

Automatic Test System

ATS 400

PROFINET IRT



1	Content	
1	Content	2
2	General	3
2.1	Licensing	3
2.2	Validation	3
2.3	Preconditions	3
2.4	Legal notice	3
2.5	Configuration file	3
2.6	Setup in Siemens TIA Portal	3
3	Data area	4
3.1	Overview	4
3.2	Output area	4
3.3	Input area	4
4	Profinet Master	6
4.1	Simulator	6
4.2	Settings	6
4.3	Data exchange	8
5	Command exchange	11
5.1	Command sequence	11
5.2	State machine	11
5.3	Time diagram	11
5.4	Interface of the function block	12
5.4.1	Input data	12
5.4.2	Output data	12
5.4.3	Implementation proposal	13
6	Notes	15

2 General

2.1 Licensing

To use the Anybus Communicator in connection with the **ATS 400** is a license for ASCII commands (article number 205060) required.

2.2 Validation

This description is valid for the Anybus Communicator for exchanging ASCII commands with the **ATS 400**.

The Anybus Communicator is already configured. It is no further configuration work necessary.

2.3 Preconditions

The count of bytes begins always from 0.

2.4 Legal notice

Third part brand names may have been used in this document, the rights are owned by them.

The rights of third parties remain inviolate by being mentioned in this document.

2.5 Configuration file

Always use the GSDML file [GSDML-V2.32-HMS-ABC_PROFINET_IRT2-20161031.xml](#) from the folder [GSDML ABC RS PROFINET 2.32](#). Do not use the file mentioned in the manual [PROFINET_IRT_Application_note_Siemens_S7_TIA.pdf](#). This is for a different model.

2.6 Setup in Siemens TIA Portal

The setup in Siemens TIA Portal is described in the document [PROFINET_IRT_Application_note_Siemens_S7_TIA.pdf](#) provided by the supplier of the Anybus Communicator.

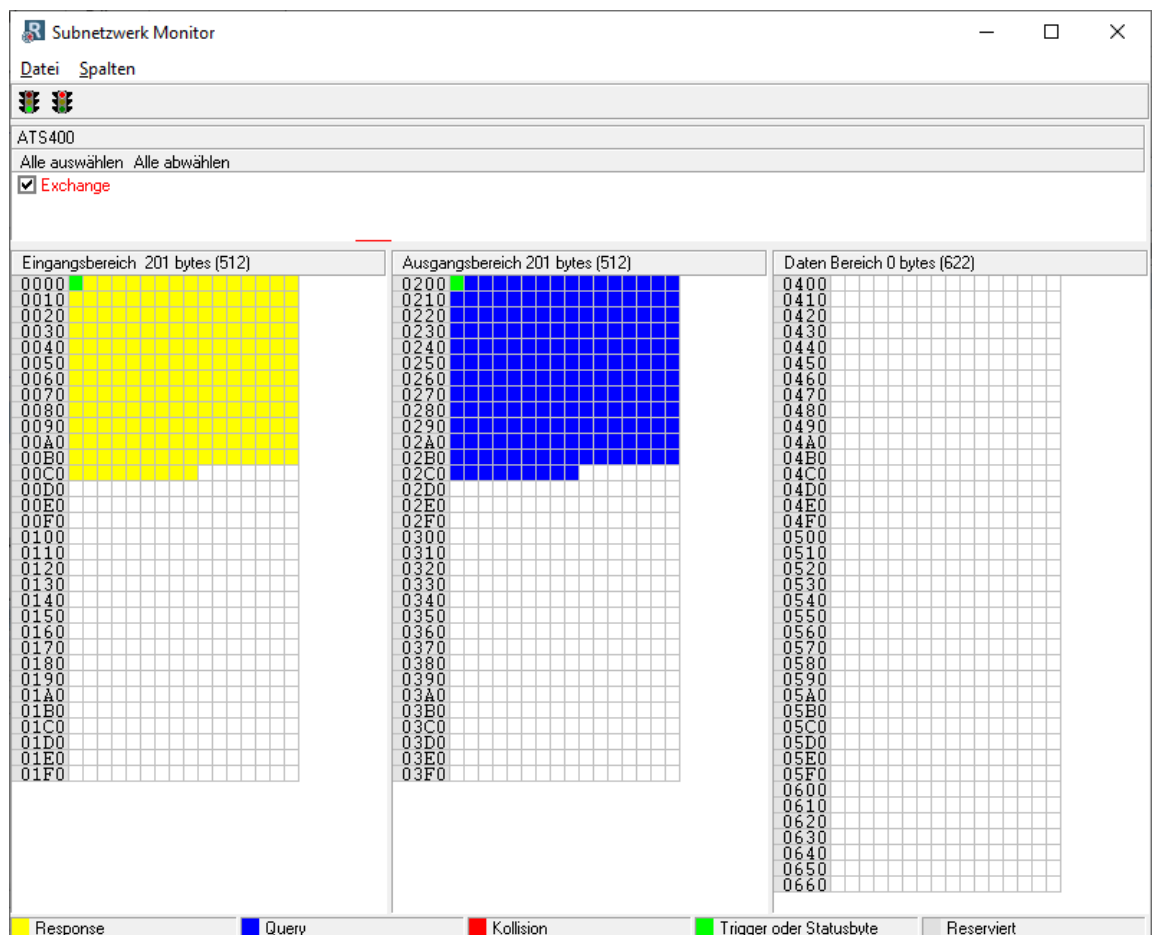
In chapter 4.1 point 6 of the document use the modules with 201 bytes described below.

