# 5E9000.29

# Transponder Reader
# 4102 / 4150
## Operating Instructions

**Version V1.2.1 (August 2010)**

| Revision | Date | Comment | Author |
|---|---|---|---|
| 1.1 | Oct. 2006 | First release | M. Hochländer |
| 1.2 | Sept. 2007 | Second release<br>Chapter 8, Automation studio transponder support included | M. Hochländer |
| 1.2.1 | Aug. 2010 | Third release<br>Chapter 7.3, Command overview, translation error of „tag“ corrected | M. Hochländer |
| | | | |

**Tabelle 1: Records of revision**

# Technical Description

| | |
|---|---|
| Model number: | 5E9000.29 |
| Project name: | Transponder reader 22mm USB |
| Description: | Transponder reader 22mm USB |
| Dimensions (WxHxD): | Approx. 45 x 46 x 31 mm (without connecting cable) |

# 1General information

## 1.1 Safety guidelines

### 1.1.1 Introduction

Programmable logic controllers (PLCs), operating and monitoring devices (industrial PCs, Power Panels, Mobile Panels, etc.) as well as B&R uninterruptible power supplies have been designed, developed, and manufactured for conventional use in industry.
They were not designed, developed and manufactured for any use involving serious risks or hazards that could lead to death, injury, serious physical damage, or loss of any kind without the implementation of exceptionally stringent safety precautions.
In particular, such risks and hazards include the use of these devices to monitor nuclear reactions in nuclear power plants, as well as flight control systems, flight safety, the control of mass transportation systems, medical life support systems, and the control of weapons systems.
When using programmable logic controllers and operating/monitoring devices as control systems together with a Soft PLC (e.g. B&R Automation Runtime or comparable products) or a Slot PLC (e.g. B&R LS251 or comparable products), safety precautions that apply to industrial control systems (e.g. the use of safety devices such as E-stop circuits, etc.) must be observed in accordance with applicable national and international regulations.
The same applies to all other connected devices, e.g. drives.
All tasks such as installation, commissioning, and service must only be carried out by qualified personnel. Qualified personnel are persons who are familiar with the transport, mounting, installation, commissioning, and operation of the product and who have the appropriate qualifications (e.g. IEC 60364).

National accident prevention guidelines must be followed.

The safety guidelines, connection descriptions (type plate and documentation), and limit values listed in the technical data are to be read carefully before installation and commissioning and must be observed.

### 1.1.2 Intended use

Electronic devices are generally not failsafe. In the event of a failure on the programmable control system, operating or monitoring device, or uninterruptible power supply, the user is responsible for ensuring that other devices that may be connected, e.g. motors, are in a secure state.

### 1.1.3 Transport and storage

During transport and storage, devices must be protected from excessive stress (mechanical load, temperature, humidity, aggressive atmosphere, etc.).

### 1.1.4 Installation

- Installation must take place according to the documentation using suitable equipment and tools.
- Devices must only be installed without voltage applied and by qualified personnel.
- General safety regulations and nationally applicable accident prevention guidelines must be observed.
- Electrical installation must be carried out according to the relevant guidelines (e.g. line cross section, fuse, protective ground connection).

### 1.1.5 Operation

#### 1.1.5.1 Protection against touching electrical parts

To operate programmable logic controllers, operating and monitoring devices, and uninterruptible power supplies, certain components must carry dangerous voltage levels of over 42 VDC.

A life-threatening electrical shock could occur if you come into contact with these parts.
This could result in death, severe injury, or material damage.

Before turning on the programmable logic controller, the operational and monitoring devices and the uninterruptible power supply, ensure that the housing is properly grounded (PE rail).

The ground connection must be established when testing the operating and monitoring devices or the uninterruptible power supply, even when operating them for only a short time.

Before turning the device on, make sure that all voltage-carrying parts are securely covered.
During operation, all covers must remain closed.

## 1.2 Organization of safety guidelines

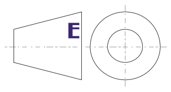
Safety notices in this document are organized as follows:

| Safety notice | Description |
|---|---|
| Danger! | Disregarding the safety regulations and guidelines can be life-threatening. |
| Caution! | Disregarding the safety regulations and guidelines can result in severe injury or major damage to material. |
| Warning! | Disregarding the safety regulations and guidelines can result in injury or damage to material. |
| Information: | Important information for preventing errors. |

**Table 2: Organization of safety notices**

## 1.3 Guidelines

European dimension standards apply to all of the dimension diagrams in this document.



## 2 Function

The necessary electronics and mechanics for a transponder reader/writer were designed for installation in a standard 22.5 mm hole.

The device can be operated with Windows, Linux, or MAC-based operating systems.
The connection is made with USB.
Data is evaluated on the PC, not on the reading/writing device!

In Automationstudio 2.6.006 the transponder reader/writer is also implemented.

The following transponder types are supported:

- 4102
  Read only
- 4150
  Read and write

Both transponders can be delivered as a key ring accessory or in check card format.

