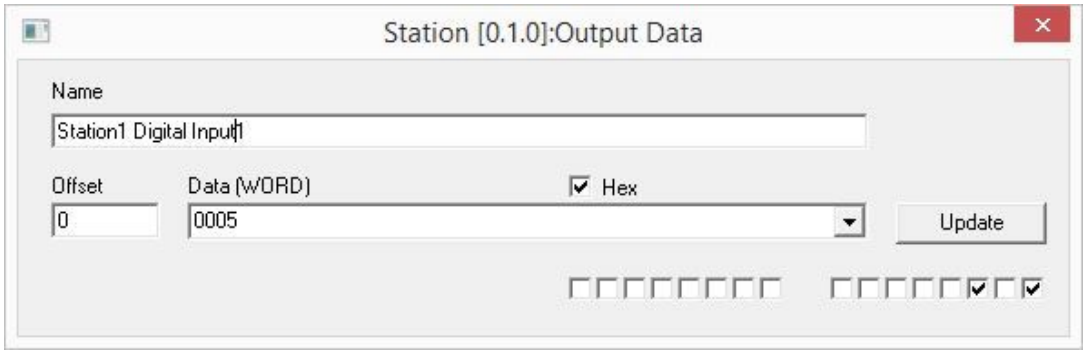
Each protocol has its own initial parameter set, like sampling period, synchronisation period, network card index. Additionally the network connection may tested by pressing the [TEST] button. If so, the network link LED will turn off and on to signal the correct adapter choice.

2.2 FlexMonitor

The Flex Monitor provides you an easy to use station control system which is particularly suitable for service applications. It allows you to display the input data, and the change of output data with different data masks. All data masks are saved within the file “MASK.PAR” (or another given name) by leaving the software. The saved dialog location and name will be used by the Auto Start function..

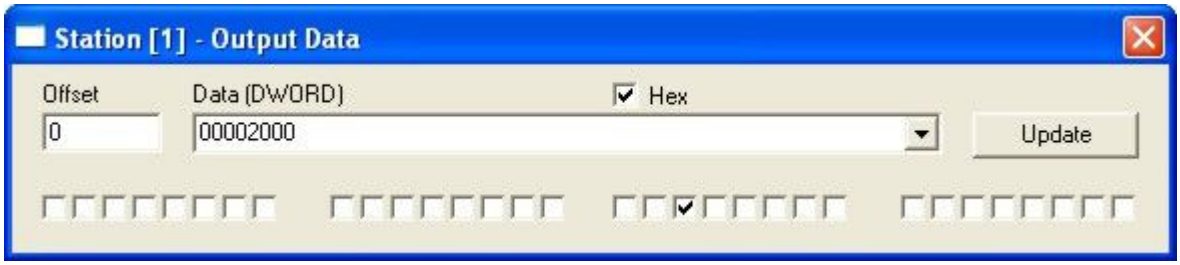


The value dialog allows reading or writing of station values. Just click on a station input or output item inside the Station Monitor opens the dialog. The data format is adjustable (due to the data length) and provides BYTE, WORD, DWORD, ASCII format (either HEX or DEC). The data offset determines the beginning of the payload data (Sercos III without CCON). The Data may be entered or read binary as well as text with an adjustable data format

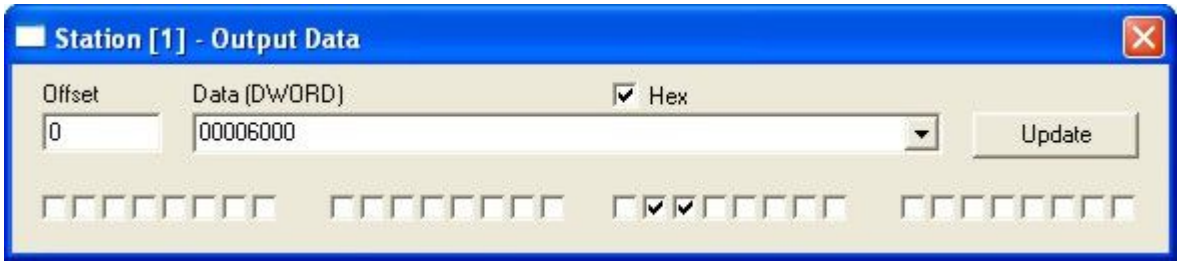


2.2.1 Sample: Drive Startup Sequence

1. Set Drive Control Word (Bit 13) : Drive Restart



2. Set Drive Control Word (Bit 14) : Drive Enable

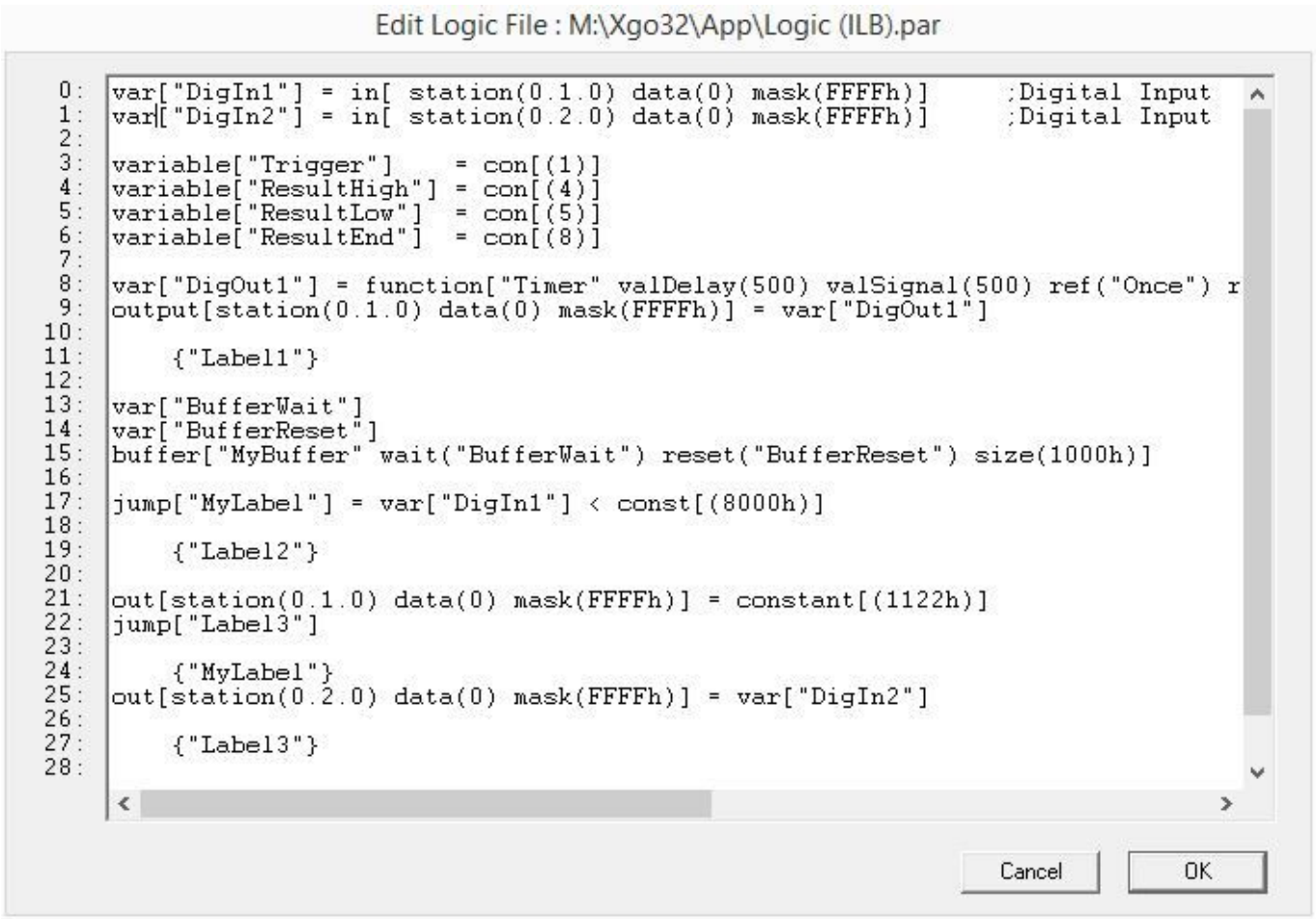


3. Set Drive Control Word (Bit 15) : Drive ON

4. Set Drive Velocity (Bit 31)

2.3 FlexCoding

A programming language adapting to your preference ? Write your PLC realtime code with your own common syntax, with all the language elements and without crashes. A code checker checks the code and displays errors. X-GO Flex provides an editor to edit the PLC program. The PLC language provides all elements of a programming language and may be extended by own function modules (see FlexExtend). The PLC program is saved as text file (default name LOGIC.PAR)



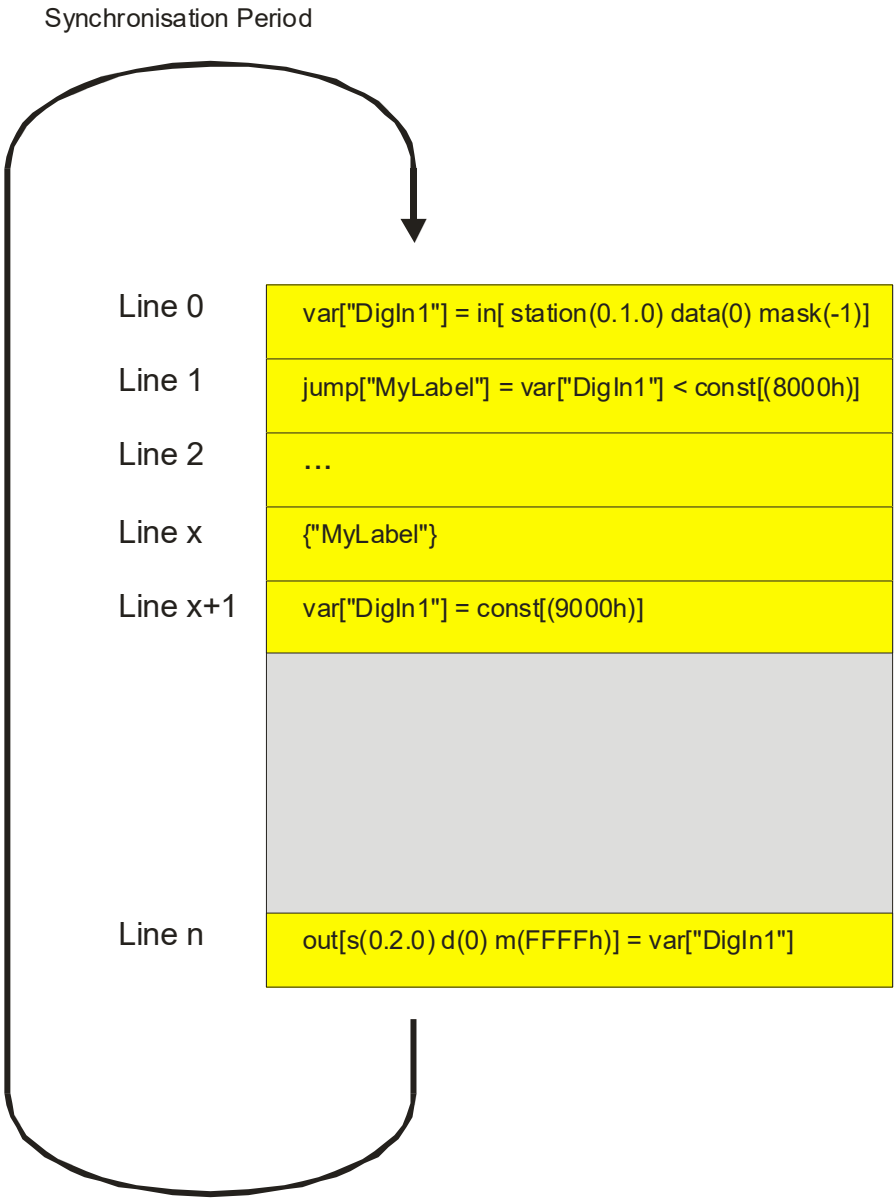
The adapting syntax allows using element declarations of your own preference. So these declarations are equal:

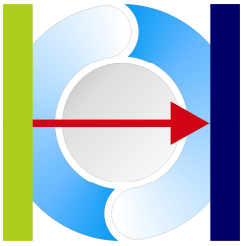
```
v["Sensor"]
var["Sensor"]
variable["Sensor"]
vMyVariable["Sensor"]
```

Only the first character (prefix) of the language element is fixed and must be common.

2.3.1 PLC Execution

All PLC program lines are executed at realtime, starting from the first line to the last and starting again in a loop by the given realtime synchronization period:





2.3.2 PLC Language structure

A PLC program line consists of 1, 3 or 5 language elements:

Element1	Element2	Element3	Element4	Element5
Variable / Buffer / Station /Jump	Operator [=]	Variable / Buffer / Station / Constant / Function	Logical Operator [+,-,/.*,<,>,&,|,^,?,:,#]	Variable / Buffer / Station /Constant / Function

1 element PLC line:

Element1 = Variable / Buffer / Station /Jump

```
jump["MyLabel"]    ;jump to MyLabel

var["BufferWait"]  ;global variable
```

3 elements PLC line:

Element1 : Variable / Buffer / Station /Jump
Element2 : Operator [=]
Element3 : Variable / Buffer / Station / Constant / Function

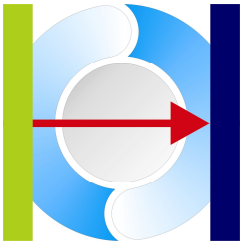
```
variable[“DigIn1”] = input[station(0.1.0 data(0) mask(-1)]

var[“ResultLow”] = const[(9000h)]    ;set result low for timer

jump["MyLabel"] = var["Cond"]
```

5 language elements line:

Element1 : Variable / Buffer / Station /Jump
Element2 : Operator [=]
Element3 : Variable / Buffer / Station / Constant / Function
Element4 : Logical Operator [+,-,/.*,<,>,&,|,^,?,:,#]
Element3 : Variable / Buffer / Station / Constant / Function

```
variable[“DigIn1”] = input[station(0.1.0 data(0) mask(-1)] + constant[(5)]

jump["MyLabel"] = var["Test1"]< const[(8000h)]
```

The size of each value is upto 4 bytes. Each program line may be decorated optionally by comments. Comments are separated by [;]. Even if a line contains only comments, it`s included to the logic cycle as NOP program line.



2.3.3 Station Value

The station value defines a location inside the station data input or output area. The size of each element is upto 4 bytes. The station element may consists of a combination of the parameters StationAddress (see FlexMonitor), DataOffset, BitMask and DataType. Due to the controlled fieldbus, the station address contains of the station index (EtherCAT, Sercos III) or the station index , module index and submodule index (ProfiNET). All station indexes starting from zero and have decimal format.

```
output[station(0.2.0) data(9) mask(FFh) type(4)] ;output station address 0.2.0
                                                 ;data Offset 9
                                                 ;bit mask FFh
                                                 ;data type string

out[s(3) d(0) m(2)]           ;output station address 0
;data offset 0
;bit mask 2

input[station(1) d(2) m(-1)]  ;input station address 1
;data offset 2
;bit mask 0xFFFFFFFF
```

Data Types:
0 : None (DWORD)
1 : BYTE
2 : WORD
3 : DWORD
4 : WORD STRING
5 : DWORD STRING

2.3.4 Constant Value

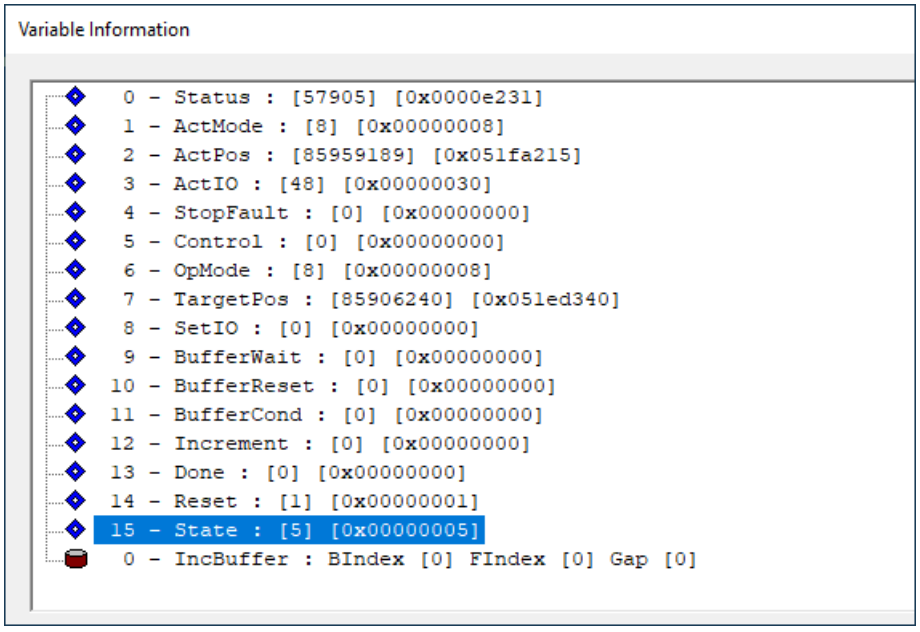
Constant values may have decimal or hexadecimal format. The size of each element is upto 4 bytes.

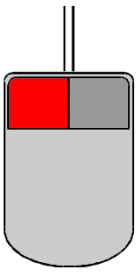
```
constant[(3)]      ;constant value decimal 3
con[(-1)]          ;constant value decimal -1 (0xFFFFFFFF)
c[(15h)]           ;constant value hexadecimal 0x15
```

2.3.5 Variables

Variables are used as place holders for any language element. The size of each element is upto 4 bytes. A variable element may consists of the parameters Name in combination of the parameter BitMask.

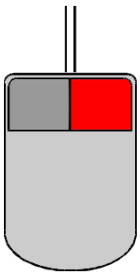
```
v["Sensor1"]            ;sensor 1
var["DigIn1"]           ;digital input 1
variable["Motor2"]      ;motor 2
vMyVariable["Test"]     ;test variable
```





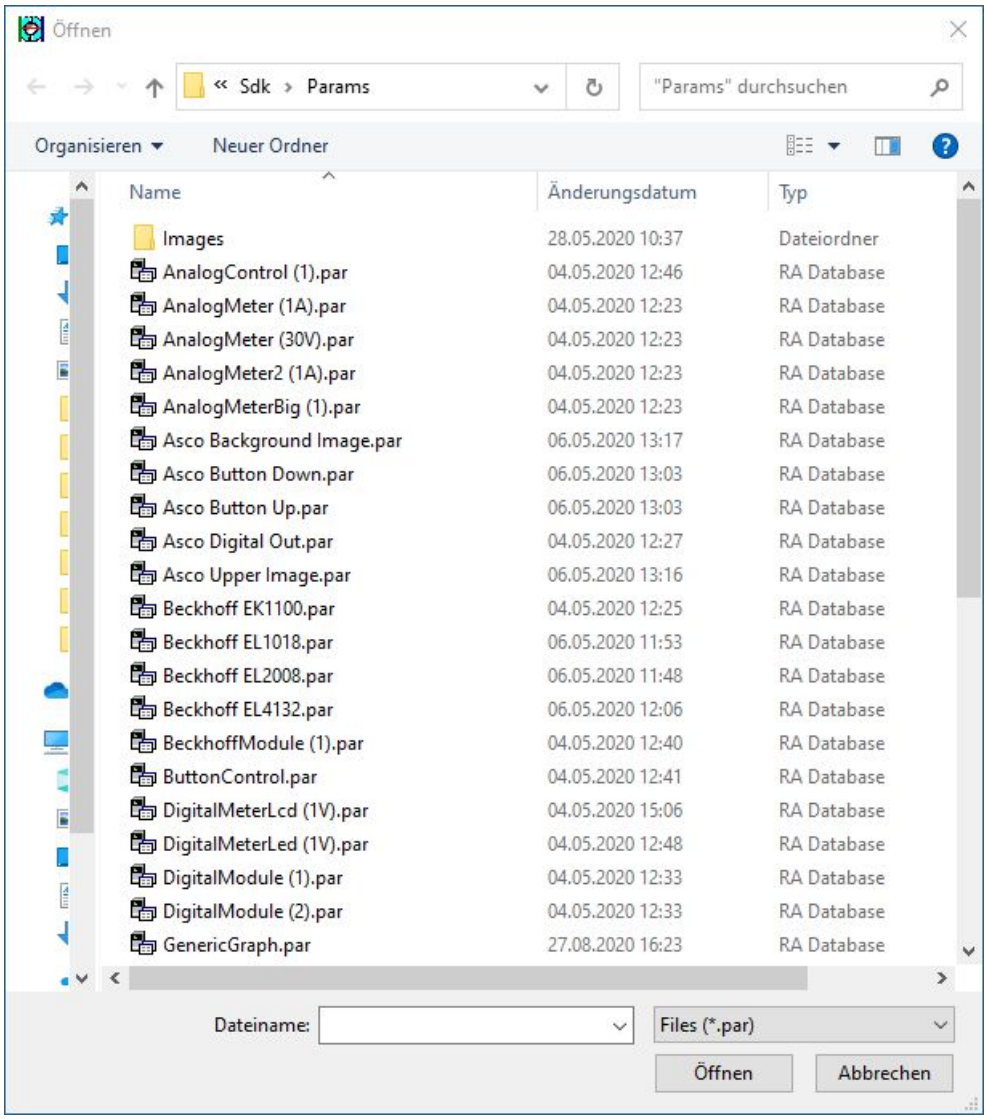
2.3.5.1 Changing a variable value

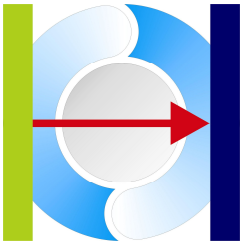
On pressing the left button at the selected variable, the value dialog appears. In debug mode, the value can be changed immediately



2.3.5.2 Selecting a panel

On pressing the right mouse button, a panel can be selected for the variable (see FlexPanel)





2.3.6 Buffers

Buffers can be used to share information with the programming interface between the PLC program and an outside application (see FlexAttach, part of the X-GO SDK). The buffer elements consist of the parameters Name, Size and a reference to the control variables BufferWait, BufferReset and BufferCond.

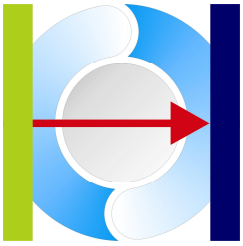
```
var["BufferWait"]  = const[(0)]
var["BufferReset"] = const[(0)]
var["BufferCond"]  = const[(0)]
buffer["MyBuffer" wait("BufferWait") reset("BufferReset") cond("BufferCond")
size(1000h)]
```

Prameters:

[“buffer_name”] buffer name
wait[“name”] (in): if variable is set (1), buffering is suspended, else (0) buffering is active
reset[“name”] (in): if variable is set (1), buffering is reset, else (0) buffering is active
condition[“name”] (out): return buffer condition:

BUFFER_EMPTY = 0
BUFFER_BUSY = 1
BUFFER_FULL = 2

size(x) (in): buffer size. If buffer is located multiple times within the PLC program, only the first location specifies the buffer size



2.3.7 Standard Operators

X-GO provides almost all standard operators, like addition, subtraction, multiplication, division, negation, greater than, less than.

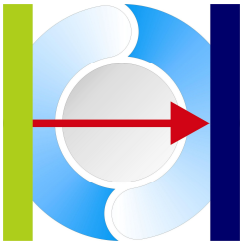
```
var["Limit"] = in[station (1) d(0) m(1)] + c[(1000)]        ;Addition
var["Hold"]  = in[station (2) d(0) m(FFFFh)] < c[(2000)]    ;if less than 2000 the
                                                            ;result becomes 1 else 0
```

Operator	Type	Result
&	LOGIC_TYPE_AND	Element1 & Element2
|	LOGIC_TYPE_OR	Element1 | Element2
^	LOGIC_TYPE_XOR	Element1 ^ Element2
~	LOGIC_TYPE_NOT	Element1 & (~Element2)
+	LOGIC_TYPE_PLUS	Element1 + Element2
-	LOGIC_TYPE_MINUS	Element1 - Element2
<	LOGIC_TYPE_LESS	(Element1 < Element2) ? 1 : 0
>	LOGIC_TYPE_GREATER	(Element1 > Element2) ? 1 : 0
*	LOGIC_TYPE_MUL	Element1 * Element2
/	LOGIC_TYPE_DIV	Element1 / Element2
?	LOGIC_TYPE_COND	Element1 ? Element2 : 0
:	LOGIC_TYPE_NOTCOND	Element1 ? 0 : Element2
#	LOGIC_TYPE_CMP	(Element1 == Element2) ? 1 : 0

2.3.8 Conditional Operators

Conditional operators result with element2 on condition of element1, if the result is logical true, and 0 if the result is false

```
jump["MyLabel"] = var["Hold"] ? const[(1)]      ;Jump if var["Hold"] > 0
jump["MyLabel"] = var["Hold"] ? const[(0)]      ;Jump if var["Hold"] = 0
```



2.3.9 Jumps

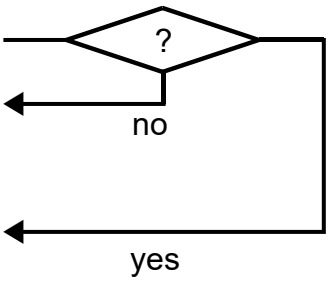
Typically all programming languages provide conditional or unconditional jumps. X-GO Flex provides 3 jump types, conditional, unconditional and conditional wait.