Configure the modules in the data area as array [0..200] of bytes.

3 Data area

3.1 Overview

In each input and output are one status byte or a trigger byte and a data buffer defined. The data buffer for payload data comprises 200 bytes corresponding to the longest ASCII command. Thus, the entire data area comprises 201 bytes and must be configured accordingly.



3.2 Output area

Data in the output area are sent to the **ATS 400**.

To send a command, the following steps have to be carried out:

1. Entry of the ASCII command including the end character starting at byte 1.
2. Increase of the trigger byte in byte 0 by 1, including overflow from 255 to 0.

After increasing the trigger byte, the command is sent to the **ATS 400**.

3.3 Input area

Data in the input area are received by the **ATS 400**.

The following steps must be carried out to collect a received response:

1. Read the status byte and wait until the value is changed.
2. Reading of the response from byte 1 and evaluation up to the end character.

When a response is received from the Anybus Communicator, the status byte is increased by 1, including overflow from 255 to 0.

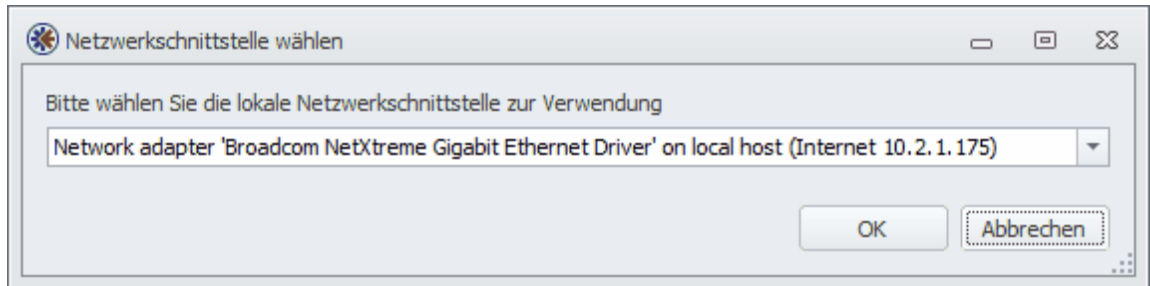
4 Profinet Master

4.1 Simulator

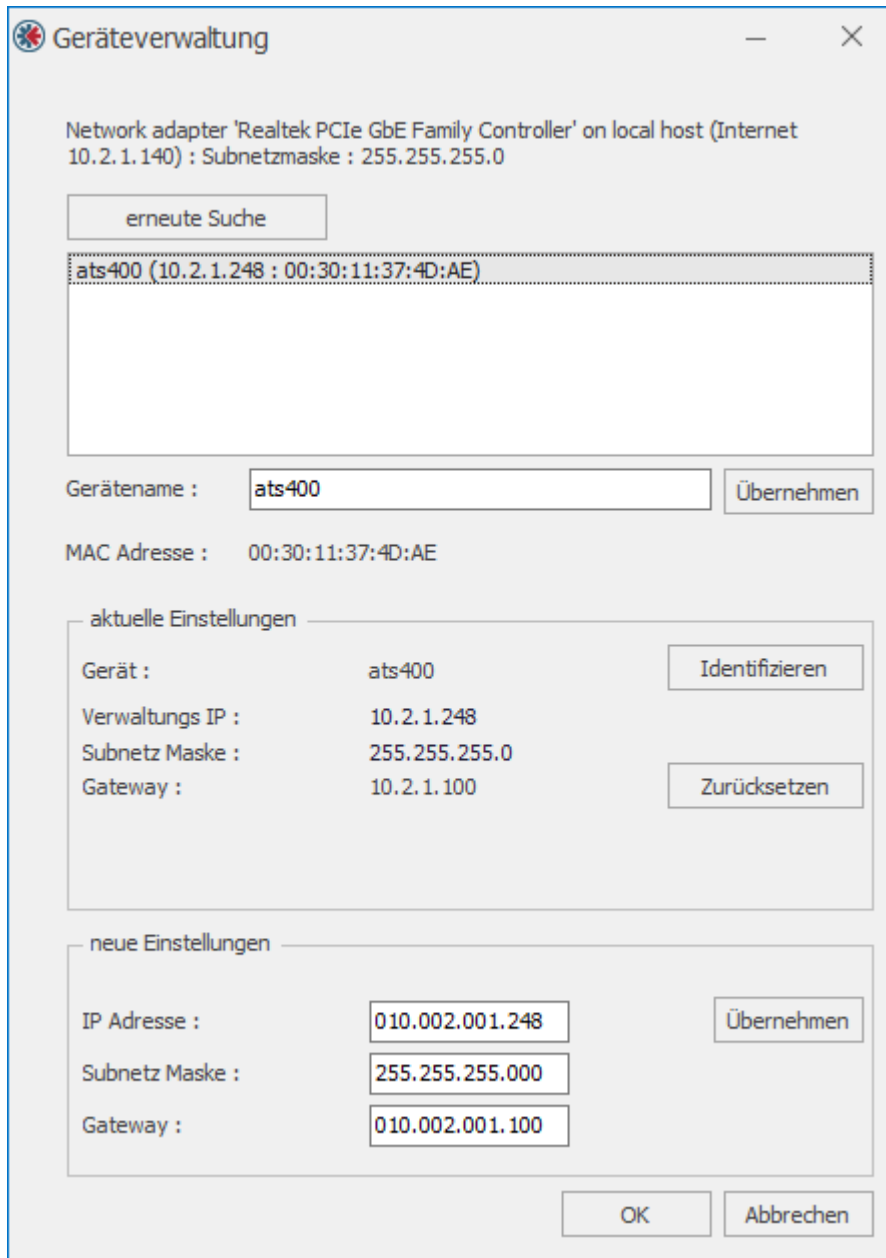
The program **PROFINET Master Simulator 1.0.1.63** was used as the Profinet master. The program was obtained from HMS and is for a fee.

4.2 Settings



The following settings have been made:



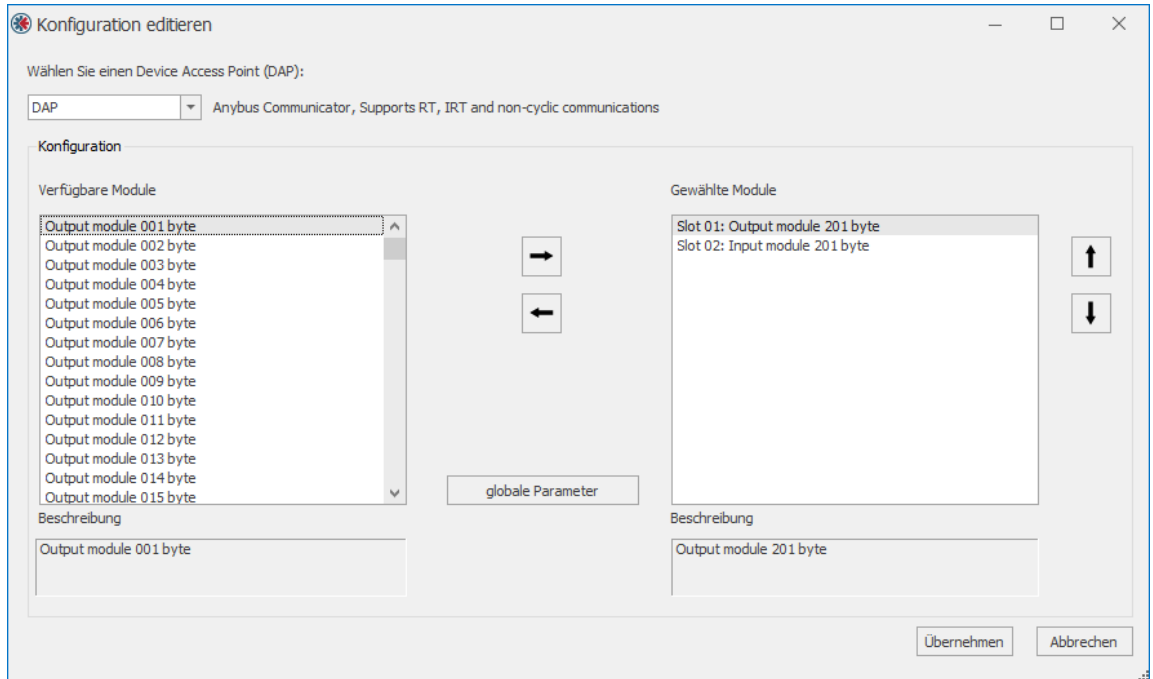
Selection of the network card on the simulation computer.



Definition of the device and assignment of an IP address. In the case of a point-to-point connection, the PROFINET master must be specified as the gateway.

Name	Datum	Typ
 GSDML-010C-0017-ABC-PIR2.bmp	31.10.2016 08:16	BMP-Datei
 GSDML-V2.32-HMS-ABC_PROFINET_IRT2-20161031.xml	11.11.2016 13:04	XML Document

Selection of the GSD file for the Anybus Communicator.

