Reading and Writing RFID Data with SIMATIC S7-300/400 and SIMATIC RF670R

SIMATIC RF670R

Application Description • April 2013

Applications & Tools

Answers for industry.

SIEMENS

Siemens Industry Online Support

This document is taken from the Siemens Industry Online Support. The following link takes you directly to the download page of this document:

http://support.automation.siemens.com/WW/view/en/23626344

Caution

The functions and solutions described in this entry predominantly confine themselves to the realization of the automation task. Please also take into account that corresponding protective measures have to be taken in the context of Industrial Security when connecting your equipment to other parts of the plant, the enterprise network or the Internet. For more information, please refer to Entry ID 50203404.

http://support.automation.siemens.com/WW/view/en/50203404

SIEMENS

SIMATIC

Reading from and Writing to Transponders with RF670R

Application Description

Warranty and Liability

Note

The application examples are not binding and do not claim to be complete regarding the circuits shown, equipping and any eventuality. The application examples do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible for ensuring that the described products are correctly used. These application examples do not relieve you of the responsibility of safely and professionally using, installing, operating and servicing equipment. When using these application examples, you recognize that Siemens cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these application examples at any time without prior notice. If there are any deviations between the recommendations provided in these application examples and other Siemens publications – e.g. Catalogs – then the contents of the other documents have priority.

We do not accept any liability for the information contained in this document.

Any claims against us – based on whatever legal reason – resulting from the use of the examples, information, programs, engineering and performance data etc. described in this application example shall be excluded. Such an exclusion shall not apply in the case of mandatory liability, e.g. under the German Product Liability Act (“Produkthaftungsgesetz”), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract (“wesentliche Vertragspflichten”). However, claims arising from a breach of a condition which goes to the root of the contract shall be limited to the foreseeable damage which is intrinsic to the contract, unless caused by intent or gross negligence or based on mandatory liability for injury of life, body or health. The above provisions do not imply a change in the burden of proof to your detriment.

It is not permissible to transfer or copy these application examples or excerpts of them without first having prior authorization from Siemens Industry Sector in writing.

Table of Contents

1 Problem

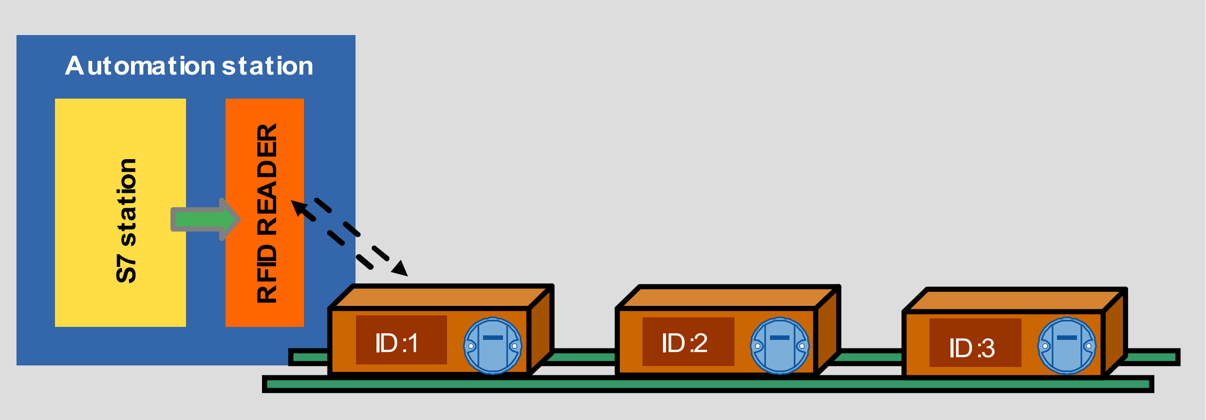
Introduction

For incoming goods, stock keeping, production logistics and distribution, RFID (Radio Frequency Identification) provides complete tracking and documentation of all delivered, stored and sent goods. For this purpose, a small data medium – referred to as a transponder or tag – that stores all essential information is attached to each product, package or pallet.
A read/write device is used to read from and write to the transponder.
To ensure complete tracking and documentation or to benefit from the use of RFID technology even during production, the connection to automation systems is required in many cases.

Overview of the automation problem

The figure below provides an overview of the automation problem.

Figure 1-1



The high-performance RF670R reader from the RFID UHF system with

- integrated processing logic that allows comprehensive filter functions
- reliable identification at large intervals
- reliable detection of fast-moving transponders and
- an Ethernet interface (XML Protocol via TCP/IP) for easy connection to the PC or IT level

is to be connected to an S7-300/400 CPU with PROFINET interface and implement the reading from and writing to transponders.

2 Solution

2.1 Overview of the overall solution

Introduction

This application example shows you

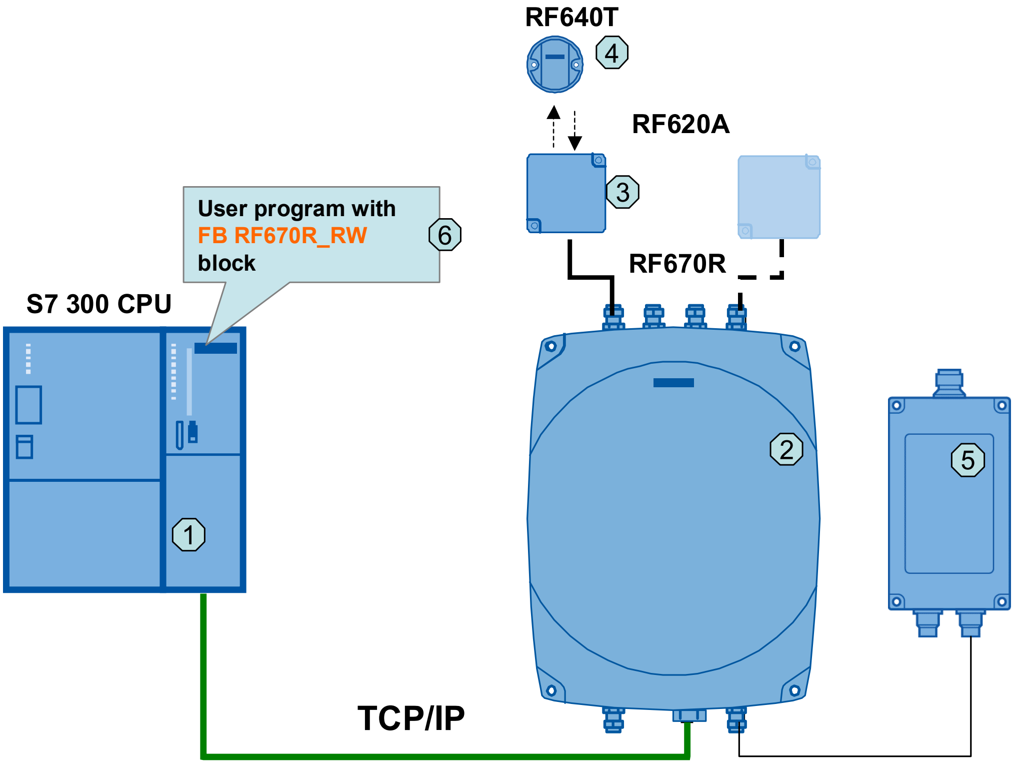
- how to parameterize the RF670R reader using RF-MANAGER Basic 2010 regarding the reading and writing of data
- how to program a block in your S7-300/400 CPU with PROFINET interface to implement the following core functions:
  - Connect to/disconnect from reader
  - Read/write transponder ID
  - Read/write RFID data
- how to format SIMATIC S7 variable types for the RF670R.

The complete functionality is encapsulated in an open SCL block and can be used as a basis for your own developments such as other RF670R functions.

Diagrammatic representation

The diagrammatic representation below shows the most important components of the solution with an S7-300 CPU with PROFINET interface. Alternatively, the solution can also be implemented with another S7-300/400 CPU with PROFINET interface that supports TCP (see \3\).

Figure 2-1



Components included

Table 2-1

No.	Component	Description
1	PROFINET S7 CPU	An S7-300 CPU, S7-400 CPU
2	SIMATIC RF670R	Reader
3	SIMATIC RF620A	Antenna
4	SIMATIC RF640T	Transponder
5	Wide-range power supply unit with Euro-plug	
6	Communication block for the reader	FB 670

Scope

This application does not include the basics of

- SIMATIC RF600. For more information, refer to document \4\ in References.
- SIMATIC RF670R. For more information, refer to document \5\ in References.
- the LAD/ FBD/ STL/ SCL programming languages.

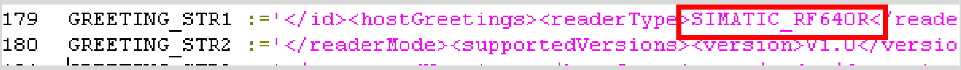
Basic knowledge of these topics is required.

Validity of application V1.1

- All PROFINET S7-300/400 CPUs from the SIMATIC product range that support TCP (see \3\)
- STEP 7 V5.5
- SIMATIC RF670R, RF640R

Note

If you use an RF640R reader, you have to change in the FB RF670R_RW (FB670) the Reader-type in hostGreetings Message.



```
GREETING_STR1 :='</id><hostGreetings><readerType>SIMATIC_RF640R</reade
GREETING_STR2 :='</readerMode><supportedVersions><version>V1.0</versio
```

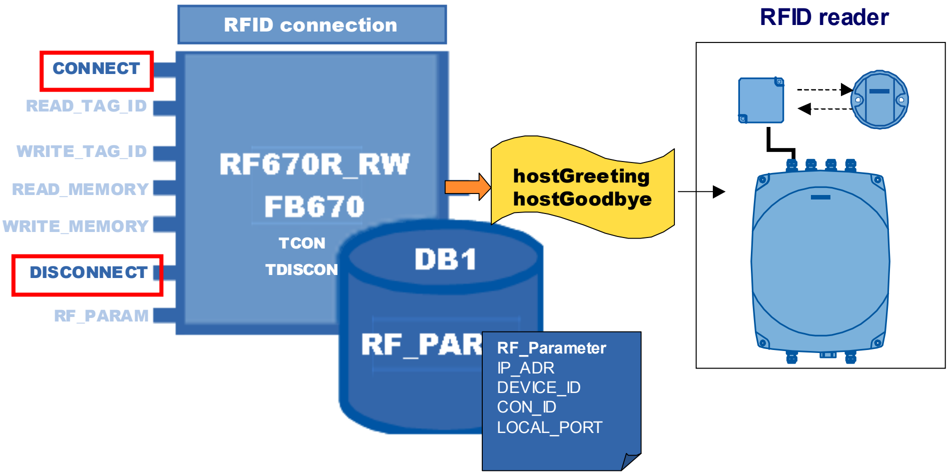
2.2 Description of the core functionality

In this example, the following functions are implemented with a user block programmed in SCL: FB RF_670R_RW (FB 670):

- Establish or terminate the connection between a PROFINET S7-300/400 station and the RF670R RFID reader
- Read transponder ID
- Write transponder ID
- Read RFID data from transponder
- Write RFID data to transponder

“Connect or disconnect”

Figure 2-2

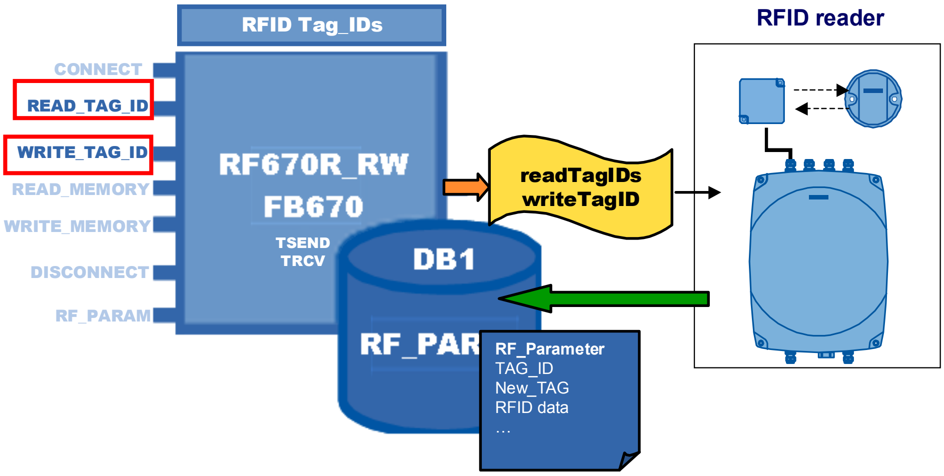


Via the “CONNECT” input, the TCP/IP connection to the RF670 reader is established and the hostGreetings message is sent.

Via the “DISCONNECT” input, the hostGoodbye message is sent to the RF670 reader and the TCP/IP connection is terminated.

“Read or write transponder IDs”

Figure 2-3

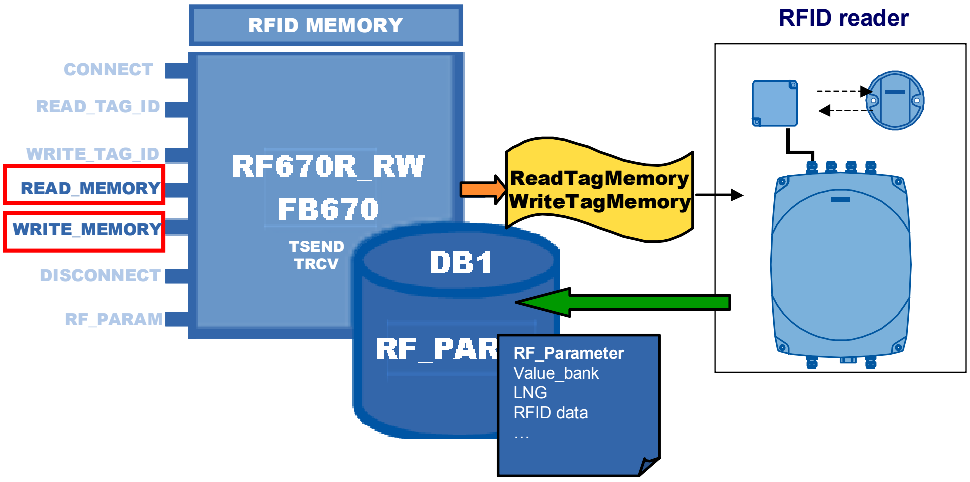


Via the “READ_TAG_ID” input, all transponder IDs of the transponders (TAGs) detected in the field of the reader are read in.

Via the “WRITE_TAG_ID” input, a new transponder ID is written to the transponder detected first by the reader.

"Read RFID data from or write RFID data to transponder"

Figure 2-4



Via the READ_MEMORY input, the RFID data is read out of the transponder memory.

Via the WRITE_MEMORY input, the RFID data is written to the transponder memory.

Advantages of this solution

The code of the FB "RF670R_RW" block (FB670)

- already includes the above-described functions on a fully implemented basis
- can be easily adapted for extensions such as other RF670R functions.

2.3 Hardware and software components used

The application was created with the following components:

Hardware components

Table 2-2

Component	Qty.	Order no.	Note
PS307 10A	1	6ES7307-1KA01-0AA0	
CPU 315-2 PN/DP	1	6ES7315-2EH14-0AB0	Alternatively, another CPU with PROFINET interface that supports TCP can also be used.
SIMATIC RF670R (reader)	1	6GT2811-0AB00-0AA0 (EU) 6GT2811-0AB00-1AA0 (USA) 6GT2811-0AB00-2AA0 (CHINA)	Alternatively, RF640R can also be used.
Connecting cable for power units	1	6GT2891-0NH50	5 m
SIMATIC RF640T	n	6GT2810-0DC00	

Component	Qty.	Order no.	Note
(transponder)			
SIMATIC RF620A (antenna)	1	6GT2812-1EA00	Please note: ETSI frequency. Up to four antennas per RF670R
Antenna cable	1	6GT2815-0BH30	3 m
Wide-range power supply unit	1	6GT2898-0AA00 (EU) 6GT2898-0AA10 (UK) 6GT2898-0AA20 (US)	

Standard software components

Table 2-3

Component	Qty.	Order no.	Note
STEP 7 V5.5	1	6ES7810-4CC08-0YA5	
S7-SCL V5.3+SP5	1	6ES7811-1CC05-0YA5	
SIMATIC RF-MANAGER Basic 2010, V2 or V3	1	Included in the scope of delivery of the RF670R	

Sample files and projects

The following list contains all files and projects that are used in this example.

Table 2-4

Component	Note
23626344_RF670R_CODE_V11.zip	This zip file contains the STEP 7 project.
23626344_Application_RF670R_DOKU_V11_en.pdf	This document.

2.4 Performance data

The following section gives you an overview of the size of the blocks of the STEP 7 project in the main memory.

Blocks used and resource requirements

The size of all program blocks in the main memory is 36,342 Bytes. They are composed as follows.

Table 2-5

Block	Symbol	Description	Size in main memory (bytes)	Classification
FB670	RF670R_RW	Function block for reading from and writing to transponders	23442	In-house development
DB670	IDB_RF670R_RW	Instance DB FB670	3442	
DB1	RF_PARAM	Global data block for parameterizing the RF670R function block (FB670)	644	
FB1	CALL_RF670R	Call of the FB670 function block with individual logic	250	
DB3	IDB_CALL_RF670R	Instance DB FB1	46	
FB2	INT_TO_HEX	Function block for converting INT to HEX format	1998	
DB4	IDB_INT_HEX	Instance DB FB2	448	
FB3	HEX_TO_INT	Function block for converting HEX format to INT	2120	
DB5	IDB_HEX_INT	Instance DB FB3	200	
DB2	In_Out_Data	Global data block for the converted data	40	
UDT 670	RF_VAR	Data structure that contains the parameters necessary for RF670R_RW (FB670)	-	
UDT 65	TCON_PAR	Data structure that contains the parameters necessary to establish the connection	-	Standard Library/ communication blocks
FB 65	TCON	Block for setting up and establishing the communication connection	1018	
FB 63	TSEND	TSEND (FB 63) sends data to the addressed remote partner via TCP/IP	416	
FB 64	TRCV	TRCV (FB 64) receives data via TCP/IP	472	
FB 66	TDISCON	This block disconnects a communication connection to a communication partner	230	

Block	Symbol	Description	Size in main memory (bytes)	Classification
FC 2	CONCAT	This function combines two STRING variables in one string	358	Standard Library/ IEC function blocks
FC 4	DELETE	This function deletes characters in a string	414	
FC10	EQ_STRNG	This function compares the contents of two strings	152	
FC 16	I_STRNG	The FC 16 function converts a variable in INT format to a string	264	
FC 21	LEN	The FC 21 function outputs the current length of a string (number of valid characters) as a return value	76	
FC 38	STRNG_I	The FC 38 function converts a string to a variable in INT format	330	
SFB4	TON	Generates a switch-on delay	-	Standard Library/ system function blocks
SFC20	BLKMOV	Copies the contents of a memory area	-	
SFC21	FILL	Pre-fills a memory area (destination field) with the contents of another memory area (source field)	-	

Local data memory requirements

Table 2-6

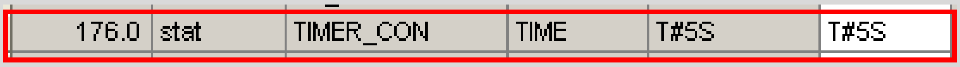
Block	Symbol	Local data
FB670	RF670R_RW	148 bytes
FB1	CALL_RF670R	6 bytes
FB2	INT_TO_HEX	48 bytes
FB3	HEX_TO_INT	22 bytes
FB 65	TCON	28 bytes
FB 63	TSEND	24 bytes
FB 64	TRCV	24 bytes
FB 66	TDISCON	12 bytes

Watchdog timer

A watchdog timer monitors connection establishment to the RF670R reader. It is preset to 5 s.

Note

If you want to change the time, you can enter the value directly in the instance data block of FB RF670R_RW (FB670).



176.0	stat	TIMER_CON	TIME	T#5S	T#5S

Application software

The following table shows the measured runtimes of the functions in the S7-300 CPU from the test setup, the size of the receive buffer and the maximum number of read transponder IDs.

Table 2-7

Criterion	Value
Connect runtime (CONNECT)	3 ms
Read transponder IDs runtime (READ_TAG_IDs)	10 ms
Write transponder ID runtime (WRITE_TAG_ID)	3 ms
Read RFID data runtime (READ_MEMORY)	8 ms
Write RFID data runtime (WRITE_MEMORY)	5 ms
Disconnect runtime (DISCONNECT)	3 ms
Maximum runtime of the sample program	12 ms
Receive buffer size	1500 bytes
Maximum number of read transponder IDs	5 **(*1)**

(*1): The number of stored transponder IDs depends on the field length of REC_DATA in DB RF_PARAMETER. If you want to read more transponder IDs, you have to adjust the receive buffer length (see Chapter 6.1).