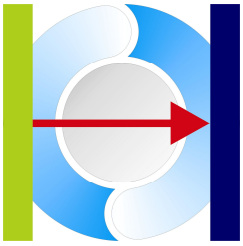
2.3.9.1 Conditional Jump

Conditional jumps are proceeded due to the result of a conditional. The destination of the jump is the label line

jump["MyLabel"] = var[“Hold”] # const[(5)]

Line 0	...
Line 1	`jump["MyLabel"] = var[“Hold”] # const[(5)]`
Line 2	...
Line 3	...
Line 4	`{"MyLabel"}`
Line n	

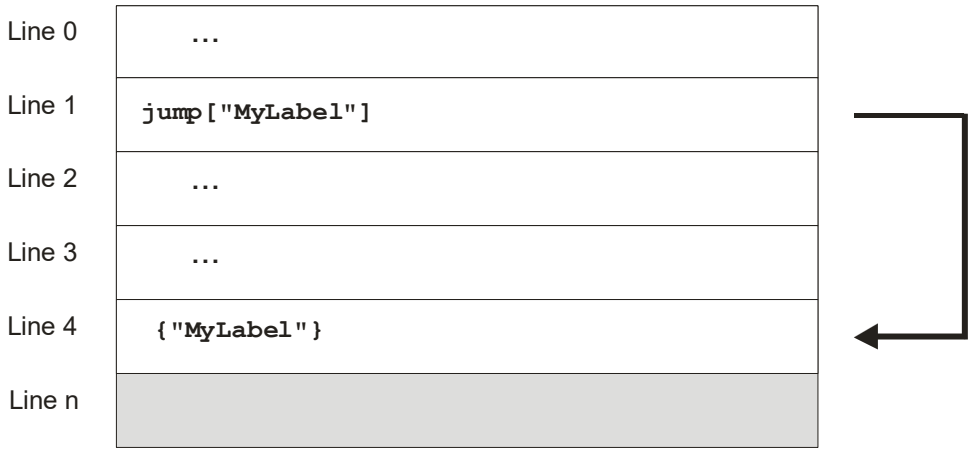




2.3.9.2 Unconditional jump

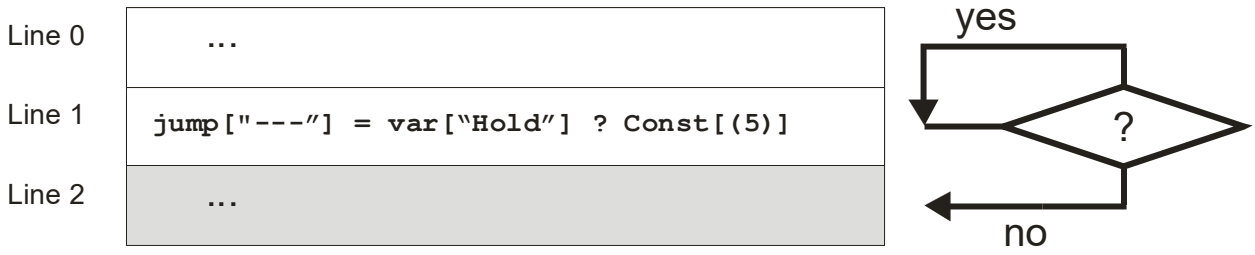
A unconditional jump is immediately executed. The destination of the jump is the label line

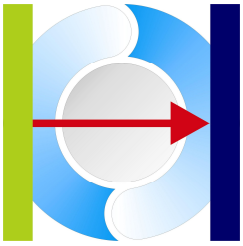
```
jump["MyLabel"]
```



2.3.9.3 Conditional Wait

Conditional wait keeps the line position while“result == TRUE”. On “result == FALSE” the destination of the jump is the next line





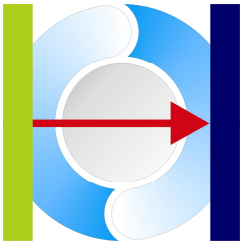
2.3.10 Functions

X-GO Flex provides a set of PLC functionality, like TIMER, COUNTER, THRESHOLD, … .Each function may be used as language element inside the PLC program. A function consists of the FunctionName followed by parameters.

```
Var[“Result”] = function[“Name”, Param1, Param2, Param3, …)
```

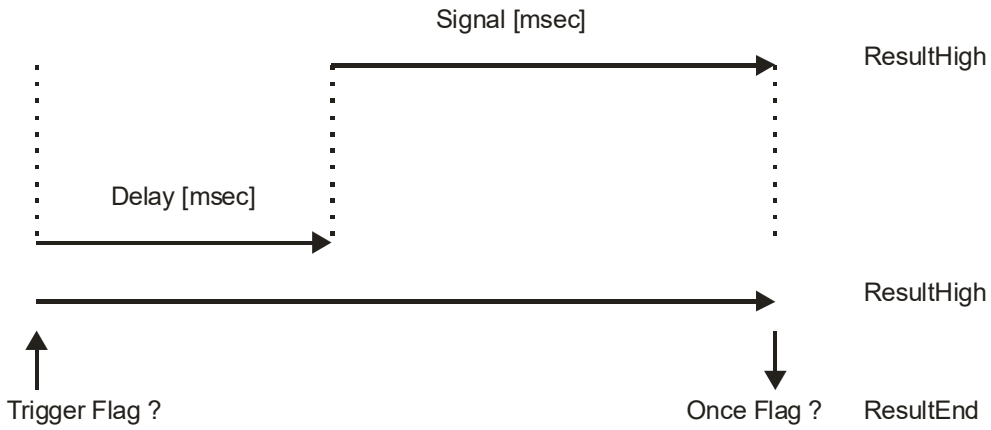
Each function parameter may be a value or a reference. Each function returns a value.

```
Value:      val***(x)              e.g. valPosistion(2)
Reference:  ref***(“VarName”)      e.g. ref (“Trigger”)
```



2.3.10.1 Function Timer

Timers may be used as periodic or single-shot timer. The timer function is defined by the parameter order FunctionName, DelayTime, SignalTime, OnceFlag, ResultLow, ResultHigh, ResultEnd and TriggerFlag. Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v). The timer resolution is 1 msec.




```
var[“Once”] = const[(0)]
var[“ResultLow”]  = const[(2)]
var[“ResultHigh”] = const[(8)]
var[“ResultEnd”]  = const[(0)]
var[“Trigger”]    = const[(1)]

var["Result"] = function["Timer" valDelay(500) valSignal(500) ref("Once")
                ref("ResultLow") ref("ResultHigh") ref("ResultEnd") ref("Trigger")]
```

equal

```
var["Result"] = function["Timer" vDelay(500) vSignal(500) vOnce(0)
                vResultLow(2) vResultHigh(8) vResultEnd(0) vTrigger(1)]
```

2.3.10.2 Function Counter

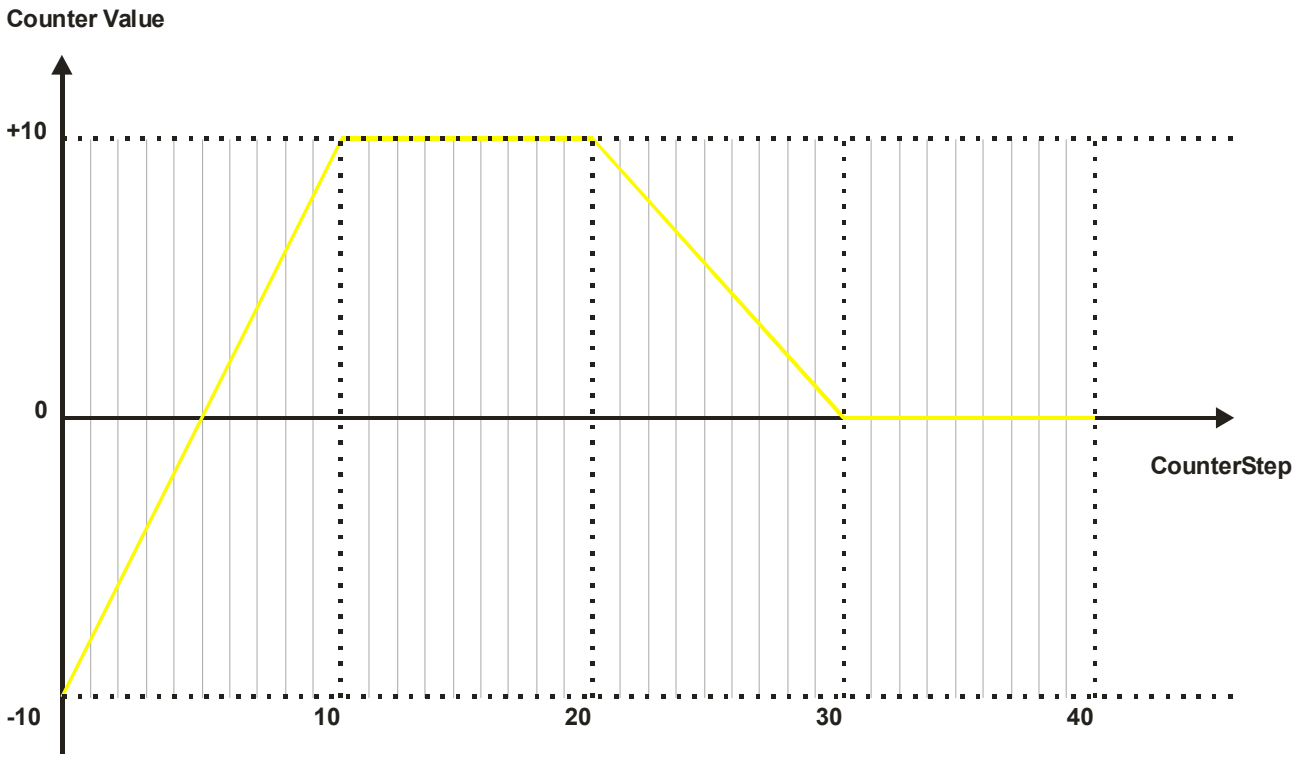
The counter function is defined by the parameter order FunctionName, IntervalCount, RepeatCount, TriggerFlag, and upto 5 positions. Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v). The position parameter is define with X and Y steps. The counter function makes use of a matrix which allows a wide range of use. The matrix is defined by counter positions (x/y). The gap between 2 positions is divided into steps given from the x distance:

e.g. pos1 = (0, -10) pos2 = (10, 10) -> 10 Steps resulting (counter = -10, -8, -6, -4, -2, 0, 2, 4, 6, 8, final 10)

Counters may be used as periodic or single-shot counter. The step time of a counter is based on the given sampling time (typically 100 µsec). This time may be multiplied by the parameter IntervalCount. The RepeatCount is the number of the periodically repeation of the matrix. A RepeatCount of -1 means endless repeation.

e.g.

```
var["Trigger"] = const[(1)]
var["Result"]  = function["Counter" valInterval(100) valRepeat(3) ref("Trigger")
                          pos(0, -10) pos(10,10) pos(20,10) pos(30,0)]
```



Note: Calculation of Y_{step} = (long) $\frac{(Y_n - Y_{n-1})}{(X_n - X_{n-1})}$

2.3.10.3 Function Threshold

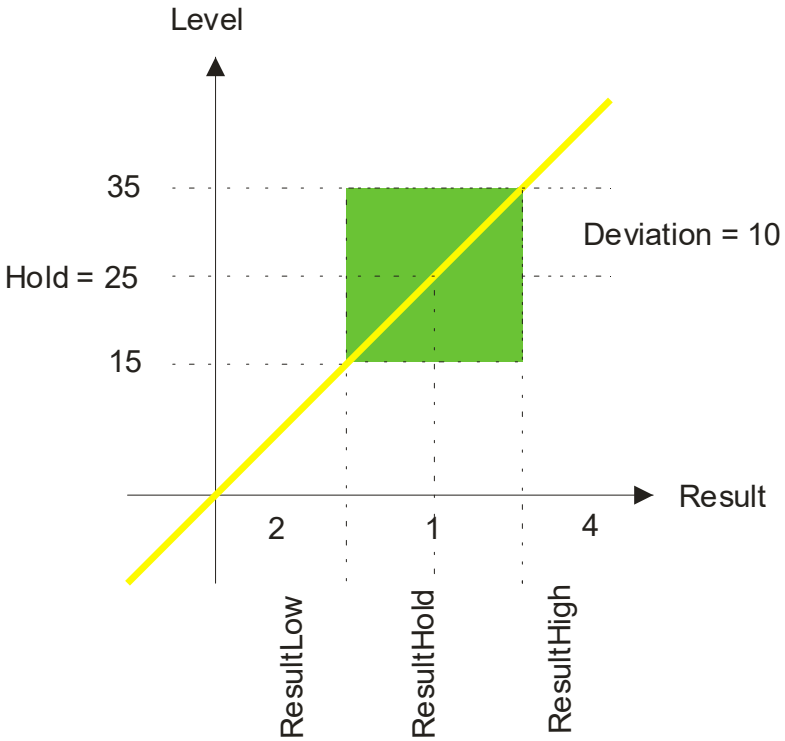
The threshold function is defined by the parameter order FunctionName, Level, HoldLevel, Deviation, ResultUp, ResultDown and ResultHold. Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v).

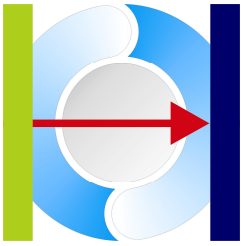
e.g.

```
var["Level"] = in[station (2) data(0) mask(FFFFh)]
var["Result"]  = function["Threshold" ref("Level") valHoldLevel(25) valDeviation(10)
                          valResultUp(4), valResultDown(1), valResultHold(2)]
```

Definition:

if (Level < HoldLevel - Deviation)	Result = ResultDown
if (Level > HoldLevel + Deviation)	Result = ResultUp
if (Level >= HoldLevel - Deviation) and (Level <= HoldLevel + Deviation)	Result = ResultHold





2.3.10.4 Function Shift(Left/Right)

The function Shift(Left/Right) is defined by the parameter order FunctionName, Value, and Shift. Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v).

e.g.

```
var["Value"] = const[(11223344h)]
var["Result"]  = function["ShiftLeft" ref("Value") valShift(8)]
var["Result"]  = function["ShiftRight" ref("Value") valShift(16)]
```

Definition: Result = Value << Shift

2.3.10.5 Function Compare

The function Compare is defined by the parameter order FunctionName, Comperand1, Comperand2, ResultEqual and ResultNotEqual. Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v).

e.g.

```
variable["Value1"] = const[(1)]
variable["Value2"] = const[(2)]
var["Result"] =
     function["Compare" ref("Value1") ref("Value2") valEqual(4) valNotEqual(5)]
```

Definition: if (Value1 == Value2) { Result = valEqual; }
else { Result = valNotEqual; }

2.3.10.6 Function BitTst

The function BitTst is defined by the parameter order FunctionName, Value, Bit, ResultEqual and ResultNotEqual. Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v).

e.g.

```
variable["Value"] = const[(4)]
var["Result"] = function["BitTst" ref("Value") valBit(2) valEqual(4) valNotEqual(5)]
```

Definition: if (Value & (1<<Bit)) { Result = valEqual; }
else { Result = valNotEqual; }

2.4 FlexExtend

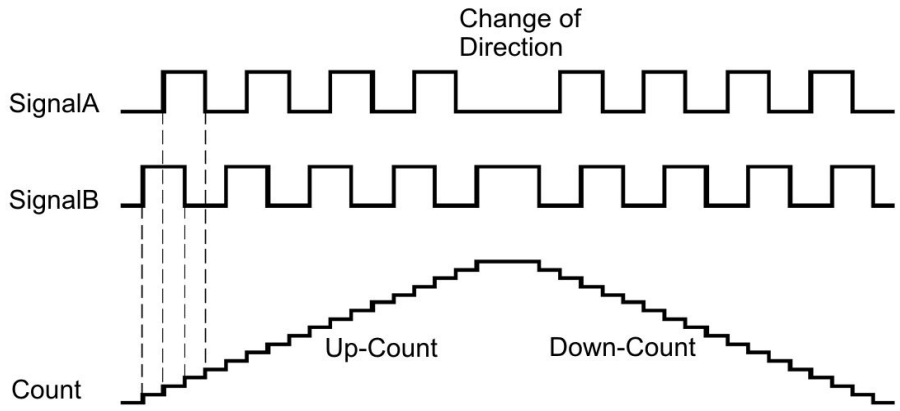
The X-GO Software Development Kit (SDK) allows the extension of the X-GO PLC language with new functional modules. With a simple programming interface new functional modules can be developed and used in the PLC program. For example Drive modules for complex curve control, or encoder modules may be added.

2.4.1 Function Encoder

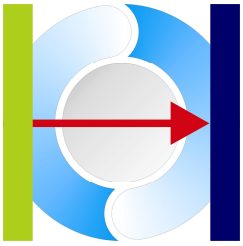
The encoder function of FlexExtend provides a counter value for the encoder signals A, B and Reset. The encoder function is defined by the parameter order SignalA, SignalB and Reset. Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v).

e.g.

```
var["SignalA"] = in[station (2) data(0) mask(1)] ;Get SignalA of incremental encoder
var["SignalB"] = in[station (2) data(0) mask(2)] ;Get SignalB of incremental encoder
var["Reset"]   = in[station (2) data(0) mask(4)] ;Get Reset signal
var["Result"]  = function["Encode" ref("SignalA") ref("SignalB") ref("Reset")]
```



For more information about FlexExtend see the manual of the X-GO Software Development Kit (SDK)



Sample Program:

```
{"Start"}

var["DigIn1"] = in[ station(0.1.0) data(0) mask(FFFFh)]     ;Digital Input
var["DigIn2"] = in[ station(0.2.0) data(0) mask(FFFFh)]     ;Digital Input

variable ["Trigger"]    = const[(1)]
variable ["ResultHigh"] = const[(4)]
variable ["ResultLow"]  = const[(5)]
variable ["ResultEnd"]  = const[(8)]
variable ["Timer1"] = function["Timer" valDelay(1000) valSignal(1000) ref("Once")
                ref("ResultLow") ref("ResultHigh") ref("ResultEnd") ref("Trigger")]

jump["MyLabel"] = var["Timer1"] ? const[(4)]     ;Jump if timer result is 4

var["BufferWait"]  = const[(0)]
var["BufferReset"] = const[(0)]
buffer["Buffer1" wait("BufferWait") reset("BufferReset") size(1000h)] = var["Timer1"]

{"MyLabel"}

var["Counter1"] = function["Counter" valInterval(10) valRepeat(-1) valTrigger(1)
                    pos(0, -10) pos(10,10) pos(20,0)]

out[station(0.1.0) data(0) mask(FFFFh)] = var["Counter1"]
```

2.4.2 Function ConvertEndian

The function “ChangeEndian” changes a value to Big/LittleEndian. The function is defined by the parameter order Value and Size. Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v).

e.g.

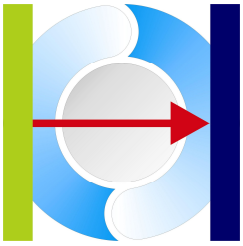
```
var["Value"]  = in[station (2) data(0) mask(-1)]
var["Result"] = function["ConvertEndian" ref(“Value”) varSize(4)]
```

2.4.3 Function Delay

The function “Delay” implies a timer which is used as functional delay in units of [msec] by returning a binary result value (TRUE / FALSE). Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v).

e.g.

```
var["Reset"] = const[(0)]
var["Done"]  = function["Delay" ref("Reset") valDelay(10000)]
```



2.4.4 PneumaticMotion

The function “PneumaticMotion” is used to drive a pneumatics cylinder due to a proportional valve. Therefore the parameter “ParamDist” is measured by a sensor, while the parameter “ParamProp” drives a proportional valve. The parameter mode defines the motion ramp (1: linear, 2: progressive, 3: degressive), defined by 3 points:

x1 , pressure1
X2 , pressure2
X3 , pressure3

Each parameter may be a reference to a variable (prefix r), or a constant value (prefix v).

e.g.

```
var["ParamDist"] = in[station(3) data(5) mask(FFh)]
var["ParamProp"]
var["Done"]  = function["PneumaticMotion" ref("Reset") valMode(1) ref("ParamDist")
                                          pos1(20,5500) pos2(15,4000) pos3(10,5000)
                                          ref("ParamProp")]

out[station(5) data(2) mask(FFFFh)] = var["ParamProp"]
```

2.5 FlexExchange

To make use of the FlexExchange Module the X-GO Software Development Kit (SDK) is required. The FlexExchange module allows easily connecting external applications to the X-GO PLC program. The whole logic stack is accessible via shared memory, so that PLC data can be processed directly with your C # or C + + program, via exported PLC variables and multiple buffers.

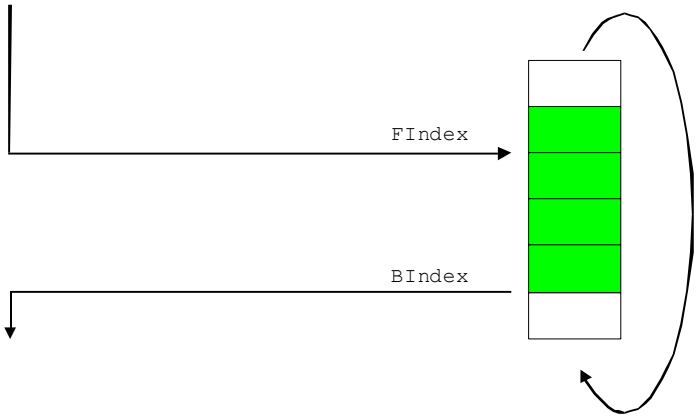
2.5.1 Buffer Exchange

Buffers can be used to share information with the programming interface between PLC program and any outside application. Therefore 2 interface functions are available:

```
Xgo32/64AttachToBuffer("Buffer1", &pBuffer, &hBuffer, &hDesc);
Xgo32/64AttachToBuffer("Buffer2", &pBuffer, &hBuffer, &hDesc);
```

```
Xgo32/64SetBufferBlock(pBuffer, pBlock, BlockLen, FALSE, FALSE, hDesc, &ValuesWritten)
```

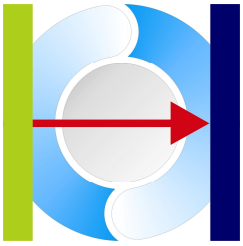
```
var["Buffer2"] = in[station(2) data(0) mask(-1)]
var["BufferWait"] = c[(0)]
var["BufferReset"]= c[(0)]
buffer["Buffer2" wait("BufferWait") reset("BufferReset") cond("CondFlag") size(1000h)] = var["Buffer2"]
```




```
Xgo32/64GetBufferBlock(pBuffer, pBlock, BlockLen, FALSE, FALSE, hDesc, &ValuesRead)
```

```
var["BufferWait"] = c[(0)]
var["BufferReset"]= c[(0)]
var["Buffer1"]= buffer["Buffer1" wait("BufferWait") reset("BufferReset") cond("CondFlag") size(1000h)]
out[station(1) data(0) mask(-1)] = var["Buffer1"]
```

```
Xgo32/64DetachFromBuffer(hBuffer);
Xgo32/64DetachFromBuffer(hBuffer);
```



e.g. set value to buffer

```
var["BufferVal"]  = const [(11223344h)]
var["BufferWait"] = const [(0)]
var["BufferReset"]= const [(0)]
var["BufferCond"]

buffer["InBuffer" wait("BufferWait")
                  reset("BufferReset")
                  cond("BufferCond")
                  size(10000h)] = var["BufferVal"]
```

e.g. get value from buffer

```
var["BufferVal"]
var["BufferWait"] = const [(0)]
var["BufferReset"]= const [(0)]
var["BufferCond"]

var["BufferVal"]= buffer["OutBuffer" wait("BufferWait")
                                     reset("BufferReset")
                                     cond("CondFlag")
                                     size(1000h)]
```

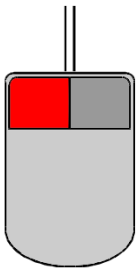
2.5.2 Buffer Synchronization

With the BufferWait flag buffers can be synchronized between PLC program and outside application. Also the buffer can be resetted from both sides using the flag BufferReset. The condition indicates following buffer states:

- BUFFER_FULL
- BUFFER_BUSY
- BUFFER_EMPTY

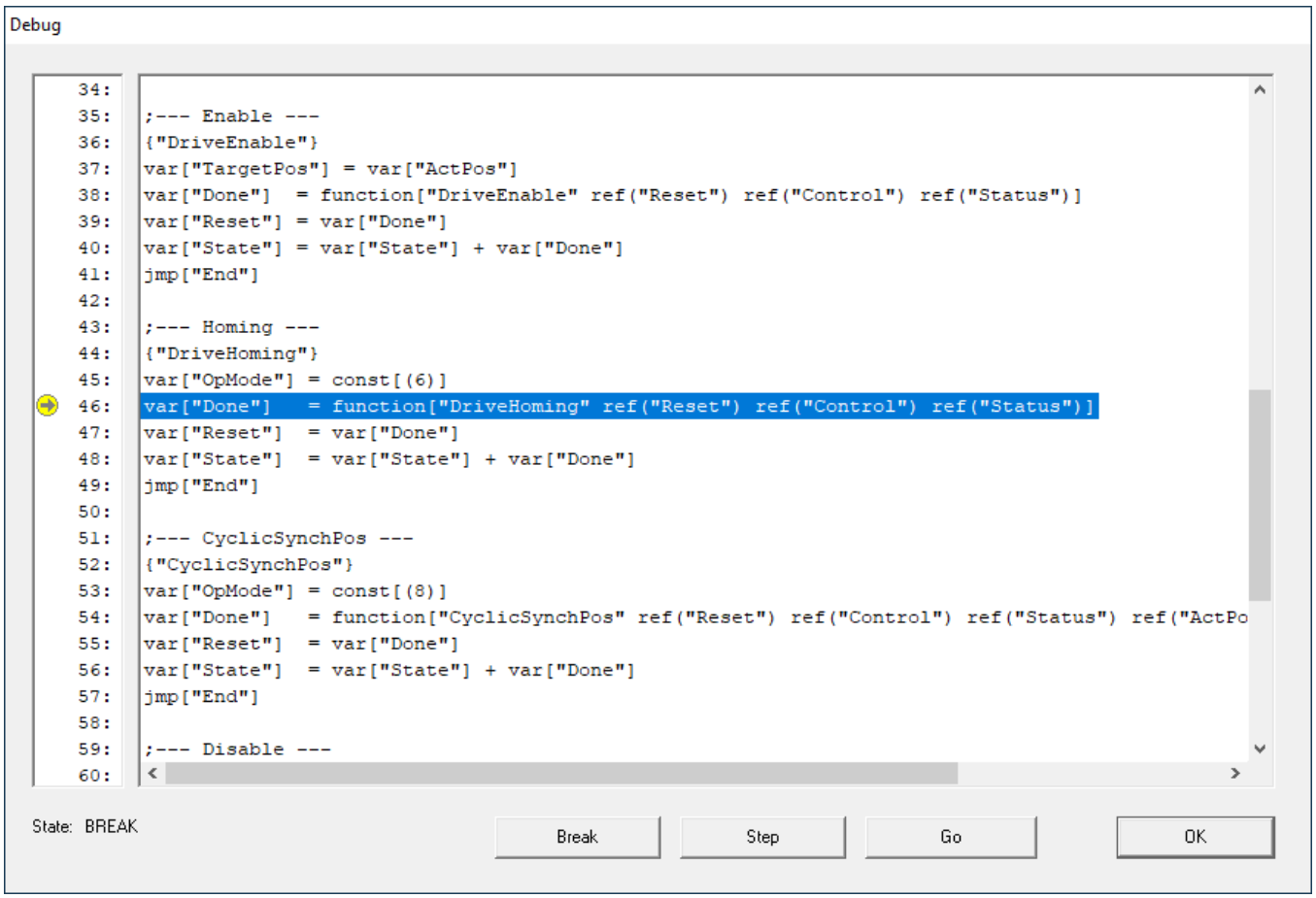
2.6 FlexDebug

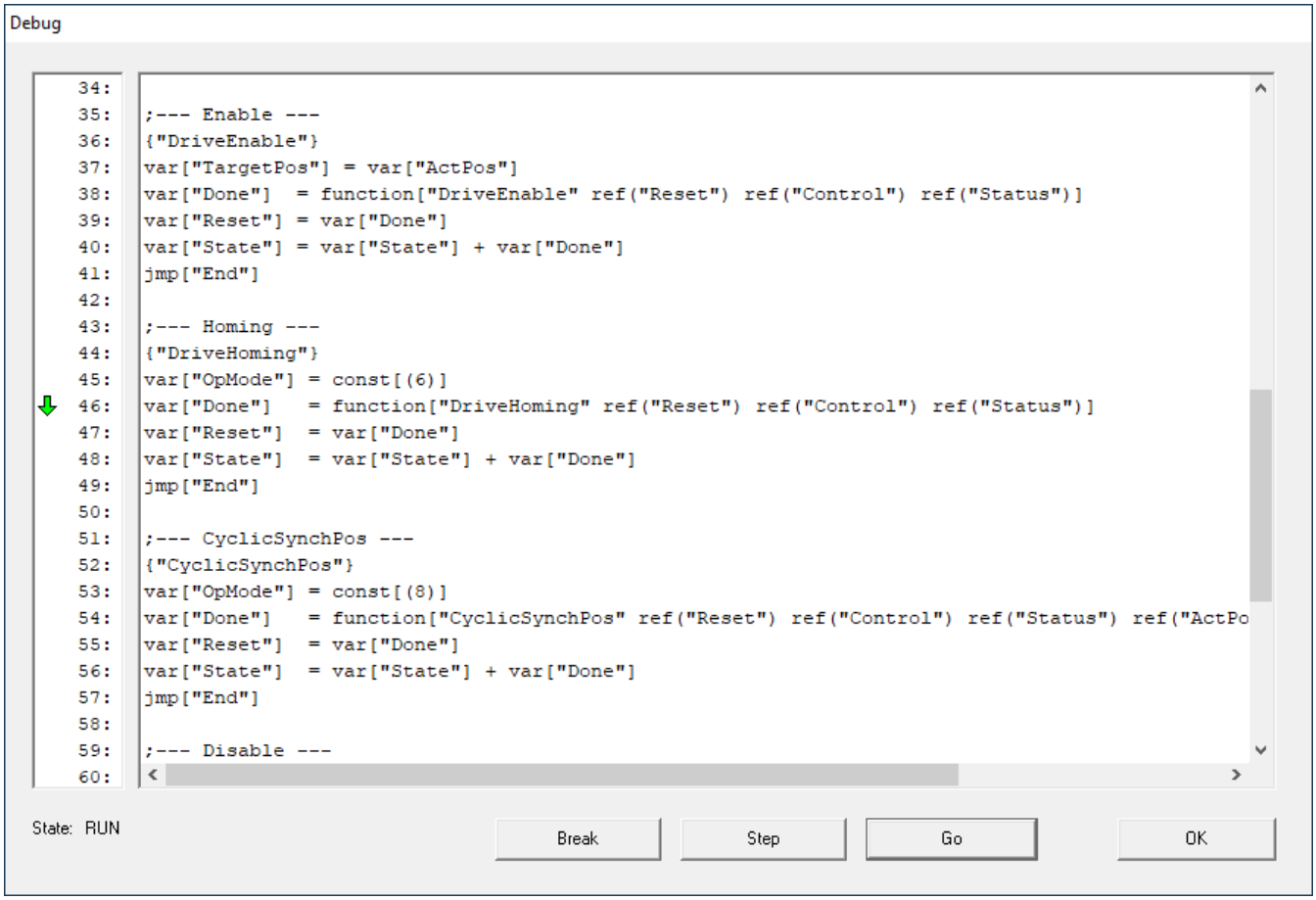
The FlexDebug allows breakpoints and single line execution of the PLC program. Each program line will be evaluated separately, while its result is updated inside FlexMonitor and FlexPanel. Flex Debug allows constant repositioning of the current execution line. When the program is running, it's interrupted on pressing the BREAK button. On BREAK, you can change or display values associated to FlexPanel elements.



2.6.1 Setting a Breakpoint

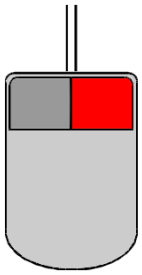
A breakpoint is set on pressing the left mouse button at the selected line. Press the GO button to run the program until the breakpoint is reached





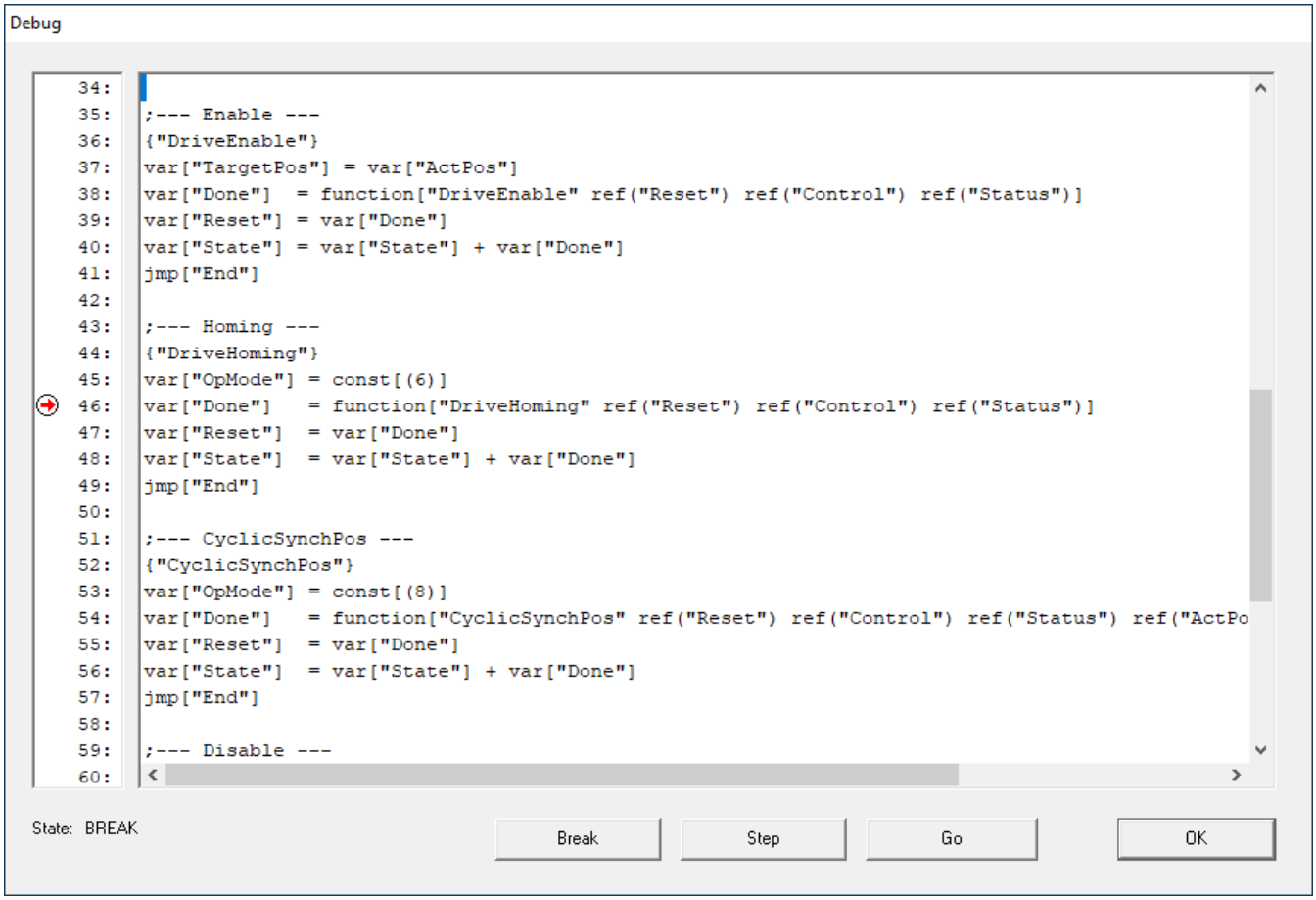
Note:

When pressing the STEP button the line will be executed. Comment lines are stepped as NOP command



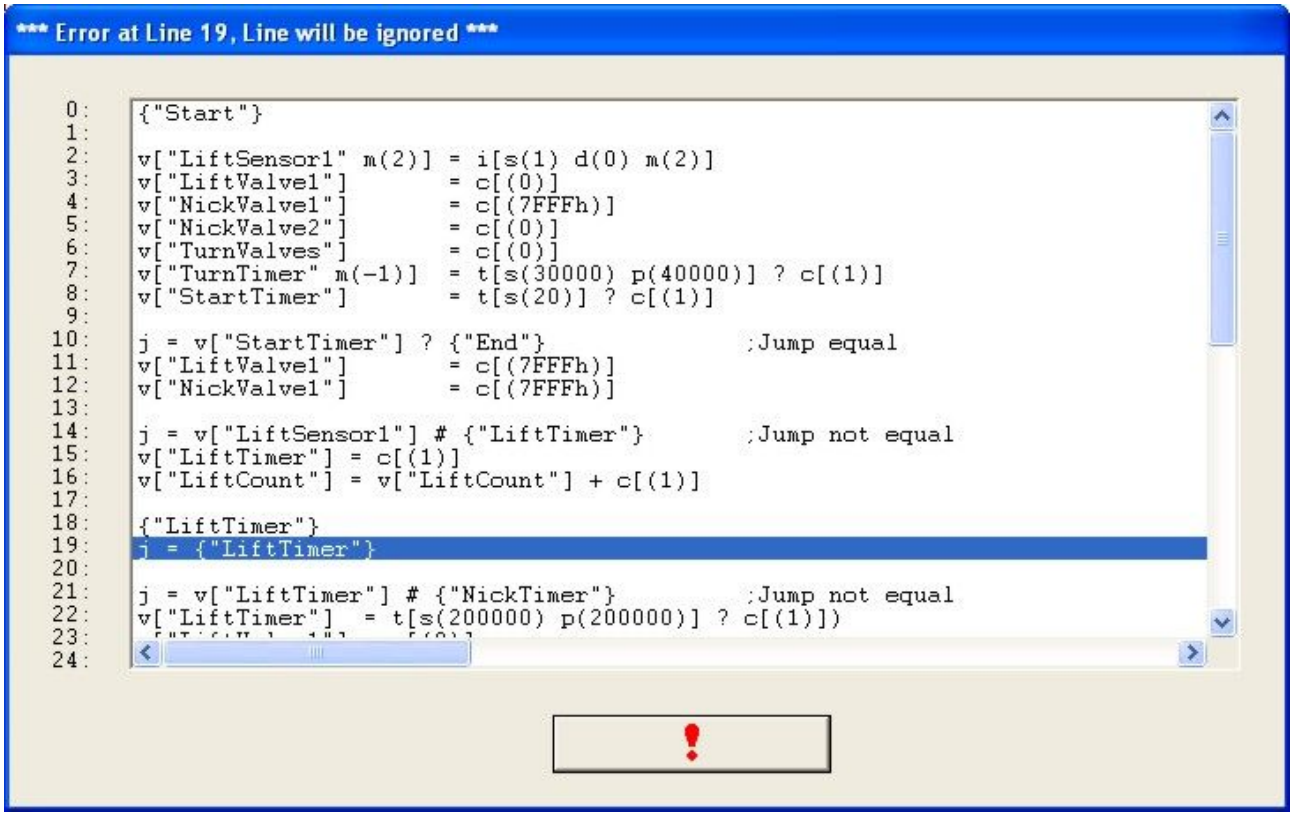
2.6.2 Setting an execution line

When pressing the right mouse button at any line, the execution will be set to this line



2.6.3 Logic Check

X-GO provides a PLC program check while stepping. Thus, e.g. endless loops will be recognized.



The error line will be ignored when pressing the button twice.

2.7 FlexPanel (HmiDesignBox)

The Flex Panel module allows the photorealistic design and integration of machine panel instruments and controls, like meters, switches, modules and backgrounds - without programming. You can use images and graphics of your machine instruments, overlaid by corresponding software templates and simply linked to PLC program variables. The overlay of the software templates is carried out through simple adjustment of the parameter files. Additionally new templates can be programmed and integrated with the X-GO Software Development Kit (SDK).

Pre-defined panel images:

Pre-defined panel functions:

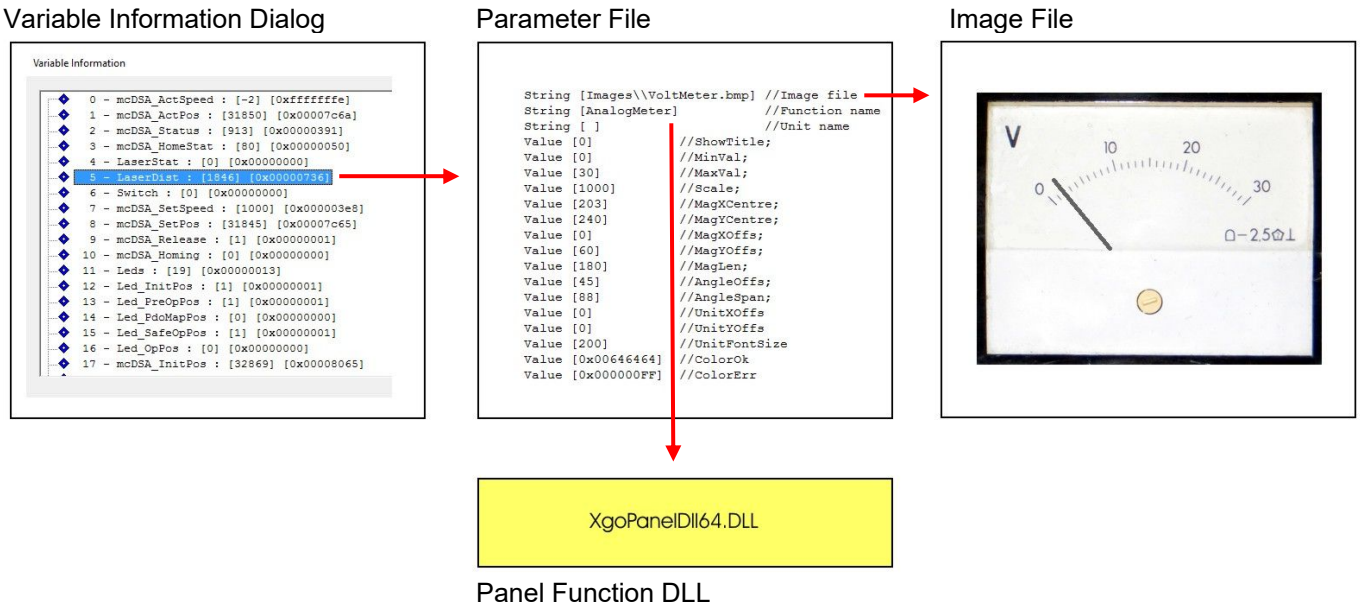
- AnalogMeter
- DigitalMeter
- LinearMeter
- AnalogControl
- DigitalControl
- DigitalModule
- GenericGraph
- Monitor
- Motion

Sample: Parameter File for AnalogMeter Panel:

```
String [Images\\VoltMeter.bmp] //Image file
String [AnalogMeter]           //Function name
String [ ]                     //Unit name
Value [0]            //ShowTitle;
Value [0]            //MinVal;
Value [30]           //MaxVal;
Value [1000]         //Scale;
Value [203]          //MagXCentre;
Value [240]          //MagYCentre;
Value [0]            //MagXOffs;
Value [60]           //MagYOffs;
Value [180]          //MagLen;
Value [45]           //AngleOffs;
Value [88]           //AngleSpan;
Value [0]            //UnitXOffs
Value [0]            //UnitYOffs
Value [200]          //UnitFontSize
Value [0x00646464]   //ColorOk
Value [0x000000FF]   //ColorErr
```

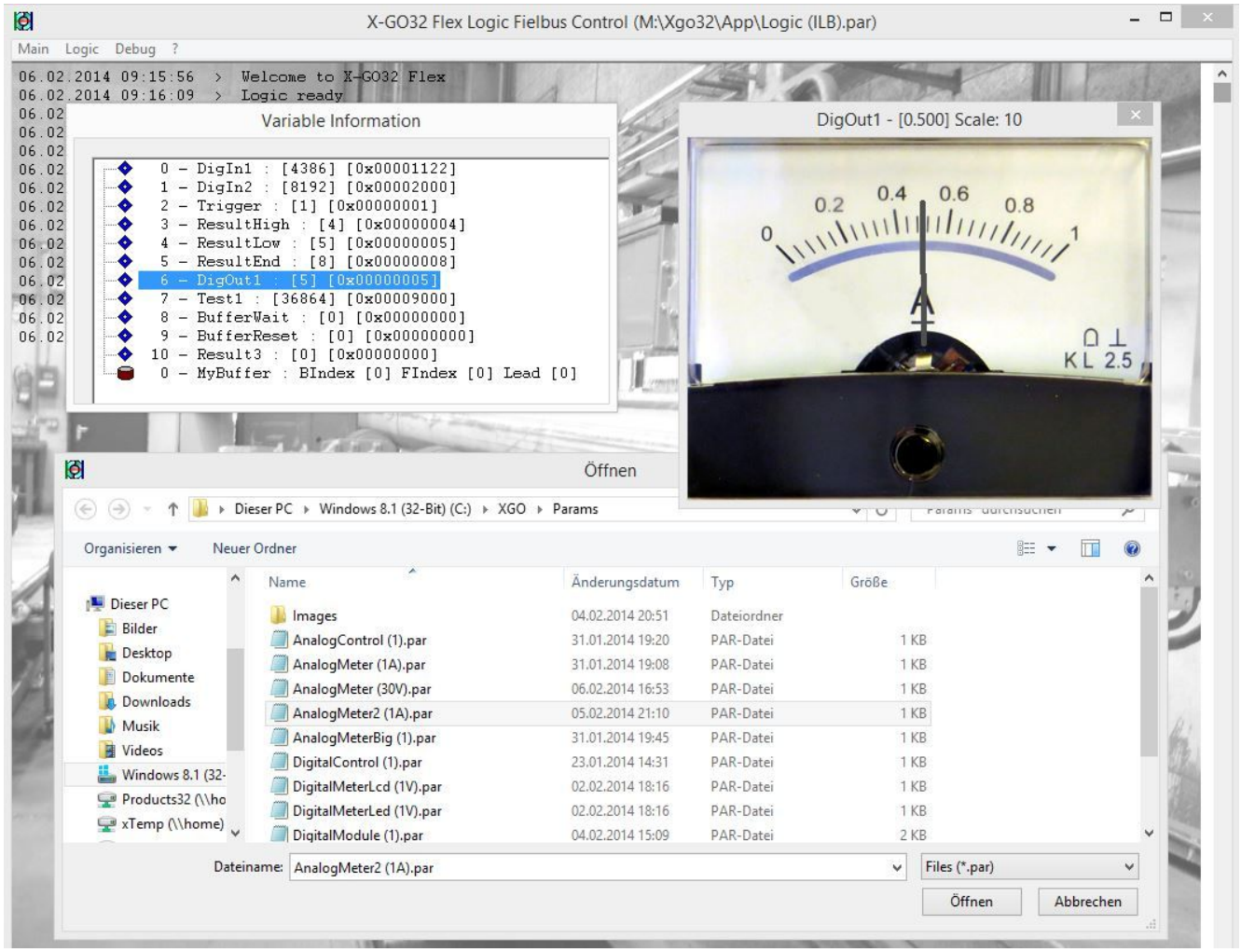
2.7.1 Panel parameter

To overlay a software panel template on a given image, simply adjust the parameter set. This can be done with any editor while X-GO is running. The default parameters set files are stored within the directory \Params, while corresponding images are stored within \Params\Images as files of Windows bitmap type BMP. To adjust the template element positions, like magnitude, unit offset, font size, by pressing the key [F5] the coordinates are shown in the left-top corner inside the panel dialog. When pressing the key [F4] the movement of the panel is adjusted to a grid.



2.7.2 Panel selection

For selecting a panel just right-click on a variable inside the Variable Information dialog.



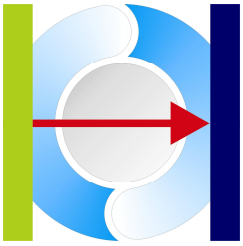
Function Keys:

F4: Lock to Grid. When pressing F4 the selected panel is moving along a grid.
F5: Print Grid Position. When pressing F5, the grid position appears within the selected
F6: Unlock for moving. The selected panel appears at the upper left corner and must be unlocked for moving by pressing the Key F6.
F7: Delete the selected panel
F8: Decrease Z-Order
F9: Increase Z-Order

Note:
For the development of new panel functions, Sybera offers the X-GO Software Development Kit

2.8 Background Image

If an image named “”MAINIMG.BMP” is present within the XGO directory, the image is used as background image for XGO. It may be used as panel background to build up a photo-realistic control desktop in combination with the AUTOSTART feature..



3 X-GO Installation

For installation of the X-GO software following steps are required:

Preparation

1. Provide a PC with INTEL or REALTEK Ethernet adapter and Windows operating system with administrator
2. rights
3. Make shure, the latest Updates are installed (especially Windows 7)
4. Check the installed Ethernet adapter has given a correct IP address
5. Install the MFC redistributables DLLs (in Folder MISC \ REDIST)
6. Install Fonts (in Folder MISC \ FONTS)
7. Install MSXML redistributables (in Folder MISC \ MSXML)

Installation

8. Run the program SYSETUP32/64
9. (make sure the directory path has no space characters)

10. On Installation the PEC information (PID, SERNUM and KEYCODE) must be
11. entered. The SERNUM for the evaluation version is: 12345678,
12. the KEYCODE is: 00001111-22223333

13. Select Network card
14. Reboot the System

Optional: Install network card manually (see chapter 2.1)
Optional: Check license with SYLICENCECHECK(32/64).EXE

Operation

15. Use additional Tools for device configuration

ProfiNET: Use PNIOVERIFY to generate device configuration file

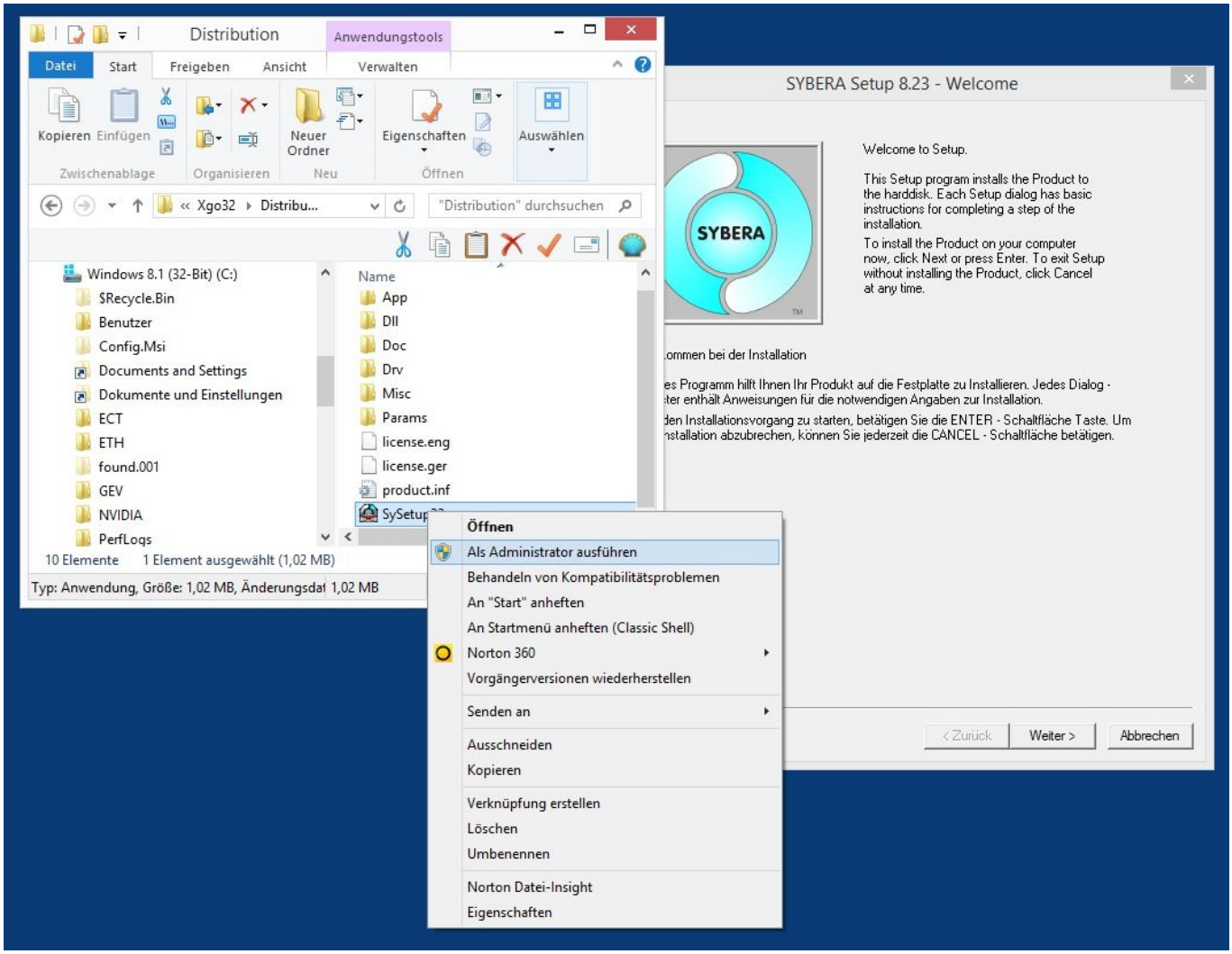
EtherCAT: Use ECATVERIFY to generate device configuration file ECATDEVICE.PAR
(to be placed in \WINDOWS\SYSTEM32)

Ethernet/IP: Use EIPCONFIG to generate device configuration file

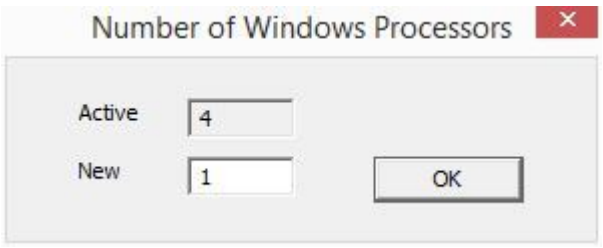
Sercos III: Use editor to generate device configuration file SC3CONFIG
(to be placed in \WINDOWS\SYSTEM32)

16. Run X-GO(32/64)

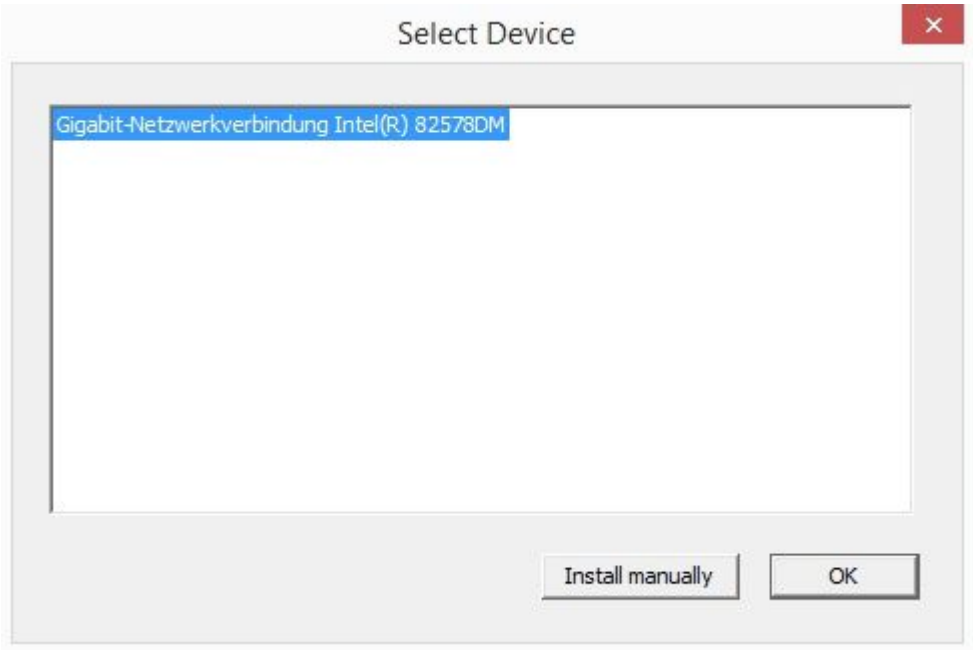
Start installation program SySetup32/64 with administrator privileges



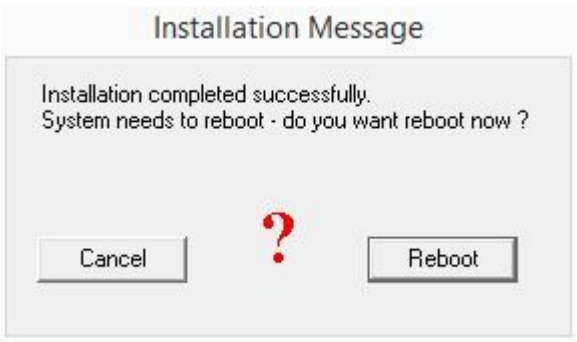
Select windows processor count (typically Active - 1 is best)

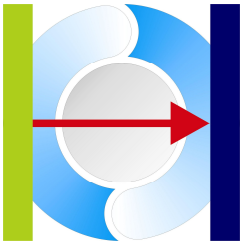


Select ethernet device and quit with button [OK]



Reboot the system





3.1 BIOS Adjustment

Note:

For proper operation, make sure within the BIOS the INTEL SpeedStep Technology, the INTEL TurboBoost Technology as well as the INTEL C-STATE Technology is turned off.