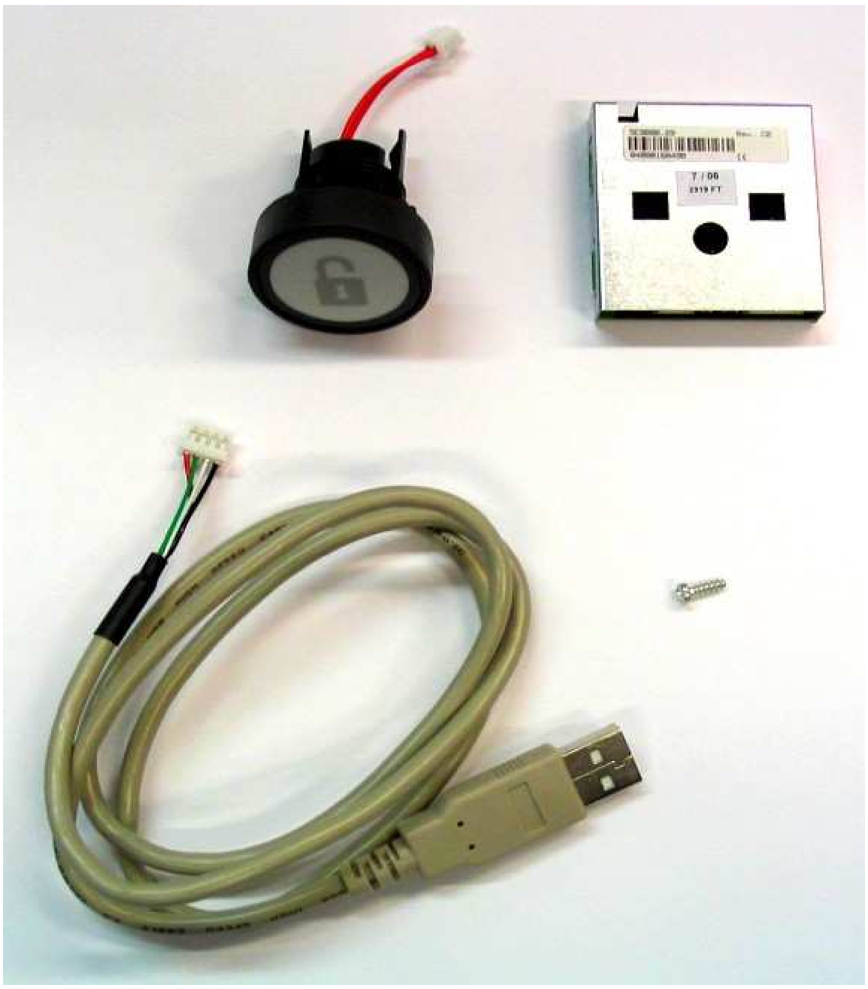
Typical applications:

- Admittance control
- Key switch function
- Assigning access levels
- Suspending safety systems
- Etc.

## 3 Construction

The 22 mm USB transponder reader consists of the following parts:

- Installation element for front mount installation including gasket and fastening ring.
  The antenna (incl. cable) for reading and writing is integrated here.
- Electronic circuit board attached to the installation element (can also be screwed on).
- Fastening screw
- 80 cm USB connecting cable

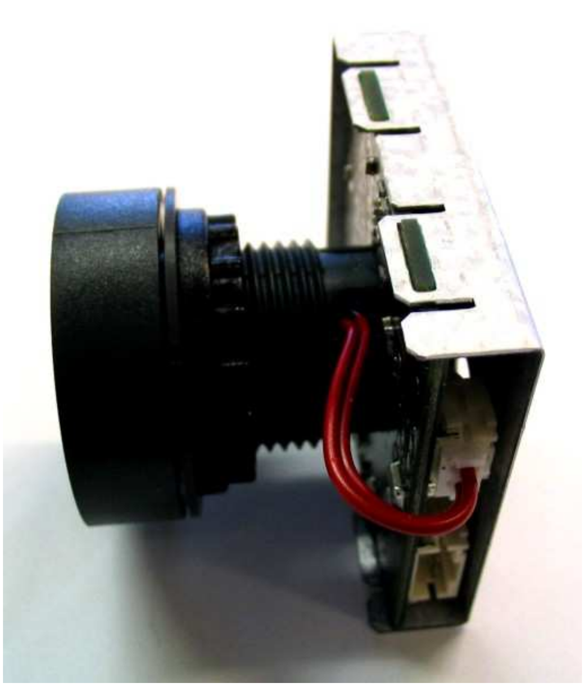


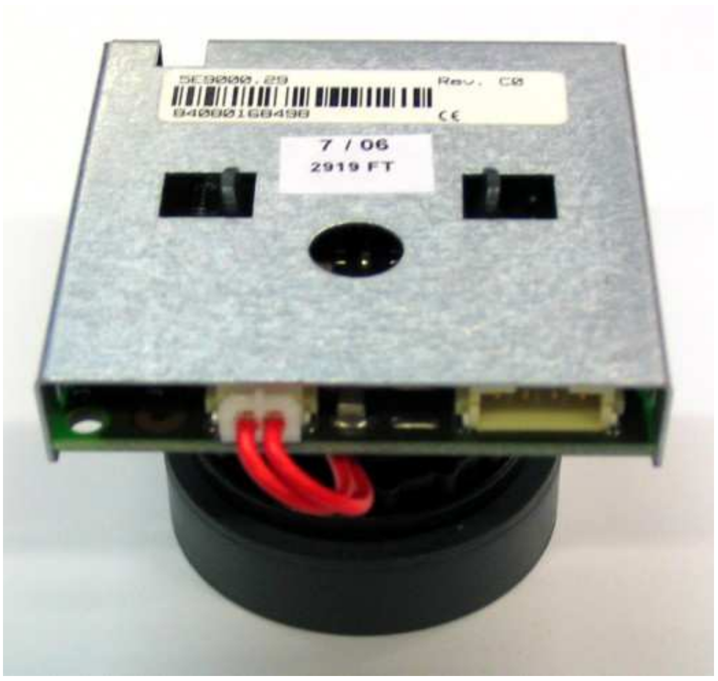
## 4 Mechanics

### 4.1 Mylar design

## 5 Installation

The installation element for front mount installation is inserted in a 22.5 mm hole and secured in place with the fastening nut.
The reading or writing electronics are snapped onto the back of the installation element.
It can be fastened with the screw that is provided.
When doing so, make sure that the antenna cable is not clamped down.
This is then inserted into the corresponding socket.
To connect to a USB interface (e.g. PC), the USB cable is inserted in the 4-pin socket.