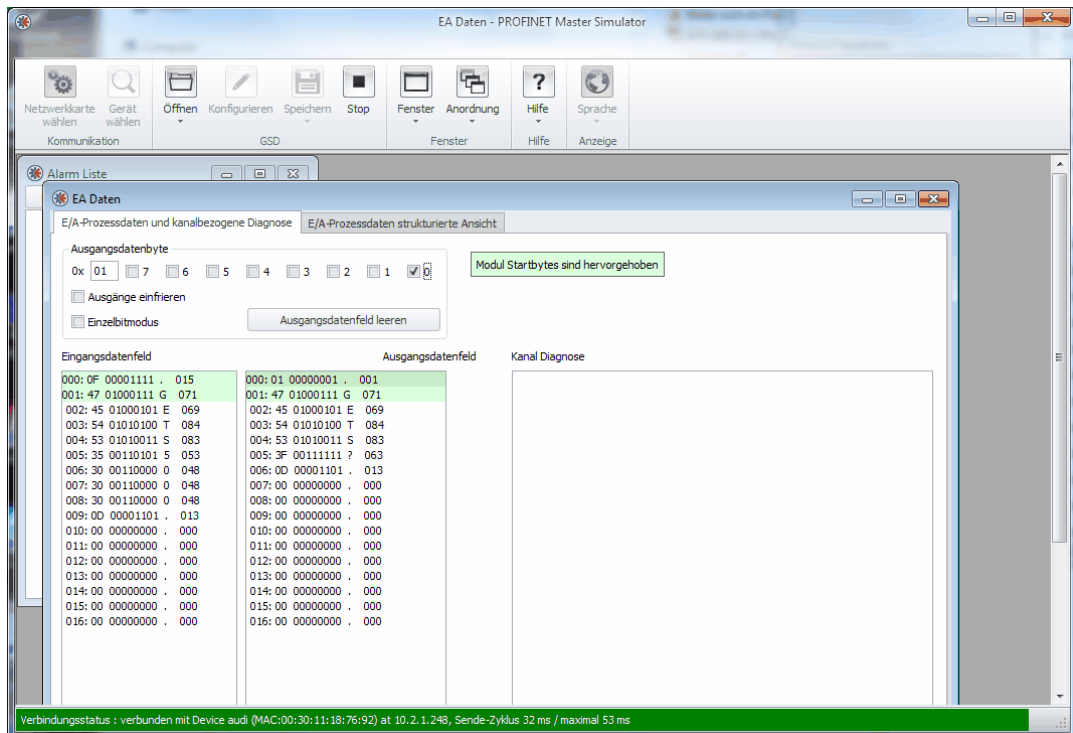


Configuration of the data areas. This configuration must match the data areas in the Anybus Communicator.

Communication can be started using the Start button.

4.3 Data exchange

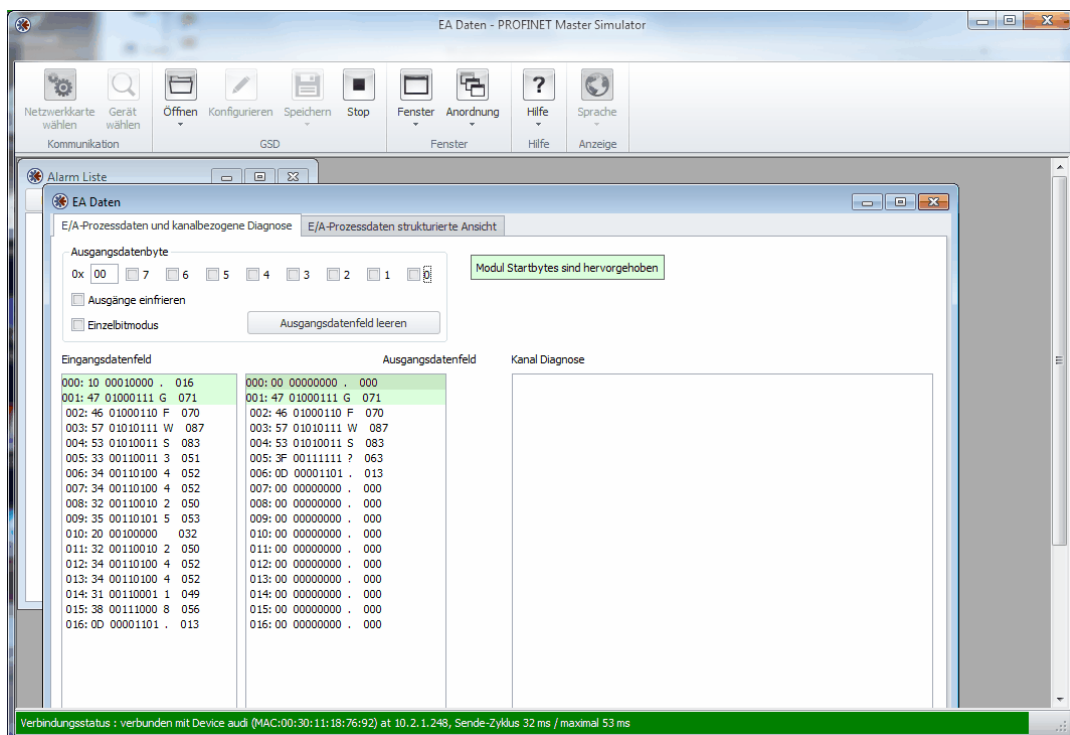
Two commands are shown as an example. One is the query of the status information and one the query of the firmware version.