3.1.1 Disable Intel SpeedStep Technology

Enhanced SpeedStep — SpeedStep also modulates the CPU clock speed and voltage according to load, but it is invoked via another mechanism. The operating system must be aware of SpeedStep, as must the system BIOS, and then the OS can request frequency changes via ACPI. SpeedStep is more granular than C1E halt, because it offers multiple rungs up and down the ladder between the maximum and minimum CPU multiplier and voltage levels.

3.1.2 Disable C-States

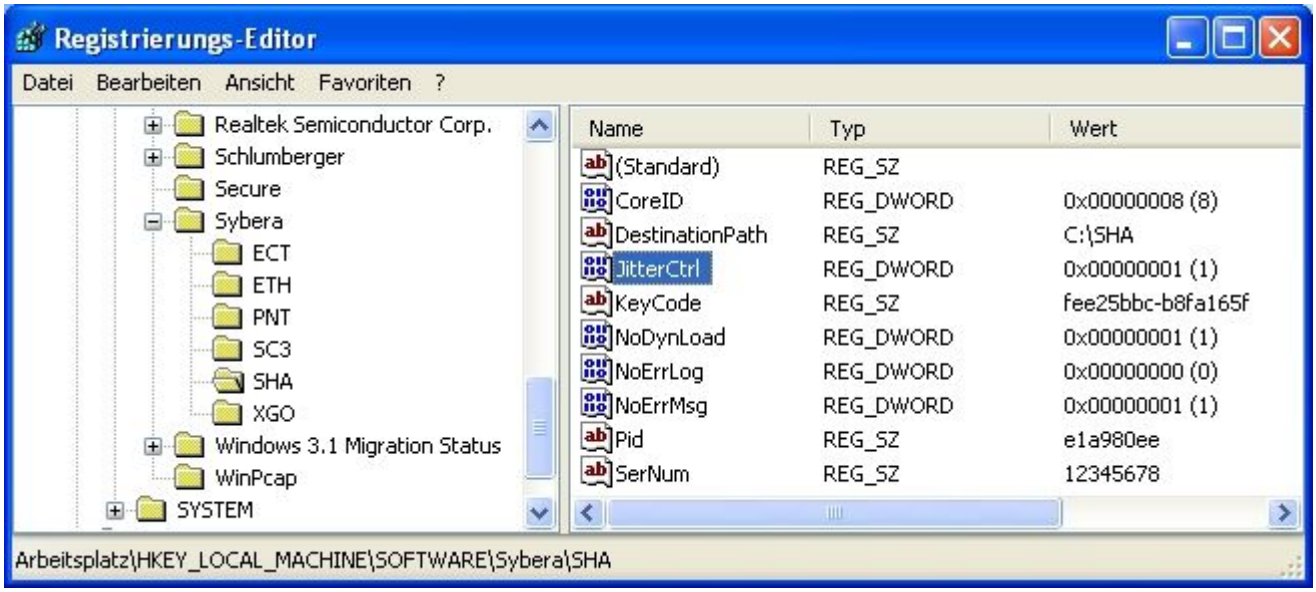
C1E enhanced halt state — Introduced in the Pentium 4 500J-series processors, the C1E halt state replaces the old C1 halt state used on the Pentium 4 and most other x86 CPUs. The C1 halt state is invoked when the operating system's idle process issues a HLT command. (Windows does this constantly when not under a full load.). C0 is the operating state. C1 (often known as Halt) is a state where the processor is not executing instructions, but can return to an executing state essentially instantaneously. All ACPI-conformant processors must support this power state. Some processors, such as the *Pentium 4*, also support an Enhanced C1 state (C1E or Enhanced Halt State) for lower power consumption. C2 (often known as Stop-Clock) is a state where the processor maintains all software-visible state, but may take longer to wake up. This processor state is optional. C3 (often known as Sleep) is a state where the processor does not need to keep its *cache* coherent, but maintains other state. Some processors have variations on the C3 state (Deep Sleep, Deeper Sleep, etc.) that differ in how long it takes to wake the processor. This processor state is optional.

3.1.3 Disable TurboMode

Intel® Turbo Boost Technology automatically allows processor cores to run faster than the base operating frequency, increasing performance. Under some configurations and workloads, Intel® Turbo Boost technology enables higher performance through the availability of increased core frequency. Intel® Turbo Boost technology automatic allows processor cores to run faster than the base operating frequency if the processor is operating below rated power, temperature, and current specification limits. Intel® Turbo Boost technology can be engaged with any number of cores or logical processors enabled and active. This results in increased performance of both multi-threaded and single-threaded workloads.

3.2 Jitter Control

Note: Since a notebook has a quiet different jitter behaviour than desktop systems, an enhanced jitter control mechanism is required. Therefore SYBERA provides a registry entry called “JitterCtrl”. This entry allows an adaptive iteration to the best jitter behaviour of the notebook.



Following values are valid:

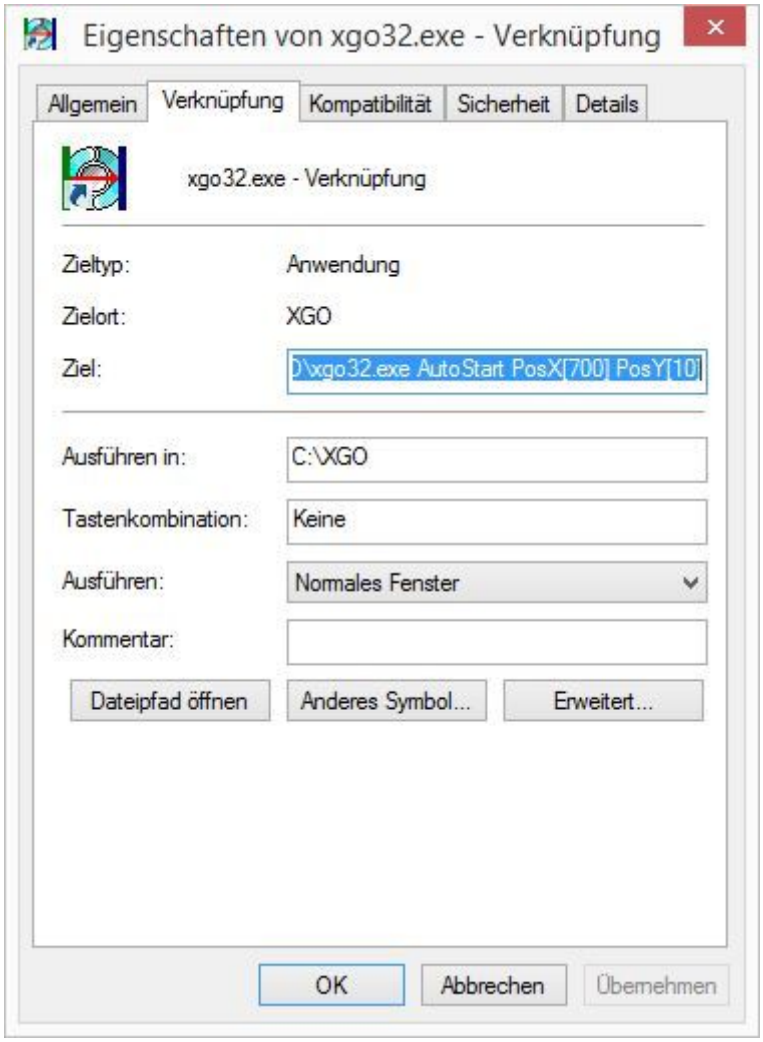
0: No enhanced jitter control
1: Enhaced Jitter Control, Step 1 (first choice together with BIOS settings)
2: Enhaced Jitter Control, Step 2 (for INTEL platforms only)
3: Enhaced Jitter Control, Step 3 (for INTEL platforms only, together with BIOS settings)

Note:
In order to operate SYBERA software under VISTA or Windows 7, the program must be started with Administrator privilleges. It's not sufficient to be logged on as administrator, since VISTA or Windows 7 does not share the rights with the processes automatically.

3.3 Autostart

Note: For frequent operation X-GO may be started with command line options:

AutoStart X-GO uses last saved startup parameters



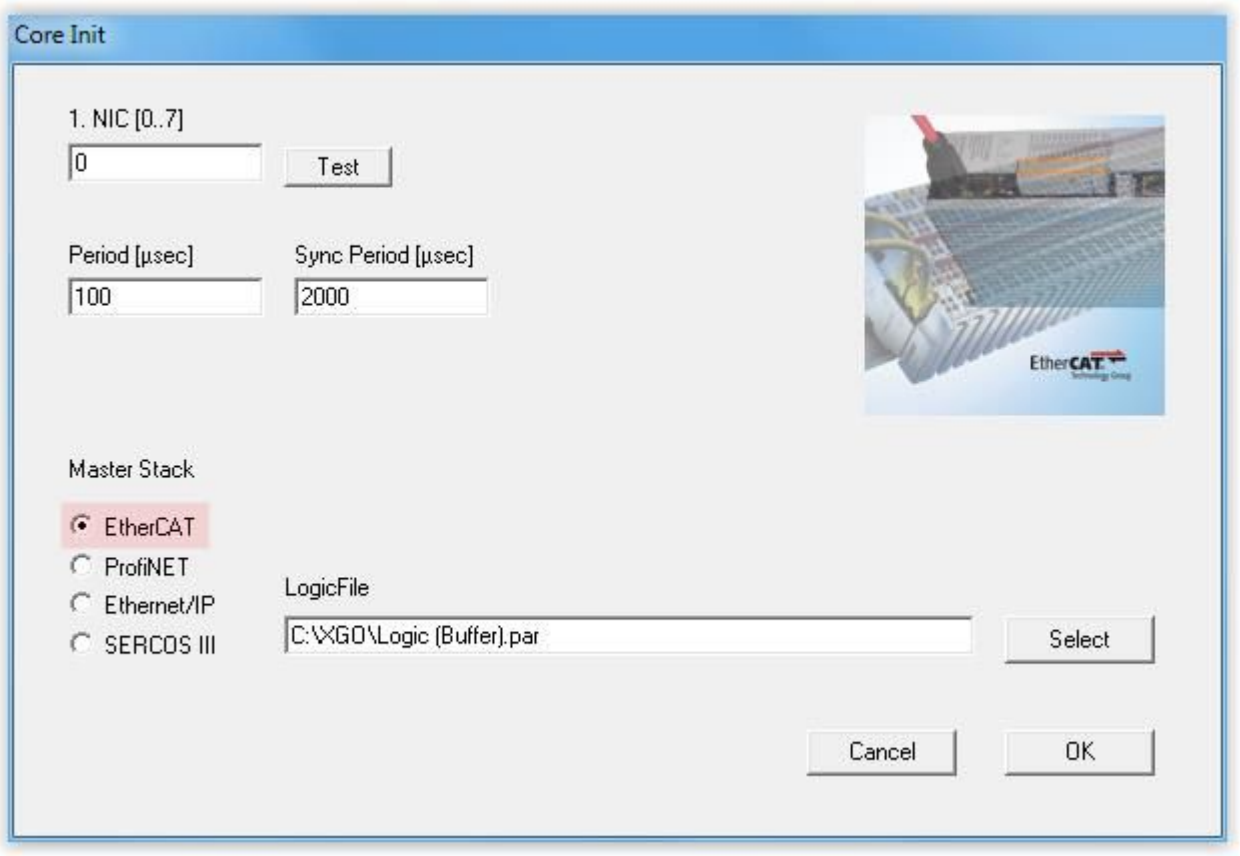
PosX[Pixel] X position of the main window
PosY[Pixel] Y position of the main window

Note:

On Autostart, all Dialogs (Station Information, Variable Information and Value Dialogs) are hidden. The Main windows is minimized. Only Panel elements are displayed

4 EtherCAT

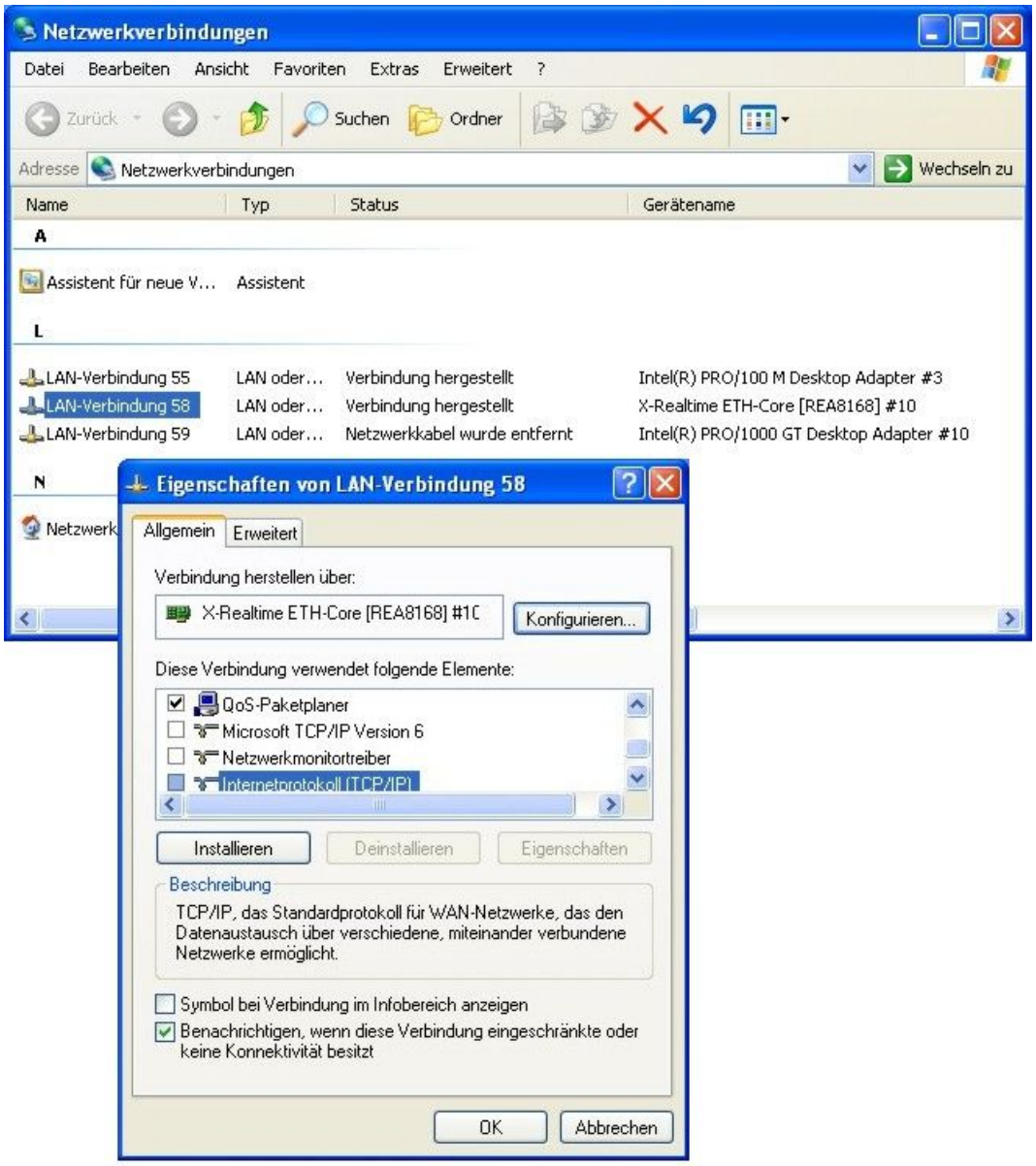
When using X-GO EtherCAT following parameters have to be used:



1.NIC	Index of the first realtime Ethernet dirver (typically 0)
Period	Sampling period (typically 100 µsec)
Sync Period	Synchronisation (Update) Period (typically 2000 µsec)

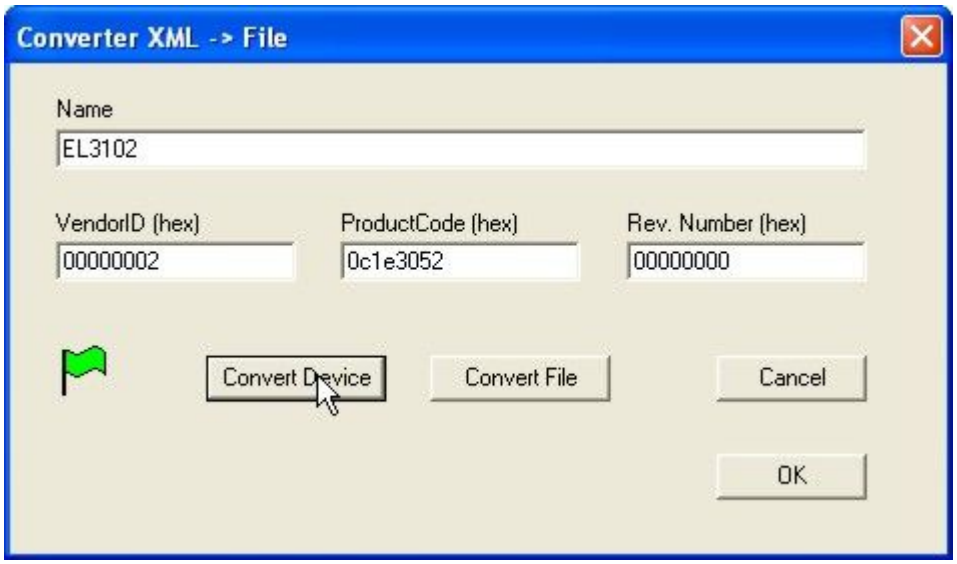
Note:

For proper EtherCAT operation its recommended to use the SERCOS III network as standalone network. This requires to turn off the Windows protocols for this network connection:



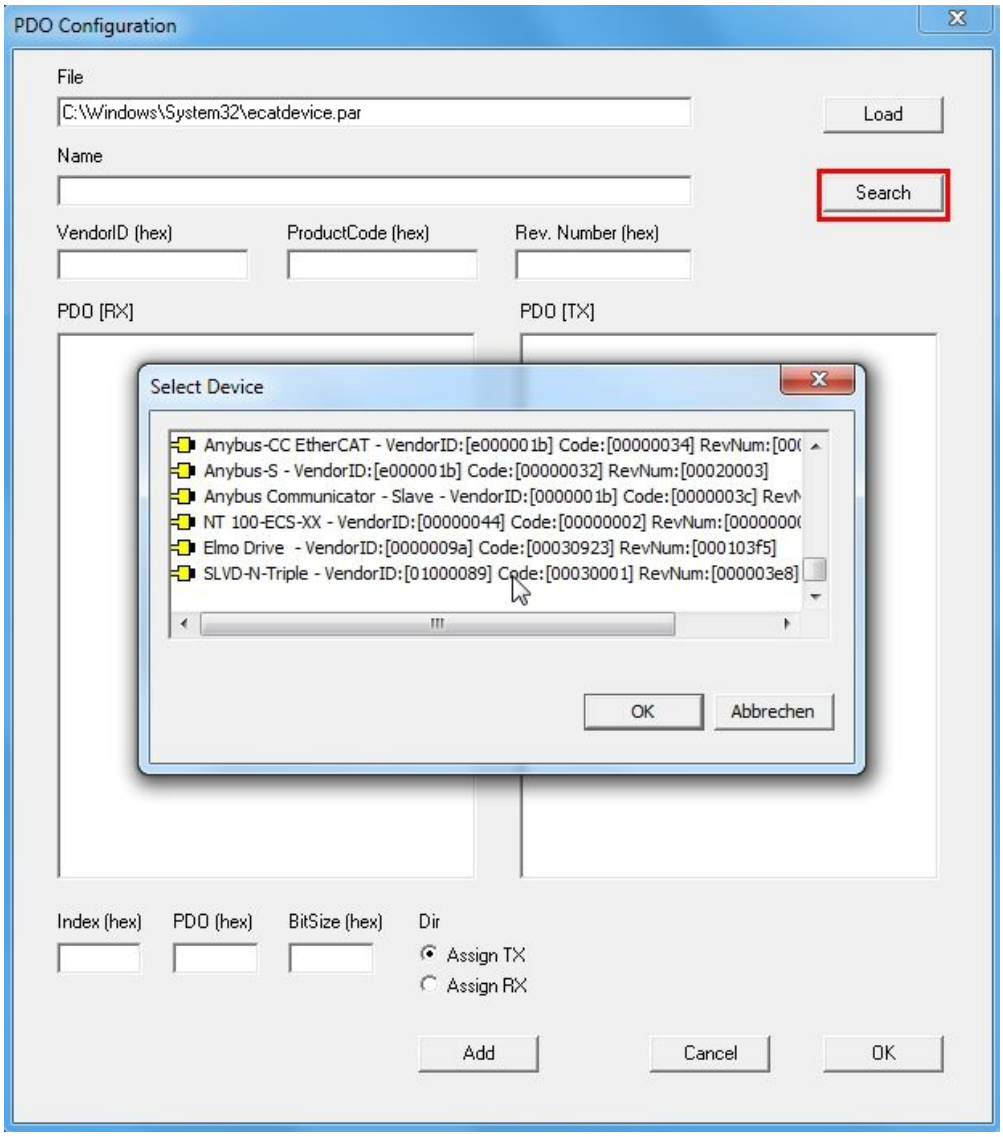
4.1 XML Converter

Since the X-GO EtherCAT Master uses a native parameter format, the XML (ESI) device information has to be converted. The tool ECATVERIFY has implemented a XML parser which allows converting XML device information into native parameter information and save it into the file ECATDEVICE.PAR (to be placed in the directory \WINDOWS\SYSTEM32). For conversion the XML files must be placed in the directory where ECATVERIFY is located. The device which is to be converted may be searched within an XML file by its Name, Product Code, Vendor ID or Revision Number. It is also possible to convert the whole XML file to the native format. Devices which are already present in ECATDEVICE.PAR will be updated.