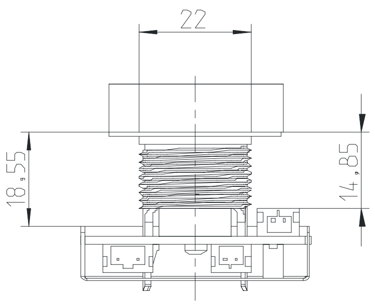


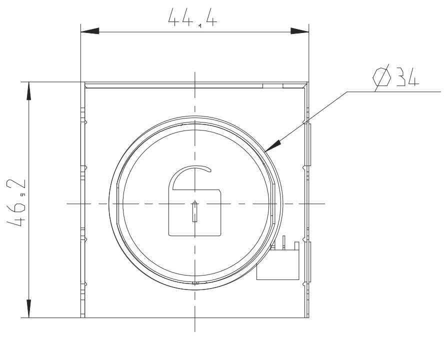


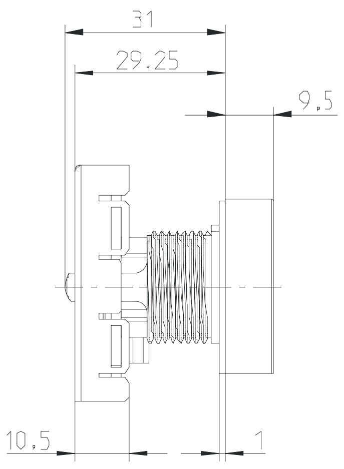
Complete device (without front plate)

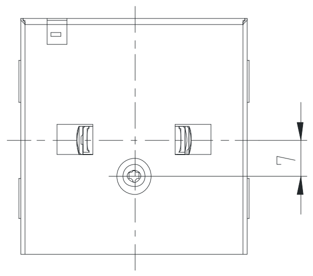
To loosen the electronics from the installation element, press the latches outwards and pull out!

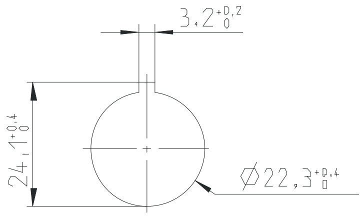
## 5.1 Installation diagrams









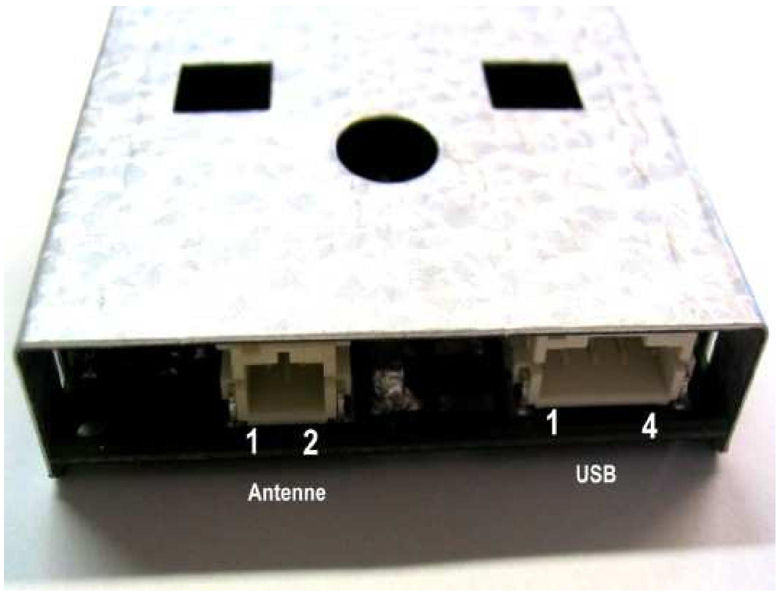


## 6 Interfaces

### 6.1 USB

4-pin PHR socket

| Pin | Assignment |
|---|---|
| 1 | GND |
| 2 | Data - |
| 3 | Data + |
| 4 | VCC |



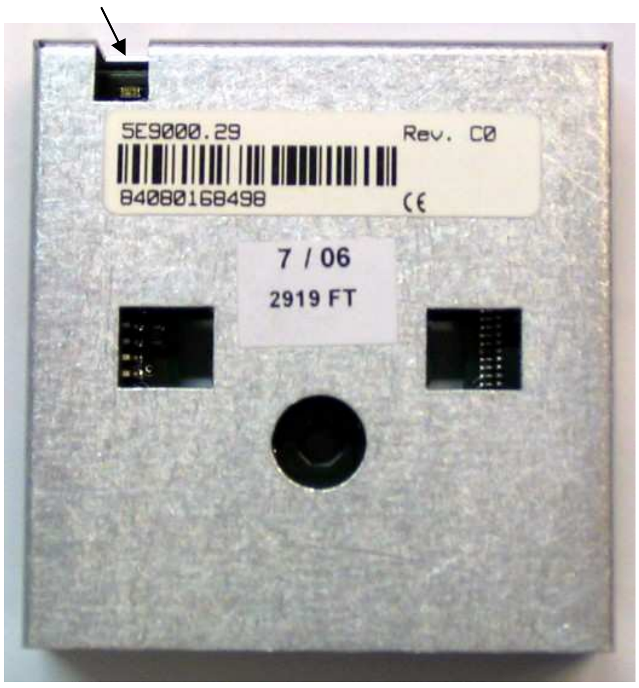
### 6.2 Antenna

2-pin PHR socket

| Pin | Assignment |
|---|---|
| 1 | Antenna |
| 2 | Antenna |

## 6.3 LED display

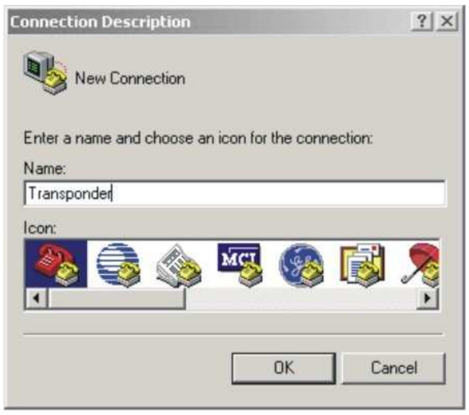
The LED is continuously lit if a transponder is not in range.
If a transponder is in range, the LED goes dark.