3 Functional Mechanisms of this Application

Introduction

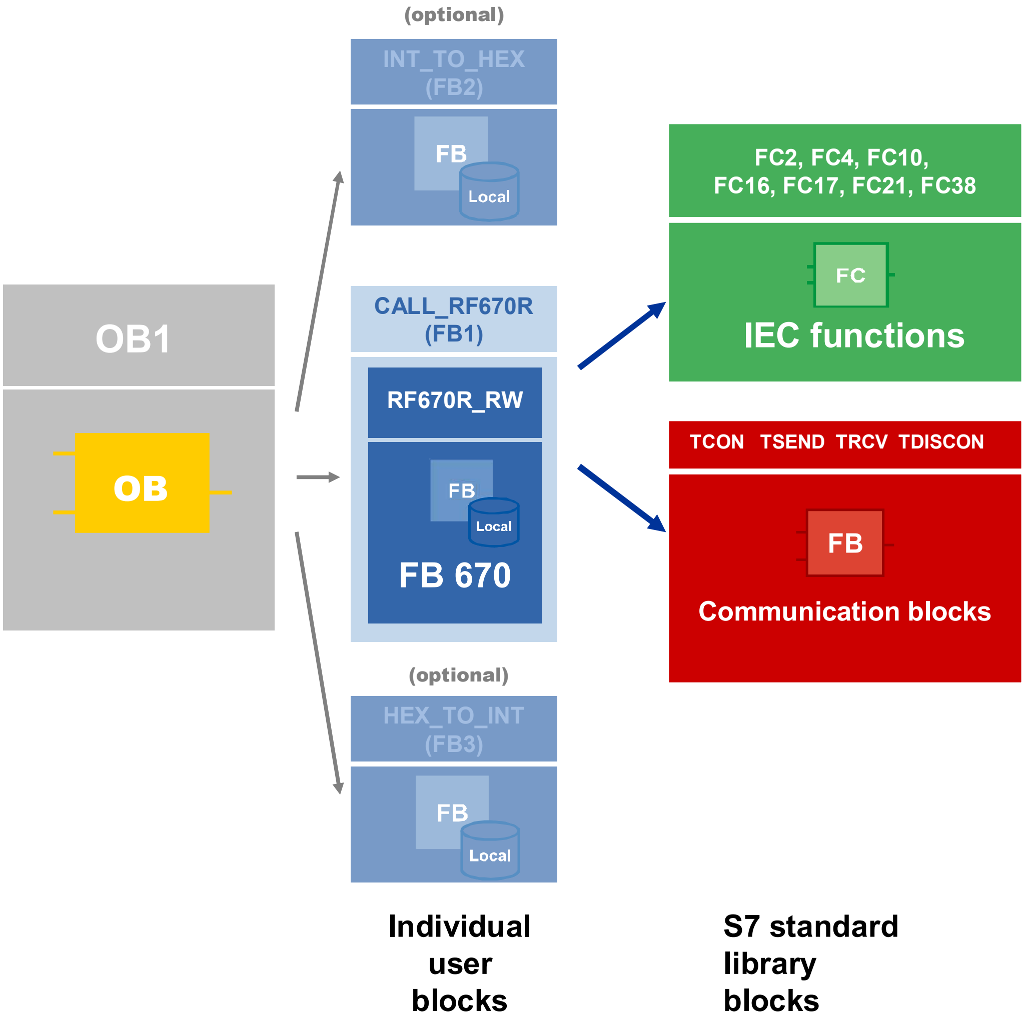
This chapter provides a detailed description of the FB RF670R_RW user block (FB670) in terms of the internal functional sequences and programming.

3.1 Program overview

Diagrammatic representation

The figure below shows the program structure of the entire STEP 7 project.

Figure 3-1



3.1.1 FB RF670R_RW parameter

The following figure and table show the call interface of the FB RF670R_RW user block (FB670).

Figure 3-2

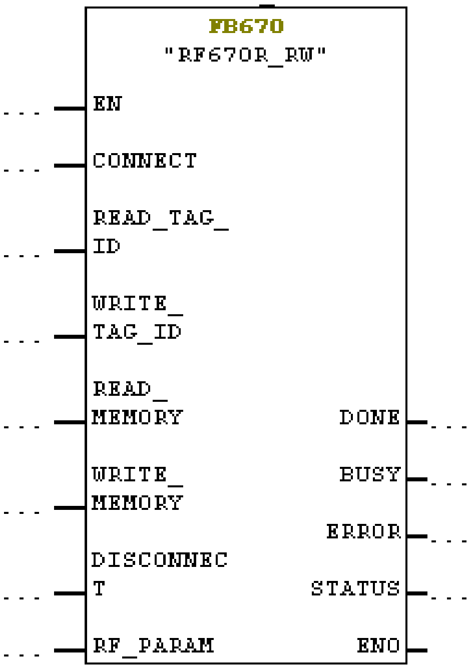


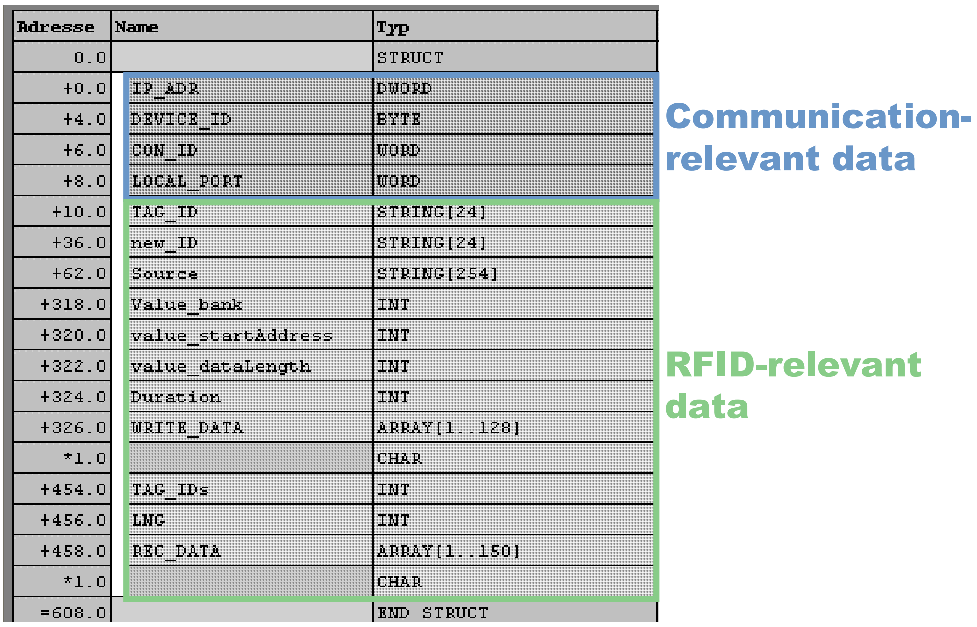
Table 3-1

Symbol	Data type	Explanation
EN	BOOL	Enable input. Relevant only in FBD and LAD representation.
CONNECT	BOOL	• Activates the connection establishment routine to the reader • Reacts to a positive edge
READ_TAG_ID	BOOL	• Activates the routine for reading the transponder IDs of the tags in the field • Reacts to a positive edge
WRITE_TAG_ID	BOOL	• Activates the routine for writing the transponder ID of the transponder in the field • Reacts to a positive edge
READ_MEMORY	BOOL	• Activates the routine for reading out the RFID data • Reacts to a positive edge
WRITE_MEMORY	BOOL	• Activates the routine for writing the RFID data • Reacts to a positive edge
DISCONNECT	BOOL	• Activates the connection termination routine to the reader • Reacts to a positive edge

Symbol	Data type	Explanation
RF_PARAM	STRUCT	Via the RF_VAR UDT, it contains the parameters of the TCP connection and all parameters required for RFID data processing (see Figure 3-3 UDT RF_VAR).
DONE	BOOL	TRUE when the last job has been completed without errors. Set to FALSE if a new routine is activated.
BUSY	BOOL	Set to TRUE when the RF670R_RW block is active. Set to the FALSE status as soon as the operation is completed or an error occurs.
ERROR	BOOL	TRUE if an error occurs when executing the routine Set to FALSE if a new routine is activated. Default value: FALSE
STATUS	DWORD	Status if ERROR=TRUE Set to DW#16#00 if a new routine is activated.
ENO	BOOL	Enable output. Relevant only in FBD and LAD representation.

3.1.2 RF_VAR structure

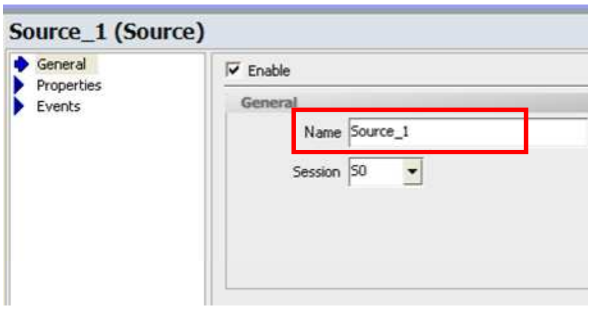
Figure 3-3 UDT RF_VAR



Adresse	Name	Typ
0.0		STRUCT
+0.0	IP_ADR	DWORD
+4.0	DEVICE_ID	BYTE
+6.0	CON_ID	WORD
+8.0	LOCAL_PORT	WORD
+10.0	TAG_ID	STRING[24]
+36.0	new_ID	STRING[24]
+62.0	Source	STRING[254]
+318.0	Value_bank	INT
+320.0	value_startAddress	INT
+322.0	value_dataLength	INT
+324.0	Duration	INT
+326.0	WRITE_DATA	ARRAY[1..128]
*1.0		CHAR
+454.0	TAG_IDs	INT
+456.0	LNG	INT
+458.0	REC_DATA	ARRAY[1..150]
*1.0		CHAR
=608.0		END_STRUCT

Table 3-2 Description of the parameters of the RF_VAR structure

Parameter	Description
IP_ADR (DWORD)	IP address of the RFID reader
DEVICE_ID (BYTE)	DEVICE_ID of the S7 CPU (see \6\)
CON_ID (BYTE)	Unique connection ID for the TCP connection
LOCAL_PORT (WORD)	Local port

Parameter	Description
TAG_ID (STRING)	ID of the transponder to be read from/written to. (Hex format example of a 96-bit tag ID: 3005FB63AC1F3681EC880468)
new_TAG (STRING)	The new transponder ID with which the transponder is to be written to (Hex format example of a 96-bit tag ID: 3005FB63AC1F3681EC880468)
Source (STRING)	Name of the data source. This entry must match the entry in RF-MANAGER Basic.  Source_1 (Source) ☑ Enable General Name Source_1 Session S0
Value_bank (INT)	Memory area of the transponder: 0 Reserved 1 EPC 2 TID 3 USER MEMORY Default value: 3.
value_startAddress (INT)	Number of the first byte in the memory area of the transponder
value_dataLength (INT)	Number of bytes to be read in. The maximum length of the data to be read per job is 64 bytes.
WRITE_DATA (ARRAY [1..128] of CHAR)	Data to be written to the memory area of the transponder. Each nibble (half a byte) must be represented as a hexadecimal character (see Figure 3-4). Example: You want to write the bytes 0x12 and 0x34 to the memory area of the transponder. To do so, you have to store the bytes as characters in DB RF_PARAMETER as follows: RF_VAR.WRITE_DATA[1]:=‘1’ RF_VAR.WRITE_DATA[2]:=‘2’ RF_VAR.WRITE_DATA[3]:=‘3’ RF_VAR.WRITE_DATA[4]:=‘4’ In this example, value_dataLength has the value 2.
Duration (INT)	Read duration in ms. Indicates how long the reader must read the transponder IDs. The command will be active for the entire duration.
TAG_Ids (INT)	Number of transponder IDs in the field that were read by the reader.
LNG (INT)	Length of the received byte data **(*1)**

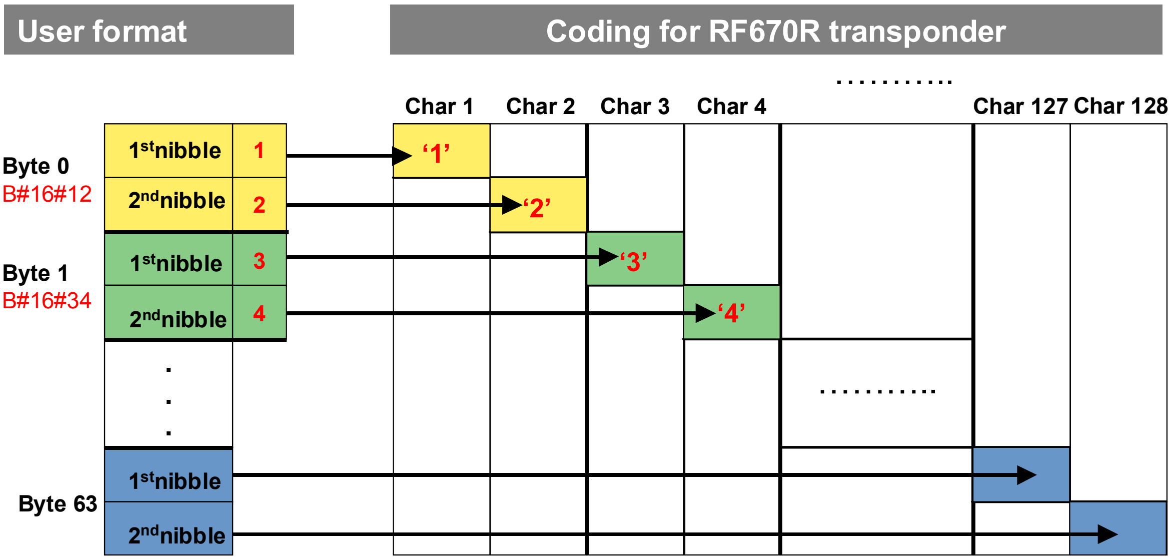
Parameter	Description
REC_DATA (ARRAY [1..150] of CHAR)	Received data. Example: RF_VAR.REC_DATA [1]:='1' RF_VAR.REC_DATA [2]:='2' RF_VAR.REC_DATA [3]:='3' RF_VAR.REC_DATA [4]:='4'. The LNG of the RFID data has the value 2 and the received data has the values 0x12, 0x34 **(*2)**.

(*1): If you receive more than one transponder ID, the length of the last transponder ID will be copied to the LNG field.

(*2): Max. up to 5 transponder IDs can be read for this field length. If you want to read more transponder IDs, you have to adjust this length (see Chapter 6.1).

The following figure shows how each nibble (half a byte) is to be represented as a hexadecimal character.

Figure 3-4 Coding specification for RF670R transponder

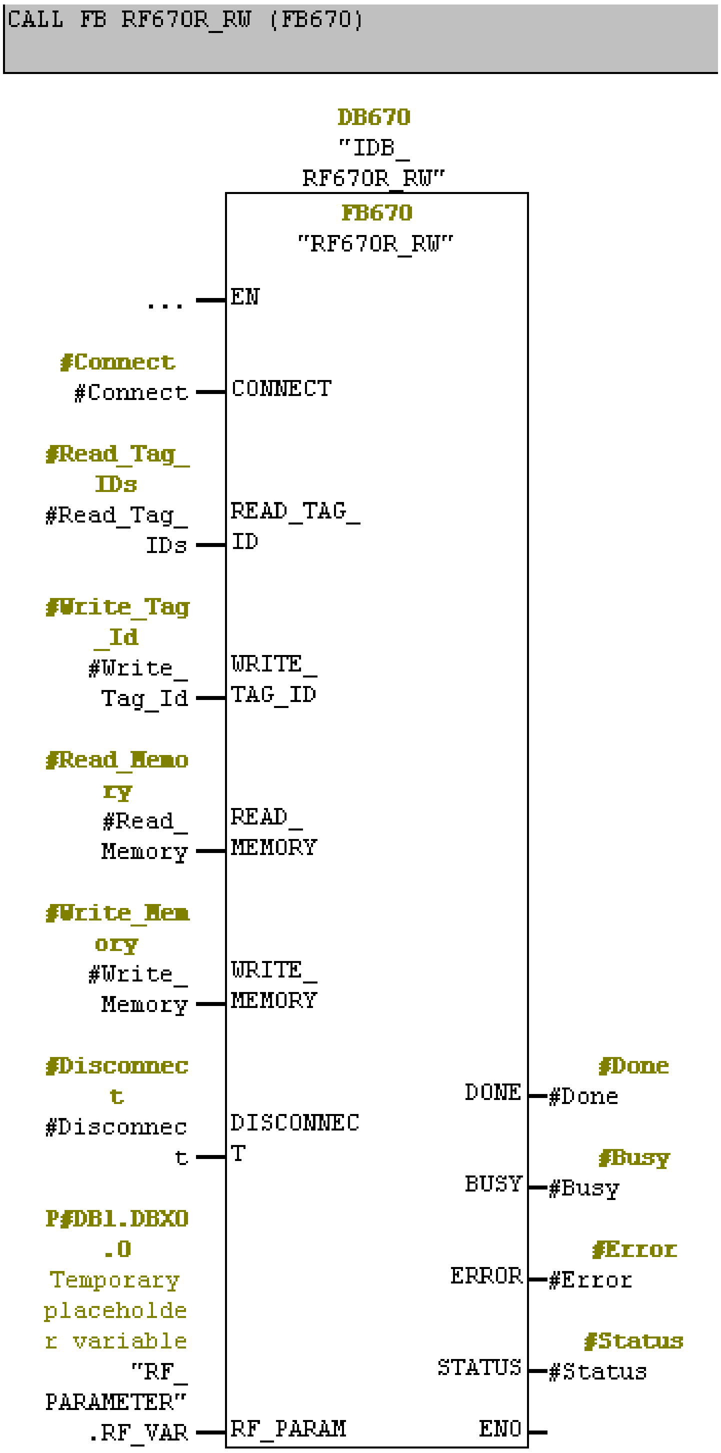


Note For a more detailed overview of the RFID-relevant parameters, please refer to the RF670R Function Manual (/5/, Chapter 3).

3.1.3 Call example: FB RF670R_RW (FB670) in FB CALL_RF670R (FB1)

Function block FB RF670R_RW (FB670) is called in FB CALL_RF670R (FB1). FB CALL_RF670R is called cyclically in OB1. The figure below shows the call of FB RF670R_RW in FB CALL_RF670R (FB1).

Figure 3-5 Call of FB RF670R_RW (FB670)



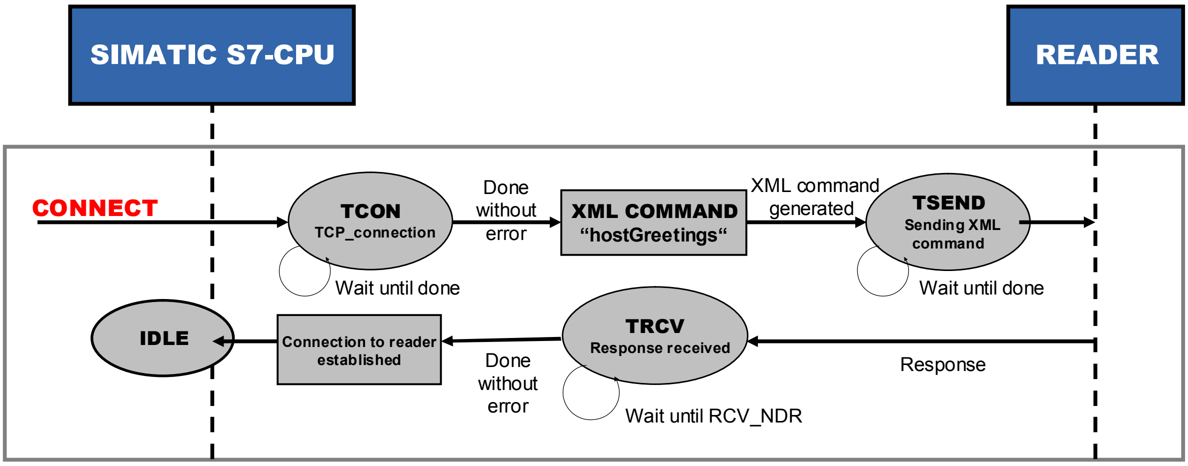
CALL FB RF670R_RW (FB670)

3.1.4 State diagram of FB RF670R_RW (FB670)

The following figures show the state diagrams of the above-described functions. The figures are read from left to right.
The S7 station sends XML commands to the RF670R reader. The RF670R reader responds with an XML message that contains the required data. This message is evaluated and stored to the respective fields of DB RF_PARAMETER.