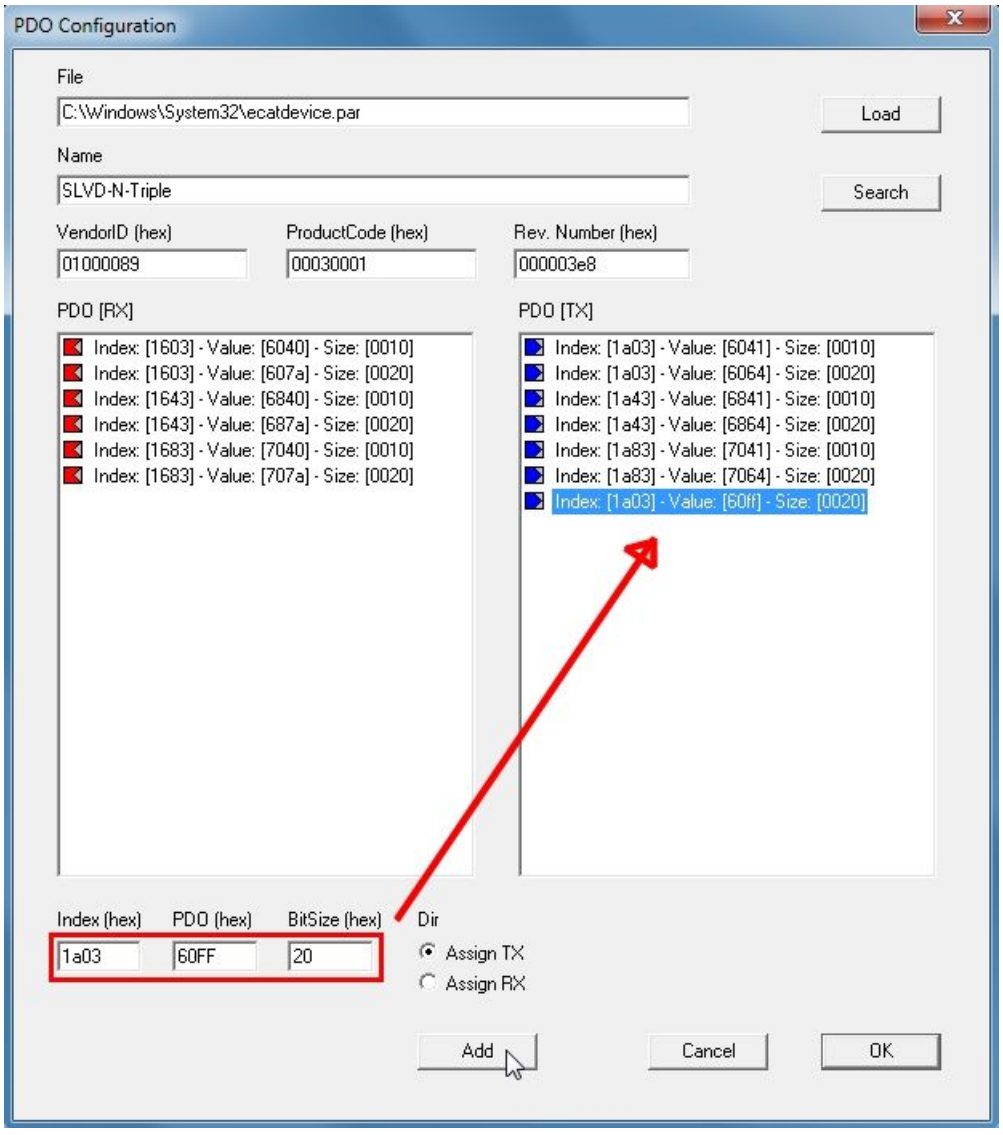
4.2 PDO Configurator

The integrated PDO configurator allows easy determination of the EtherCAT PDO mapping. The PDO Configurator allows adding, removing, and deleting PDO mapping objects. With the PDO-Configurator devices located in the file ECATDEVICE.PAR can be listed or searched for editing the PDO mappings.



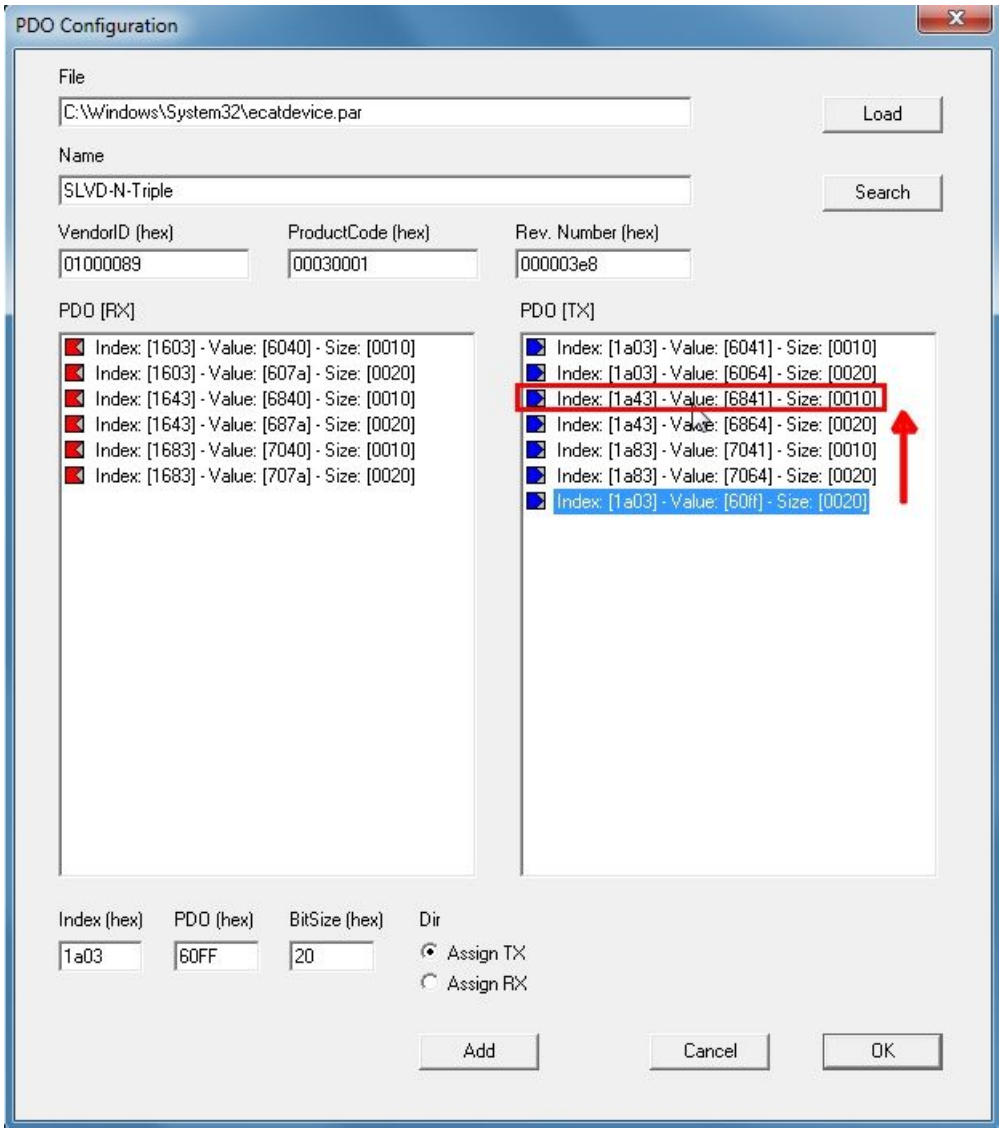
Note:
Existing PDO-Mappings need to have an already listed PDO assignment (1C12 / 1C13). Otherwise the PDO mapping has to setup newly.

New PDO mappings are entered by index, PDO and bit size for assigning it to the corresponding PDO mapping list (TX / RX).

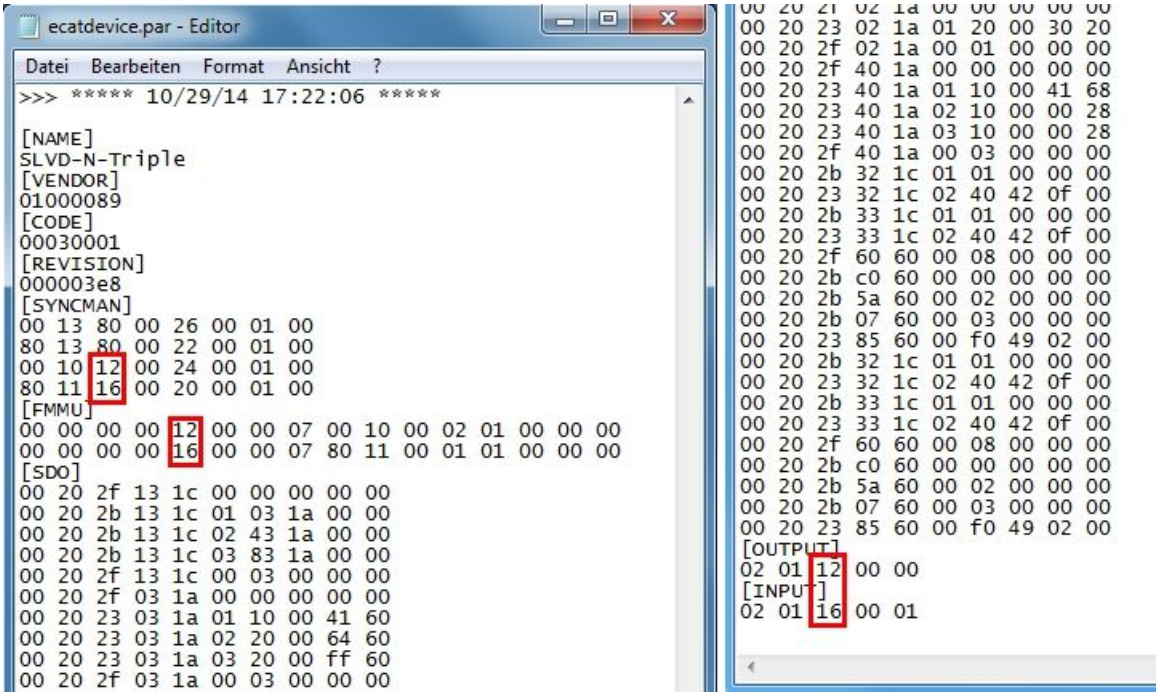


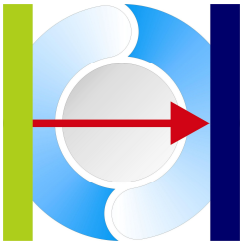
Selected PDO mappings may be deleted by pressing the key „DELETE“.

The new PDO mapping entries can be moved to the appropriate position. For this, the corresponding entry is selected to be moved and swapped with the entry of the desired position by clicking on it.



Once configured, the device located in the file ECATDEVICE.PAR file is automatically updated and the value “length” of the corresponding FMMU-, SYNCMAN- and INPUT / OUTPUT descriptor entries is automatically updated.



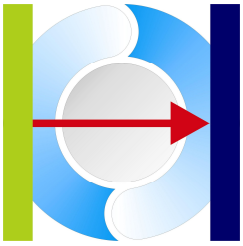


4.3 Device Parameter File

Usually device information is provided by a corresponding XML configuration file. Since the development of software with the EtherCAT Master Library has special needs for programming, the XML file must be parsed and translated into a native format. Therefore a configuration file named **ECATDEVICE.PAR** has to be generated with the tool **ECATVERIFY** and placed into the directory **\windows\system32**. The ECATDEVICE.PAR is a text based file with sections for Product Code, Name, SYNC Manager, FMMU Manager, SDO and Data Description. A new device description must start with the signature **“>>>”**

Sample:

```
>>> ***** 09/15/10 14:56:37 *****

[NAME]
EL3102
[VENDOR]
00000002
[CODE]
0c1e3052
[REVISION]
00100000
[SYNCMAN]
00 10 80 00 26 00 01 00
80 10 80 00 22 00 01 00
00 11 00 00 04 00 00 00
80 11 06 00 20 00 01 00
[FMMU]
00 00 00 00 06 00 00 07 80 11 00 01 01 00 00 00
00 00 00 00 01 00 00 00 0d 08 00 01 01 00 00 00
[SDO]
00 20 2f 13 1c 00 00 00 00 00
00 20 2b 13 1c 01 00 1a 00 00
00 20 2b 13 1c 02 01 1a 00 00
00 20 2f 13 1c 00 02 00 00 00
[OUTPUT]
[INPUT]
02 01 01 00 00
02 06 02 00 00
02 01 01 00 00
02 06 02 00 00
```



Note: With newer devices the configuration is stored inside the EEPROM. The EtherCAT Master Library is able to configure the devices by parsing the EEPROM information, even without XML file or Native file. But without using the configuration file, the configuration time increases by parsing EEPROM information. The Software ECATVERIFY parses XML information and EEPROM information and converts it into the native format and gives additional help for configuration.

4.3.1 Section [NAME]

This section contains the name of the device:

```
[NAME]
EL3102
```

4.3.2 Section [VENDOR]

This section contains the vendor ID of the device:

```
[VENDOR]
00000002
```

4.3.3 Section [CODE]

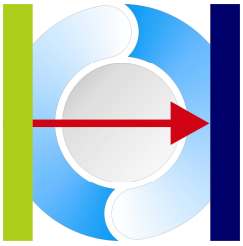
This section contains the product code of the device:

```
[CODE]
0C1E3052
```

4.3.4 Section [REVISION]

This section contains the revision number of the device:

```
[CODE]
00100000
```



4.3.5 Section [SYNCMAN]

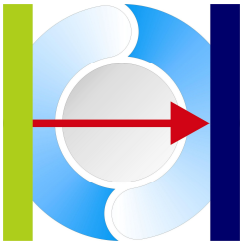
This section contains the binary data for the synchronisation manager of the device:

```
[SYNCMAN]
00 18 F6 00 26 00 01 00
F6 18 F6 00 22 00 01 00
00 10 00 00 24 00 00 00
00 11 06 00 20 00 01 00
```

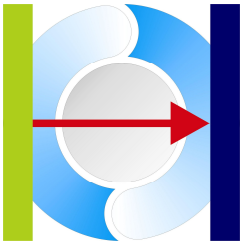
Meaning:

```
| Addr   |   Len   |Cntr    |ChEn
                     |Stat    |Res
|        |         |    |    |    |    |
 00   18   F6   00   26   00   01   00      <- SYNMAN0
 F6   18   F6   00   22   00   01   00      <- SYNMAN1
 00   10   00   00   24   00   00   00      <- SYNMAN2
 00   11   06   00   20   00   01   00      <- SYNMAN3
```

Parameter	relative address (offset)	Data type	Access type	Access type PDI	Value/description
Physical start address	0x0000	WORD	RW	R	
Length	0x0002	WORD	RW	R	
Buffer type	0x0004	Unsigned2	RW	R	0x00: buffered 0x02: mailbox
Direction	0x0004	Unsigned2	RW	R	0x00: area shall be read from the master 0x01: area shall be written by the master
reserved	0x0004	Unsigned1	RW	R	0x00
DLS-user event enable	0x0004	Unsigned1	RW	R	0x00: DLS-user event is not active 0x01: DLS-user event is active (when area was accessed and is no longer locked)
Watchdog enable	0x0004	Unsigned1	RW	R	0x00: watchdog disabled 0x01: watchdog enabled
reserved	0x0004	Unsigned1	RW	R	0x00
Write event	0x0005	Unsigned1	R	R	0x00: no write event 0x01: write event
Read event	0x0005	Unsigned1	R	R	0x00: no read event 0x01: read event



reserved	0x0005	unsigned1	R	R	0x00
Mailbox state	0x0005	Unsigned1	R	R	0x00: mailbox empty 0x01: mailbox full
Buffered state	0x0005	Unsigned2	R	R	0x00: first buffer 0x01: second buffer 0x02: third buffer 0x03: buffer locked
reserved	0x0005	Unsigned2	R	R	0x00
Channel enable	0x0006	Unsigned1	RW	R	0x00: channel disabled 0x01: channel enabled
Repeat	0x0006	Unsigned1	RW	R	
reserved	0x0006	Unsigned4	RW	R	0x00
DC Event 0 with Bus write	0x0006	Unsigned1	RW	R	0x00: no Event 0x01: DC Event if master writes complete buffer
DC Event 0 with local write	0x0006	Unsigned1	RW	R	0x00: no Event 0x01: DC Event if DL-user writes complete buffer
Channel enable PDI	0x0007	Unsigned1	R	RW	0x00: channel disabled 0x01: channel enabled
RepeatAck	0x0007	Unsigned1	R	RW	shall follow repeat after data recovery
reserved	0x0007	Unsigned6	R	RW	0x00



4.3.6 Section [FMMU]

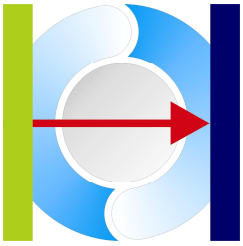
This section contains the binary data for the FMMU manager of the device:

```
[FMMU]
06 00 00 00 01 00 00 00 0D 08 00 01 01 00 00 00
00 00 00 00 06 00 00 07 00 11 00 01 01 00 00 00
```

Meaning:

```
| LogAddr(Offs) |  Len  |LogStartBit    |PhysStartBit
                          |LogEndBit      |RdWrEnable
                              |PhysAddr       |ChEnable
|               |       |   |   |       |   |   |               |
 00  00  00  00  01  00  00  00  0D  08  00  01  01  00  00  00 <- FMMU0
 00  00  00  00  06  00  00  07  00  11  00  01  01  00  00  00 <- FMMU1
```

Parameter	relative address (offset)	Data type	Access type	Access type PDI	Value/description
Logical start address	0x0000	DWORD	RW	R	
Length	0x0004	WORD	RW	R	
Logical start bit	0x0006	Unsigned3	RW	R	
reserved	0x0006	Unsigned5	RW	R	0x00
Logical end bit	0x0007	Unsigned3	RW	R	
reserved	0x0007	Unsigned5	RW	R	0x00
Physical start address	0x0008	WORD	RW	R	
Physical start bit	0x000A	Unsigned3	RW	R	
reserved	0x000A	Unsigned5	RW	R	0x00
Read enable	0x000B	Unsigned1	RW	R	0x00: entity will be ignored for read service 0x01: entity will be used for read service
Write enable	0x000B	Unsigned1	RW	R	0x00: entity will be ignored for write service 0x01: entity will be used for write service
reserved	0x000B	Unsigned6	RW	R	0x00
Enable	0x000C	Unsigned1	RW	R	0x00: entity not active 0x01: entity active
reserved	0x000C	Unsigned15	RW	R	0x0000
reserved	0x000E	WORD	RW	R	0x0000



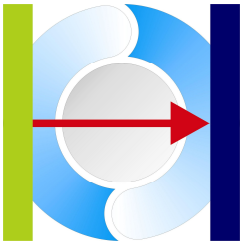
4.3.7 Section [SDO]

This section contains the binary SDO data of the device:

```
[SDO]
00 20 2F 12 1C 00 00 00 00 00
00 20 2F 13 1C 00 00 00 00 00
00 20 2B 13 1C 01 00 1A 00 00
00 20 2B 13 1C 02 01 1A 00 00
00 20 2F 13 1C 00 02 00 00 00
```

Meaning:

```
|NumServ|Cmd|Index   |SubIndex
                      |Data
|       |   |        |   |               |
 00   20   2F   12   1C   00   00   00   00   00 <- COE Cmd0
 00   20   2F   13   1C   00   00   00   00   00 <- COE Cmd1
 00   20   2B   13   1C   01   00   1A   00   00 <- COE Cmd2
 00   20   2B   13   1C   02   01   1A   00   00 <- COE Cmd3
 00   20   2F   13   1C   00   02   00   00   00 <- COE Cmd4
```

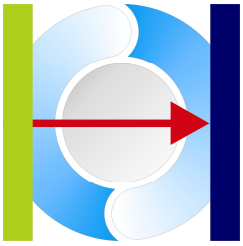


SDO Header Word and Command Byte

Frame part	Data Field	Data Type	Value/Description
CANopen Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x02: SDO Request
SDO	Size Indicator	Unsigned1	0x00: size of Data (1..4) unspecified 0x01: size of Data in Data Set Size specified
	Transfer Type	Unsigned1	0x01: Expedited transfer
	Data Set Size	Unsigned2	0x00: 4 Octet Data 0x01: 3 Octet Data 0x02: 2 Octet Data 0x03: 1 Octet Data
	Complete Access	Unsigned1	0x00
	Command	Unsigned3	0x01: Initiate Download Request

Sample:

COE Header 2000h : SDO Request
SDO Cmd 2Fh : Data in Data Set Size, exp. Transfer, 1 Oct. Data, Download Req.
Index 1C10h : Sync Manager 0 PDO Assignment (UNSIGNED16)
Index 1C11h : Sync Manager 1 PDO Assignment (UNSIGNED16)
Index 1C12h : Sync Manager 2 PDO Assignment (UNSIGNED16)
Index 1C13h : Sync Manager 3 PDO Assignment (UNSIGNED16)



4.3.8 Section [OUTPUT] / [INPUT]

This section contains the output/input data description of the device:

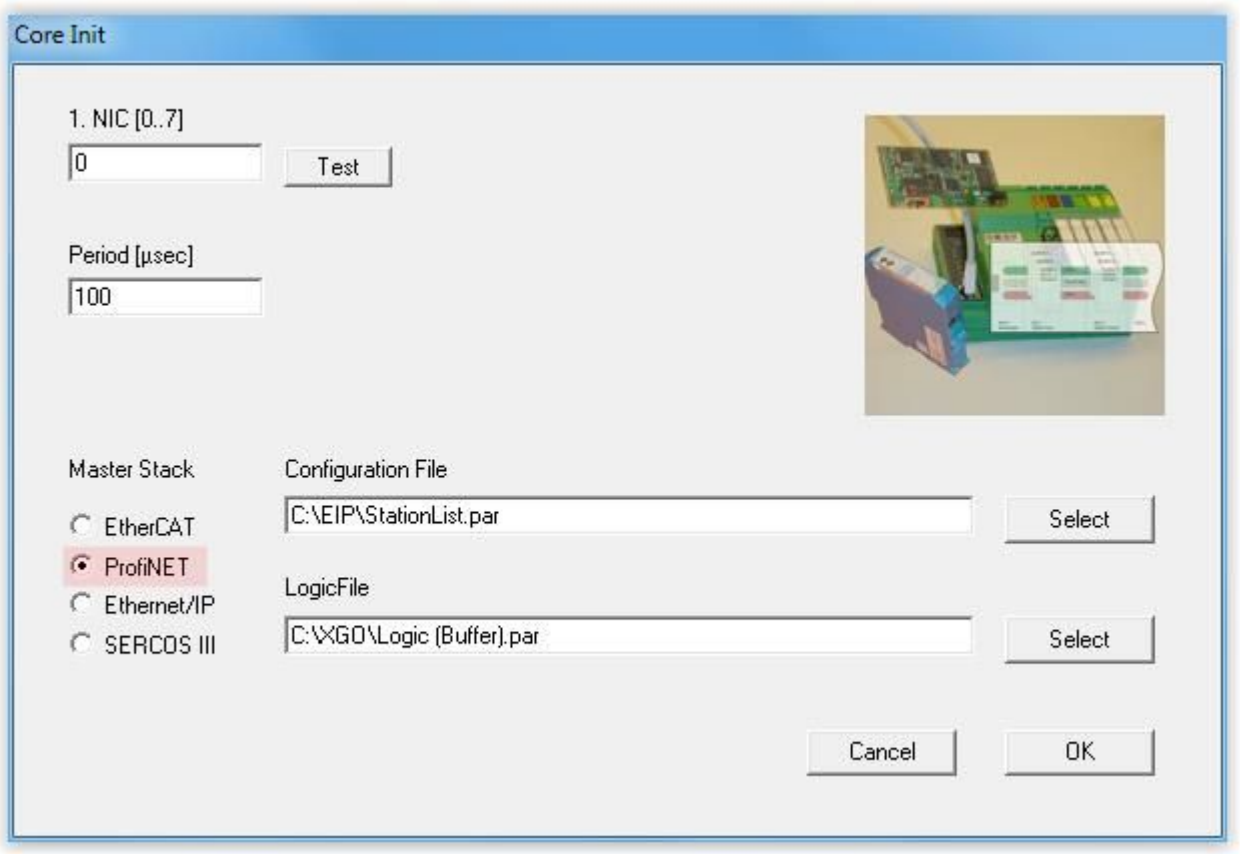
```
[OUTPUT]
01 01 01 00 00
02 02 02 00 00
01 01 01 00 00
02 02 02 00 00
03 02 02 00 00
03 02 02 00 00
```

Meaning:

```
The output data is based on 6 data items:

Item Type   (01 : DATA_ITEM_STATUS)
(02 : DATA_ITEM_VALUE)
(03 : DATA_ITEM_SCALE)

Data Type   (01 : DATA_TYPE_U8)
                  (02 : DATA_TYPE_U16)


Item Type
    Data Type
        Data Len
                FMMU Index
|   |   |       |   |
 01  01  01  00  00 <- Item 0
 02  02  02  00  00 <- Item 1
 01  01  01  00  00 <- Item 2
 02  02  02  00  00 <- Item 3
 03  02  02  00  00 <- Item 4
 03  02  02  00  00 <- Item 5
```

5 ProfiNET

When using X-GO ProfiNET following parameters have to be used:

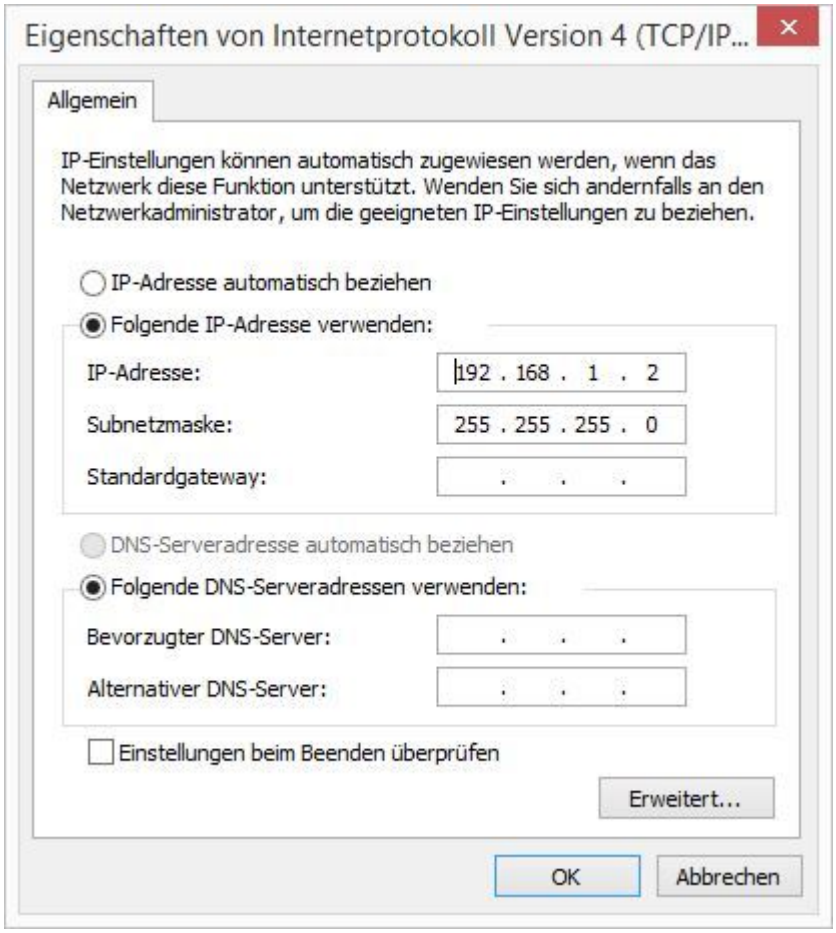


1.NIC Index of the first realtime Ethernet driver (typically 0)