## 7 Operating Instructions

The transponder reader/writer works with the transponders 4102 and 4150.
A driver from Prolific is necessary for operation. (not for AutomationStudio) This creates a virtual COM port that the transponder uses for communication.
This driver can be downloaded at no cost from Prolific's support page.
http://www.prolific.com.tw
driver for PL2303X : wd_pl2303h-hx-x_v……

The connection is established with e.g. HyperTerminal.


Connection Description

New Connection

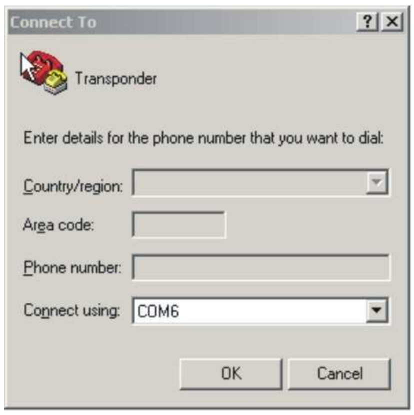
Enter a name and choose an icon for the connection:

Name: Transponder

Icon:

OK Cancel

**Figure 1: HyperTerminal setup**


Connect To

Transponder

Enter details for the phone number that you want to dial:

Country/region:

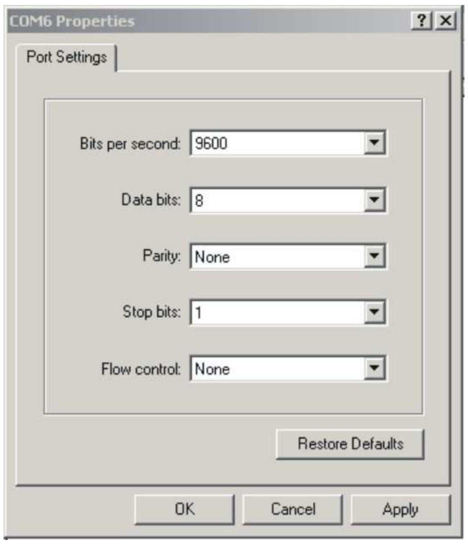
Area code:

Phone number:

Connect using: COM6

OK Cancel

**Figure 2: Selecting the interface in HyperTerminal**


COM6 Properties

Port Settings

| | |
|---|---|
| Bits per second: | 9600 |
| Data bits: | 8 |
| Parity: | None |
| Stop bits: | 1 |
| Flow control: | None |

Restore Defaults

OK Cancel Apply

**Figure 3: HyperTerminal connection settings**

## 7.1 Transponder 4102

This transponder can only be read. (read only)
The 40-bit unique ID number is programmed by laser.
Data is transferred using amplitude modulation.
Manchester data encoding is used.
The carrier frequency is 125 kHz.

The "tag" command can be used to query the status.

**Format:** :tag,SC,#crc <Return>

**Example:** tag,0,0,#crc <Return>

Example response with transponder 4102:

ack,0,4102,f080990414,#0532  Note: f080990414 stands for the 40-bit unique ID number.

## 7.2 Transponder 4150

This is a read/write transponder with a memory size of 1 Kbit.
It is organized into 32 words, each with 32 bits.
This data is stored in an EEProm memory.
100,000 delete and write procedures are guaranteed.
Data can be retained in the EEProm for at least 10 years.
In addition, this transponder has a 32-bit unique ID number. (read only)
The password function can be used to write-or read-protect individual registers.
Data is transferred by amplitude modulation.
The carrier frequency is 125 kHz.

**Register overview and function:**
The following table shows the internal memory assignments of the transponder 4150.

| Register number | Description |
|---|---|
| 0 | Password register. Write only. |
| 1 | Protection word register |
| 2 | Control word register |
| 3 to 31 | User memory. Read / Write. |
| 32 | Device serial number. Laser-programmed. Read only. |
| 33 | Device identification. Laser-programmed. Read only. |

**Register 0:**
The password is located in Register 0.
The password is always 0 when delivered.
This register can only be written.
Reading this register always returns the value 0, regardless of the password assigned.

**Register 1:**
The protection word is located in Register 1.
This register is always 0 when delivered.
The 32-bit protection word specifies the protected read and write area on the transponder.
If this register is 0, then all user registers (3-31) can be read and written.
Writing to this register specifies which registers should be read- or write-protected.

| | |
|---|---|
| Bits 0 – 7 | First register that should be read-protected. (First word read inhibited.) |
| Bits 8 – 15 | Last register that should be read-protected. (Last word read inhibited.) |
| Bits 16 – 23 | First register that should be write-protected. (First word write inhibited.) |
| Bits 24 – 31 | Last register that should be write-protected. (Last word write inhibited.) |

**Register 2:**

The 32-bit control word is located in Register 2.
This register is always 0 when delivered.
This register should normally not be modified since individual settings must be matched to your transponder reading device.

| | |
|---|---|
| Bits 0 – 7 | First word read (FWR), first read cell. |
| Bits 8 – 15 | Last word read (FWR), last read cell. |
| Bit 16 | Password check ON (1) / OFF (0). |
| Bit 17 | Read after write ON (1) / OFF (0). |
| Bits 18 – 31 | Available. |

**Registers 3 – 31:**

Registers 3 to 31 are available for the user to store data.

**Register 32:**

Register 32 holds the transponder's read-only serial number.
This unique number is programmed by laser when manufactured.
It can be read but not modified.