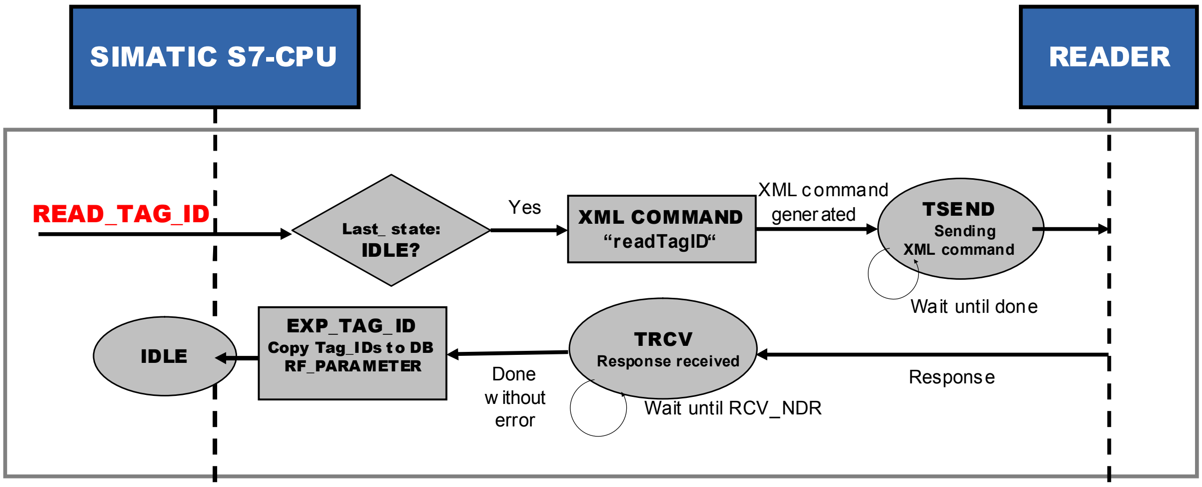
Connect

Figure 3-6



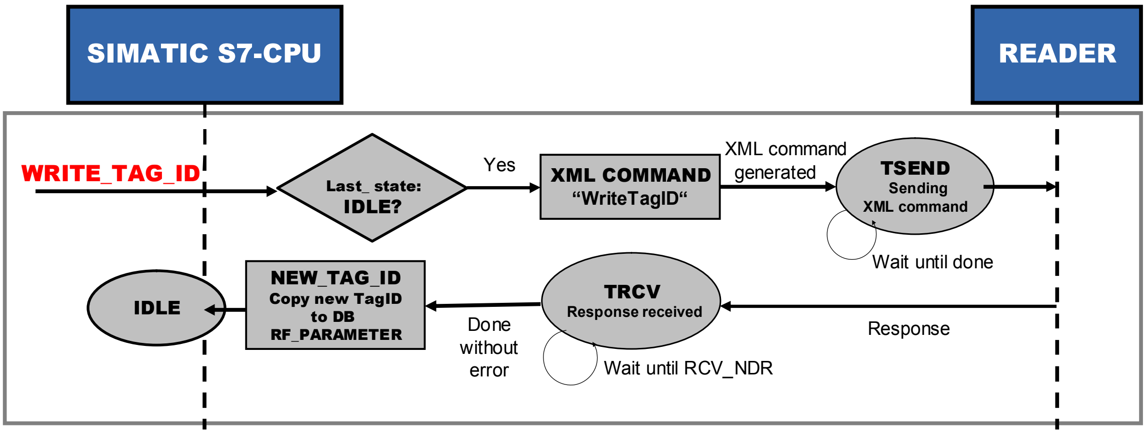
Read transponder IDs

Figure 3-7



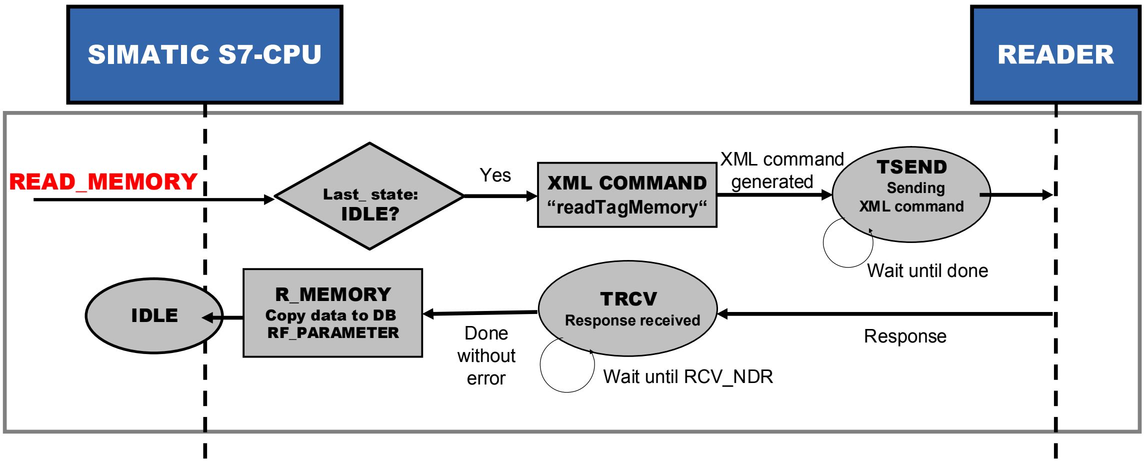
Write transponder ID

Figure 3-8



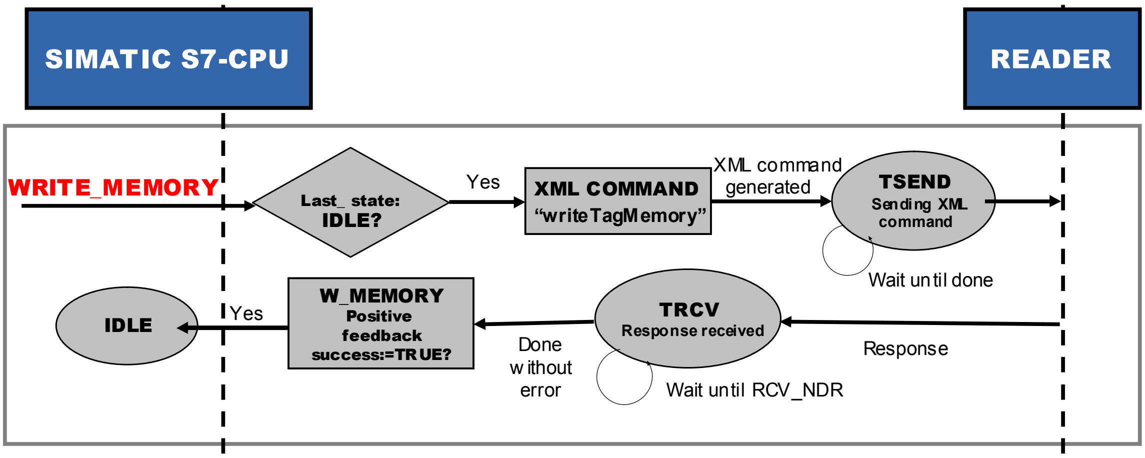
Read RFID data

Figure 3-9



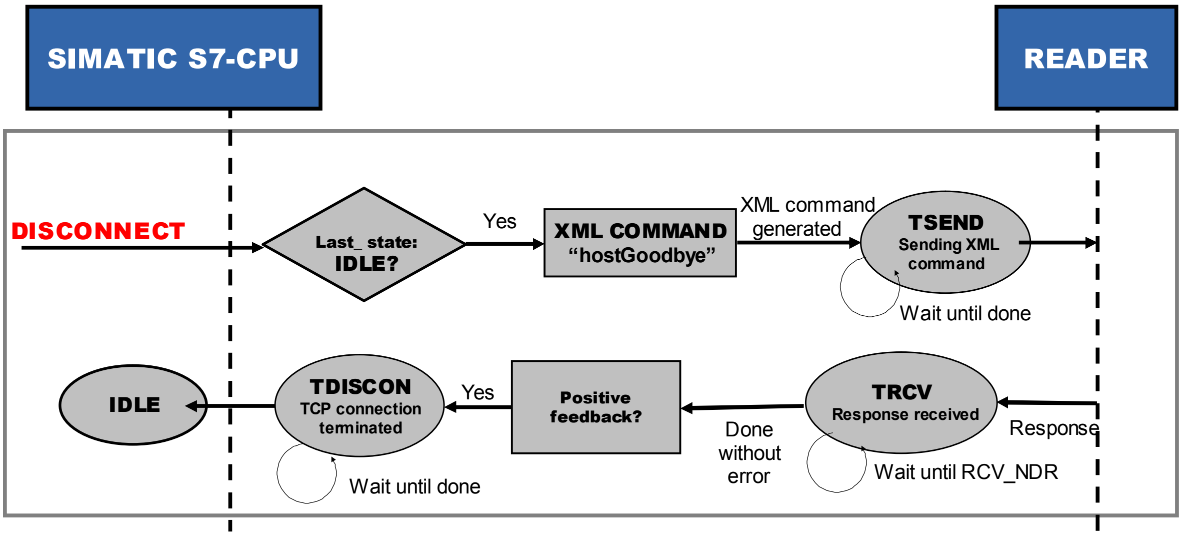
Write RFID data

Figure 3-10



Disconnect

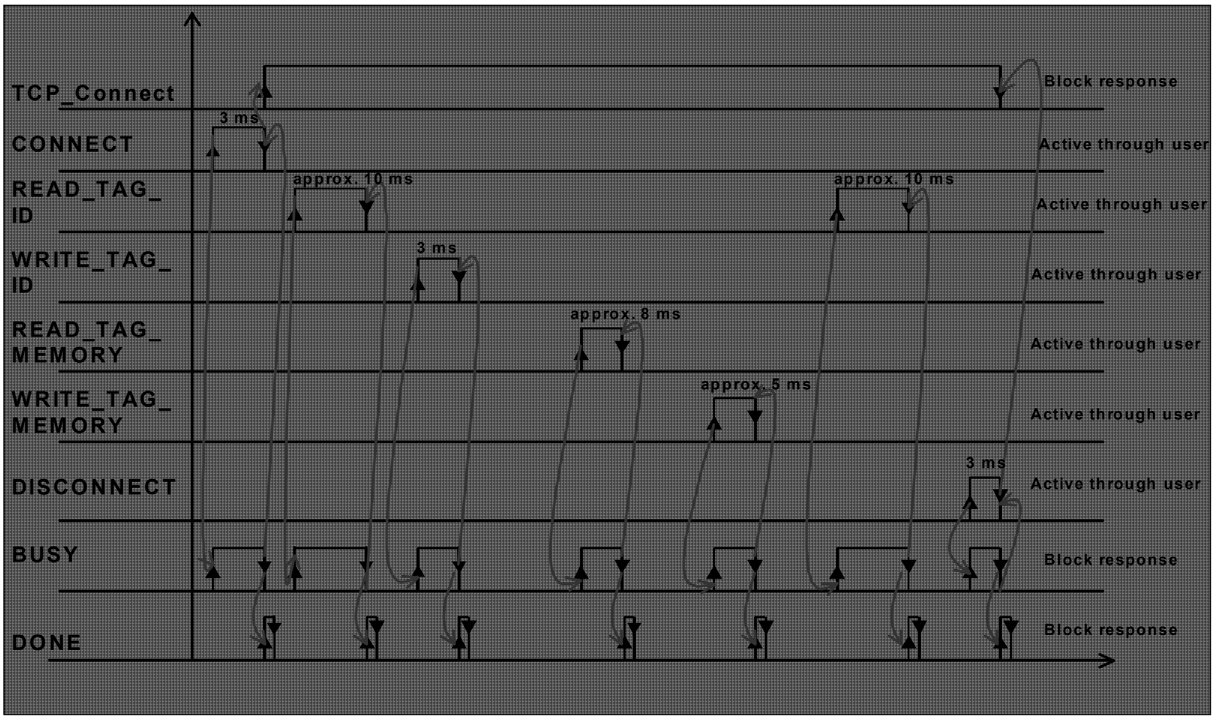
Figure 3-11



3.1.5 Function chart

The following chart shows the graphical representation of the time sequences in the FB RF670R_RW function block (FB670).

Figure 3-12



3.2 Explanation of the functions implemented in FB RF670R_RW

In the following chapters, we show you the details of the functions implemented in FB RF670R_RW (FB670). We explain the most important SCL code fragments. For a complete overview, the well-documented SCL code is available to you in the project.

3.2.1 “Connect or disconnect”

Overview of functions

Via the CONNECT input parameter, the FB RF670R_RW function block establishes the TCP/IP connection to the RF670R reader and sends the hostGreetings message.

Via the DISCONNECT input parameter, the FB RF670R_RW function block sends the hostGoodbye message and terminates the TCP/IP connection to the RF670 reader.

Sequence of the function

The diagrammatic representation below shows the steps FB RF670R uses to establish/terminate a connection to the reader.

Figure 3-13

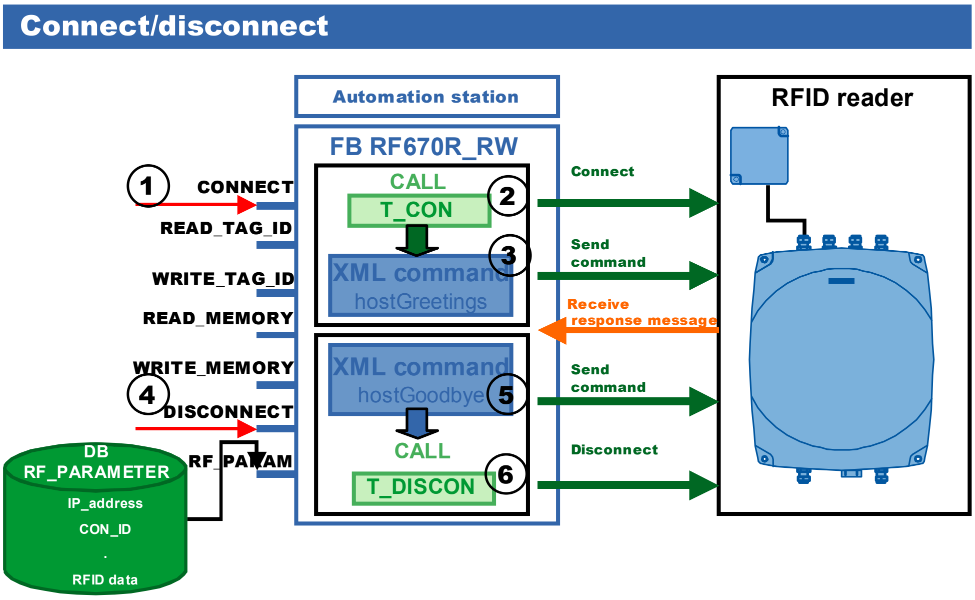


Table 3-3

No.	Description
1	CONNECT is triggered.

No.	Description
2	The variables that are necessary to establish the TCP connection are read out of DB RF_PARAMETER. The TCON block establishes the connection.
3	The XML-formatted “hostGreetings” command is sent to the reader with TSEND. If FB RF670R_RW receives positive feedback from the reader via TRCV, connection establishment has been successfully completed.
4	DISCONNECT is triggered.
5	The XML-formatted “hostGoodbye” command is sent to the reader with TSEND. If FB RF670R_RW receives positive feedback from the reader via TRCV, TDISCON will be called.
6	The TDISCON block terminates the TCP connection.

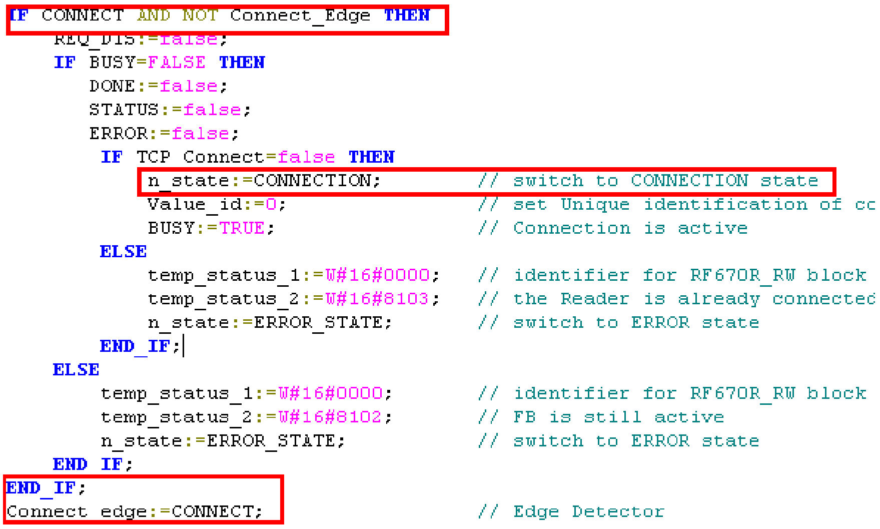
Program details

In this section, we show you the most important code fragments of this function from the documented source code of this example.

1. Triggering the “CONNECT” input

In the SCL code, a positive edge is generated to check the signal changes of the “CONNECT” input. When a rising edge of the input is detected, the block will start establishing the connection.

Figure 3-14

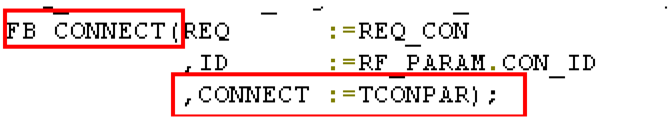


```
IF CONNECT AND NOT Connect_Edge THEN
    REQ_DIS:=false;
    IF BUSY=FALSE THEN
       DONE:=false;
       STATUS:=false;
       ERROR:=false;
        IF TCP Connect=false THEN
            n_state:=CONNECTION;          // switch to CONNECTION state
            Value_id:=0;                  // set Unique identification of cc
            BUSY:=TRUE;                   // Connection is active
        ELSE
            temp_status_1:=W#16#0000;     // identifier for RF670R_RW block
            temp_status_2:=W#16#8103;     // the Reader is already connectec
            n_state:=ERROR_STATE;         // switch to ERROR state
        END_IF;
    ELSE
        temp_status_1:=W#16#0000;         // identifier for RF670R_RW block
        temp_status_2:=W#16#8102;         // FB is still active
        n_state:=ERROR_STATE;             // switch to ERROR state
    END IF;
END_IF;
Connect edge:=CONNECT;                    // Edge Detector
```

2. Establishing the TCP connection

To parameterize the communication connections for TCP, the data structure from UDT 65 “TCONPAR” is created in the instance DB of FB RF670R_RW. This data structure contains the parameters you need to establish the connection. The TCON block is called and the TCP connection is established.

Figure 3-15

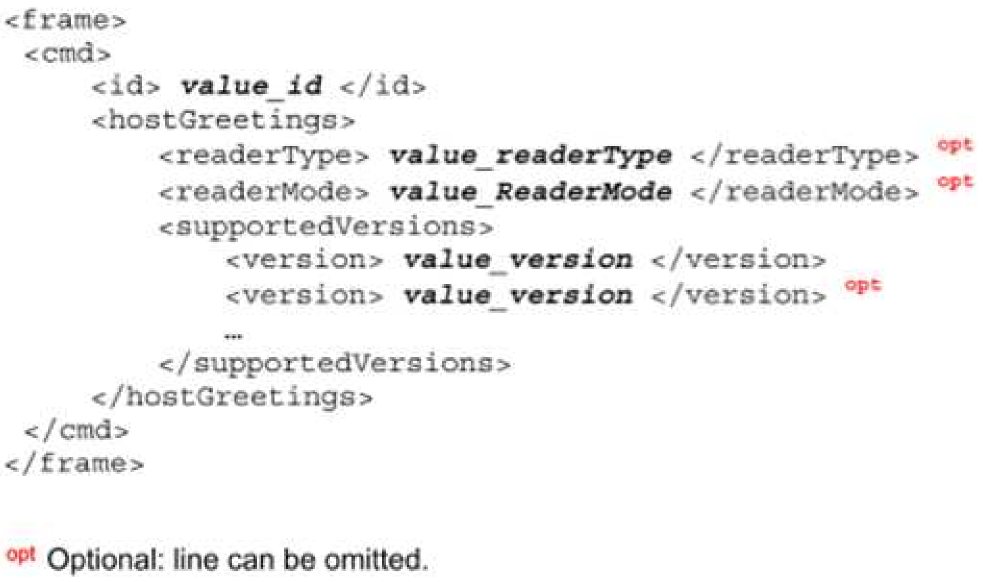


```
FB_CONNECT(REQ      :=REQ_CON
          ,ID       :=RF_PARAM.CON_ID
          ,CONNECT  :=TCONPAR);
```

3. Formatting the “hostGreetings” XML_command and sending it to the reader with TSEND

The “hostGreetings” command must have the following structure:

Figure 3-16

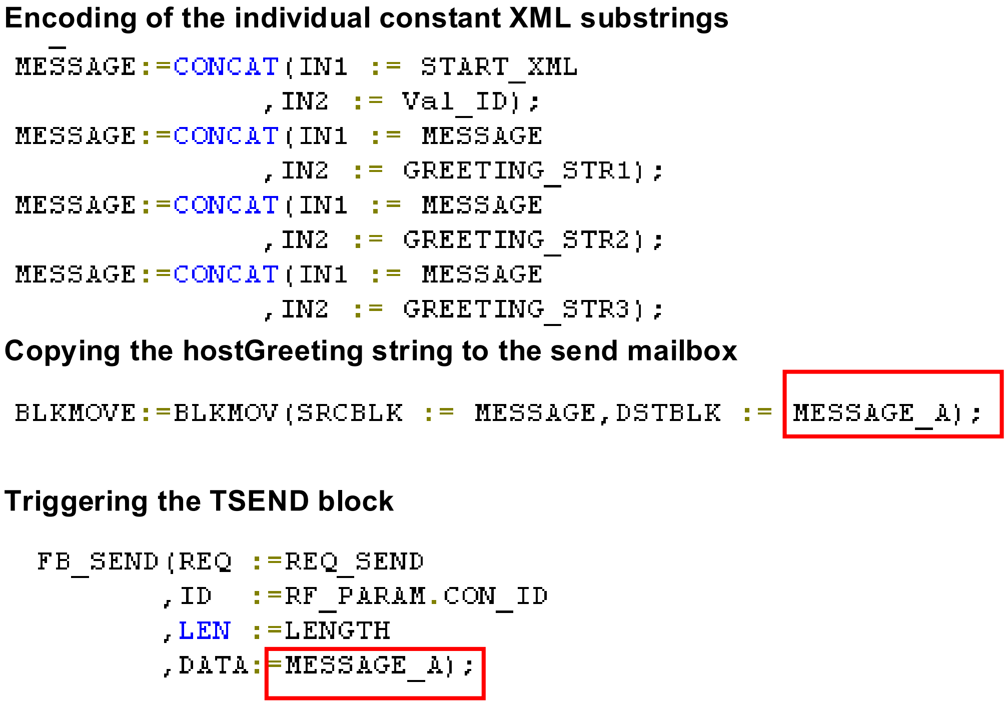


```
<frame>
 <cmd>
     <id> value_id </id>
     <hostGreetings>
         <readerType> value_readerType </readerType> opt
         <readerMode> value_ReaderMode </readerMode> opt
         <supportedVersions>
             <version> value_version </version>
             <version> value_version </version> opt
             ...
         </supportedVersions>
     </hostGreetings>
 </cmd>
</frame>
```

opt Optional: line can be omitted.

The figure below shows how the “hostGreetings” structure was implemented in the SCL code.