Period Sampling period (typically 100 µsec)

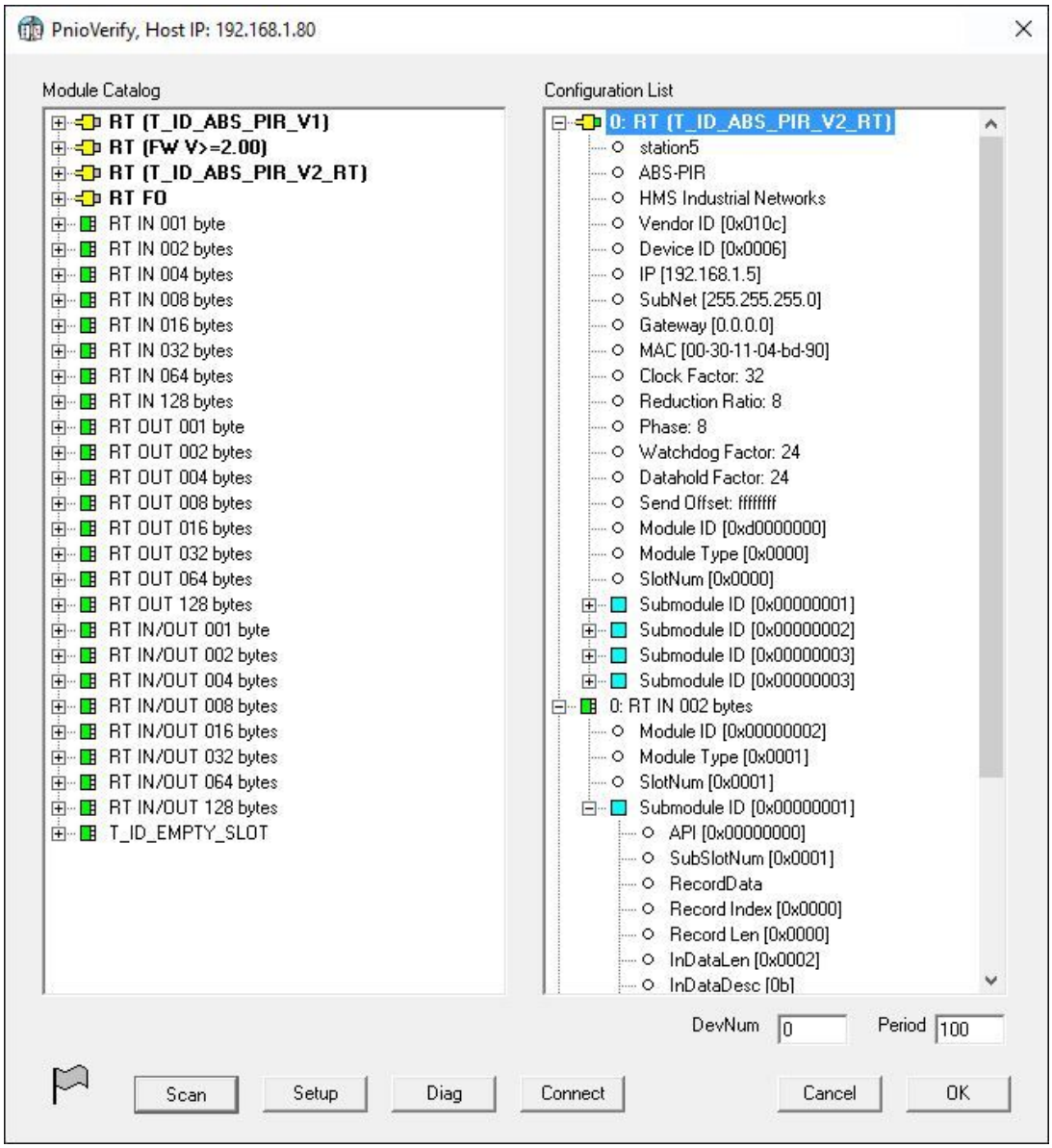
Configuration File Path to configuration file (e.g. STATIONLIST.PAR)

Note: ProfiNET requires the Host PC to have a valid IP-Address. So, check the netadapter setting.



5.1 Creating a configuration file

A ProfinetIO fieldbus system consists of several station devices (typically buscoupler devices). A station consists at least of one module (SLOT) and a module consists at least of one submodule (SUBSLOT). For proper operation the ProfinetIO devices needs first to be configured (by Station Name and IP) and a native STATIONLIST for operating the ProfiNET realtime library has to be created. Therefore SYBERA provides a program called PNIOVERIFY.EXE.



Note: Make shure a valid IP address is provided for the network connection.

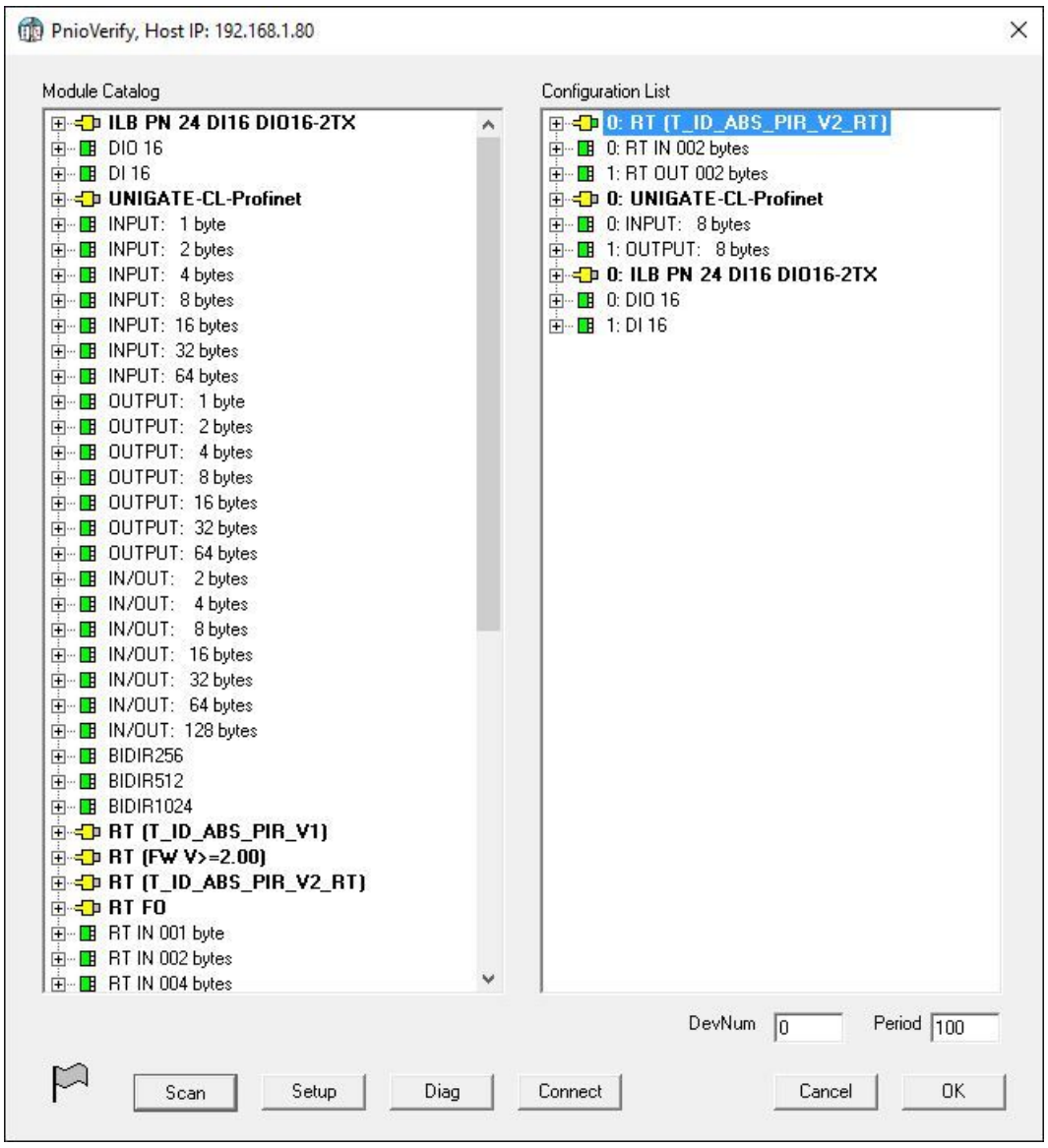
Note: If the application fails to run, check if the lastest Microsoft XML Parser has been installed. If not, install in the directory \APP\MSXML\MSXML6

PNIOVERIFY allows creating a native stationlist by selecting modules from a module catalog (leftside view). The catalog get its entries by the provides GSDML files which must be present in the same directory as PNIOVERIFY. A module is inserted to the station list configuration (rightside view) by a DRAG and DROP operation (just drag a module from the catalog to the station list configuration). There are two types of modules:

	Accesspoint Module	(SLOT 0)
	Functional Module	(SLOT 1 .. n)

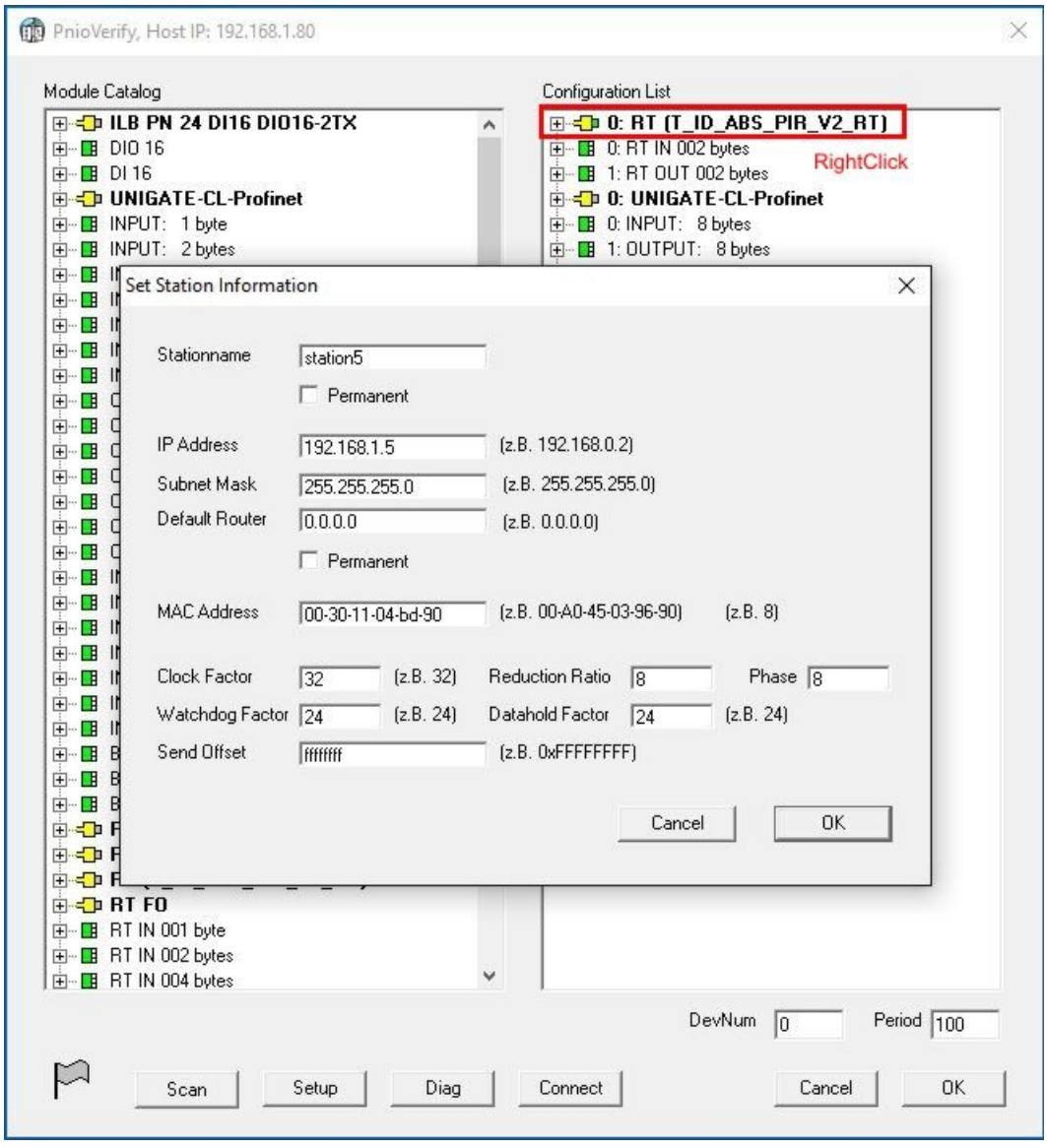
5.2 Accesspoint Module

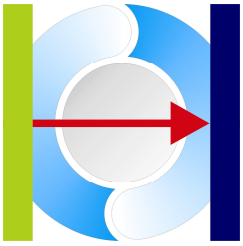
The accesspoint module keeps all information required for connecting to the fieldbus, as station name, IP parameters, MAC address, timing parameters. Therefore first task is to collect information about the ProfinetIO configuration by scanning the bus.



5.3 Station Settings

The scan gets information about manufacturer name and MAC address. Now individual assignment must set (e.g. IP address, station name, timings). On a right button click at the accesspoint module a dialog appears, which allows setting of station name , IP and timing parameters.





The timing settings of each station are based on a clock unit of 31,25 µsec. The synchronisation period is calculated as follow:

SyncTime = 31,25 µsec * ClockFactor * ReductionRatio

(e.g. 31,25 µsec * 32 * 8 = 8000 µsec = 8 msec)

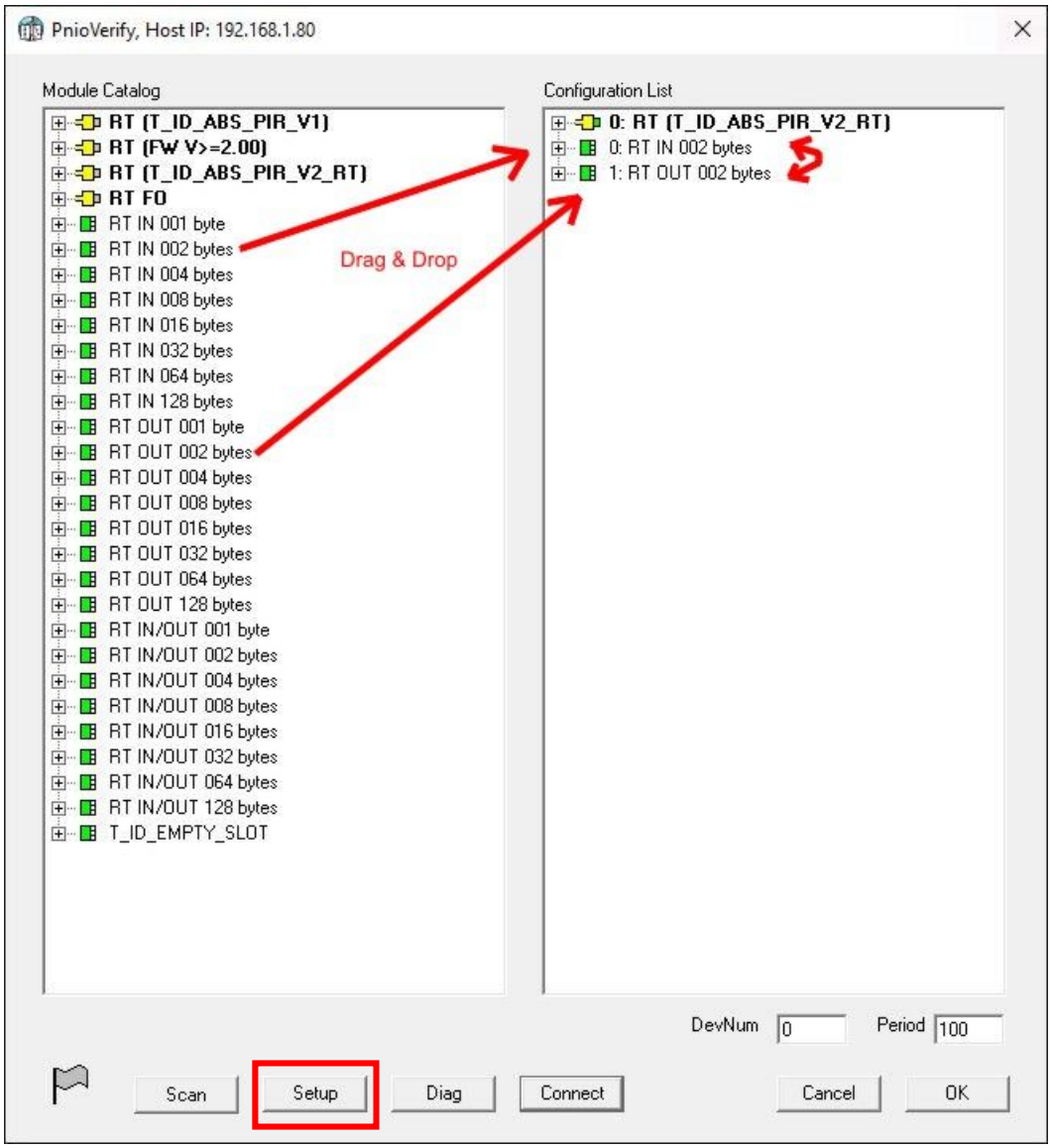
WatchdogTime = SyncTime * WatchdogFactor

(e.g. 8 msec * 24 = 192 msec)

The SendOffset must be set to 0xFFFFFFFF

5.4 Functional Module

Each station typically consists of multiple functional modules (SLOT 1..n). Function Modules have to be inserted from the catalog by DRAG and DROP operations. As well the nmodules may be sorted below the AccessPoint. A station configuration should contain all functional modules (in the order these modules are physically connected). When inserting a new module from the catalog, after dropping, it appears at the end of the configuration list and may be pushed to the correct slot location.



When the settings are done, the station my be initialized by pressing the button [Setup]

The resulting stationlist is stored to a choosen text file (sample):

Sample:

> Station

[NAME]
station8
[MFG_NAME]
ABS-PIR
[VENDOR_NAME]
HMS Industrial Networks
[STATION_ID]
0c 01 06 00 01 00
[MAC_ADDR]
00 30 11 04 bd 90
[IP_PARAMS]
c0 a8 01 08 ff ff ff 00 00 00 00 00
[TI_PARAMS]
20 00 08 00 08 00 18 00 18 00 ff ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00

>> Module

[NAME]
T_ID_ABS_PIR_V2_RT
[MOD_ID]
00 00 00 d0
[MOD_TYPE]
00 00
[SLOT_NUM]
00 00

>>> Submodule

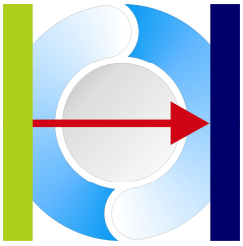
[SUBMOD_ID]
01 00 00 00
[SUBSLOT_NUM]
01 00
[OBJ_INPUT]
01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 01 00 00 00 01 00 00 00

>>> Submodule

[SUBMOD_ID]
02 00 00 00
[SUBSLOT_NUM]
00 80
[OBJ_INPUT]
01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 01 00 00 00 01 00 00 00

>>> Submodule

[SUBMOD_ID]



03 00 00 00
[SUBSLOT_NUM]
01 80
[OBJ_INPUT]
01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 01 00 00 00 01 00 00 00

>>> Submodule

[SUBMOD_ID]
03 00 00 00
[SUBSLOT_NUM]
02 80
[OBJ_INPUT]
01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 01 00 00 00 01 00 00 00

>> Module

[NAME]
T_ID_RT_IN2
[MOD_ID]
02 00 00 00
[MOD_TYPE]
01 00
[SLOT_NUM]
01 00

>>> Submodule

[SUBMOD_ID]
01 00 00 00
[SUBSLOT_NUM]
01 00
[OBJ_INPUT]
01 0b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02 00 00 00 01 00 00 00 01 00 00 00

>> Module

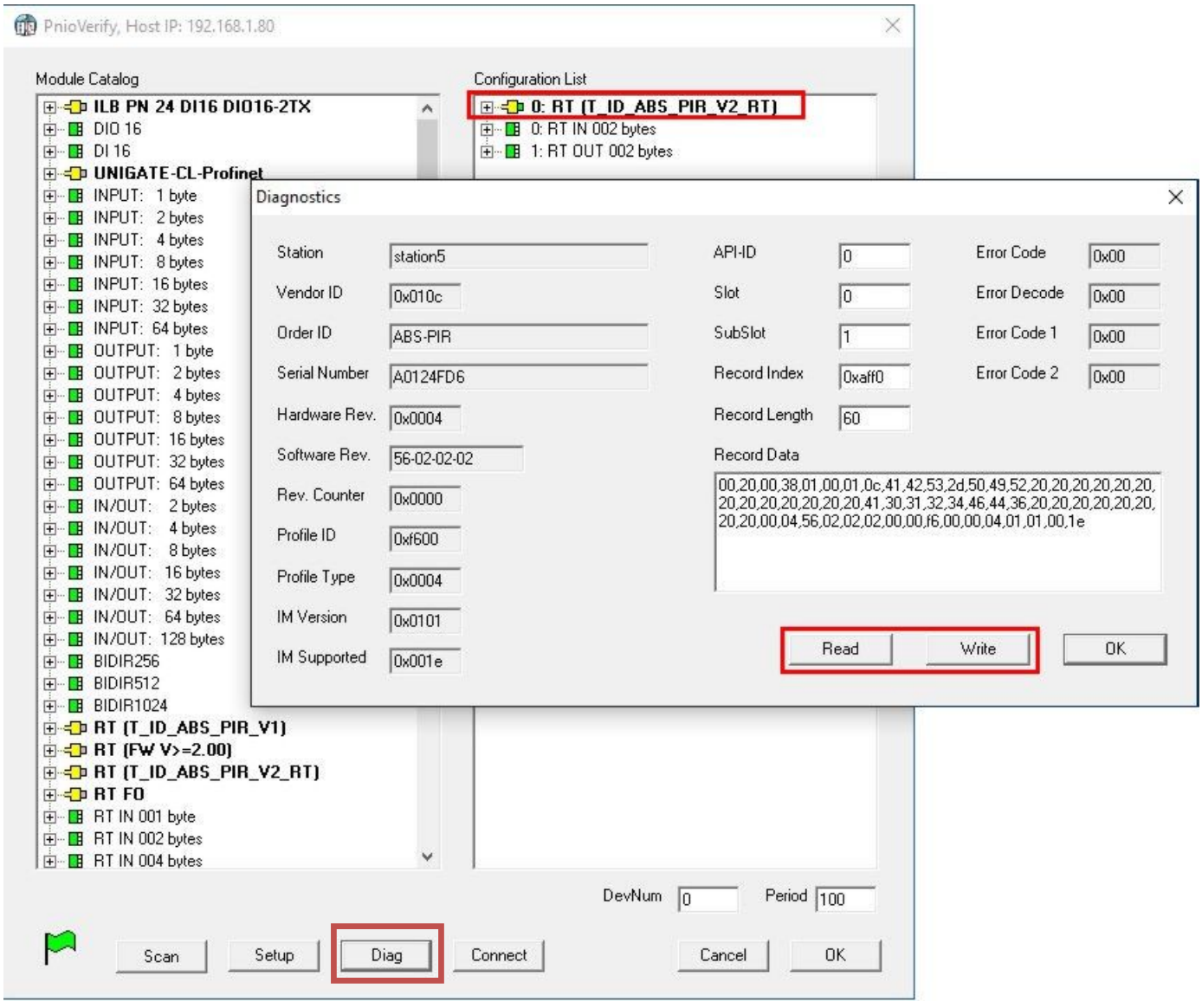
[NAME]
T_ID_RT_OUT2
[MOD_ID]
20 00 00 00
[MOD_TYPE]
01 00
[SLOT_NUM]
02 00

>>> Submodule

[SUBMOD_ID]
01 00 00 00
[SUBSLOT_NUM]
01 00
[OBJ_OUTPUT]
01 0b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02 00 00 00 01 00 00 00 01 00 00 00

5.5 Station Diagnostics

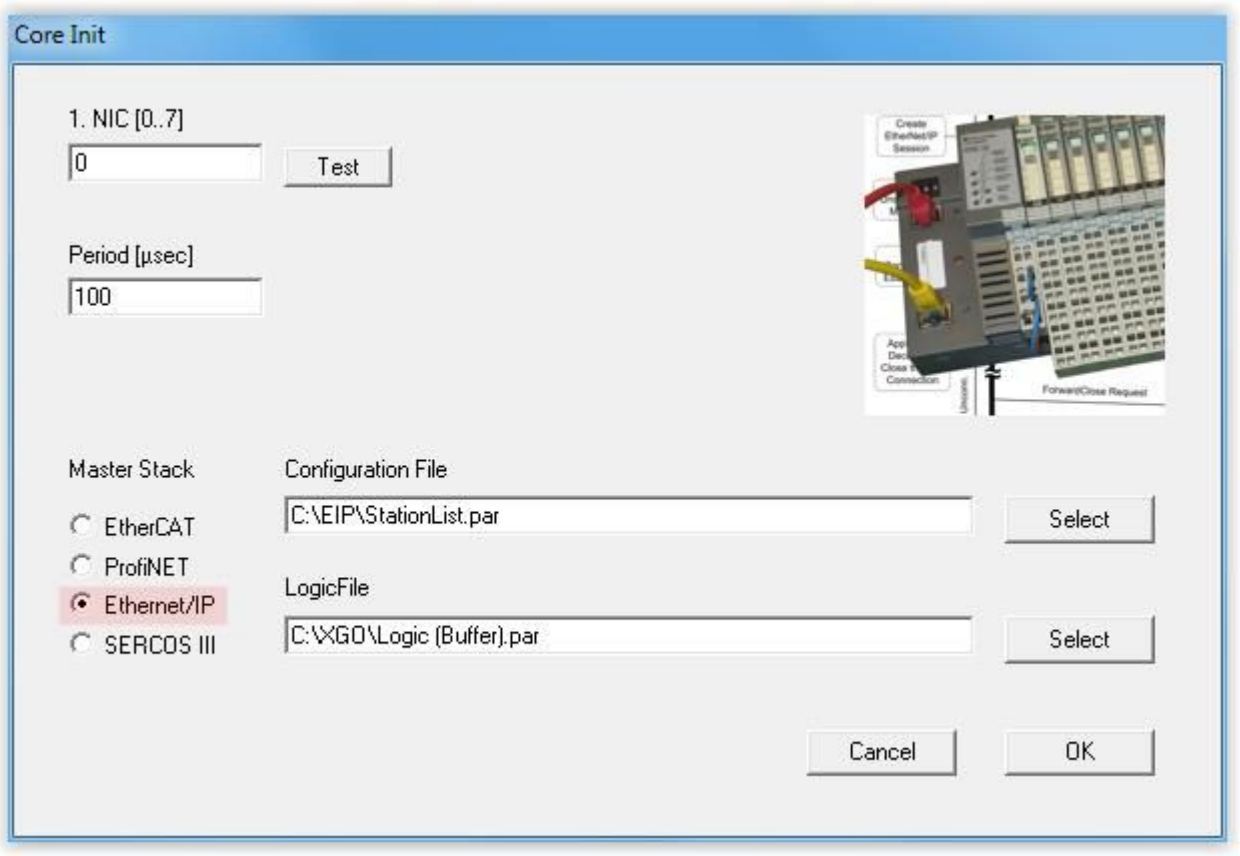
The station diagnostics allows gathering of I&M data, as well as reading and writing acyclic information (by API, SlotNum, SubSlotNum and RecordIndex). Therefore select an AccessPoint and press button [Diag].