**Register 33:**

The identification number is also laser-programmed and can only be read.
The individual bits are assigned as follows:

| | |
|---|---|
| Bit 0 = 5 | V4050 code = 32 hex. |
| Bit 6 = 15 | Transponder chip version code. |
| Bits 16 – 23 | Reserved |
| Bits 24 – 31 | Check bits. |

The following table provides an overview of when individual registers may be modified.

| Write to register | Check password bit (Bit 16 control word) | Write inhibit (protection word) | Write operation |
|---|---|---|---|
| 0 | X | X | Only with command. Password writing possible. |
| 1-2 | X | OFF | Always login first. |
| 1-2 | X | ON | Always write-protected. |
| 3-31 | OFF | OFF | Freely programmable |
| 3-31 | ON | OFF | Login required. |
| 3-31 | X | ON | Login first. Then change protection word. |

## Information:

**If you use your own password to protect the transponder and then forget the password, the protected area can no longer be accessed!**

## General information:

All commands that are entered on the terminal are printed blue in this documentation.
**Example:** cc,0,#crc <Return>

Positive acknowledgements from the transponder reader/writer are shown in green.
**Example:** ack,0,0,#0236 <CR> <LF>

Negative acknowledgements from the transponder reader/writer are shown in red.
**Example:** nak,0,101,#02a3 <CR> <LF>

| | |
|---|---|
| x[xxxxxxx] | = A hex number with 1-8 digits. |
| SC | = Sequence counter 0- 65535 dec. |
| crc | = Checksum xxxx 4 digits hex. |
| <Return> | = Return key (0x0d hex). |
| <CR> | = Carriage return (0x0d hex). |
| <LF> | = Line feed (0x0a hex). |

## Checksum calculation:

Each command string contains a checksum (CRC) that has to be calculated by the program.
It is calculated by adding up all characters in the command string up to and including the # character in hexadecimal format.

**Example: cc,0,0,#**01cd <Return>
**c** = 63 Hex
**c** = 63 Hex
**,** = 2C Hex
**0** = 30 Hex
**,** = 2C Hex
**0** = 30 Hex
**,** = 2C Hex
**#** = 23 Hex
======
Sum = **1cd** Hex
The checksum is always specified as a **4-digit** hex number.
In this case, we also need to add a leading zero at the beginning.
Therefore, **01cd**.
The complete command thus looks like this:

**Example: cc,0,0,#01cd** <Return>
As an alternative, the three letters **crc** can be specified in place of the calculated checksum.

**Example:** cc,0,0,#crc <Return>
In this case, the transponder reader/writer **does not** check the checksum for the specified command string.

## 7.3 Command structure:

A complete command consists of a data string. The data string is subdivided into several blocks.
The individual blocks are separated by commas.
The command string must be completed by pressing <Return> (Enter).

All commands and parameters must be written in **small letters**.

**Command structure:**

Command , SC , Par1, Par2 , #crc , <Return>

There are some commands where Par1 and Par2 are not used; there are also cases where additional ParX parameters are needed.

The sequence counter (**SC**) is a number that must be specified by the user. It is then returned by the corresponding transponder reader/writer as an echo. This number should normally be incremented by one for each command. It is needed so that the response to a command can be assigned.

**Command overview:**

The following commands are currently implemented:

| **Command** | **Description** | **4102** | **4150** |
|---|---|---|---|
| Cc | Switch CRC check ON/OFF | Yes | Yes |
| du | Dump transponder register | No | Yes |
| id | Read hardware/software version | Yes | Yes |
| li | Login for transponder | No | Yes |
| rd | Read transponder register | No | Yes |
| rdm | Read multiple transponder registers | No | Yes |
| re | Reset transponder | No | Yes |
| rf | Carrier frequency ON/OFF | Yes | Yes |
| stack | Output stack level | Yes | Yes |
| tag | Query status | Yes | Yes |
| wp | Write password | No | Yes |
| wr | Write register | No | Yes |
| wrm | Write multiple registers | No | Yes |

### 7.3.1 cc "Check Checksum" command

This command can specify whether the transponder reader/writer should check the checksum when a command string is received.
If this check is disabled, then commands that don't have a valid checksum can be entered.

**Format:** cc,SC,Par1,#crc <Return>

**Par1:** 1 = Turn on checksum checking
0 = Turn off checksum checking

**Example:** cc,0,0,#01cd <Return> Turn off check.
ack,0,0,#0236 <CR> <LF>

**Example:** cc,0,1,#01ce <Return> Turn on check.
ack,0,0,#0236 <CR> <LF>

### 7.3.2 id command

The "id" command can be used to query the hardware and software version of the transponder reader/writer.
This guarantees that your application software is being operated with the correct readers.

**Format**: id,SC,#crc <Return>

| | |
|---|---|
| **Example:** id,0,#0178 <Return> | Queries versions. |
| ack,0,0120,0105,#03bb | Response. |

Hardware version: **01.20**
Software version: **01.05**

### 7.3.3 rf command

The "rf" command can be used to turn the 125 kHz carrier frequency of the transponder reader/writer on or off.

**Information:**
**Be aware that no transponders can be read if RF is turned off.**

**Format:** rf,SC,Par1,#crc <Return>

**Par1:** = 1 for activated
= 0 for deactivated

**Example:** rf,0,1,#01e0 <Return> Turn on carrier.
ack,0,0,#0236 <CR> <LF>

**Example:** rf,0,0,#01df <Return> Turn off carrier.
ack,0,0,#0236 <CR> <LF>

### 7.3.4 stack command

The "stack" command can be used to output the stack level.

**Format:** stack,SC,#crc <Return>

**Example:** stack,0,#crc <Return>
ack,0,0,#0236 <CR> <LF>

### 7.3.5 tag command

The "tag" command can be used to query the status.

**Format:** :tag,SC,#crc <Return>

**Example:** tag,0,0,#crc <Return>

Possible responses:
nak,SC,3,#0244 -> No tag in range

or (4100 in range)
ack,SC,4102,xxxxxxxxxx,#CRC xx = 10-digit serial number (40-bit unique ID number)

or (4x50 in range)
ack,SC,4150,#CRC

## 7.4 4150 transponder-specific commands

The following is a brief overview of 4150 transponder registers:

| Register number | Description |
|---|---|
| 0 | Password register. Write only. |
| 1 | Protection word register |
| 2 | Control word register |
| 3 to 31 | User memory. Read / Write. |
| 32 | Device serial number. Laser-programmed. Read only. |
| 33 | Device identification. Laser-programmed. Read only. |

When delivered, the password for the transponder is always **00000000**.