Figure 3-17



Encoding of the individual constant XML substrings

```
MESSAGE:=CONCAT(IN1  := START_XML
              ,IN2  := Val_ID);
MESSAGE:=CONCAT(IN1  := MESSAGE
              ,IN2  := GREETING_STR1);
MESSAGE:=CONCAT(IN1  := MESSAGE
              ,IN2  := GREETING_STR2);
MESSAGE:=CONCAT(IN1  := MESSAGE
              ,IN2  := GREETING_STR3);
```

Copying the hostGreeting string to the send mailbox

```
BLKMOVE:=BLKMOV(SRCBLK := MESSAGE,DSTBLK := MESSAGE_A);
```

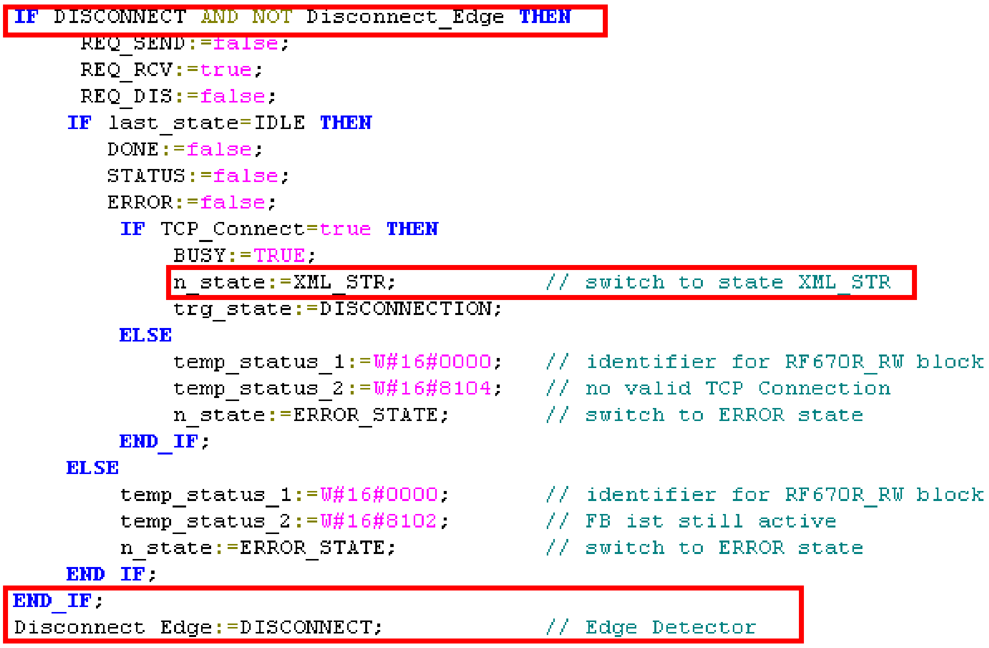
Triggering the TSEND block

```
 FB_SEND(REQ  :=REQ_SEND
        ,ID   :=RF_PARAM.CON_ID
        ,LEN  :=LENGTH
        ,DATA:=MESSAGE_A);
```

4. **Triggering the “DISCONNECT” input**

In the SCL code, a positive edge is generated to check the signal changes of the “DISCONNECT” input. When a rising edge of the input is detected, the block will start terminating the connection to the reader.

Figure 3-18

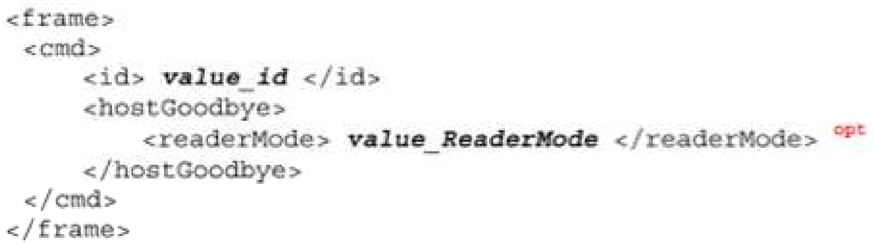


```
IF DISCONNECT AND NOT Disconnect_Edge THEN
     REQ_SEND:=false;
     REQ_RCV:=true;
     REQ_DIS:=false;
    IF last_state=IDLE THEN
       DONE:=false;
       STATUS:=false;
       ERROR:=false;
        IF TCP_Connect=true THEN
            BUSY:=TRUE;
            n_state:=XML_STR;           // switch to state XML_STR
            trg_state:=DISCONNECTION;
        ELSE
            temp_status_1:=W#16#0000;   // identifier for RF670R_RW block
            temp_status_2:=W#16#8104;   // no valid TCP Connection
            n_state:=ERROR_STATE;       // switch to ERROR state
        END_IF;
    ELSE
        temp_status_1:=W#16#0000;       // identifier for RF670R_RW block
        temp_status_2:=W#16#8102;       // FB ist still active
        n_state:=ERROR_STATE;           // switch to ERROR state
    END IF;
END_IF;
Disconnect Edge:=DISCONNECT;            // Edge Detector
```

5. **Formatting the “hostGoodbye” XML_command and sending it to the reader with TSEND**

The “hostGoodbye” command must have the following structure:

Figure 3-19

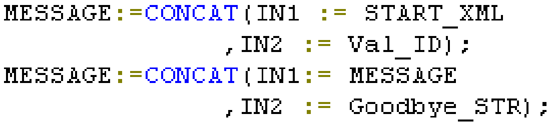


```
<frame>
 <cmd>
     <id> value_id </id>
     <hostGoodbye>
         <readerMode> value_ReaderMode </readerMode> opt
     </hostGoodbye>
 </cmd>
</frame>
```

The figure below shows how the “hostGoodbye” structure is implemented in the SCL code.

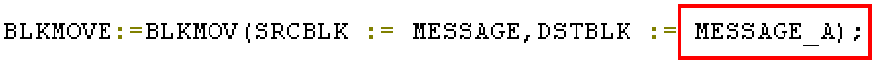
Figure 3-20

Encoding of the individual constant XML substrings



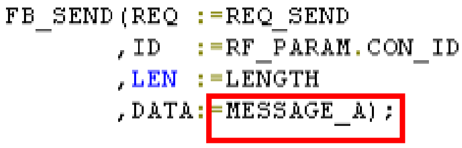
```
MESSAGE:=CONCAT(IN1 := START_XML
               ,IN2 := Val_ID);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2 := Goodbye_STR);
```

Copying the hostGoodbye string to the send mailbox



```
BLKMOVE:=BLKMOV(SRCBLK := MESSAGE,DSTBLK := MESSAGE_A);
```

Triggering the TSEND block

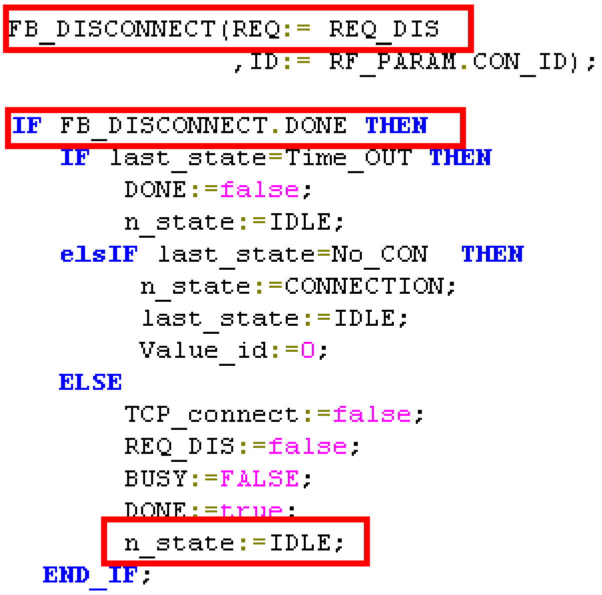


```
FB_SEND(REQ  :=REQ_SEND
       ,ID   :=RF_PARAM.CON_ID
       ,LEN  :=LENGTH
       ,DATA:=MESSAGE_A);
```

6. Terminating the TCP connection

If FB RF670R_RW receives positive feedback from the reader via TRCV, TDISCON will be called and the connection will be terminated.

Figure 3-21



```
FB_DISCONNECT(REQ:= REQ_DIS
             ,ID:= RF_PARAM.CON_ID);

IF FB_DISCONNECT.DONE THEN
   IF last_state=Time_OUT THEN
       DONE:=false;
       n_state:=IDLE;
   elsIF last_state=No_CON  THEN
        n_state:=CONNECTION;
        last_state:=IDLE;
        Value_id:=0;
   ELSE
       TCP_connect:=false;
       REQ_DIS:=false;
       BUSY:=FALSE;
       DONE:=true;
       n_state:=IDLE;
  END_IF;
```

3.2.2 “Read or write transponder IDs”

Overview of functions

The FB RF670R_RW function block generates a readTagIDs command and sends it to the RF670R reader.
All transponder IDs of the transponders the RF670R could read are now sent to the CPU. They are filtered and stored in the REC_DATA area of DB RF_PARAMETER.
Via the WRITE_TAG_ID input parameter, the FB RF670R_RW function block generates a writeTagID command and sends it to the RF670R reader to write the transponder ID of the transponder in the field. If a positive response message is received, the WRITE process is successfully completed.

Sequence of the function

The diagrammatic representation below shows the steps FB RF670R uses to read/write the transponder ID.

Figure 3-22

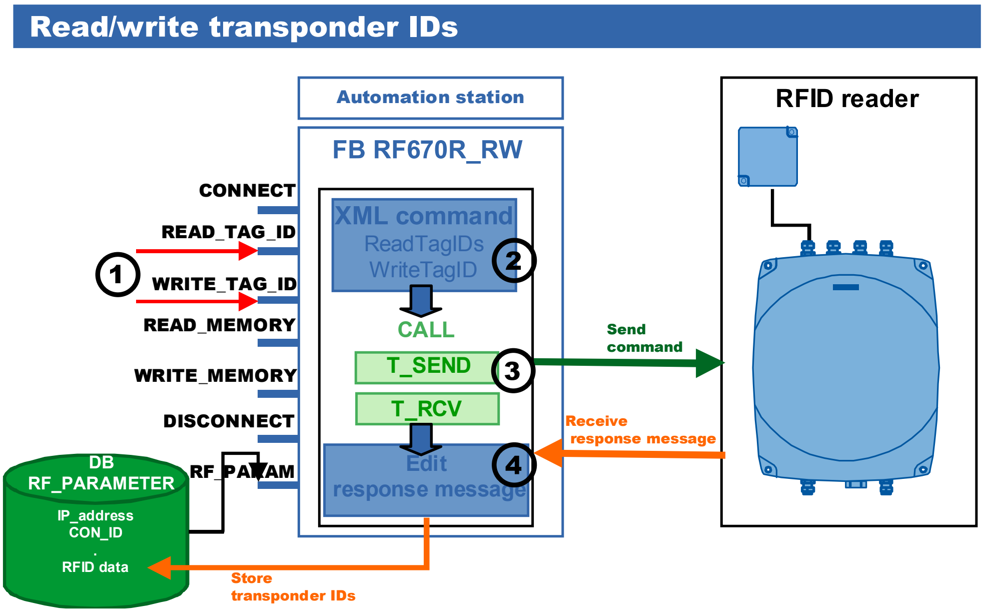


Table 3-4

No.	Description
1	READ_TAG_ID or WRITE_TAG_ID is triggered.
2	The “readTagID” or “writeTagID” XML command is formatted.
3	The XML-formatted command is sent to the reader in TSEND.
4	If FB RF670R_RW receives positive feedback from the reader via TRCV, the received data will be evaluated and copied to DB RF_PARAMETER.

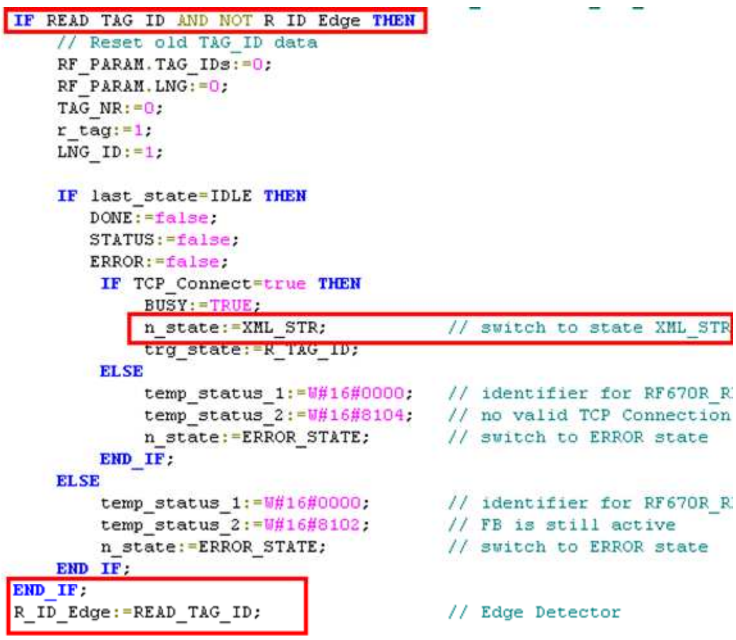
Program details

In this section, we show you the most important code fragments of this function from the documented source code of this example.

1. Triggering the “READ_TAG_ID” or “WRITE_TAG_ID” input

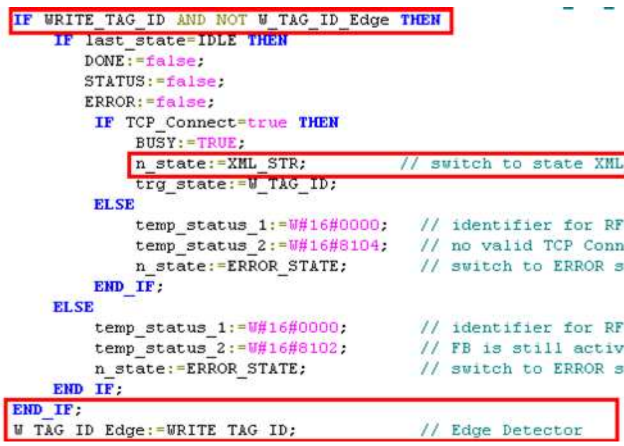
In the SCL code, a positive edge is generated to check the signal changes of the “READ_TAG_ID” or “WRITE_TAG_ID” input. When a rising edge of the input is detected, the block will start reading or writing the transponder_ID.

Figure 3-23



```
IF READ_TAG_ID AND NOT R_ID_Edge THEN
    // Reset old TAG_ID data
    RF_PARAM.TAG_IDs:=0;
    RF_PARAM.LNG:=0;
    TAG_NR:=0;
    r_tag:=1;
    LNG_ID:=1;

    IF last_state=IDLE THEN
       DONE:=false;
       STATUS:=false;
       ERROR:=false;
        IF TCP_Connect=true THEN
            BUSY:=TRUE;
            n_state:=XML_STR;            // switch to state XML_STR
            trg_state:=R_TAG_ID;
        ELSE
            temp_status_1:=W#16#0000;    // identifier for RF670R_R
            temp_status_2:=W#16#8104;    // no valid TCP Connection
            n_state:=ERROR_STATE;        // switch to ERROR state
        END_IF;
    ELSE
        temp_status_1:=W#16#0000;        // identifier for RF670R_R
        temp_status_2:=W#16#8102;        // FB is still active
        n_state:=ERROR_STATE;            // switch to ERROR state
    END_IF;
END_IF;
R_ID_Edge:=READ_TAG_ID;                  // Edge Detector
```

or

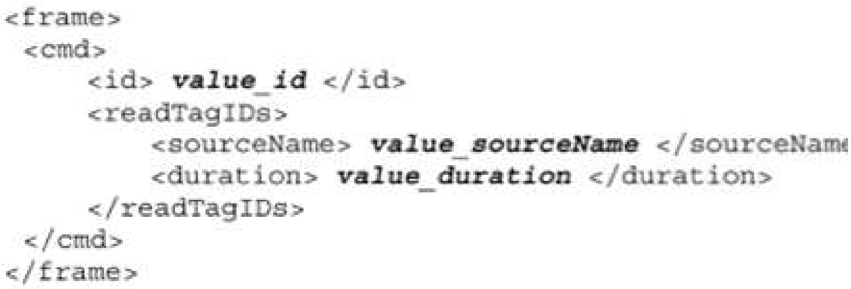


```
IF WRITE_TAG_ID AND NOT W_TAG_ID_Edge THEN
   IF last_state=IDLE THEN
      DONE:=false;
      STATUS:=false;
      ERROR:=false;
       IF TCP_Connect=true THEN
          BUSY:=TRUE;
          n_state:=XML_STR;         // switch to state XML_STR
          trg_state:=W_TAG_ID;
       ELSE
          temp_status_1:=W#16#0000;   // identifier for RF670R_RW
          temp_status_2:=W#16#8104;   // no valid TCP Connection
          n_state:=ERROR_STATE;       // switch to ERROR state
       END_IF;
   ELSE
       temp_status_1:=W#16#0000;      // identifier for RF670R_RW
       temp_status_2:=W#16#8102;      // FB is still active
       n_state:=ERROR_STATE;          // switch to ERROR state
   END_IF;
END_IF;
W_TAG_ID_Edge:=WRITE_TAG_ID;          // Edge Detector
```

2. Formatting the “readTagID” or “writeTagID” XML command

The readTagIDs command has the following structure:

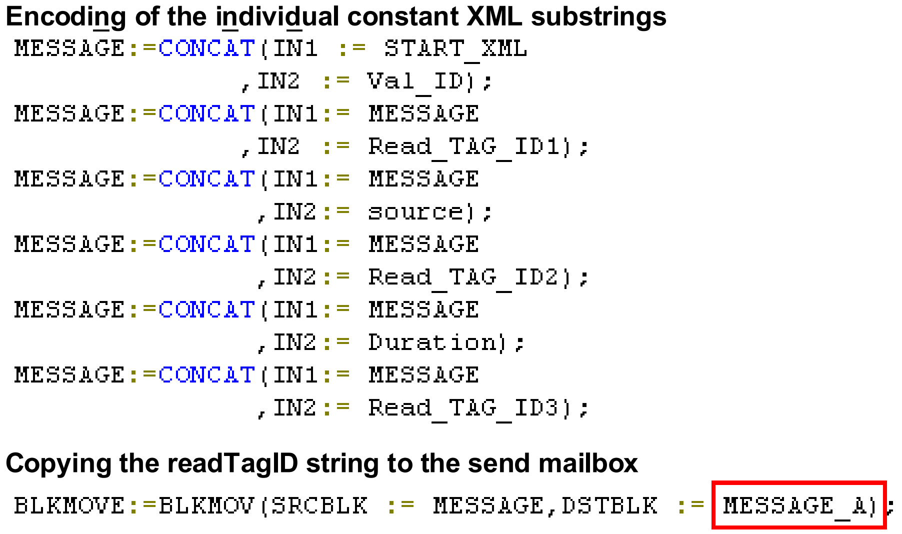
Figure 3-24



```
<frame>
 <cmd>
     <id> value_id </id>
     <readTagIDs>
         <sourceName> value_sourceName </sourceName
         <duration> value_duration </duration>
     </readTagIDs>
 </cmd>
</frame>
```

The figure below shows how the “readTagID” structure was implemented in the SCL code.

Figure 3-25



Encoding of the individual constant XML substrings

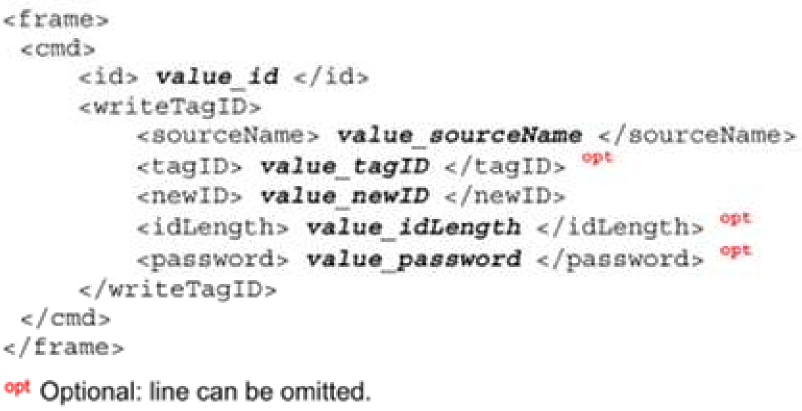
```
MESSAGE:=CONCAT(IN1 := START_XML
               ,IN2 := Val_ID);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2 := Read_TAG_ID1);
MESSAGE:=CONCAT(IN1:= MESSAGE
                ,IN2:= source);
MESSAGE:=CONCAT(IN1:= MESSAGE
                ,IN2:= Read_TAG_ID2);
MESSAGE:=CONCAT(IN1:= MESSAGE
                ,IN2:= Duration);
MESSAGE:=CONCAT(IN1:= MESSAGE
                ,IN2:= Read_TAG_ID3);
```

Copying the readTagID string to the send mailbox

```
BLKMOVE:=BLKMOV(SRCBLK := MESSAGE,DSTBLK := MESSAGE_A);
```

The writeTagIDs command has the following structure:

Figure 3-26

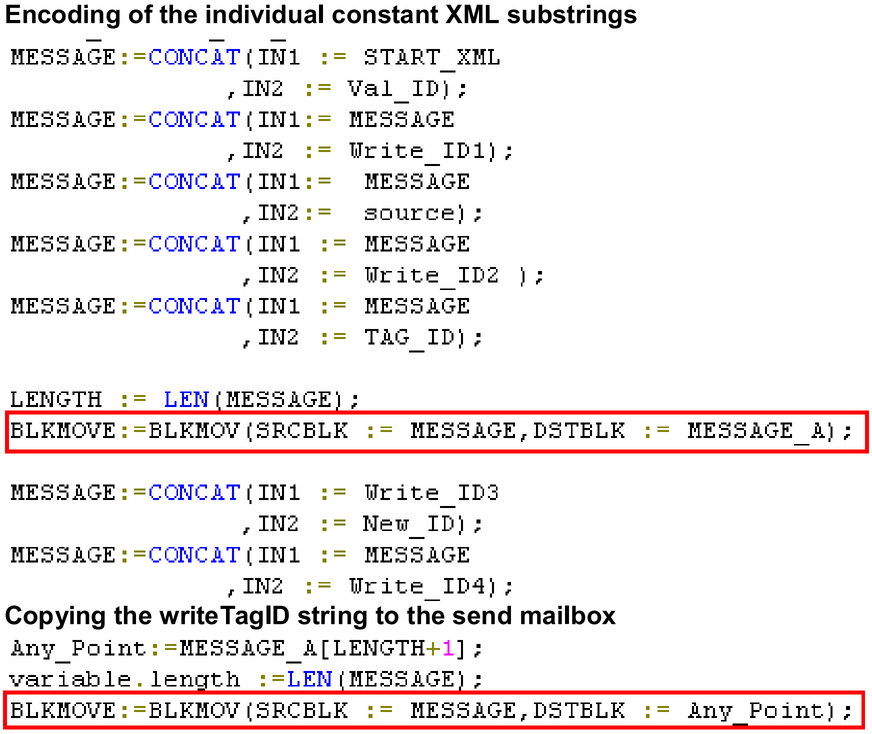


```
<frame>
 <cmd>
     <id> value_id </id>
     <writeTagID>
         <sourceName> value_sourceName </sourceName>
         <tagID> value_tagID </tagID> opt
         <newID> value_newID </newID>
         <idLength> value_idLength </idLength> opt
         <password> value_password </password> opt
     </writeTagID>
 </cmd>
</frame>
```

opt Optional: line can be omitted.