To read or write acyclic information, put in the API-ID, Slot, SubSlot and Record Index. If the function fails, you'll get the corresponding PNIO error code.

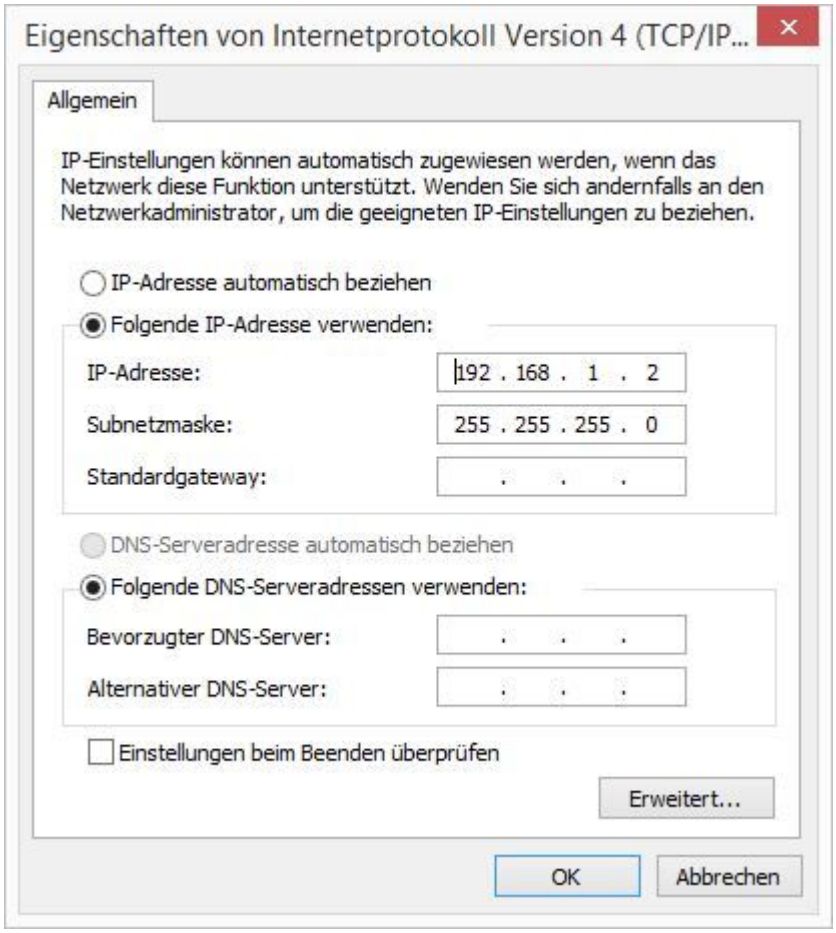
6 Ethernet/IP

When using X-GO SERCOS III following parameters have to be used:



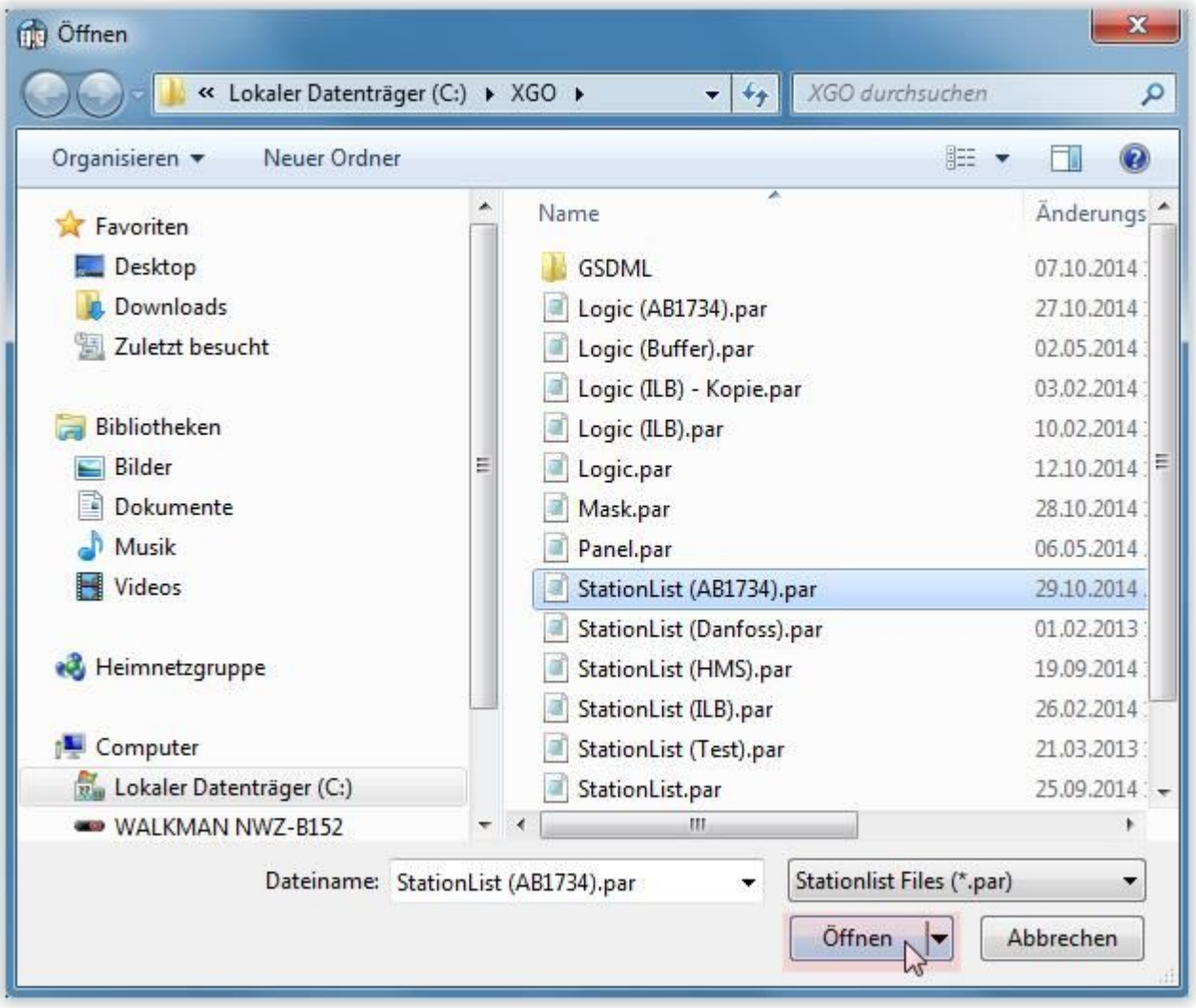
1.NIC	Index of the first realtime Ethernet driver (typically 0)
Period	Sampling period (typically 100 µsec)
Configuration File	Path to configuration file (e.g. STATIONLIST.PAR)

Note: ProfiNET requires the Host PC to have a valid IP-Address. So, check the ethnet adapter setting.



6.1 Creating a configuration file

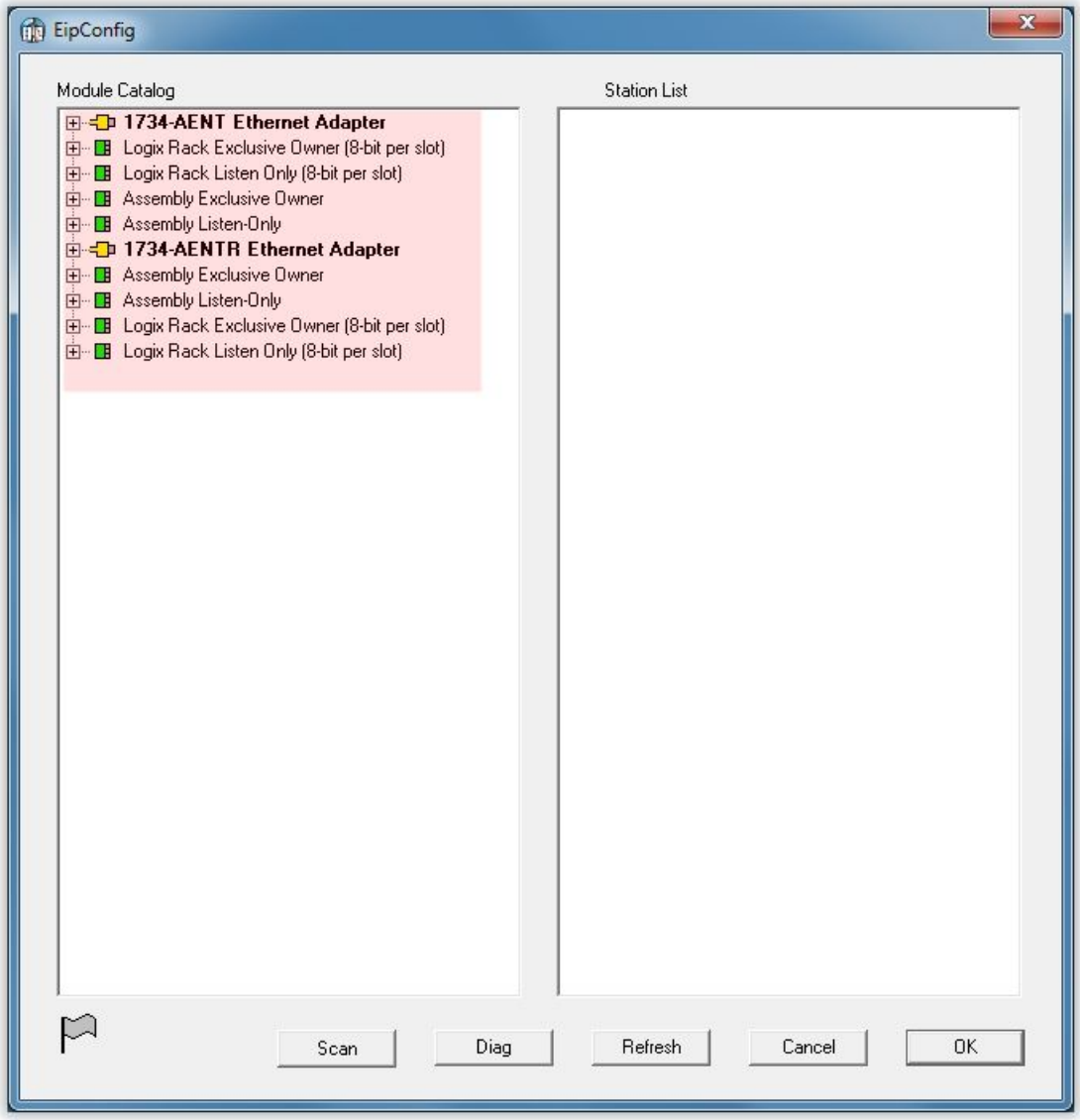
A Ethernet/IP fieldbus configuration consists typically of several station devices. A station device owns at least one connection. For proper operation the Ethernet/IP device needs first to be configured and a configuration file must be created. Therefore SYBERA provides a program called EIPCONFIG.EXE. When starting the program you will be asked for a already existing configuration file. If a file is present, choose it, otherwise just cancel the dialog.



Note: If the program fails to run, check if the latest Microsoft XML Parser has been installed. If not, install in the directory \APP\MSXML\MSXML6

X-GO Flex Logic Control Documentation

EIPCONFIG allows creating a configuration file by selecting devices and corresponding connections from a module catalogue (left side view). The catalogue gets its entries by the provided ESD files. A module is inserted to the configuration (right side view) by a DRAG and DROP operation (just drag a module from the catalogue to the station list configuration). There are two types of modules:

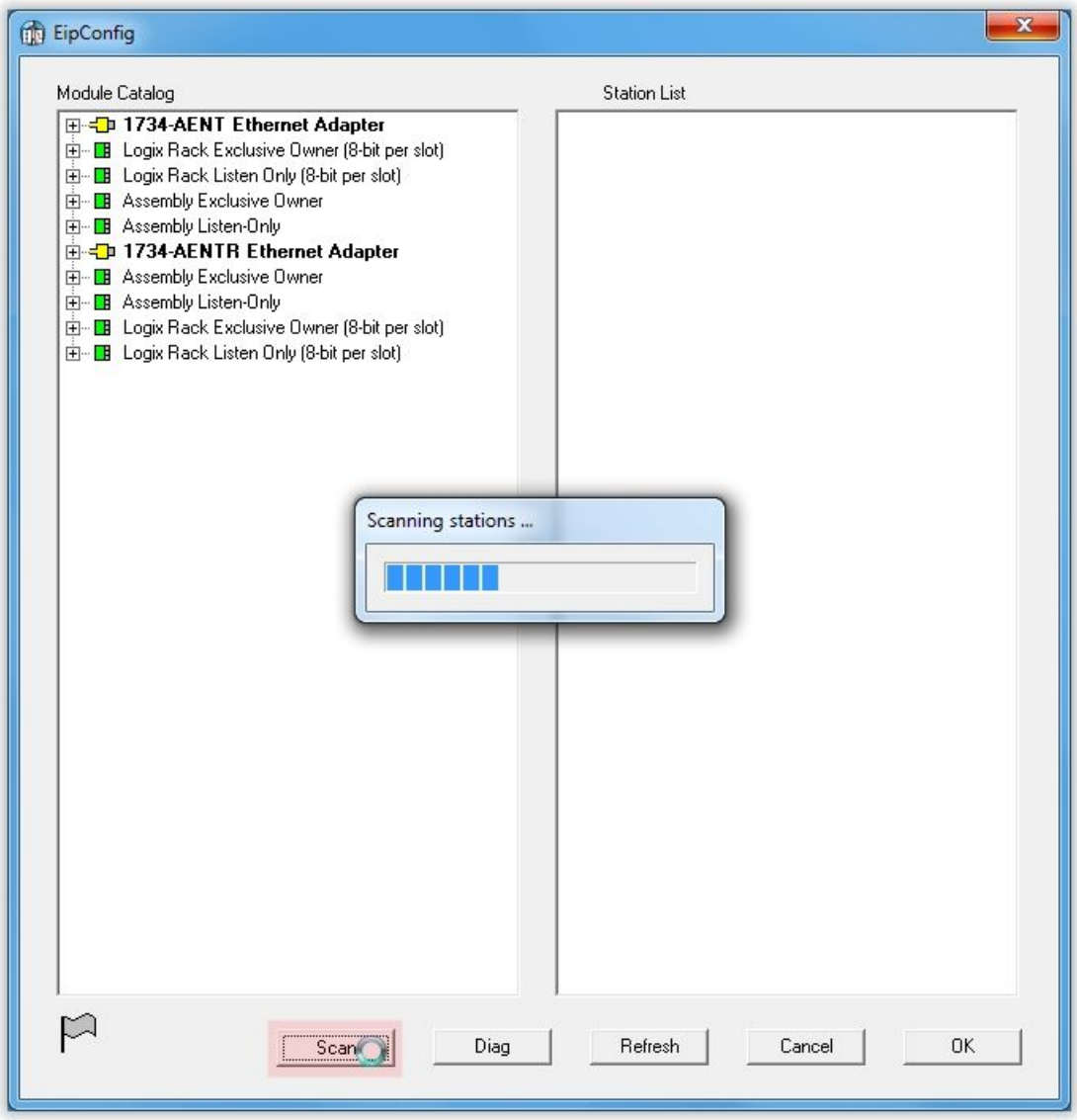



dule


Module

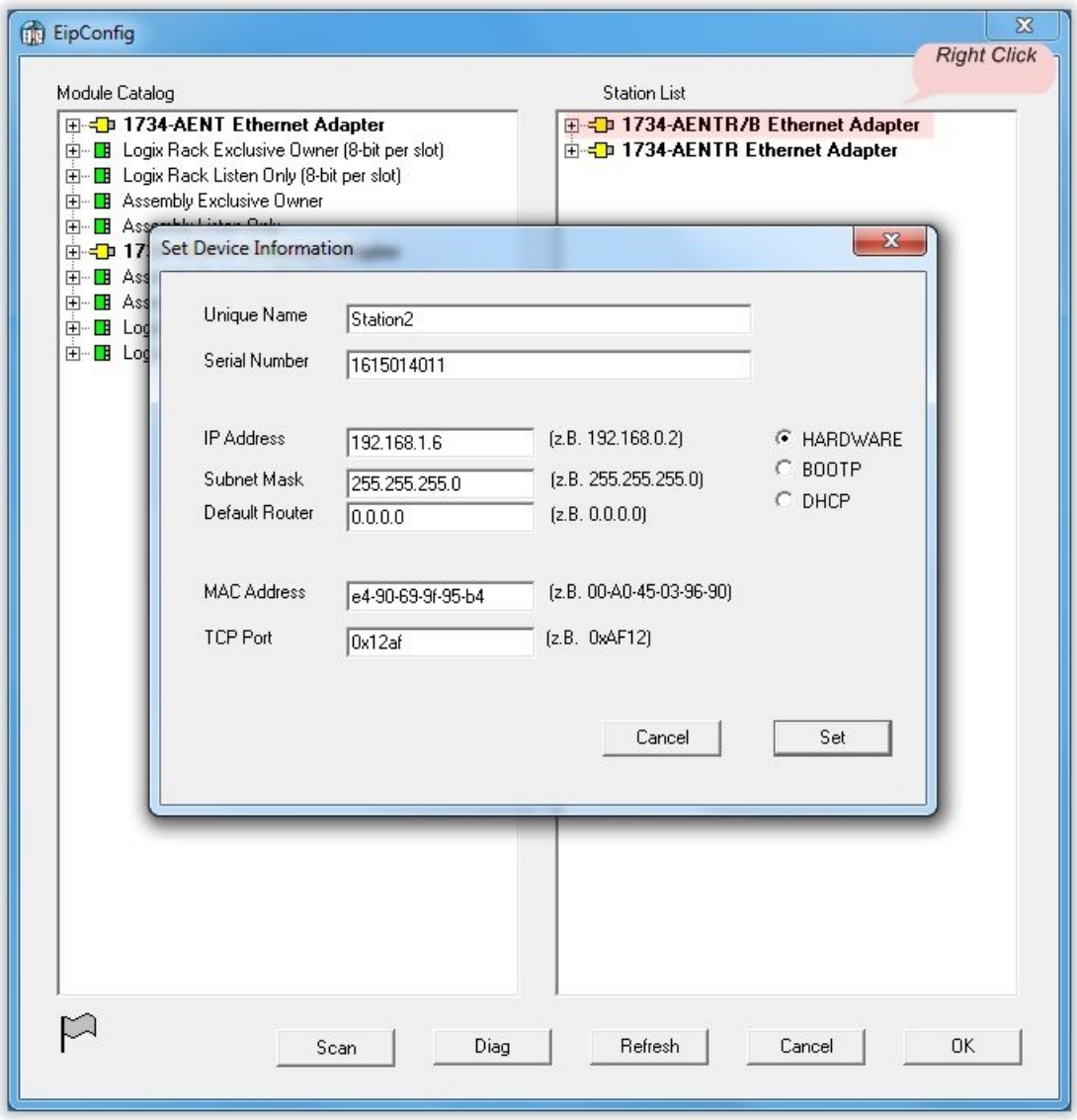
6.1.1 Device Module

The device module keeps all information required for identifying to the fieldbus, such as station name, product code IP parameters, MAC address. Therefore, the first task is to collect information about the available devices by scanning the bus.



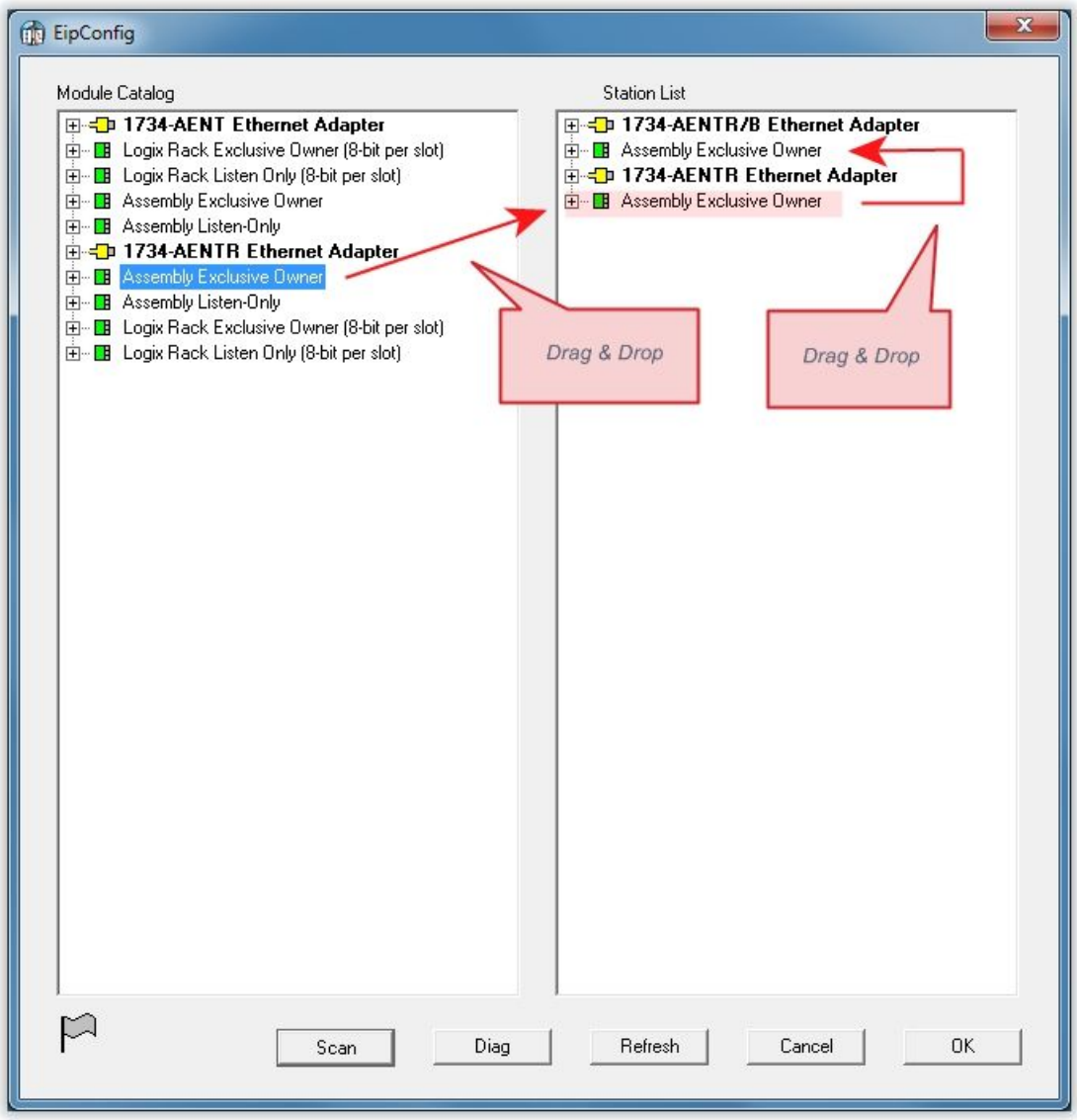
6.1.2 Device Settings

The scan gets all information to identify the device. Now individual assignment must set (e.g. IP address and station name). On a right button click at the device module the “Set Device Information” dialog appears.