**Information:**
**Only modify registers 0, 1, and 2 if you are very familiar with the transponder.**
**Improper programming of these registers may make the transponder unusable.**

### 7.4.1 rd (read) command for the 4150 transponder

This command allows you to read the individual 32-bit registers of the 4150 transponder.
For this to work, the transponder must not be password-protected or a valid li (login) needs to be performed beforehand.

**Format:** rd,SC,Par1,#crc <Return>

**Par1:** 1 to 33 Register number. Decimal.

**Example:** rd,0,5,#01e2 <Return> Read Register 5.
ack,0,2031373a#03ca <CR> <LF> Register 5 contains the data "**2031373a**".
ack,0,0,#0236

### 7.4.2 rdm (read multiple) command for the 4150 transponder

This command lets you read multiple words on the 4150 transponder.

Important! A read error aborts the procedure!

**Format:** rdm,SC,start_adr.number_of_words,#crc <Return>

**Example:** rdm,0,0,34,#crc -> Reads all words from the tag.

```
ack,0,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00
000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,0
0000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
00000000,00000000,00000000,00000000,00000000,00000000,011d7722,ea001032,#3b66
```

### 7.4.3 wr (write) command for the 4150 transponder

The "wr" command (write) is used to write individual 32-bit registers on the 4150 transponder.
For this to work, the transponder must not be password-protected or a valid li (login) must be carried out beforehand.

> **Information:**
> **Be aware that Registers 1 and 2 are used for the control and protection words.**
> **Only modify these registers if you are very familiar with their function.**
> **In addition, a valid login is needed.**

**Format:** wr,SC,Par1,Par2,#crc <Return>

**Par1:** = Register number. 1 to 31 decimal.

**Par2:** = Register data. 8 ASCII hex characters.

**Example:** wr,0,5,12345678,#03c5 <Return> Writes 12345678 to Register 5.
ack,0,0,#0236 <CR> <LF>

### 7.4.4 wrm (write multiple) command for the 4150 transponder

This command allows you to write multiple words to the 4150 transponder's registers.

ATTENTION! The number of words is limited to 2!

**Format:** wrm,SC,start_adr,xxxxxxxx,[xxxxxxxx],#crc <Return>

**Example:** wrm,0,4,12345678,33335555,#crc <Return> Writes 12345678 to Register 4 and 33335555 to Register 5.
ack,0,0,#0236 <CR> <LF>

### 7.4.5 li (login) command for the 4150 transponder

The "li" (login) command can be used to open a password-protected transponder so that all register contents can be edited.

**Format:** li,SC,Par1,#crc <Return>

**Par1:** = Password 8 ASCII hex characters
li,0,00000000,#032c <Return> Login with password 00000000
ack,0,0,#0236 <CR> <LF>

The password, protection word, and control word can only be modified after a valid login.

### 7.4.6 re (Reset) command for the 4150 transponder

If a password-protected transponder was opened with the "li" (login) command, it can be locked again with the reset command. From this point on, password-protected areas can no longer be edited.
If the transponder is removed from the reading range of the transponder reader/writer, then it is locked automatically.

**Format:** re,SC,#crc Reset transponder.

re,0,#0182 <Return> Reset transponder.
ack,0,0,#0236 <CR> <LF>

### 7.4.7 wp (write password) command for the 4150 transponder

The "wp" (write password) command is used to write the password.

Before a new password can be written, a valid login must have been carried out.

**Format:** wp,SC,Par1,#crc4 Write password.

**Par1:** = Password 8 ASCII hex characters
wp,0,12345678,#0362 <Return> Write password **12345678**.
ack,0,0,#0236 <CR> <LF>

### 7.4.8 du (dump) command for the 4150 transponder

The "du" (dump) command allows you to display the complete data contents of the transponder.

**Format:** du,SC,#crc4

```
du,0,#0184 <Return> Display dump.
Dump of the 4150 Tag
00:00000000 00000000 00000000 00000003
04:00000004 00000005 00000006 00000007
08:88888888 99999999 10101010 00000011
12:00000012 00000013 00000014 00000015
16:16161616 17171717 18181818 19191919
20:00000020 00000021 00000022 00000023
24:24242424 25252525 26262626 27272727
28:00000028 00000029 00000030 00000031
32:0002ef3d df001032
ack,0,0,#0236
```

The two-digit number on the left is the register number. It is followed by the corresponding register data.

## 7.5 4150 transponder error messages

For each command transferred to the transponder reader/writer, the transponder reader/writer responds with an appropriate return message that can be either positive or negative.

If a complete transferred command is detected without errors, a positive message is returned with ack (acknowledge). Messages returned from the transponder reader/writer have the format shown below.

**Format:** ack,SC,DATA,#crc

SC = The specified sequence counter returned as an echo.
DATA = Return data or 0

**Example:** ack,0,0,#0236 <CR> <LF>

If a command is not recognized by the transponder reader/writer, a negative message is returned with nak (negative acknowledgement).

**Format:** nak,SC,ERROR,#crc <CR> <LF>

**SC** = The specified sequence counter returned as an echo.
**ERROR** = Error number. (see below)

**Example:** nak,0,101,#02a3 <CR> <LF>

The following error numbers are currently defined for the 4150 transponder.

**201** = Undefined command
**202** = Read error
**203** = Write error
**204** = Password write error
**205** = Login error
**206** = Reset error
**207** = Multiple write error
**208** = Multiple read error
**209** = Overflow block size

**Example: Set Registers 4 to 8 to read-protected.**

Before you perform this procedure, it's a good idea to write data to Registers 4 to 11 so that the effect can be seen. Use the wr command for this.

**Input:** du,0,#0184 <Return> Display data contents

```
Dump of the 4150 Tag
00:00000000 00000000 00000000 00000000
04:372e3230 11111111 30393a30 77777777
08:4a757070 20526f68 726d816c 6c657200
12:00000000 00000000 00000000 00000000
……..
ack,0,0,#0236
```

**Input:** li,0,00000000,#032x <Return> Login
**Input:** wr,0,1,00000804,#crc <Return>
This specifies that Registers 4 to 8 should be read-protected.