Eingangsdatenfeld	Ausgangsdatenfeld
000: 0F 00001111 . 015	000: 01 00000001 . 001
001: 47 01000111 G 071	001: 47 01000111 G 071
002: 45 01000101 E 069	002: 45 01000101 E 069
003: 54 01010100 T 084	003: 54 01010100 T 084
004: 53 01010011 S 083	004: 53 01010011 S 083
005: 35 00110101 5 053	005: 3F 00111111 ? 063
006: 30 00110000 0 048	006: 0D 00001101 . 013
007: 30 00110000 0 048	007: 00 00000000 . 000
008: 30 00110000 0 048	008: 00 00000000 . 000
009: 0D 00001101 . 013	009: 00 00000000 . 000
010: 00 00000000 . 000	010: 00 00000000 . 000
011: 00 00000000 . 000	011: 00 00000000 . 000
012: 00 00000000 . 000	012: 00 00000000 . 000
013: 00 00000000 . 000	013: 00 00000000 . 000
014: 00 00000000 . 000	014: 00 00000000 . 000
015: 00 00000000 . 000	015: 00 00000000 . 000
016: 00 00000000 . 000	016: 00 00000000 . 000

After sending the command GETS? the **ATS 400** sends as a response GETS5000. Depending on the equipment of the **ATS 400** and the conditions of the device, other values may occur.

To send the command, the value in the trigger byte was changed from 0 to 1.



Eingangsdatenfeld	Ausgangsdatenfeld
000: 10 00010000 . 016	000: 00 00000000 . 000
001: 47 010001111 G 071	001: 47 010001111 G 071
002: 46 01000110 F 070	002: 46 01000110 F 070
003: 57 01010111 W 087	003: 57 01010111 W 087
004: 53 01010011 S 083	004: 53 01010011 S 083
005: 33 00110011 3 051	005: 3F 00111111 ? 063
006: 34 00110100 4 052	006: 0D 00001101 . 013
007: 34 00110100 4 052	007: 00 00000000 . 000
008: 32 00110010 2 050	008: 00 00000000 . 000
009: 35 00110101 5 053	009: 00 00000000 . 000
010: 20 00100000 032	010: 00 00000000 . 000
011: 32 00110010 2 050	011: 00 00000000 . 000
012: 34 00110100 4 052	012: 00 00000000 . 000
013: 34 00110100 4 052	013: 00 00000000 . 000
014: 31 00110001 1 049	014: 00 00000000 . 000
015: 38 00111000 8 056	015: 00 00000000 . 000
016: 0D 00001101 . 013	016: 00 00000000 . 000

After sending the command GFWS? the **ATS 400** sends as a response GFWS34425 24418. The values corresponds to the firmware versions available on the device, other values may occur.

To send the command, the value in the trigger byte was changed from 1 to 0.

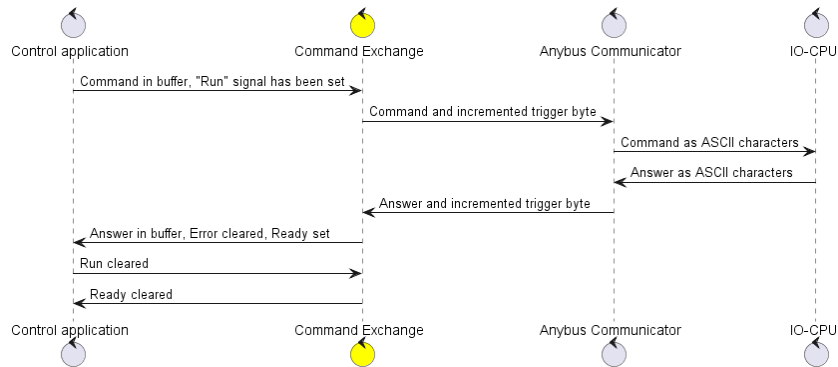
5 Command exchange

In the following, the command exchange, a state machine and a time diagram are described. This should help to create a basic module for communication with the Anybus Communicator. The case of a missing response is also considered here.