The figure below shows how the “writeTagID” structure was implemented in the SCL code.

Figure 3-27



Encoding of the individual constant XML substrings

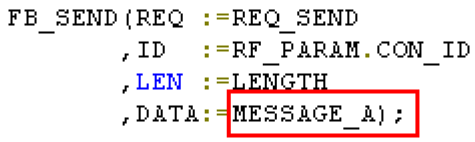
```
MESSAGE:=CONCAT(IN1  := START_XML
              ,IN2  := Val_ID);
MESSAGE:=CONCAT(IN1:= MESSAGE
              ,IN2  := Write_ID1);
MESSAGE:=CONCAT(IN1:=  MESSAGE
               ,IN2:=  source);
MESSAGE:=CONCAT(IN1  := MESSAGE
               ,IN2  := Write_ID2 );
MESSAGE:=CONCAT(IN1  := MESSAGE
               ,IN2  := TAG_ID);

LENGTH  :=  LEN(MESSAGE);
BLKMOVE:=BLKMOV(SRCBLK  := MESSAGE,DSTBLK  := MESSAGE_A);

MESSAGE:=CONCAT(IN1  := Write_ID3
               ,IN2  := New_ID);
MESSAGE:=CONCAT(IN1  := MESSAGE
              ,IN2  := Write_ID4);
```

Copying the writeTagID string to the send mailbox

```
Any_Point:=MESSAGE_A[LENGTH+1];
variable.length  :=LEN(MESSAGE);
BLKMOVE:=BLKMOV(SRCBLK  := MESSAGE,DSTBLK  := Any_Point);
```

3. Sending the XML-formatted command to the reader

Figure 3-28



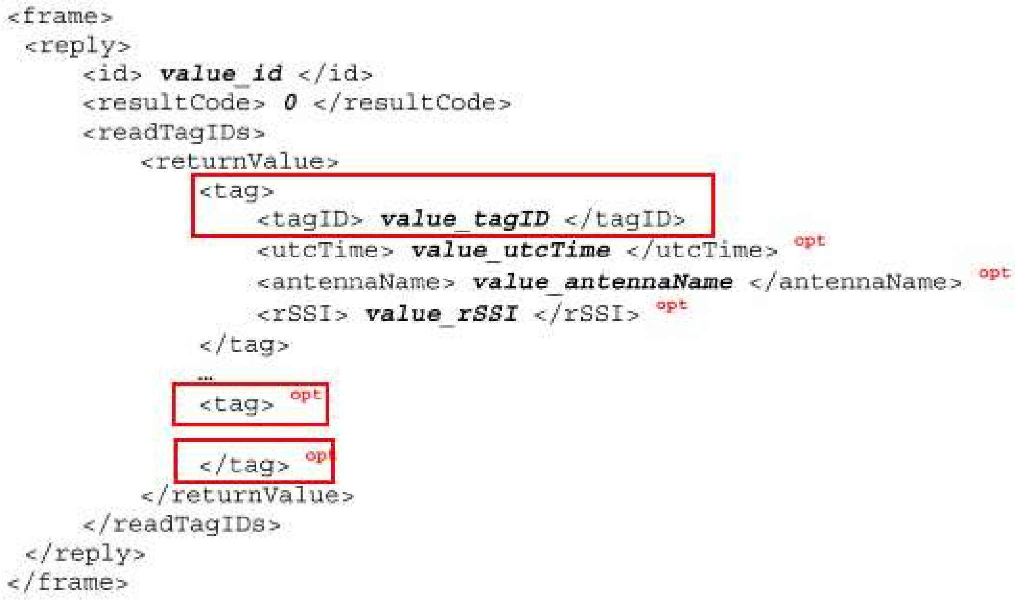
```
FB_SEND(REQ  :=REQ_SEND
       ,ID   :=RF_PARAM.CON_ID
       ,LEN  :=LENGTH
       ,DATA:=MESSAGE_A);
```

4. Receiving and evaluating data

When the response message is received, the read or written transponder IDs will be extracted and stored in the global data block RF_PARAMETER.

The response message for the ReadTagID command has the following structure:

Figure 3-29

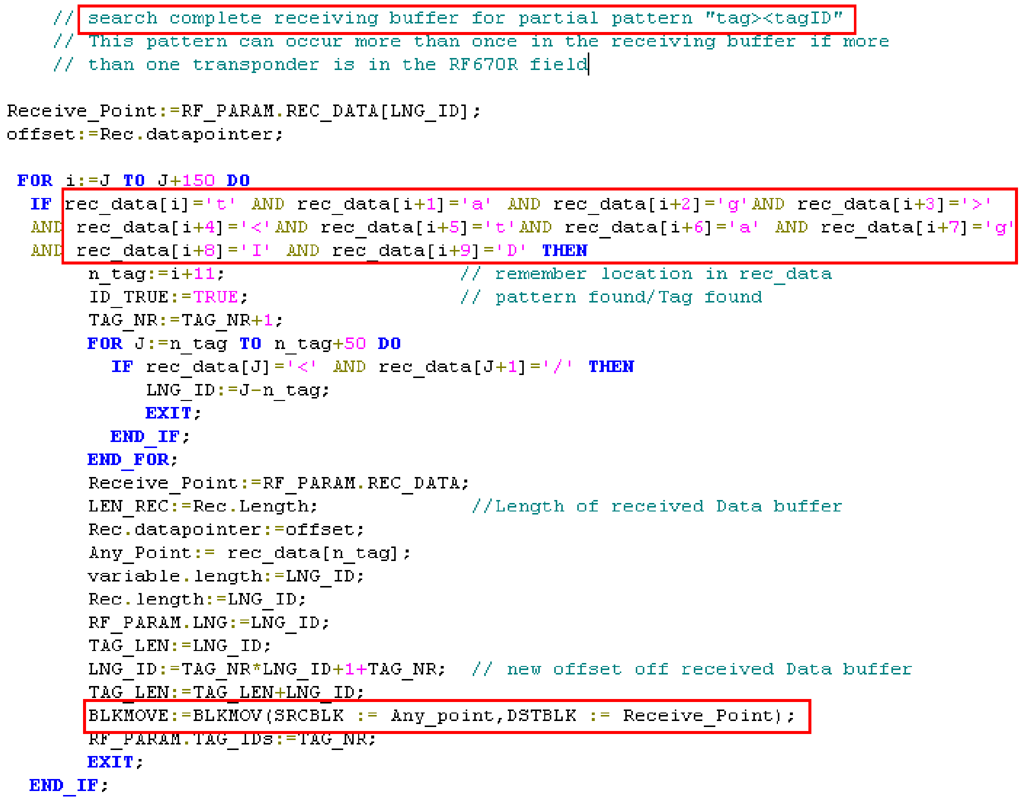


```
<frame>
 <reply>
     <id> value_id </id>
     <resultCode> 0 </resultCode>
     <readTagIDs>
         <returnValue>
             <tag>
                 <tagID> value_tagID </tagID>
                 <utcTime> value_utcTime </utcTime> opt
                 <antennaName> value_antennaName </antennaName> opt
                 <rSSI> value_rSSI </rSSI> opt
             </tag>
             ...
             <tag> opt

             </tag> opt
         </returnValue>
     </readTagIDs>
 </reply>
</frame>
```

opt Optional: line could be omitted.

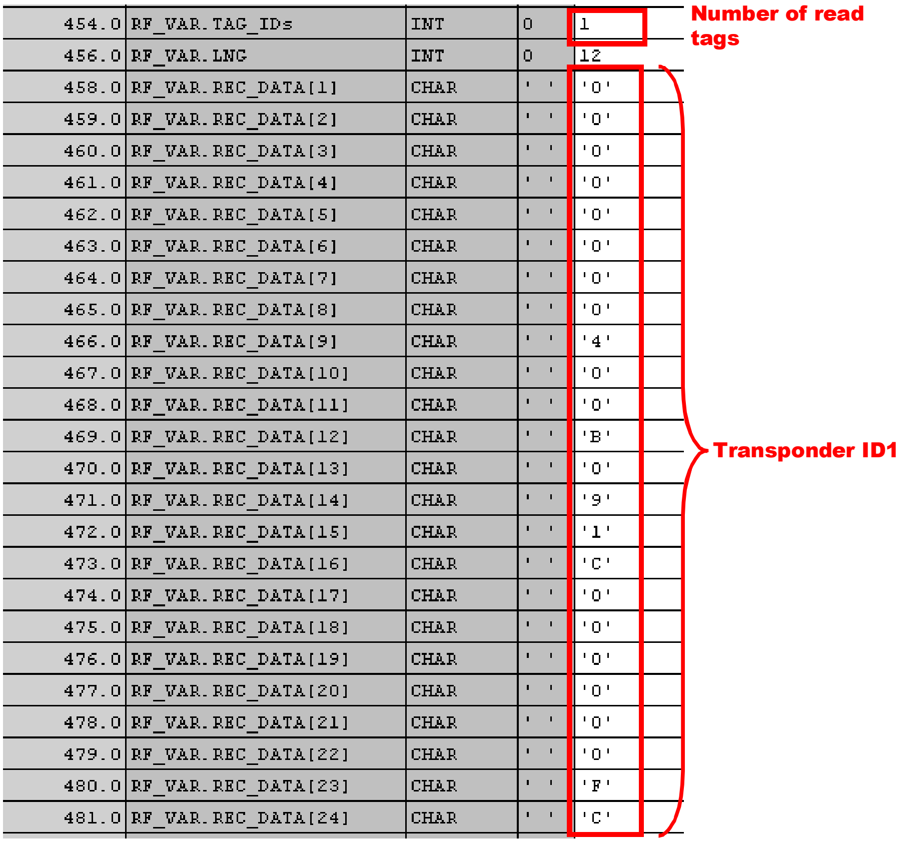
The following code fragment shows how the individual transponder IDs are extracted from the receive buffer and written to the respective RF_PARAMETER area.

Figure 3-30



```
    // search complete receiving buffer for partial pattern "tag><tagID"
    // This pattern can occur more than once in the receiving buffer if more
    // than one transponder is in the RF670R field

Receive_Point:=RF_PARAM.REC_DATA[LNG_ID];
offset:=Rec.datapointer;

 FOR i:=J TO J+150 DO
  IF rec_data[i]='t' AND rec_data[i+1]='a' AND rec_data[i+2]='g'AND rec_data[i+3]='>'
  AND rec_data[i+4]='<'AND rec_data[i+5]='t'AND rec_data[i+6]='a' AND rec_data[i+7]='g'
  AND rec_data[i+8]='I' AND rec_data[i+9]='D' THEN
      n_tag:=i+11;                        // remember location in rec_data
      ID_TRUE:=TRUE;                      // pattern found/Tag found
      TAG_NR:=TAG_NR+1;
      FOR J:=n_tag TO n_tag+50 DO
        IF rec_data[J]='<' AND rec_data[J+1]='/' THEN
           LNG_ID:=J-n_tag;
           EXIT;
        END_IF;
      END_FOR;
      Receive_Point:=RF_PARAM.REC_DATA;
      LEN_REC:=Rec.Length;                 //Length of received Data buffer
      Rec.datapointer:=offset;
      Any_Point:= rec_data[n_tag];
      variable.length:=LNG_ID;
      Rec.length:=LNG_ID;
      RF_PARAM.LNG:=LNG_ID;
      TAG_LEN:=LNG_ID;
      LNG_ID:=TAG_NR*LNG_ID+1+TAG_NR;  // new offset off received Data buffer
      TAG_LEN:=TAG_LEN+LNG_ID;
      BLKMOVE:=BLKMOV(SRCBLK := Any_point,DSTBLK := Receive_Point);
      RF_PARAM.TAG_IDs:=TAG_NR;
      EXIT;
  END_IF;
```

Figure 3-31 RF_PARAMETER/ read transponder IDs



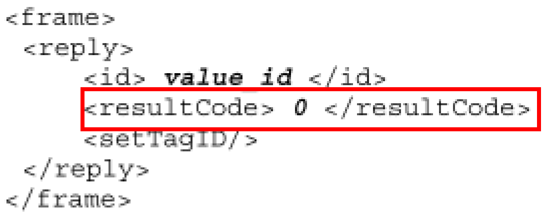
454.0	RF_VAR.TAG_IDs	INT	0	1
456.0	RF_VAR.LNG	INT	0	12
458.0	RF_VAR.REC_DATA[1]	CHAR	' '	'0'
459.0	RF_VAR.REC_DATA[2]	CHAR	' '	'0'
460.0	RF_VAR.REC_DATA[3]	CHAR	' '	'0'
461.0	RF_VAR.REC_DATA[4]	CHAR	' '	'0'
462.0	RF_VAR.REC_DATA[5]	CHAR	' '	'0'
463.0	RF_VAR.REC_DATA[6]	CHAR	' '	'0'
464.0	RF_VAR.REC_DATA[7]	CHAR	' '	'0'
465.0	RF_VAR.REC_DATA[8]	CHAR	' '	'0'
466.0	RF_VAR.REC_DATA[9]	CHAR	' '	'4'
467.0	RF_VAR.REC_DATA[10]	CHAR	' '	'0'
468.0	RF_VAR.REC_DATA[11]	CHAR	' '	'0'
469.0	RF_VAR.REC_DATA[12]	CHAR	' '	'B'
470.0	RF_VAR.REC_DATA[13]	CHAR	' '	'0'
471.0	RF_VAR.REC_DATA[14]	CHAR	' '	'9'
472.0	RF_VAR.REC_DATA[15]	CHAR	' '	'1'
473.0	RF_VAR.REC_DATA[16]	CHAR	' '	'C'
474.0	RF_VAR.REC_DATA[17]	CHAR	' '	'0'
475.0	RF_VAR.REC_DATA[18]	CHAR	' '	'0'
476.0	RF_VAR.REC_DATA[19]	CHAR	' '	'0'
477.0	RF_VAR.REC_DATA[20]	CHAR	' '	'0'
478.0	RF_VAR.REC_DATA[21]	CHAR	' '	'0'
479.0	RF_VAR.REC_DATA[22]	CHAR	' '	'0'
480.0	RF_VAR.REC_DATA[23]	CHAR	' '	'F'
481.0	RF_VAR.REC_DATA[24]	CHAR	' '	'C'

Number of read tags

Transponder ID1

The response message for the WriteTagID command has the following structure:

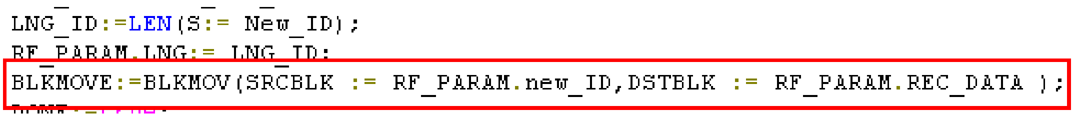
Figure 3-32



```
<frame>
 <reply>
    <id> value id </id>
    <resultCode> 0 </resultCode>
    <setTagID/>
 </reply>
</frame>
```

If FB RF670R_RW receives positive feedback from the reader via TRCV, the new_TAGid will be copied to DB RF_PARAMETER.

Figure 3-33



```
LNG_ID:=LEN(S:= New_ID);
RF_PARAM.LNG:= LNG_ID;
BLKMOVE:=BLKMOV(SRCBLK := RF_PARAM.new_ID,DSTBLK := RF_PARAM.REC_DATA );
```

3.2.3 “Read RFID data from or write RFID data to transponder”

Overview of functions

With the aid of these functions, payload can be read from or written to the transponder memory.
The readTagMemory message is used to read from and the writeTagMemory message is used to write to the transponder.

When there are several transponders in the read field, the following variants apply.

Variant 1: TAG_ID field in DB RF_PARAMETER is blank:

The payload of the transponder the reader detects first is read or written.

Variant 2: TAG_ID field in DB RF_PARAMETER exists:

Only data of the transponder with the specified transponder ID is read/written.

Sequence of the function

Figure 3-34

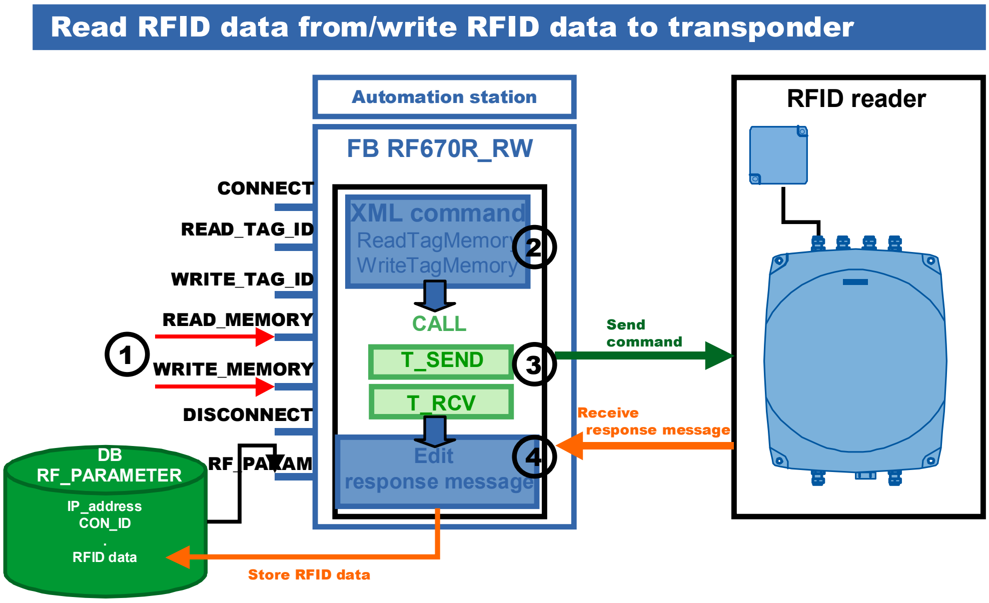


Table 3-5

No.	Description
1	READ_MEMORY or WRITE_MEMORY is triggered.
2	The readTagMemory or writeTagMemory command is formatted.
3	The XML-formatted command is sent to the reader in TSEND.
4	If FB RF670R_RW receives positive feedback from the reader via TRCV, the received data will be evaluated and copied to DB RF_PARAMETER.

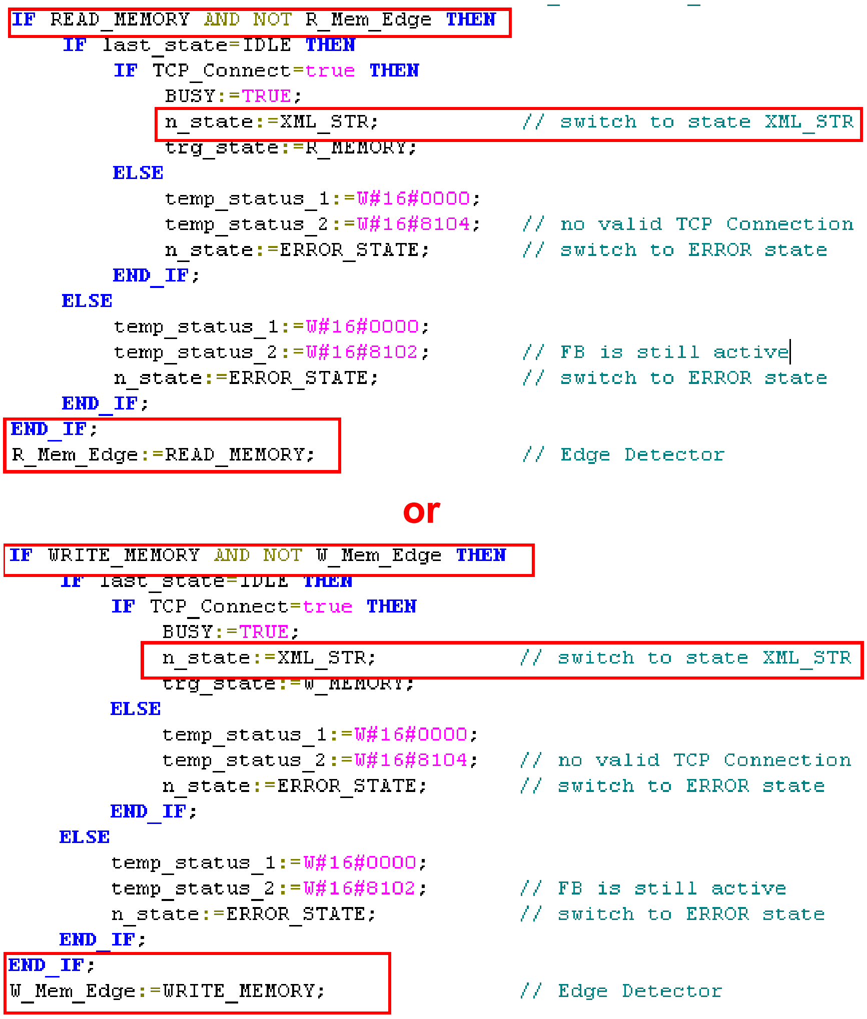
Program details

In this section, we show you the most important code fragments of this function from the documented SCL code of this example.