**Input:** du,0,#0184 <Return> Display data contents
We see that Register 1 contains the data 00000804. Registers 4 through 11 contain the data 372e3230, 11111111, 30393a30, 77777777, 4a757070, 20526f68, 726d816c, and 6c657200.

```
Dump of the 4150 Tag
00:00000000 00000804 00000000 00000000
04:372e3230 11111111 30393a30 77777777
08:4a757070 20526f68 726d816c 6c657200
12:00000000 00000000 00000000 00000000
……..
ack,0,0,#0236
```

**Input:** re,0,#0182 <Return> Reset
**Input:** du,0,#0184 <Return> Display data contents

```
Dump of the 4150 Tag
00:00000000 00000804 00000000 00000000
04:00000000 00000000 00000000 00000000
08:00000000 20526f68 726d816c 6c657200
12:00000000 00000000 00000000 00000000
……..
ack,0,0,#0236
```

The output shows us that only **00000000** was read for Registers 4 through 8 since these registers are read-protected.

**If you want to see the data in Registers 4 to 8, you must always log in before performing the read operation.**

**Input:** li,0,00000000,#032x <Return> Login
**Input:** du,0,#0184 <Return> Display data contents

```
Dump of the 4150 Tag
00:00000000 00000804 00000000 00000000
04:372e3230 11111111 30393a30 77777777
08:4a757070 20526f68 726d816c 6c657200
12:00000000 00000000 00000000 00000000
……..
ack,0,0,#0236
```

The data is visible again after logging in.

**Undo read protection for Registers 4 to 8.**

**Input:** li,0,00000000,#032x <Return> Login
**Input:** wr,0,1,00000000,#0371 <Return> Delete Reg. 1
**Input:** du,0,#0184 <Return> Display data contents

```
Dump of the 4150 Tag
00:00000000 00000000 00000000 00000000
04:372e3230 11111111 30393a30 77777777
08:4a757070 20526f68 726d816c 6c657200
12:00000000 00000000 00000000 00000000
…….
ack,0,0,#0236
```

## 8 Automation Studio transponder support

Automation Studio supports the use of transponder readers with the two libraries **AsUSB** and **DVFrame**.

### 8.1 AsUSB library

The AsUSB library allows you to search transponder readers on the target using the function blocks UsbNodeListGet and UsbNodeGet. The search is performed via the VendorID(0x67b) and the ProductID (0x2303). The ifName of the output structure from the UsbNodeGet function block can be used to communicate with the DVFrame library with the transponder reader.

C example of searching a transponder on the target

```
#define TRANSPONDER_VENDOR_ID    0x067b
#define TRANSPONDER_PRODUCT_ID 0x2303
#define TRANSPONDER_BCD           0x0300

_INIT void init(void)

{
int i;
/* search transponder */
    UsbNodeListGetFub.enable = 1;
    UsbNodeListGetFub.pBuffer = (UDINT)&usbNodeList;
    UsbNodeListGetFub.bufferSize = sizeof(usbNodeList);
    UsbNodeListGet(&UsbNodeListGetFub);
    if (UsbNodeListGetFub.status == OK && UsbNodeListGetFub.allNodes)
    {
       for (i=0;i<UsbNodeListGetFub.allNodes;i++)
       {
    UsbNodeGetFub.enable = 1;
    UsbNodeGetFub.nodeId = usbNodeList[i];
    UsbNodeGetFub.pBuffer = (UDINT)&usbDevice;
    UsbNodeGetFub.bufferSize = sizeof(usbDevice);
    UsbNodeGet(&UsbNodeGetFub);
    if (UsbNodeGetFub.status == OK )
    {
        /* TRANSPONDER ? */
if (usbDevice.vendorId == TRANSPONDER_VENDOR_ID
    && usbDevice.productId == TRANSPONDER_PRODUCT_ID
    && usbDevice.bcdDevice == TRANSPONDER_BCD)
   {
    /* TRANSPONDER found */
strcpy(StringDevice, usbDevice.ifName);
break;
}
}
} /* endfor */
    }
}
```

## 8.2 DVFrame library

The DVFrame library allows you to exchange data with the transponder reader 5E9000.29.

The DVFrame library function blocks enable data to be transmitted and received as "frames". The frame driver takes care of the hardware side of the data exchange, without modifying the data in the frame. This enables fast and easy application-specific communication.

Further information can be found in the section of the Automation Studio help about the DVFrame library

### FRM_xopen

To open the transponder reader device, the parameters FLAGS=2 and MODE=0 must be set using a parameter string in parameter mode.

```
       strcpy(StringMode, "/FLAGS=2 /MODE=0");
/* initialize open structure */
FrameXOpenStruct.device = (UDINT) StringDevice;
FrameXOpenStruct.mode = (UDINT) StringMode;
FrameXOpenStruct.config = (UDINT)0;
FrameXOpenStruct.enable = 1;
FRM_xopen(&FrameXOpenStruct);    /* open an interface */
```

### FRM_read

A frame end ID is implemented in the USB transponder driver. The frame terminator in the driver is 0x0a (LF). A complete frame including frame terminator 0x0a (LF) or the maximum length of a frame (=256 characters) is always received with the FRM_read function block. According to the transponder description, frames longer than 256 are possible (up to 319 bytes, e.g. command rdm,0,0,34,#crc). If the total length in the FRM_read is equal to 256 bytes, then the FRM_read function block must return up to a length < 256 bytes, if called. A maximum of 4 frames with a total length of 256 bytes per device can be buffered intermediately in the transponder driver. If multiple commands are sent with the FRM_write function block to the transponder and more than 4*256 bytes are received, then all additional data is rejected by the transponder reader in the driver. The sender of FRM_read and FRM_write does **not** receive an error message.

---

Information: Between two instructions you have to wait for the answer of the transponder reader/writer (ACK or NACK).