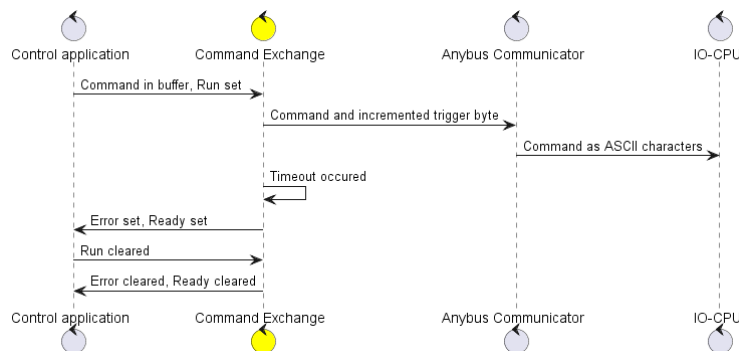
This basic block may only occur once in the program for each Anybus Communicator.

5.1 Command sequence

The command flow for a normal communication is shown here graphically.

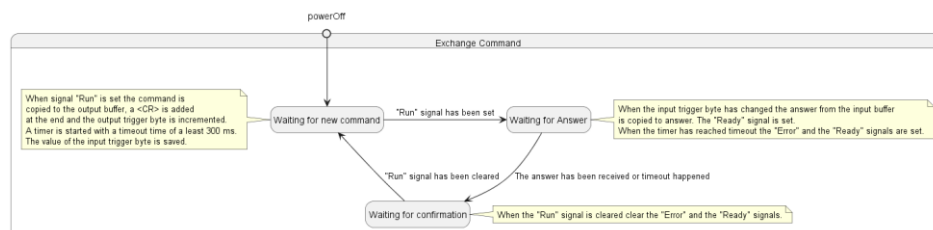


If there is no response, a timeout is triggered. The communication is shown here graphically.



5.2 State machine

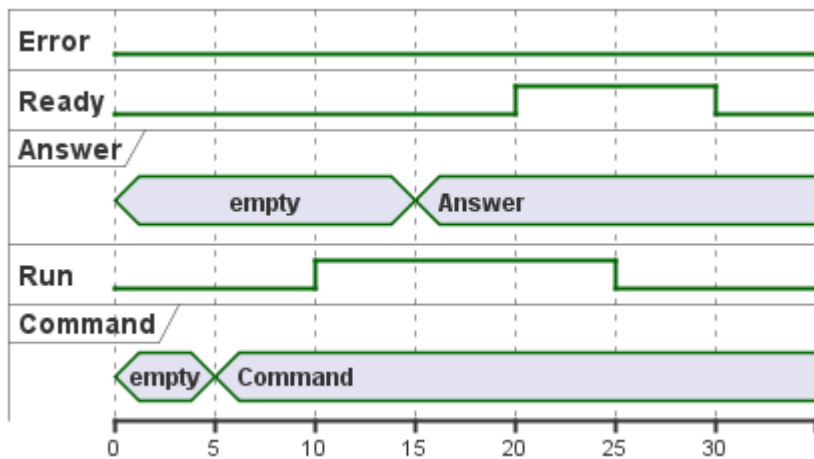
The state machine, also referred to as a step chain in the PLC environment, ensures the exchange of commands according to the master/slave principle.



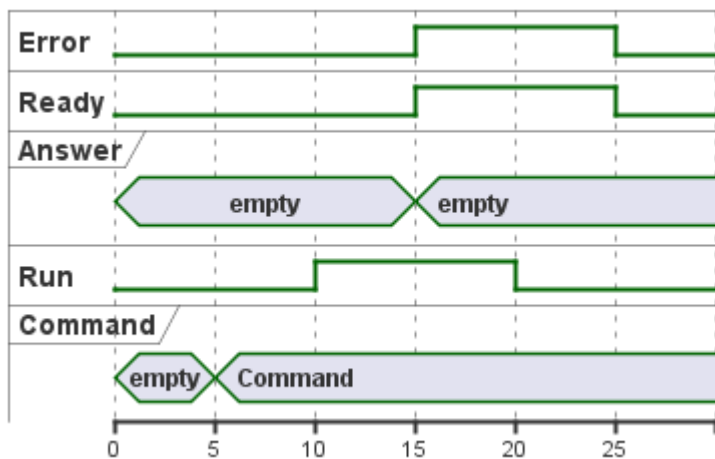
5.3 Time diagram

The following time diagrams represent the process only qualitatively and not quantitatively.