6.1.3 Connection Module

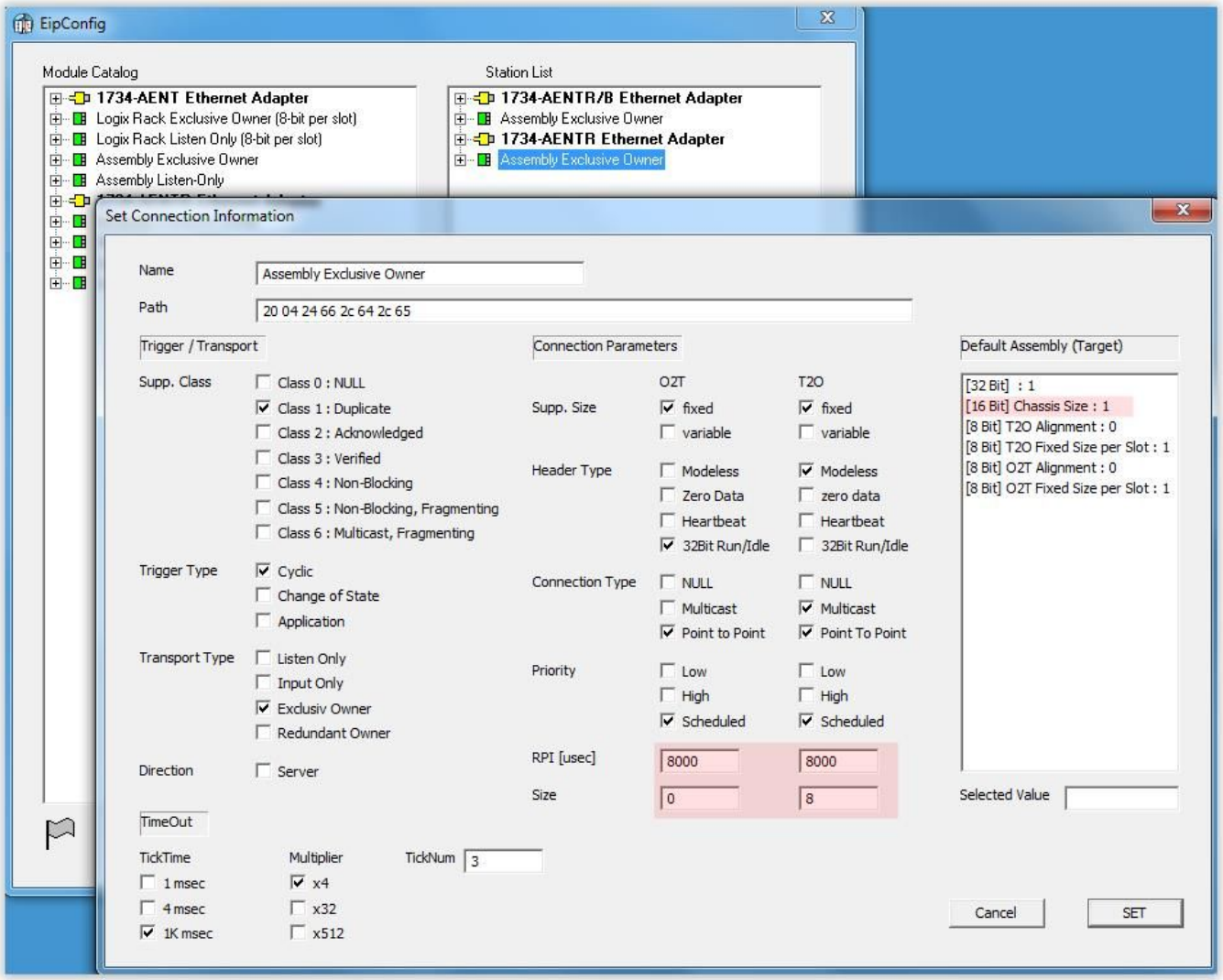
Each device typically owns at least one connection module. Connection modules have to be inserted from the catalogue by DRAG and DROP operations. A station configuration must contain all functional modules in the correct order (the modules will be addressed in this order). When inserting a module from the catalogue, after dropping, it appears at the end of the configuration list and must be pushed to the correct location.



6.1.4 Connection Settings

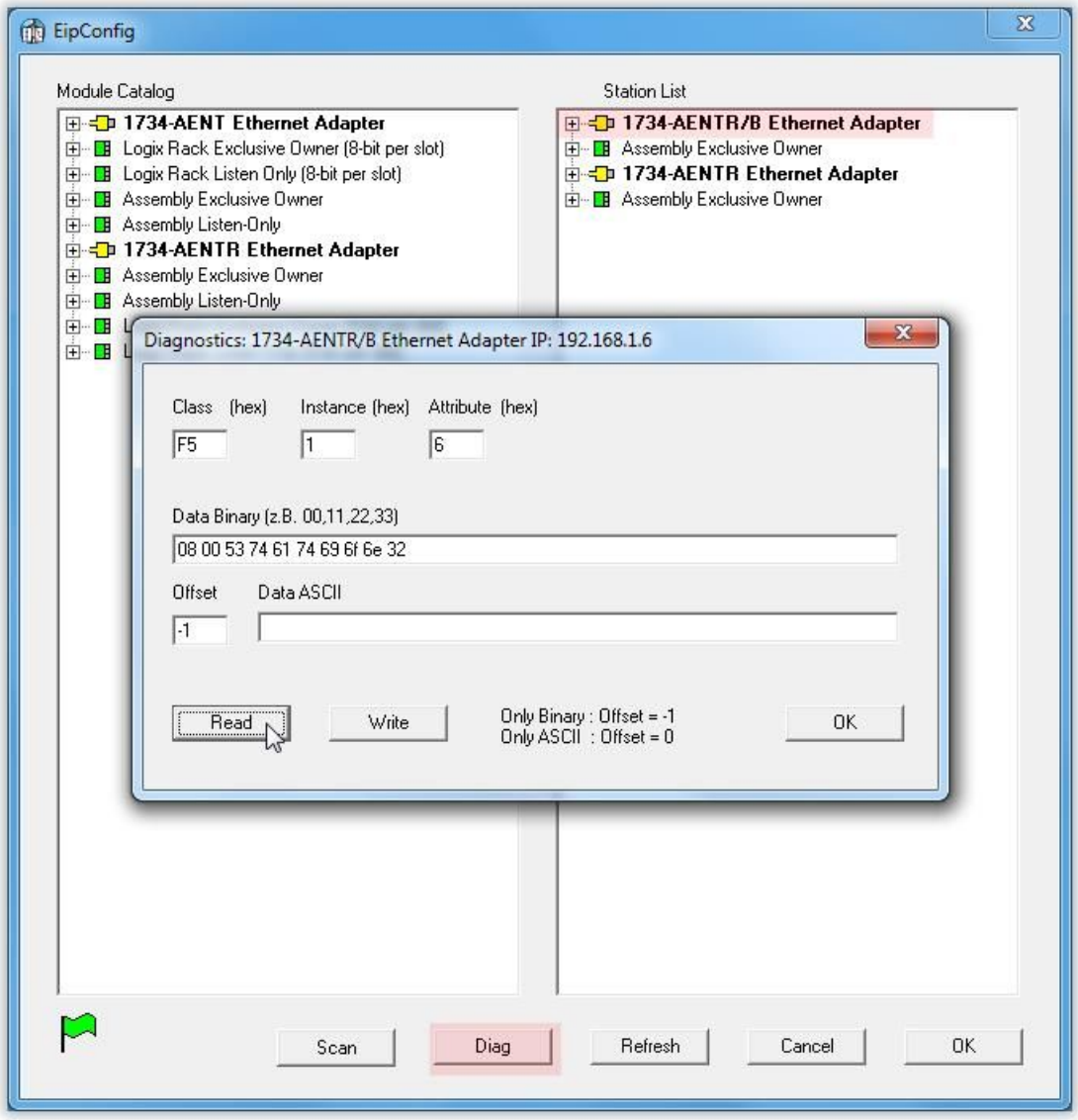
On a right button click at the device module the “Set Connection Information” dialog appears. Within the connection typically 3 elements need to be adjusted:

- RPI (Requested Packet Interval)
- Payload Size
- Chassis Size

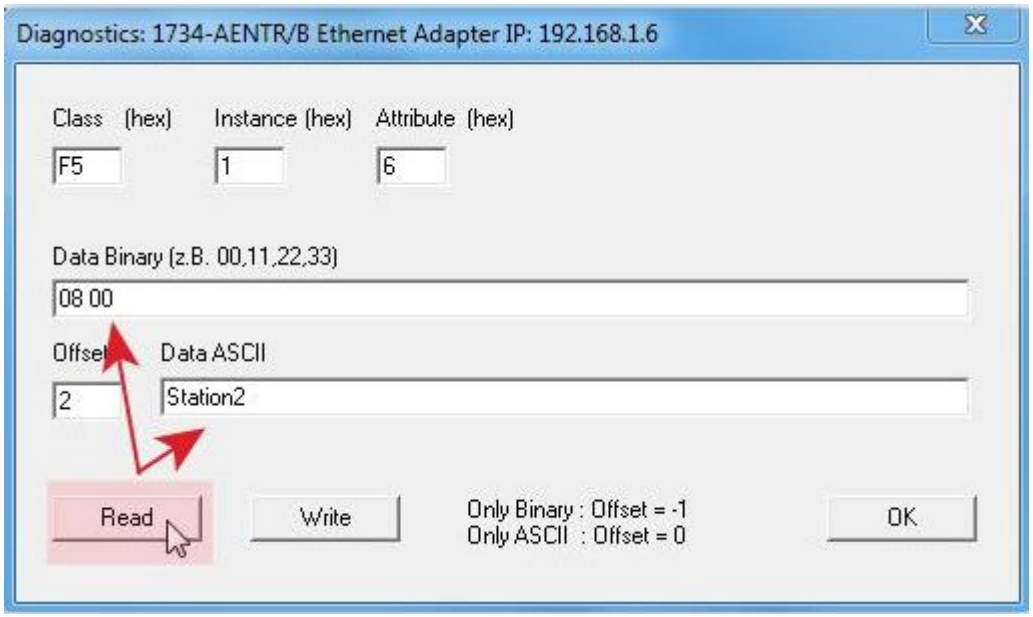


6.1.5 Device Diagnostics

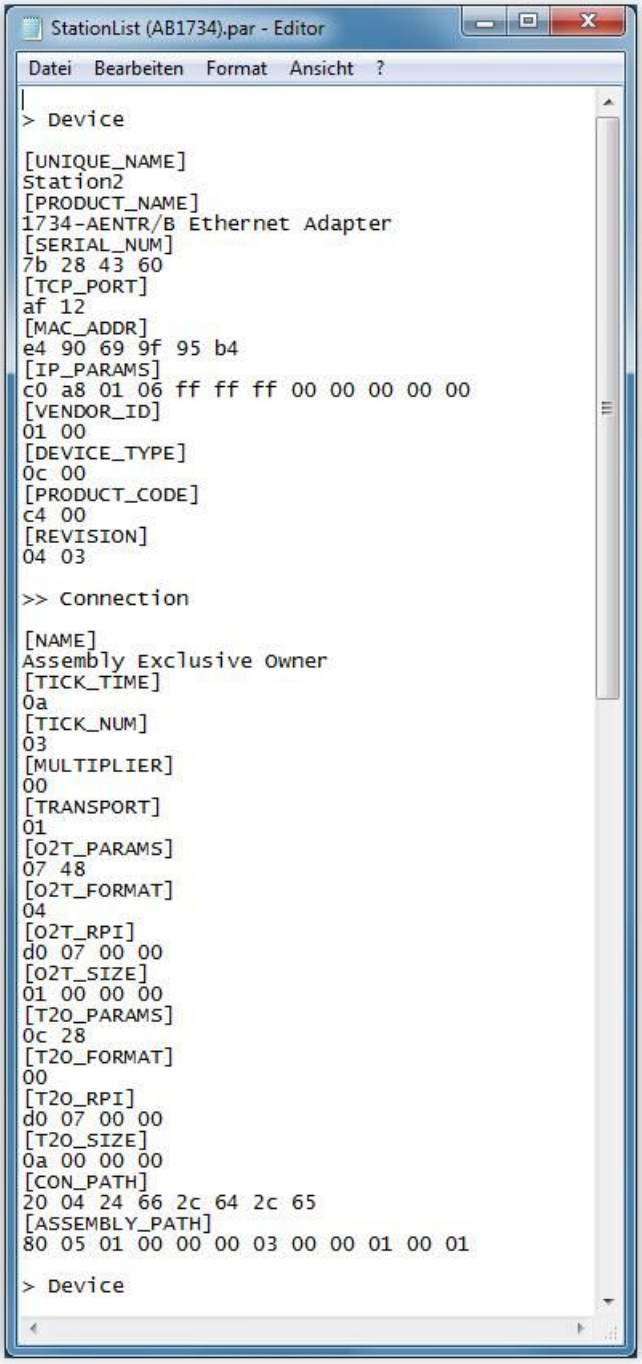
The device diagnostics allows gathering all available attributes. Therefore, select a device within the configuration and press button [DIAG]. Now, the attribute information can be read or write to the device by its class ID, instance ID and attribute ID. For instance, reading or writing the device name has class ID : 0xF5, instance ID: 0x01 and attribute ID: 0x06 (see Ethernet/IP specification).



Reading of attributes will be shown as binary data. Some attributes consist of a structured data. For instance string data consist typically of 2 bytes size trailing information and the ASCII chars following. Therefore an offset can be set, to separate these structure elements.

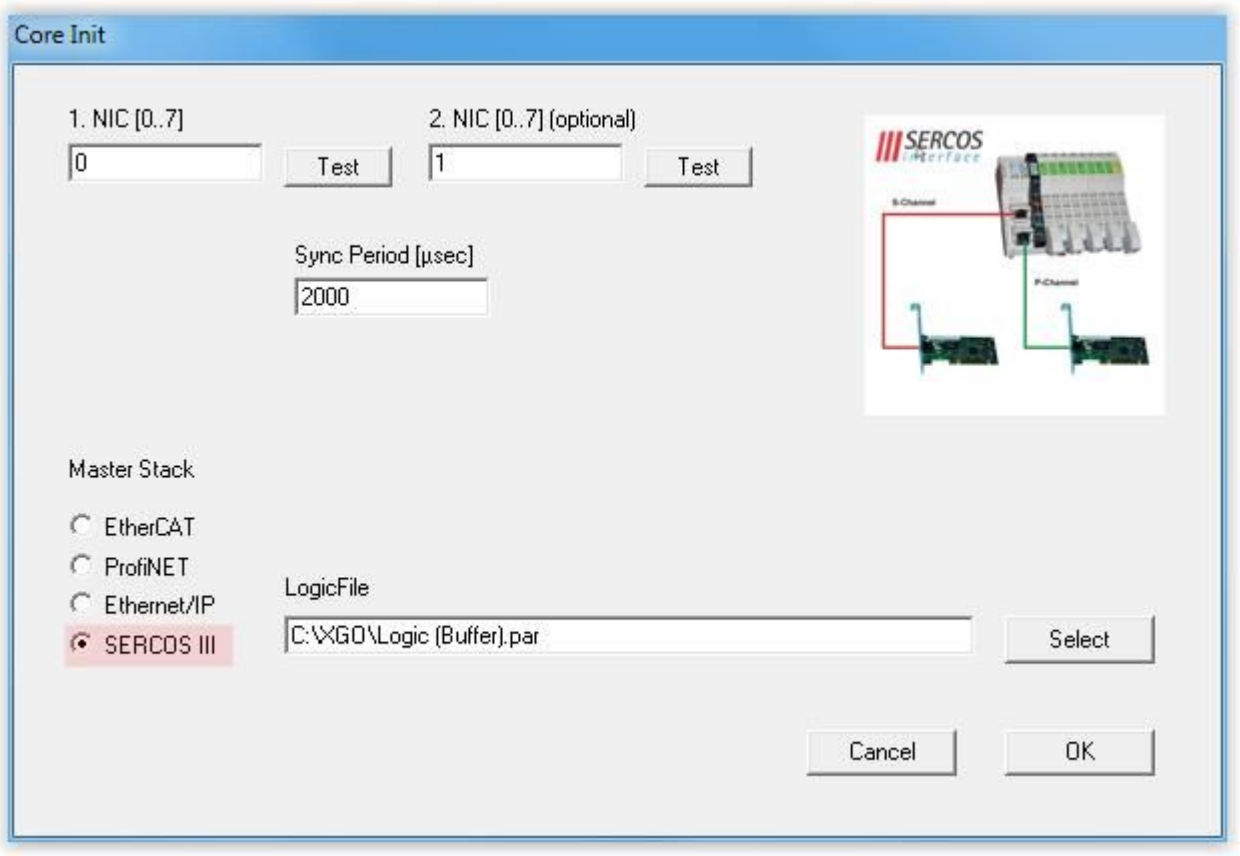


When pressing the button [OK] the configuration file will be saved. The resulting configuration is stored in a text file which must be provided to the Ethernet/IP Master for operation.



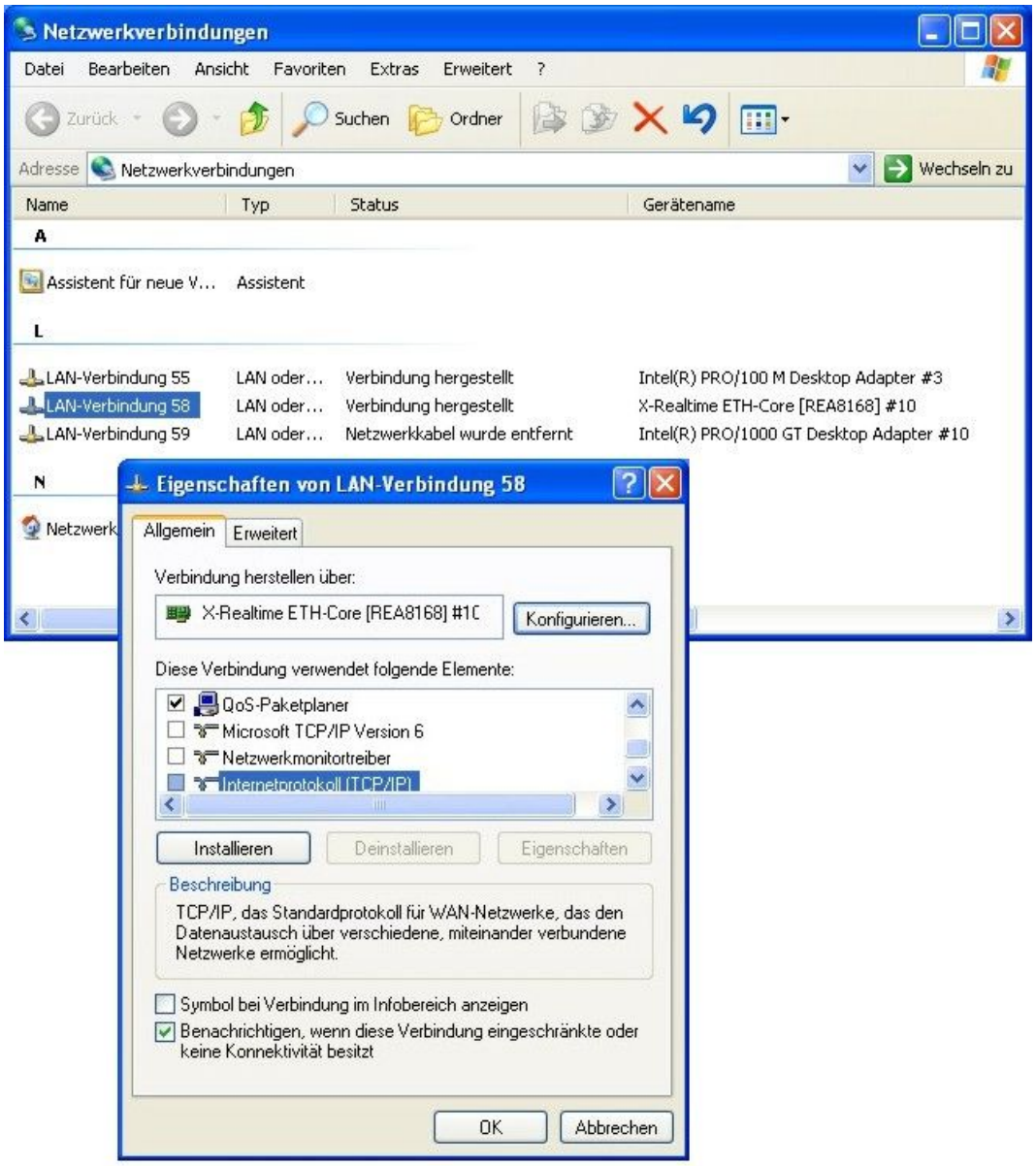
7 SERCOS III

When using X-GO SERCOS III following parameters have to be used:



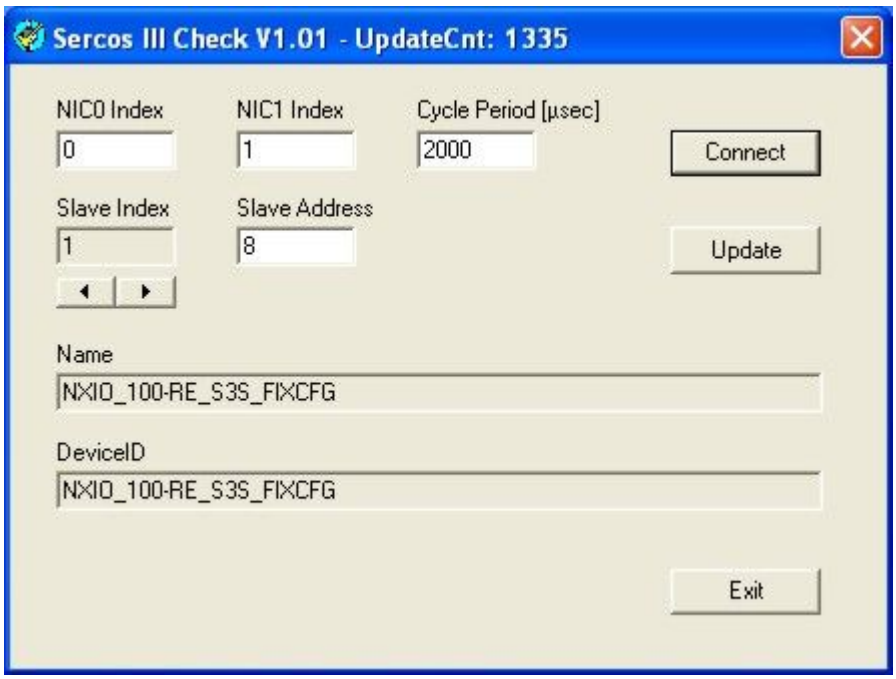
1.NIC	Index of the first realtime Ethernet driver (typically 0)
2.NIC	Index of the second realtime Ethernet driver (typically 1)
Sync Period	Synchronisation (Update) Period (typically 2000 µsec)

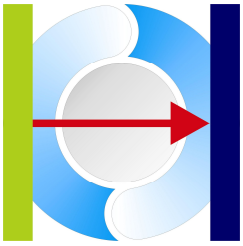
Note: For proper operation of Sercos III its recommended to use the SERCOS III network as a standalone network. This requires to turn off the Windows protocols for this network connection:



7.1 Sercos III Check

The tool SC3CHECK allows the adjustment of Addresses for Sercos III modules



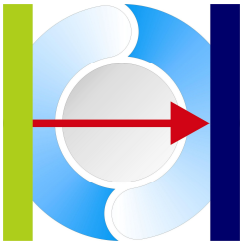


7.2 Configuration Parameter File

Usually device information is provided by a corresponding XML configuration file. Since the development of software with the SERCOS III Master Library has special needs for programming, the XML file must be parsed and translated into a native format. Therefore the SERCOS III Master Library provides a configuration file called **SC3CONFIG.PAR**, which is located in the directory **\windows\system32** after installation. The SC3CONFIG.PAR is a text based file with sections for Device Name, Device ID, configuration parameters and data description. All sections of a device must follow the item order:

NAME -> DEVICEID -> PARAM -> CONFIG -> DESC

```
[NAME]
HCS01.1E-W0003-A-02-E-S3-EC-NN-NN-NN-FW
[DEVICEID]
FWA-INDRV*-MPE-16
[PARAM]
S-0-32.0.0 (02 00)
[CONFIG]
S-0-0134.0.0  <0,0>
S-0-0036.0.0  <0,0>
S-0-0047.0.0  <0,0>
S-0-0135.0.0  <1,1>
S-0-0040.0.0  <1,1>
S-0-0386.0.0  <1,1>
[DESC]
00 00 01 01 02 00
00 00 04 02 02 00
00 00 04 03 04 00
00 00 04 03 04 00
00 01 01 01 02 00
00 01 04 02 02 00
00 01 04 03 04 00
00 01 04 03 04 00
```



7.2.1 Section [NAME]

This section contains the name of the device:

```
[NAME]
HCS01.1E-W0003-A-02-E-S3-EC-NN-NN-NN-FW
```

7.2.2 Section [DEVICEID]

This section contains the device ID of the station:

```
[DEVICEID]
FWA-INDRV*-MPE-16
```

7.2.3 Section [PARAM]

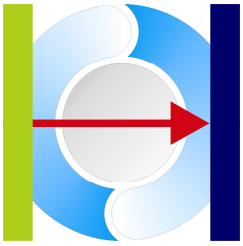
This section contains the IDN and binary data for configuration:

```
[PARAM]
S-0-32.0.0 (02 00)
```

7.2.4 Section [CONFIG]

This section contains the IDN list for the connections:

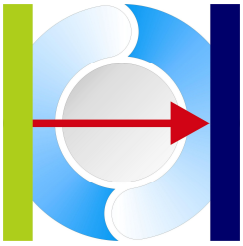
```
[CONFIG]
S-0-0134.0.0  <0,0>
S-0-0036.0.0  <0,0>
S-0-0047.0.0  <0,0>
S-0-0135.0.0  <1,1>
S-0-0040.0.0  <1,1>
S-0-0386.0.0  <1,1>


IDN
|              Connection Index
|              | Telegam Type
|              | |
S-0-0135.0.0  <1,1>
```



7.2.5 Section [DESC]

This section contains the output/input data description of the device:

```
[DESC]
00 00 01 01 02 00
00 00 04 02 02 00
00 00 04 03 04 00
00 00 04 03 04 00
00 01 01 01 02 00
00 01 04 02 02 00
00 01 04 03 04 00
00 01 04 03 04 00
```



```
Telegram Type      00 : MDT (Output)
     01 : AT (Input)

Data Item   00 : DATA_ITEM_NONE
     01 : DATA_ITEM_CCON
     02 : DATA_ITEM_IOCTRL
     03 : DATA_ITEM_IOSTAT
     04 : DATA_ITEM_VALUE
     05 : DATA_ITEM_SCALE
     06 : DATA_ITEM_DIAG
     07 : DATA_ITEM_NAME

Data Type   00 : DATA_TYPE_NONE
     01 : DATA_TYPE_U8
     02 : DATA_TYPE_U16
     03 : DATA_TYPE_U32
     04 : DATA_TYPE_U64
     05 : DATA_TYPE_I8
     06 : DATA_TYPE_I16
     07 : DATA_TYPE_I32
     08 : DATA_TYPE_I64
     09 : DATA_TYPE_F32
     10 : DATA_TYPE_F64


Connection Index
|  Telegram Type
|  |  Data Item
|  |  |  Data Type
|  |  |  |  Data Len
|  |  |  |  |     |
 00 00 01 01 02 00 <- Desc 0
 00 00 02 01 02 00 <- Desc 1
 00 00 04 02 02 00 <- Desc 2
 00 01 01 01 02 00 <- Desc 3
 00 01 03 01 02 00 <- Desc 4
 00 01 04 02 02 00 <- Desc 5
 00 01 04 02 02 00 <- Desc 6
```

8 Error Handling

X-GO and all implemented Master Stacks provides an error handling and tracing mechanism.

8.1 Debug LOG File

On execution X-GO creates sequence files in Text-Format:

EtherCAT:	ECATDBG.LOG
ProfiNET:	PNTDBG.LOG
Ethernet/IP:	EIPDBG.LOG
Sercos III:	SC3DBG.LOG

Note: These files are not accessible while X-GO is running

8.2 Event File

On execution X-GO logs error event to the Windows Event Manager. The X-GO software logs Application and System events. These events can be exported to a file and provided for support purposes.