1. Triggering the “READ_MEMORY” or “WRITE_MEMORY” input

In the SCL code, a positive edge is generated to check the signal changes of the “READ_MEMORY” or “WRITE_MEMORY” input. When a rising edge of the input is detected, the block will start reading or writing the RFID data.

Figure 3-35



```
IF READ_MEMORY AND NOT R_Mem_Edge THEN
    IF last_state=IDLE THEN
        IF TCP_Connect=true THEN
            BUSY:=TRUE;
            n_state:=XML_STR;           // switch to state XML_STR
            trg_state:=R_MEMORY;
        ELSE
            temp_status_1:=W#16#0000;
            temp_status_2:=W#16#8104;   // no valid TCP Connection
            n_state:=ERROR_STATE;       // switch to ERROR state
        END_IF;
    ELSE
        temp_status_1:=W#16#0000;
        temp_status_2:=W#16#8102;       // FB is still active
        n_state:=ERROR_STATE;           // switch to ERROR state
    END_IF;
END_IF;
R_Mem_Edge:=READ_MEMORY;                // Edge Detector
```

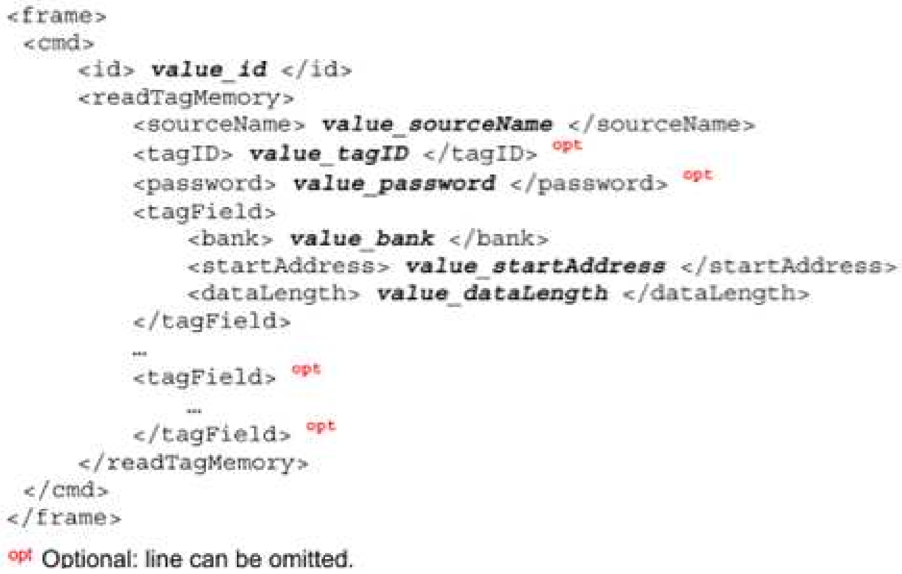
or

```
IF WRITE_MEMORY AND NOT W_Mem_Edge THEN
    IF last_state=IDLE THEN
        IF TCP_Connect=true THEN
            BUSY:=TRUE;
            n_state:=XML_STR;           // switch to state XML_STR
            trg_state:=W_MEMORY;
        ELSE
            temp_status_1:=W#16#0000;
            temp_status_2:=W#16#8104;   // no valid TCP Connection
            n_state:=ERROR_STATE;       // switch to ERROR state
        END_IF;
    ELSE
        temp_status_1:=W#16#0000;
        temp_status_2:=W#16#8102;       // FB is still active
        n_state:=ERROR_STATE;           // switch to ERROR state
    END_IF;
END_IF;
W_Mem_Edge:=WRITE_MEMORY;               // Edge Detector
```

2. Formatting the “readTagMemory” or “writeTagMemory” XML command

The readTagMemory command has the following structure:

Figure 3-36

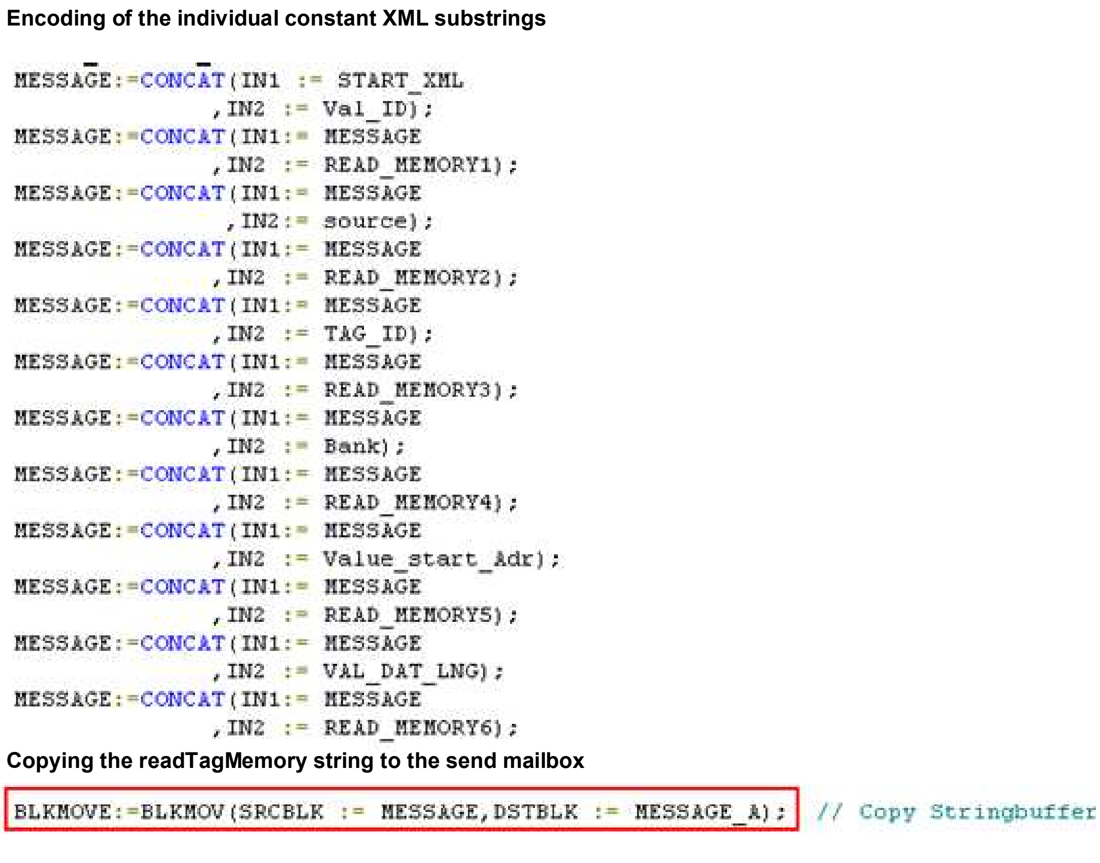


```
<frame>
 <cmd>
     <id> value_id </id>
     <readTagMemory>
         <sourceName> value_sourceName </sourceName>
         <tagID> value_tagID </tagID> opt
         <password> value_password </password> opt
         <tagField>
             <bank> value_bank </bank>
             <startAddress> value_startAddress </startAddress>
             <dataLength> value_dataLength </dataLength>
         </tagField>
         ...
         <tagField> opt
             ...
         </tagField> opt
     </readTagMemory>
 </cmd>
</frame>
```

opt Optional: line can be omitted.

The figure below shows how the “readTagMemory” structure was implemented in the SCL code.

Figure 3-37



Encoding of the individual constant XML substrings

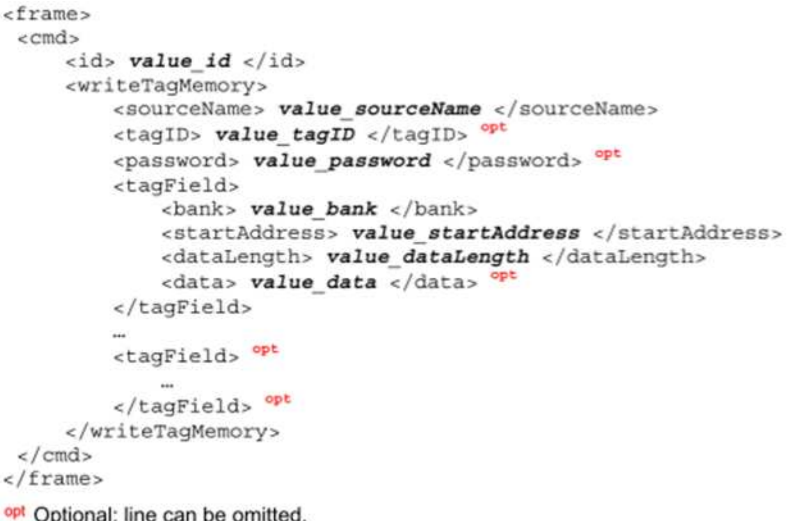
```
MESSAGE:=CONCAT(IN1  := START_XML
               ,IN2  := Val_ID);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY1);
MESSAGE:=CONCAT(IN1:= MESSAGE
                ,IN2:= source);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY2);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := TAG_ID);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY3);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := Bank);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY4);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := Value_start_Adr);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY5);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := VAL_DAT_LNG);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY6);
```

Copying the readTagMemory string to the send mailbox

```
BLKMOVE:=BLKMOV(SRCBLK := MESSAGE,DSTBLK := MESSAGE_A);  // Copy Stringbuffer
```

The writeTagMemory command has the following structure:

Figure 3-38

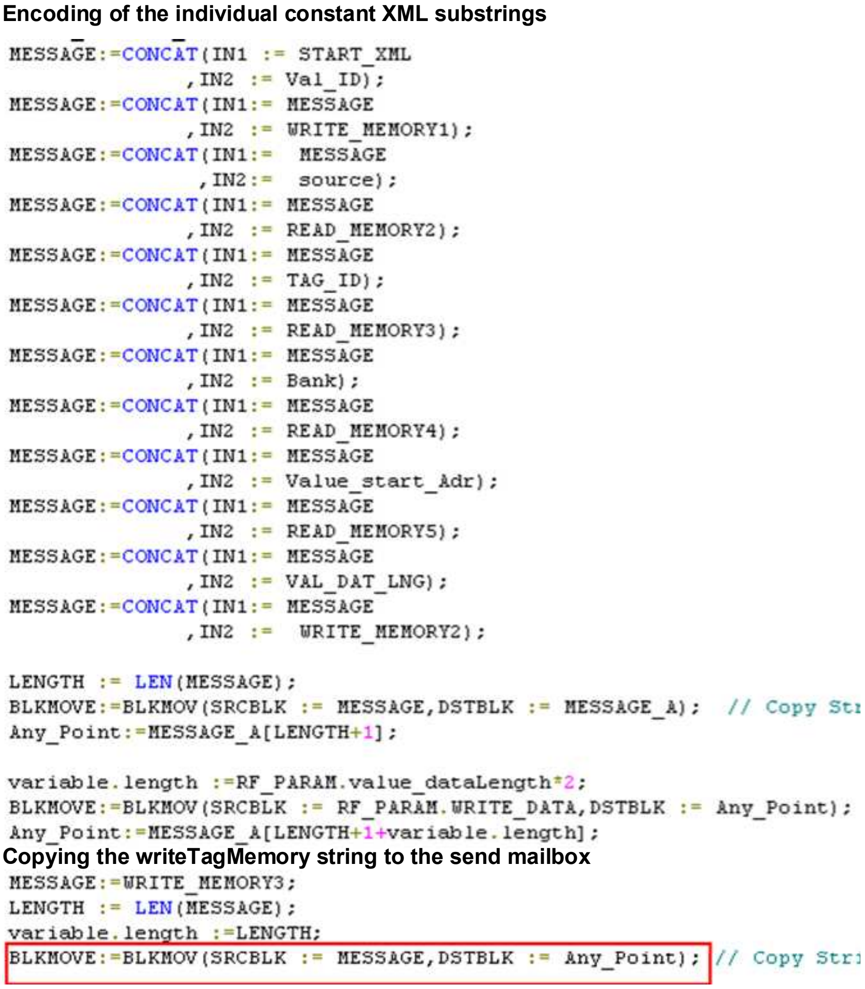


```
<frame>
 <cmd>
     <id> value_id </id>
     <writeTagMemory>
         <sourceName> value_sourceName </sourceName>
         <tagID> value_tagID </tagID> opt
         <password> value_password </password> opt
         <tagField>
             <bank> value_bank </bank>
             <startAddress> value_startAddress </startAddress>
             <dataLength> value_dataLength </dataLength>
             <data> value_data </data> opt
         </tagField>
         …
         <tagField> opt
             …
         </tagField> opt
     </writeTagMemory>
 </cmd>
</frame>
```

opt Optional: line can be omitted.

The figure below shows how the “writeTagMemory” structure was implemented in the SCL code.

Figure 3-39



Encoding of the individual constant XML substrings

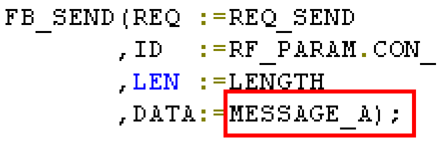
```
MESSAGE:=CONCAT(IN1  := START_XML
               ,IN2  := Val_ID);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := WRITE_MEMORY1);
MESSAGE:=CONCAT(IN1:=  MESSAGE
                ,IN2:=  source);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY2);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := TAG_ID);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY3);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := Bank);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY4);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := Value_start_Adr);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := READ_MEMORY5);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  := VAL_DAT_LNG);
MESSAGE:=CONCAT(IN1:= MESSAGE
               ,IN2  :=  WRITE_MEMORY2);

LENGTH  := LEN(MESSAGE);
BLKMOVE:=BLKMOV(SRCBLK := MESSAGE,DSTBLK := MESSAGE_A);  // Copy Str
Any_Point:=MESSAGE_A[LENGTH+1];

variable.length  :=RF_PARAM.value_dataLength*2;
BLKMOVE:=BLKMOV(SRCBLK  := RF_PARAM.WRITE_DATA,DSTBLK  := Any_Point);
Any_Point:=MESSAGE_A[LENGTH+1+variable.length];
```

Copying the writeTagMemory string to the send mailbox

```
MESSAGE:=WRITE_MEMORY3;
LENGTH  := LEN(MESSAGE);
variable.length  :=LENGTH;
BLKMOVE:=BLKMOV(SRCBLK  := MESSAGE,DSTBLK  := Any_Point); // Copy Str
```

3. Sending the XML-formatted command to the reader

Figure 3-40



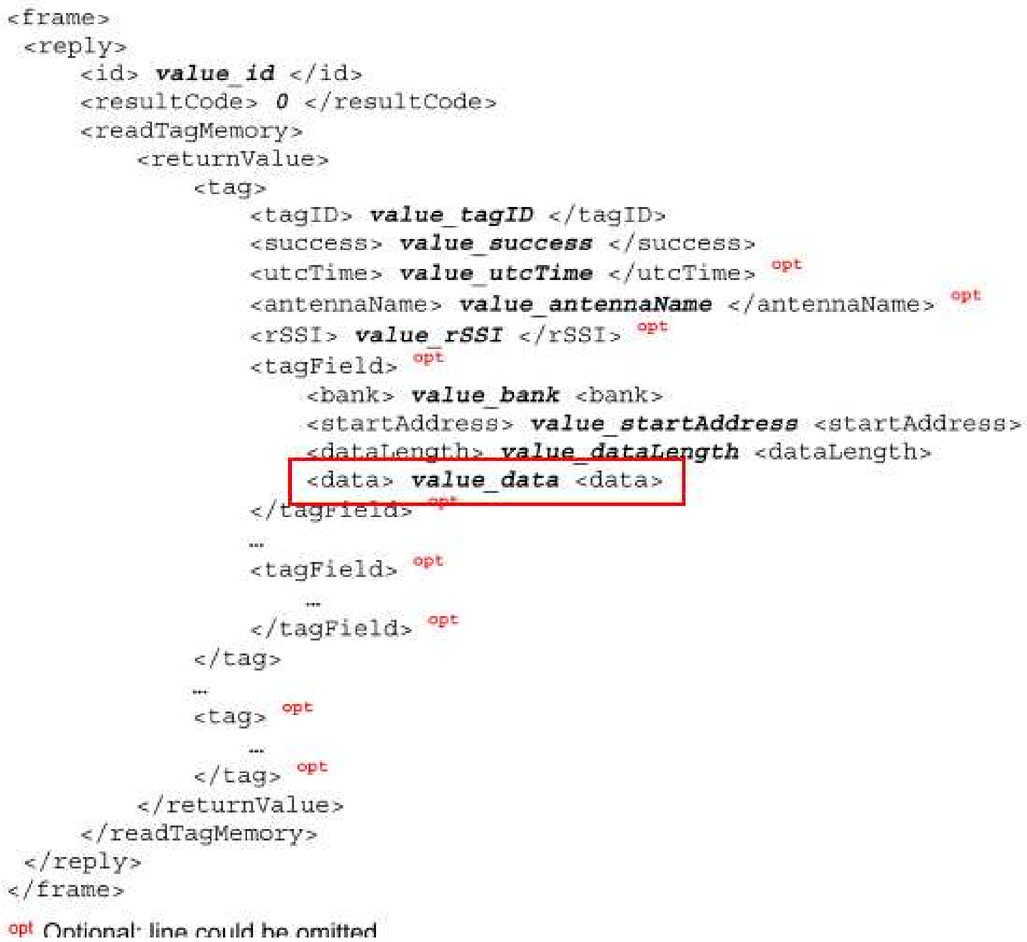
```
FB_SEND(REQ  :=REQ_SEND
       ,ID   :=RF_PARAM.CON_ID
       ,LEN  :=LENGTH
       ,DATA:=MESSAGE_A);
```

4. Receiving and evaluating data

When the response message is received, the read data will be extracted and stored in the global data block RF_PARAMETER.

The response message for the ReadTagMemory command has the following structure:

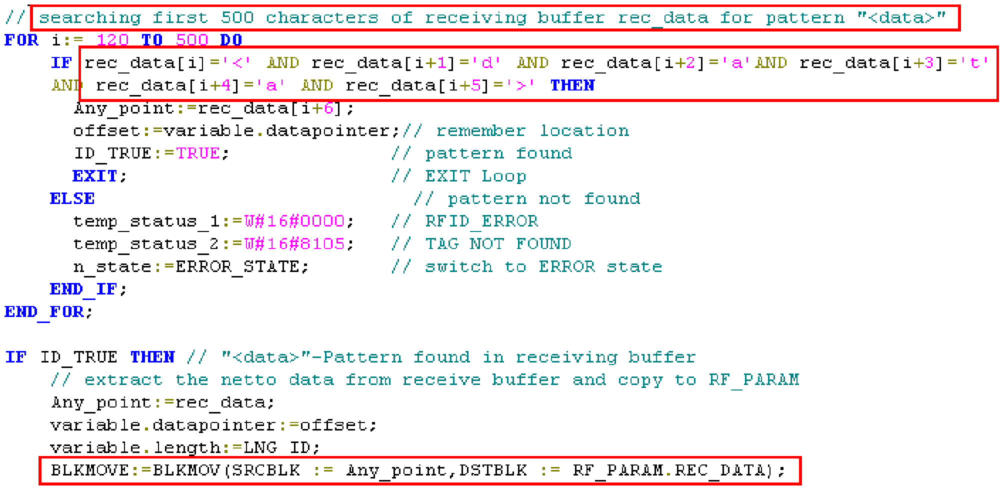
Figure 3-41



```
<frame>
 <reply>
     <id> value_id </id>
     <resultCode> 0 </resultCode>
     <readTagMemory>
         <returnValue>
             <tag>
                 <tagID> value_tagID </tagID>
                 <success> value_success </success>
                 <utcTime> value_utcTime </utcTime> opt
                 <antennaName> value_antennaName </antennaName> opt
                 <rSSI> value_rSSI </rSSI> opt
                 <tagField> opt
                     <bank> value_bank <bank>
                     <startAddress> value_startAddress <startAddress>
                     <dataLength> value_dataLength <dataLength>
                     <data> value_data <data>
                 </tagField> opt
                 ...
                 <tagField> opt
                     ...
                 </tagField> opt
             </tag>
             ...
             <tag> opt
                 ...
             </tag> opt
         </returnValue>
     </readTagMemory>
 </reply>
</frame>
opt Optional: line could be omitted
```

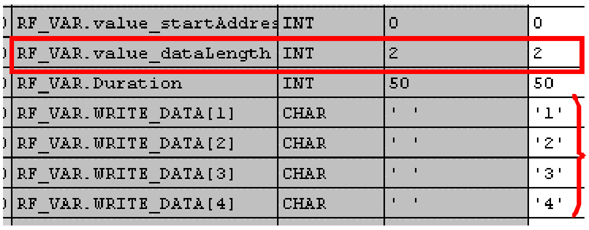
The following code fragment shows how the data is extracted from the receive buffer and written to the respective RF_PARAMETER area.