The time diagram for a normal communication is shown here graphically.



If there is no response, a timeout is triggered. The time diagram is shown here graphically.



5.4 Interface of the function block

5.4.1 Input data

Data type	Variable
BOOL	Run
STRING(200)	Command

```
VAR_INPUT
    Run : BOOL := FALSE;
    Command : STRING(200);
END_VAR
```

5.4.2 Output data

Data type	Variable
BOOL	Ready
BOOL	Error
STRING(200)	Answer

```

VAR_OUTPUT
  Ready : BOOL := FALSE;
  Error : BOOL;
  Answer : STRING(200);
END_VAR

```

5.4.3 Implementation proposal

```

(*)
* Example of a function block in structured text for the
exchange of a command.
*
* This example is only intended to illustrate the basic
principle and is not executable code and is
* not complete.
*
* For a concrete implementation are missing:
* - Conversions between data types, e.g. characters of a string
and BYTE.
* - Handling of the timeout
*)
VAR_INPUT
  Run : BOOL := FALSE; // At the transition from FALSE to TRUE
the command exchange is started.
  Kommando : STRING(200); // Command to be sent, without
terminating semicolon.
END_VAR
VAR_OUTPUT
  Ready : BOOL := FALSE; // Becomes TRUE if the response was
received or a timeout occurred. Becomes FALSE if Run is FALSE
again.
  Error : BOOL := FALSE; // Becomes TRUE if a timeout occurred.
  Antwort : STRING(200); // Response received or blank if a
timeout occurred.
END_VAR
VAR
  Output : ARRAY(0..200) of BYTE; // This represents the
output buffer of the ABC.
  Input : ARRAY(0..20) of BYTE; // This represents the input
buffer of the ABC.
  Index : INT; // Index for transferring the characters.
  OldRun : BOOL := FALSE;
  OldReceiveIndex : BYTE;
  (* Values for the state machine:
  * 0 : Waiting for new command
  * 10 : Waiting for response
  * 20 : Waiting for acknowledgement
  *)
  State : INT := 0; // Condition of the machine.
END_VAR

CASE State OF
  0 :
    IF OldRun = FALSE AND Run = TRUE THEN // New command is
to be sent.
      (* Accept command in send buffer and change state. *)
      Index := 0;

```

```
        WHILE Command[Index] <> 0 DO // Transfer command
            Output[Index + 1] := Kommando[Index];
            Index := Index + 1;
        END_WHILE;
        Output[Index + 1] = 13; // Append final <CR>.
        OldReceiveIndex := Input[0]; // Remember receive
trigger byte.
        Output[0] := Output[0] + 1; // Increase send trigger
byte.
        OldRun := Run; // Remember start flag.
        State := 10; // Change state to "Wait for response".
    END_IF

    10 :
        IF OldReceiveIndex <> Input[0] THEN // Receive trigger
byte has changed.
        (* Accept response from receive buffer and change state.
*)
            Index := 0;
            WHILE Input[Index + 1] <> 13 DO // Transmit response.
                Answer[Index] := Input[Index + 1];
                Index := Index + 1;
            END_WHILE;
            Answer[Index] := 0; // Finalize answer.
            Error := FALSE; // Reset error.
            Ready := TRUE; // Set ready message.
            State := 20; // Change state to "Wait for
acknowledgement".
        END_IF

        20 :
            IF OldRun = TRUE AND Run = FALSE THEN // Acknowledgement
occurs
            (* Reset signals and change state. *)
                Error := FALSE; // Reset error.
                Ready := FALSE; // Reset ready message.
                state := 0; // Change state to "Wait for new
command".
            END_IF

            ELSE
                state = 0;

    END_CASE
```

6 Notes



Lembergstraße 23
70825 Korntal

Telefon: +49 711 83 99 39-0
Telefax: +49 711 83 99 39-9
Internet: www.etl-prueftechnik.de
E-Mail: info@etl-prueftechnik.de