---

## Automation Studio C-Example:

```
/* header files */

#include <bur/plctypes.h>
#include <string.h>
#include <dvframe.h>

/* variable declaration */

_LOCAL FRM_xopen_typ FrameXOpenStruct;                      /* variable of type FRMXOPEN */
_LOCAL FRM_close_typ FrameCloseStruct;                      /* variable of type FRMCLOSE */
_LOCAL XOPENCONFIG XOpenConfigStruct;                       /* variable of type XOPENCONFIG */
_LOCAL FRM_gbuf_typ FrameGetBufferStruct;                   /* variable of type FRMGBUF */
_LOCAL FRM_rbuf_typ FrameReleaseBufferStruct;               /* variable of type FRMRBUF */
_LOCAL FRM_robuf_typ FrameReleaseOutputBufferStruct;        /* variable of type FRMROBUF */
_LOCAL FRM_write_typ FrameWriteStruct;                      /* variable of type FRMWRITE */
_LOCAL FRM_read_typ FrameReadStruct;                        /* variable of type FRMREAD */
_LOCAL USINT Error, Start, StartRead, Close, Stop,open;
_LOCAL UDINT Ident;                                         /* pointer for Ident info */
_LOCAL UDINT* SendBuffer;                                   /* pointer to send buffer */
_LOCAL UDINT* ReadBuffer;                                   /* pointer to read buffer */
_LOCAL UINT SendBufferLength, ReadBufferLength;
_LOCAL USINT StringDevice[30];                              /* initialize strings for FRM_xopen */
_LOCAL USINT WriteData[256], ReadData[256];                 /* arrays for write and read data */
```

```
/* initialize program starts here */

/* cyclic program start here */
_CYCLIC void CyclicProgram(void)
{
        if (Stop)
        {
                Start=0;
                StartRead=0;
        }
        if (open)
        {
                open=0;
                /* initialize start values */
                Error = 0;
/*              Start = 1; */
                StartRead = 0;
                Close = 0;
                Stop = 0;

                strcpy(StringDevice, "IF5.ST1");


                /* initialize open structure */
                FrameXOpenStruct.device = (UDINT) StringDevice;
                FrameXOpenStruct.config = (UDINT)0;
                FrameXOpenStruct.enable = 1;

                FRM_xopen(&FrameXOpenStruct);                   /* open an interface */
                Ident = FrameXOpenStruct.ident;                 /* get ident */

                if (FrameXOpenStruct.status != 0)               /* check status */

                {
                        Error = 1;                              /* set error level for FRM_xopen */

                }
        }


        if ((Error == 0) && (Start == 1))
        {
                Start = 0;

                /* initialize get buffer structure */
                FrameGetBufferStruct.enable = 1;
                FrameGetBufferStruct.ident = Ident;

                FRM_gbuf(&FrameGetBufferStruct);                /* get send buffer */


                SendBuffer = (UDINT*) FrameGetBufferStruct.buffer;      /* get adress of send buffer */
                SendBufferLength = FrameGetBufferStruct.bufIng;         /* get length of send buffer */

                if (FrameGetBufferStruct.status == 0)           /* check status */
                {
                        strcpy(WriteData, "tag,0,0,#crc\r\n");
                        memcpy(SendBuffer, WriteData, strlen(WriteData));  /* copy write data into send buffer */

                        /* initialize write structure */
                        FrameWriteStruct.ident = Ident;
                        FrameWriteStruct.buffer = (UDINT) SendBuffer;
                        FrameWriteStruct.bufIng = strlen(WriteData);
                        FrameWriteStruct.enable = 1;
                        FRM_write(&FrameWriteStruct);                        /* write data to interface */
                        /*StartRead = 1;*/
                        if (FrameWriteStruct.status != 0)                    /* check status */
                        {
                                /* initialize release output buffer structure */
                                FrameReleaseOutputBufferStruct.enable = 1;
                                FrameReleaseOutputBufferStruct.ident = Ident;
                                FrameReleaseOutputBufferStruct.buffer = (UDINT) SendBuffer;
                                FrameReleaseOutputBufferStruct.buflng = strlen(WriteData);
                                FRM_robuf(&FrameReleaseOutputBufferStruct);  /* release output buffer */
                        }
                }
                else
                {
                        strcpy(WriteData, "tag,0,0,#crc\r\n");
                }
        }

        if (StartRead == 1)
        {

                /* initialize read structure */
                FrameReadStruct.enable = 1;
                FrameReadStruct.ident = Ident;

                FRM_read(&FrameReadStruct);                                          /* read data form interface */

                ReadBuffer = (UDINT*) FrameReadStruct.buffer;                /* get adress of read buffer */
                ReadBufferLength = FrameReadStruct.buflng;                   /* get length of read buffer */

                if (FrameReadStruct.status == 0)                             /* check status */
                {
                        memset (ReadData,0,sizeof(ReadData));
                        memcpy(ReadData, ReadBuffer,ReadBufferLength); /* copy read data into array */

                        /* initialize release buffer structure */
                        FrameReleaseBufferStruct.enable = 1;
                        FrameReleaseBufferStruct.ident = Ident;
                        FrameReleaseBufferStruct.buffer = (UDINT) ReadBuffer;
                        FrameReleaseBufferStruct.buflng = ReadBufferLength;

                        FRM_rbuf(&FrameReleaseBufferStruct);                 /* release read buffer */

                        StartRead = 0;
                        /*Start = 1; */
                }
                else
                {
                        memset (ReadData,0,sizeof(ReadData));
                }

        }
}
```

## 8.3 General technical data

| | |
|---|---|
| Transponder reader/writer | For transponders 4102 and 4150<br>amplitude modulation, carrier frequency 125kHz |
| Supply voltage | 5 VDC ± 20% (via USB) |
| Power consumption | Approx. 0.15 Watt |
| Read/write range in air | Min. 16 mm |
| Protection | IP65 (front side) |
| Environmental temperature<br>Operation<br>Storage | <br>0 to 50 ℃<br>-20 ℃ to +60 ℃ |
| Humidity<br>Operation<br>Storage | 5% to 85%, non-condensing<br>T = 40 ℃: 5% to 90%, non-condensing<br>T > 40 ℃: < 90%, non-condensing |