Figure 3-42



```
// searching first 500 characters of receiving buffer rec_data for pattern "<data>"
FOR i:= 120 TO 500 DO
    IF rec_data[i]='<' AND rec_data[i+1]='d' AND rec_data[i+2]='a'AND rec_data[i+3]='t'
    AND rec_data[i+4]='a' AND rec_data[i+5]='>' THEN
      Any_point:=rec_data[i+6];
      offset:=variable.datapointer;// remember location
      ID_TRUE:=TRUE;              // pattern found
      EXIT;                       // EXIT Loop
    ELSE                            // pattern not found
      temp_status_1:=W#16#0000;   // RFID_ERROR
      temp_status_2:=W#16#8105;   // TAG NOT FOUND
      n_state:=ERROR_STATE;       // switch to ERROR state
    END_IF;
END_FOR;

IF ID_TRUE THEN // "<data>"-Pattern found in receiving buffer
    // extract the netto data from receive buffer and copy to RF_PARAM
    Any_point:=rec_data;
    variable.datapointer:=offset;
    variable.length:=LNG_ID;
    BLKMOVE:=BLKMOV(SRCBLK := Any_point,DSTBLK := RF_PARAM.REC_DATA);
```

Figure 3-43 RF_PARAMETER



RF_VAR.value_startAddres	INT	0	0
RF_VAR.value_dataLength	INT	2	2
RF_VAR.Duration	INT	50	50
RF_VAR.WRITE_DATA[1]	CHAR	' '	'1'
RF_VAR.WRITE_DATA[2]	CHAR	' '	'2'
RF_VAR.WRITE_DATA[3]	CHAR	' '	'3'
RF_VAR.WRITE_DATA[4]	CHAR	' '	'4'

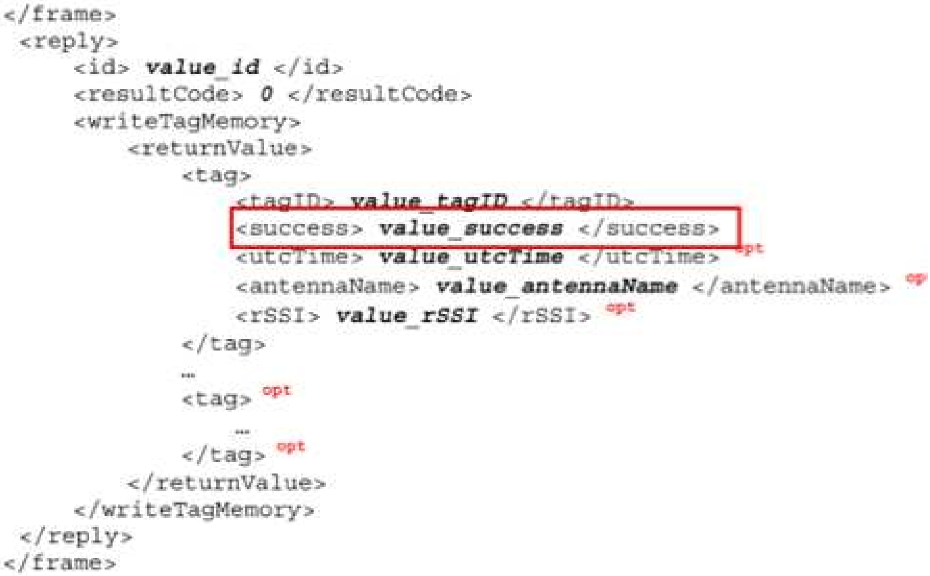
Length of the data

RFID data to be written to the transponder

In HEX format, the payload corresponds to 0x12 and 0x9F. The length of the data has the value 2.

The response message for the WriteTagMemory command has the following structure:

Figure 3-44

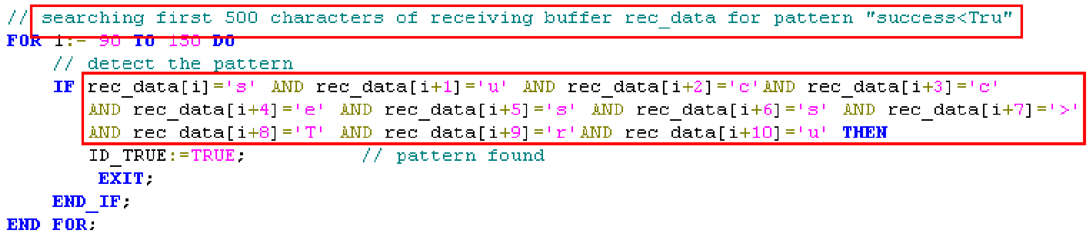


```
</frame>
 <reply>
     <id> value_id </id>
     <resultCode> 0 </resultCode>
     <writeTagMemory>
         <returnValue>
             <tag>
                 <tagID> value_tagID </tagID>
                 <success> value_success </success>
                 <utcTime> value_utcTime </utcTime> opt
                 <antennaName> value_antennaName </antennaName> opt
                 <rSSI> value_rSSI </rSSI> opt
             </tag>
             …
             <tag> opt
                 …
             </tag> opt
         </returnValue>
     </writeTagMemory>
 </reply>
</frame>
```

opt Optional: line could be omitted.

The following code fragment shows how the data from the receive buffer is evaluated.

Figure 3-45



```
// searching first 500 characters of receiving buffer rec_data for pattern "success<Tru"
FOR i:= 90 TO 150 DO
    // detect the pattern
    IF rec_data[i]='s' AND rec_data[i+1]='u' AND rec_data[i+2]='c'AND rec_data[i+3]='c'
       AND rec_data[i+4]='e' AND rec_data[i+5]='s' AND rec_data[i+6]='s' AND rec_data[i+7]='>'
       AND rec_data[i+8]='T' AND rec_data[i+9]='r'AND rec_data[i+10]='u' THEN
       ID_TRUE:=TRUE;           // pattern found
        EXIT;
    END_IF;
END_FOR;
```

Note For more program details, the well-documented SCL code is available to you in the project (see \1\).

3.3 Error and status display

For error diagnostics, the FB RF670R_RW function block (FB670) has a STATUS output. By reading the STATUS output of the function block, you are provided with information on logical errors and error messages that may occur during the communication between the controller and the RF670R reader.

3.3.1 Error messages of FB RF670R_RW (FB670)

Table 3-6

Status	Meaning	Support/remark
16#00008101	Watchdog timer has expired.	1. a)Check IP_ADR input parameter b)Check communication between controller and reader 2. Re-trigger CONNECT
16#00008102	The previous job has not yet been completed.	1. Wait until BUSY=FALSE 2. Restart process
16#00008103	There is already a connection to the reader.	
16#00008104	There is no connection to the reader.	Trigger CONNECT
16#00008105	No transponder in the field or incorrect transponder ID.	1. Check transponder ID 2. Restart process
16#00008106	Incorrect response message (received Value_id does not equal sent Value_id).	1. Trigger DISCONNECT 2. Trigger CONNECT 3. Restart process
16#00008107	The WRITE_TAG_ID process could not be successfully completed.	Restart process
16#00008108	The length of the evaluated characters exceeds 150 characters. Only 150 characters were transmitted.	1. Change length of RF_VAR.REC_DATA in DB RF_PARAMETER (see Chapter 6.1) 2. Restart process

Status	Meaning	Support/remark
16#00008109	The connection is interrupted, for example due a line break	1. Check communication between controller and reader 2. Re-trigger CONNECT

3.3.2 Error messages of the RF670R reader

Errors with the **16#13000xyy** status are RFID errors.
For a more detailed overview of the result codes, please refer to the RF670R Function Manual (/5/, Chapter 3.3).

3.3.3 Error messages of communication blocks

Errors with a status that is not described above are errors of communication blocks (see \8\):

TCON:	**DW#16#0001xyyy**
TSEND:	**DW#16#0010xyyy**
TRCV:	**DW#16#0012xyyy**
TDISCON:	**DW#16#0011xyyy**

4 Installation and Commissioning

4.1 Hardware configuration

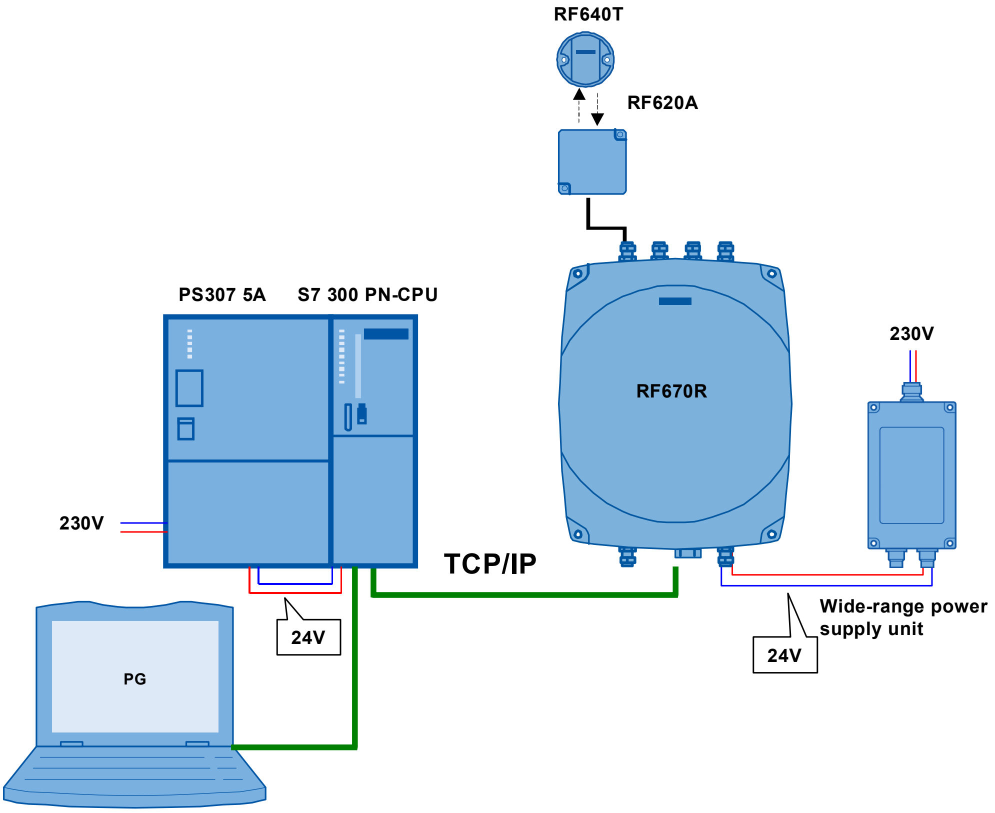
For the necessary hardware components, please refer to chapter 2.3.

NOTICE **Follow the installation guidelines for S7-300 (\7\) and RF670R (\4\). Refer to the relevant manuals.**

NOTICE **Before you switch on the power supply, complete and check the installation!**

The figure below shows the hardware configuration of the application.

Figure 4-1



The following table provides an overview of the IP addresses used in this sample program.

Table 4-1

Module	IP address
CPU 315-2PN/ DP	192.168.0.1
PC/ PG	192.168.0.3
RF670R	192.168.0.254

4.2 Hardware installation: S7 station

Table 4-2

No.	Action	Remark
1	Attach the individual modules to a suitable rack.	Table 2-1: Hardware components
2	Connect the PS307 to the network. (230 V AC)	Ensure that the polarity is correct.
3	Connect the following devices: • PROFINET interface of the engineering PG to the PROFINET interface of the CPU • PROFINET interface of the RFID reader to the second PROFINET interface of the CPU	

4.3 Hardware installation: RF670R reader

Table 4-3

No.	Action	Remark
1	Connect the antennas to the respective sockets.	You have to configure the antennas using RF-MANAGER.
2	Connect the wide-range power supply unit to the network.	
3	Connect the RF670R reader to the wide-range power supply unit.	

4.4 Installation of the standard software

The engineering station is used as the configuration computer for the S7 station.

Table 4-4

No.	Action	Remark
1	Install STEP 7 V5.5.	Follow the instructions of the installation program.
2	Install S7-SCL V5.3+SP5.	Follow the instructions of the installation program.
3	Install SIMATIC RF-MANAGER Basic 2010.	Follow the instructions of the installation program.

4.5 Installing the STEP 7 project

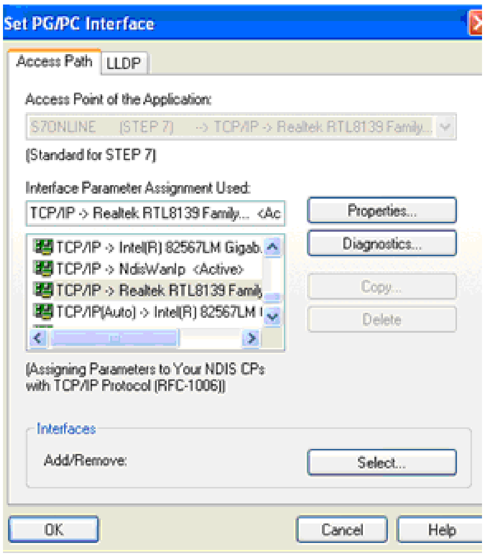
The following table lists the steps necessary to install the sample code.

Table 4-5

No.	Procedure
1	The project is available on the HTML page from which you downloaded this document. Save the **“23626344_RF670R_CODE_V10.zip”** project to your hard drive.
2	Open the **SIMATIC MANAGER** and retrieve the STEP 7 project. “File > Retrieve...”
3	The project is now available to you.

4.6 Setting the PG/PC interface

Table 4-6

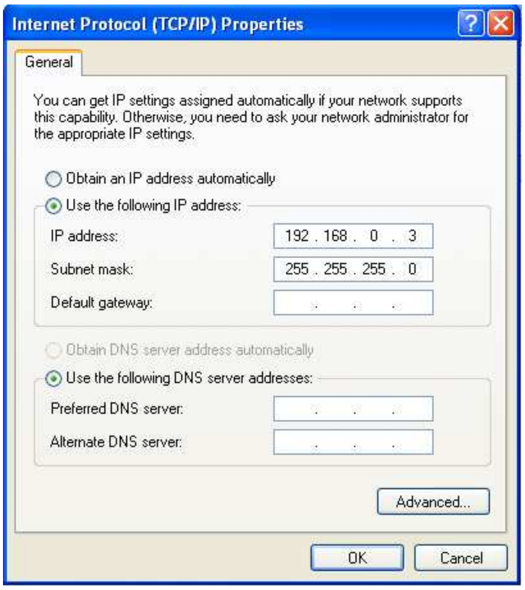
No.	Action	Remark
1	In the SIMATIC MANAGER, set the PC interface to TCP/IP “Options > Set PC/PG Interface…”.	 Set PG/PC Interface Access Path Access Point of the Application: S7ONLINE (STEP 7) --> TCP/IP -> Realtek RTL8139 Family... (Standard for STEP 7) Interface Parameter Assignment Used: TCP/IP -> Realtek RTL8139 Family... <Ac TCP/IP -> Intel(R) 82567LM Gigab. TCP/IP -> NdisWanIp <Active> TCP/IP -> Realtek RTL8139 Famil TCP/IP(Auto) -> Intel(R) 82567LM (Assigning Parameters to Your NDIS CPs with TCP/IP Protocol (RFC-1006)) LLDP Properties... Diagnostics... Copy... Delete Interfaces Add/Remove: Select... OK Cancel Help
2	Select the access path. For the used network card, select TCP/IP. Confirm with “OK”.	

4.7 Configuring the PG/PC

Changing the IP address

The figure shows the network setting to which you have to change the PG/PC!

Table 4-7

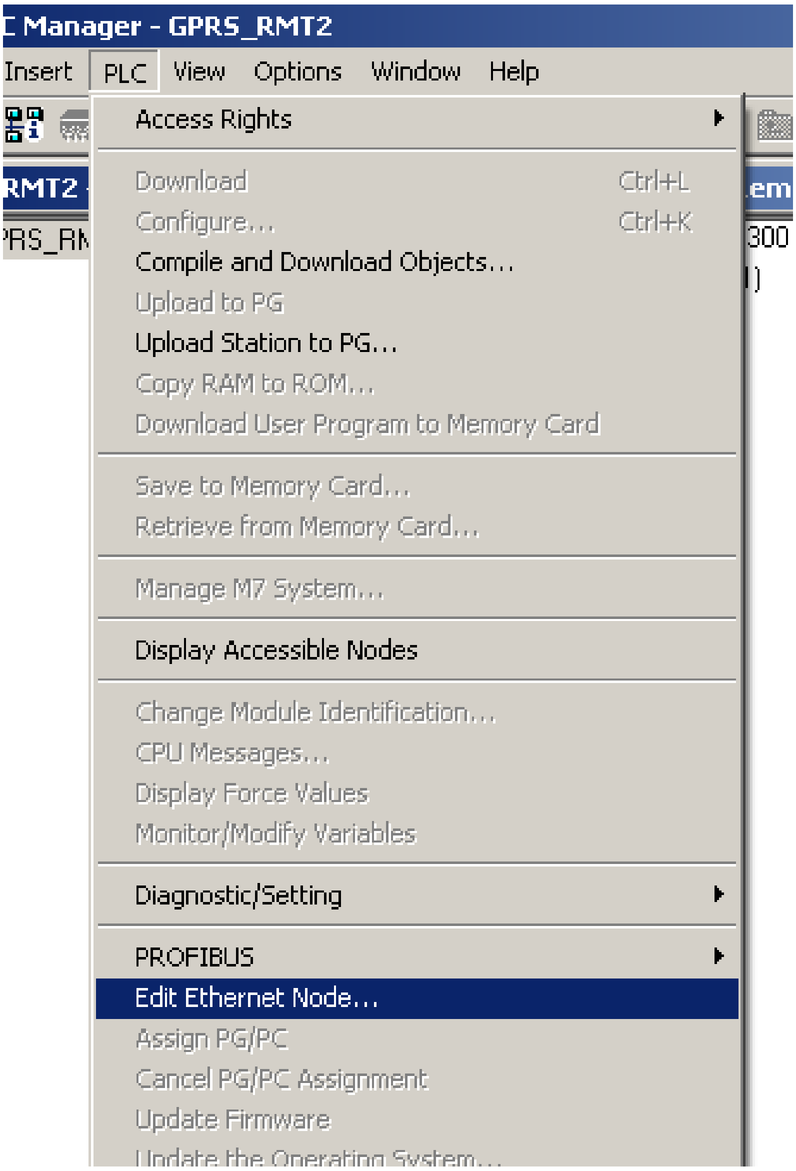
No.	Action	Remark/note
1	Open the Internet Protocol (TCP/IP) Properties by selecting "Start > Settings > Network Connection >Local Connections". In the open window, select Internet Protocol (TCP/IP) and open the Properties. Select "Use the following IP address" and fill out the field as shown in the screen shot. Close the dialog boxes with "OK".	Internet Protocol (TCP/IP) Properties General You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings. ○ Obtain an IP address automatically ● Use the following IP address: IP address: 192 . 168 . 0 . 3 Subnet mask: 255 . 255 . 255 . 0 Default gateway: . . . ○ Obtain DNS server address automatically ● Use the following DNS server addresses: Preferred DNS server: . . . Alternate DNS server: . . . Advanced... OK Cancel
2	If your PG has an IWLAN interface, disable it.	

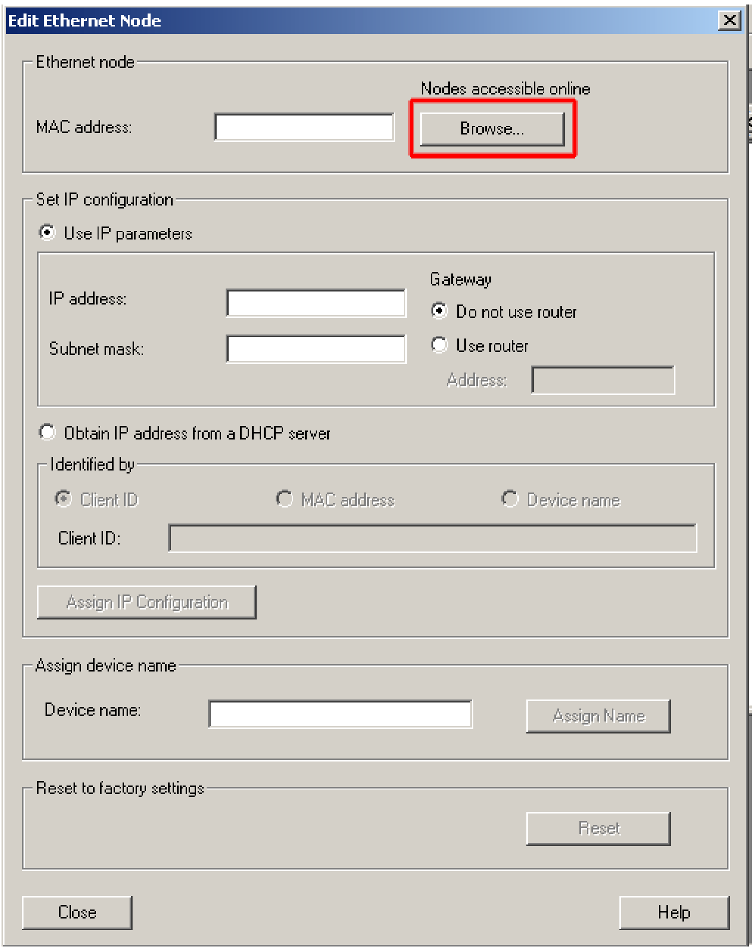
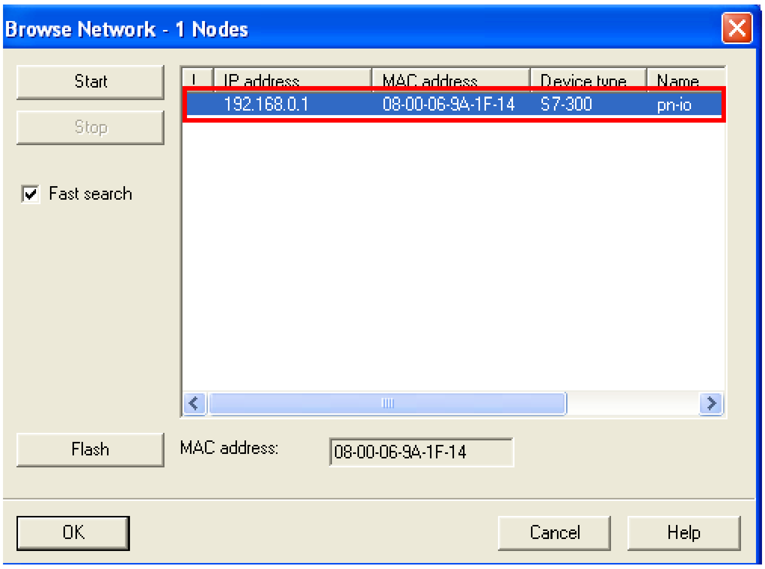
4.8 Configuring the S7 station

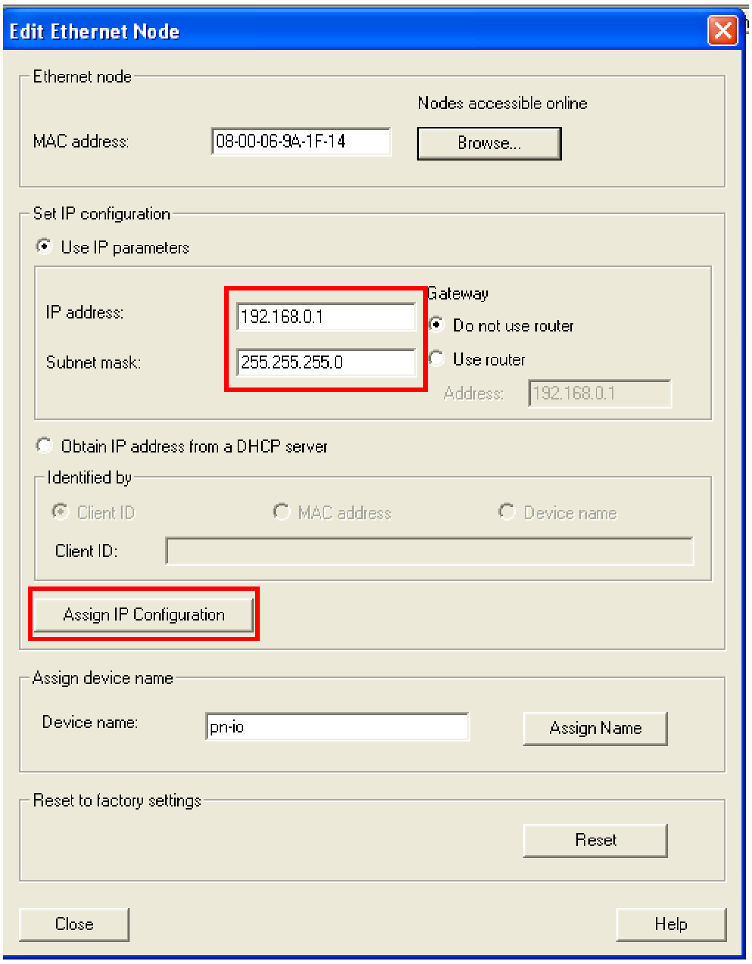
Changing the IP address of the CPU

Before the STEP 7 project can be downloaded to the CPU, you have to change the IP address of the S7-300/400 CPU as shown in Table 4-1 via which the project is downloaded to the CPU.

Table 4-8

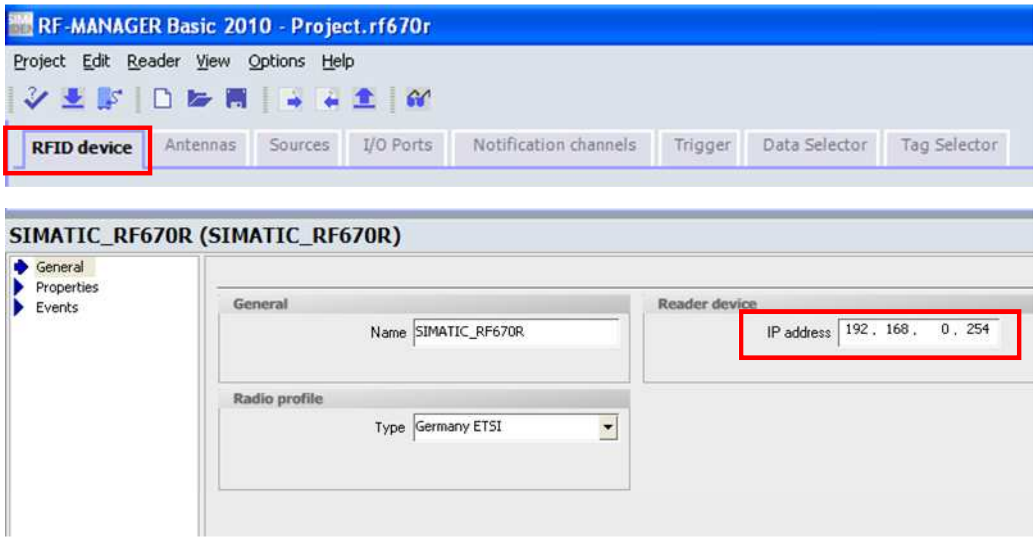
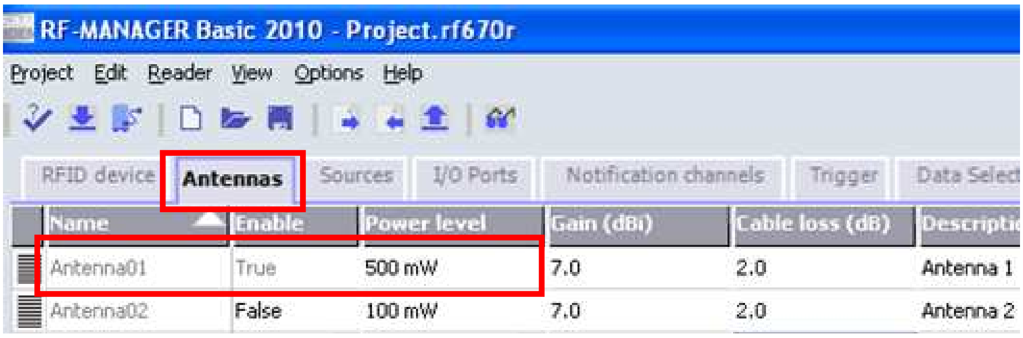
<table>
<tr><th>No.</th><th>Action</th></tr>
<tr><td>1</td><td>In the SIMATIC Manager, open a STEP 7 project.</td></tr>
<tr><td>2</td><td>In the “PLC” menu, select the “Edit Ethernet Node…” option.

</td></tr>
</table>

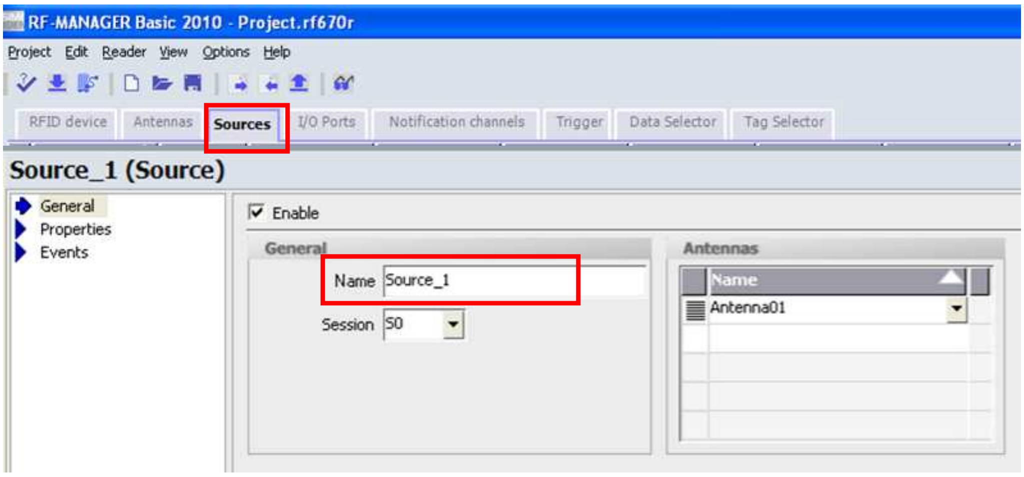
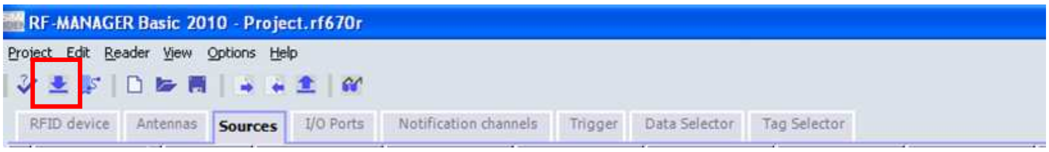
No.	Action
3	Click on the “Browse…” button.  Edit Ethernet Node Ethernet node Nodes accessible online MAC address: Browse... Set IP configuration ● Use IP parameters IP address: Subnet mask: Gateway ● Do not use router ○ Use router Address: ○ Obtain IP address from a DHCP server Identified by ● Client ID ○ MAC address ○ Device name Client ID: Assign IP Configuration Assign device name Device name: Assign Name Reset to factory settings Reset Close Help
4	Select the desired module and click on “OK” to confirm the selection.  Browse Network - 1 Nodes Start Stop ☑ Fast search IP address | MAC address | Device type | Name 192.168.0.1 | 08-00-06-9A-1F-14 | S7-300 | pn-io Flash MAC address: 08-00-06-9A-1F-14 OK Cancel Help

No.	Action
5	In the “Set IP configurations” window that appears, enter the IP address as shown in Table 4-1. Click on the “Assign IP Configuration” button. Close the dialog box with the “Close” button.  Edit Ethernet Node Ethernet node Nodes accessible online MAC address: 08-00-06-9A-1F-14 Browse... Set IP configuration ● Use IP parameters IP address: 192.168.0.1 Subnet mask: 255.255.255.0 Gateway ● Do not use router ○ Use router Address: 192.168.0.1 ○ Obtain IP address from a DHCP server Identified by ● Client ID ○ MAC address ○ Device name Client ID: Assign IP Configuration Assign device name Device name: pn-io Assign Name Reset to factory settings Reset Close Help

4.9 Configuring the RF670R reader with RF-MANAGER Basic 2010

Table 4-9

No.	Action	Remark
1	Open RF-MANAGER Basic 2010 and create a new project.	
2	Change the IP address of the RF670R "RFID device>General>IP address".	This IP address must be entered in DB RF_PARAMETER "IP_ADR" (this has already been done in the STEP 7 project).
	 RF-MANAGER Basic 2010 - Project.rf670r SIMATIC_RF670R (SIMATIC_RF670R) General: Name SIMATIC_RF670R Reader device: IP address 192 . 168 . 0 . 254 Radio profile: Type Germany ETSI	
3	Enable only the first antenna "Antennas>Antenna0X>Enable".	
4	Set the power level "Antennas>Antenna01>Power level".	The power level was increased to 500 mW. This is especially useful for writing to transponders.
	 RF-MANAGER Basic 2010 - Project.rf670r Name | Enable | Power level | Gain (dBi) | Cable loss (dB) | Descripti Antenna01 | True | 500 mW | 7.0 | 2.0 | Antenna 1 Antenna02 | False | 100 mW | 7.0 | 2.0 | Antenna 2	

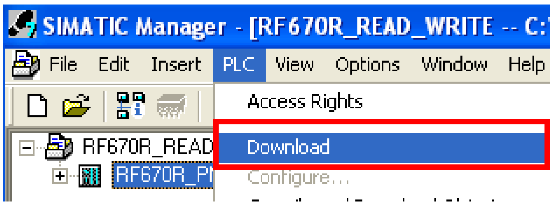
No.	Action	Remark
5	Enter the name of the data source for the antenna “Sources>Name”.	This name must match the entry in DB RF_PARAMETER “Source” (this has already been done in the STEP 7 project).
	 Source_1 (Source) ☑ Enable Name Source_1 Session S0 Antennas: Name Antenna01	
6	Download the configuration to the RF670R reader. 	

4.10 Downloading the STEP 7 project

Prerequisite:

- General CPU reset has been performed (see \7\, Chapter 8.4.3).
- The SIMATIC Micro Memory Card (MMC) of the CPU has been deleted.

Table 4-10

No.	Action	Remark
1	In the SIMATIC Manager, open the following STEP 7 project: “RF670R_READ_WRITE”.	
2	Select the “RF670R_PNCPU” S7 station and download the entire project to your CPU. “PLC > Download”	 SIMATIC Manager - [RF670R_READ_WRITE -- C: File Edit Insert PLC View Options Window Help Access Rights Download Configure... RF670R_READ RF670R_PI

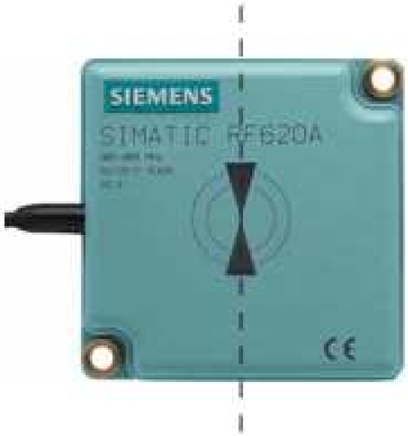
Note If you recompile all blocks, set the recompiled IDB_RF670R_RW instance data block to “Non Retain” to overwrite the respective instance DB with the initial values when restarting the CPU.

4.11 Orientation of the transponders with respect to the antenna

Polarization axis

As the RF620A antenna is linearly polarized, the correct orientation of the transponders used must be ensured with respect to the antenna. Generally, the polarization axes of antenna and transponders must run parallel to one another. The symbol on the antenna provides a graphical representation of how the polarization axis runs.

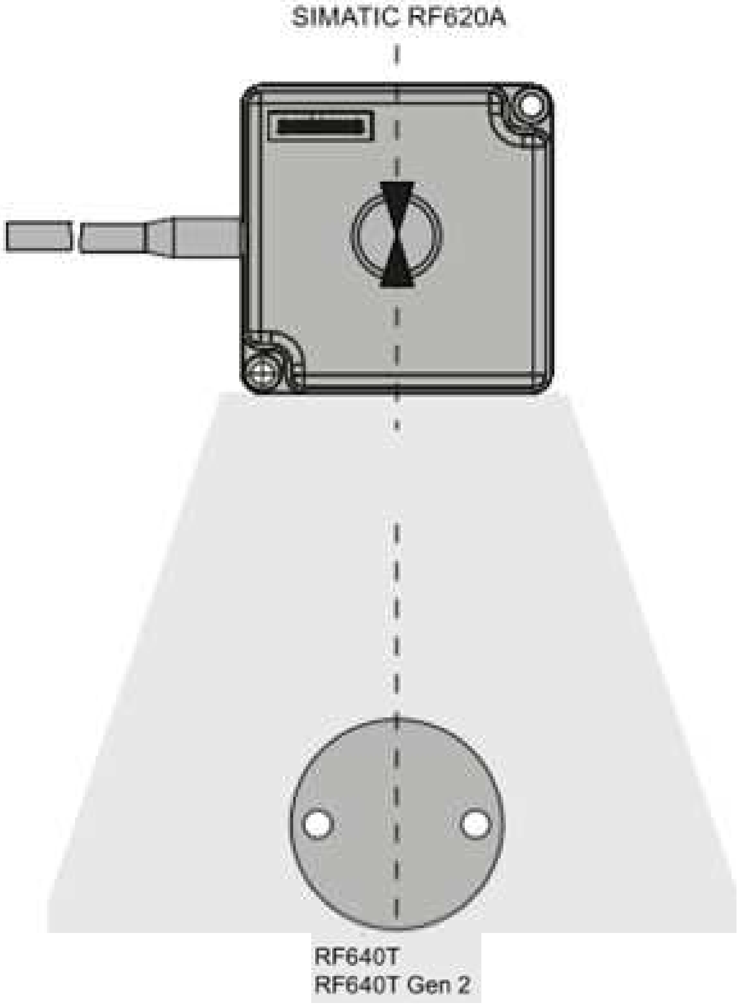
Figure 4-2 Polarization axis



Orientation

The figure below shows the optimum orientation of the RF640T transponders with respect to the RF620A antenna.

Figure 4-3 Antenna/transponder orientation

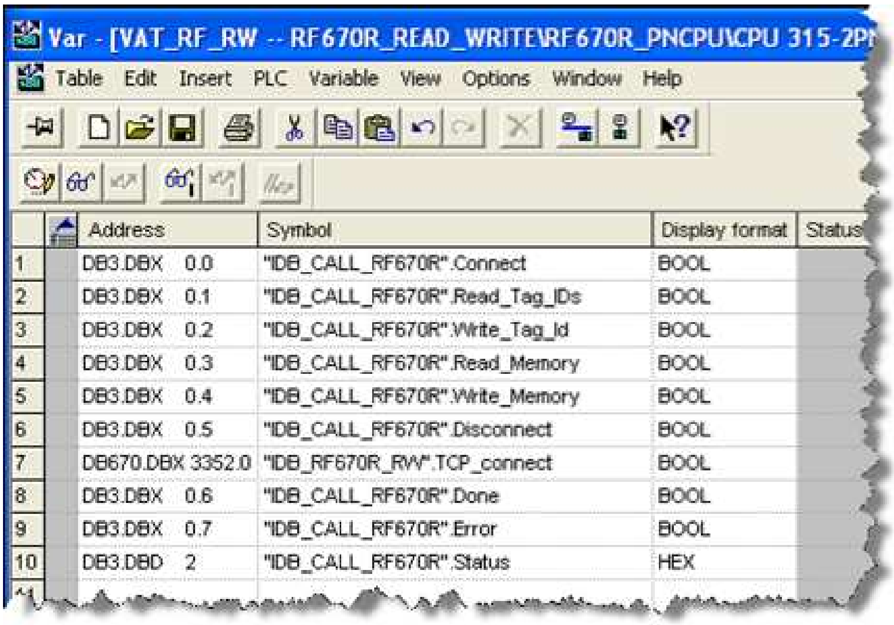


5 Operation of the Application

5.1 Overview

This chapter shows you how to operate the above-described functions of this application. All necessary variables can be found in the “VAT_RF_RW” variable table.

Figure 5-1


Var - [VAT_RF_RW -- RF670R_READ_WRITE\RF670R_PNCPU\CPU 315-2P
Table Edit Insert PLC Variable View Options Window Help

	Address	Symbol	Display format	Status
1	DB3.DBX 0.0	"IDB_CALL_RF670R".Connect	BOOL	
2	DB3.DBX 0.1	"IDB_CALL_RF670R".Read_Tag_IDs	BOOL	
3	DB3.DBX 0.2	"IDB_CALL_RF670R".Write_Tag_Id	BOOL	
4	DB3.DBX 0.3	"IDB_CALL_RF670R".Read_Memory	BOOL	
5	DB3.DBX 0.4	"IDB_CALL_RF670R".Write_Memory	BOOL	
6	DB3.DBX 0.5	"IDB_CALL_RF670R".Disconnect	BOOL	
7	DB670.DBX 3352.0	"IDB_RF670R_RW".TCP_connect	BOOL	
8	DB3.DBX 0.6	"IDB_CALL_RF670R".Done	BOOL	
9	DB3.DBX 0.7	"IDB_CALL_RF670R".Error	BOOL	
10	DB3.DBD 2	"IDB_CALL_RF670R".Status	HEX	

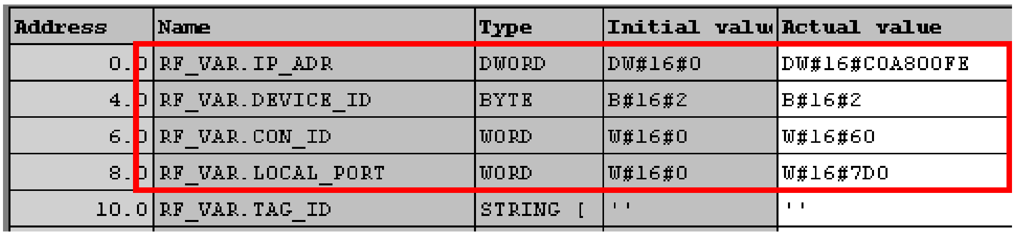
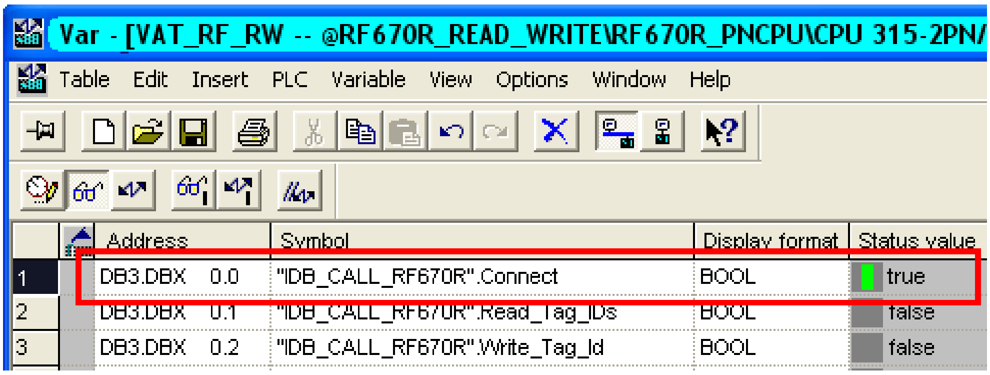
Tabelle 5-1

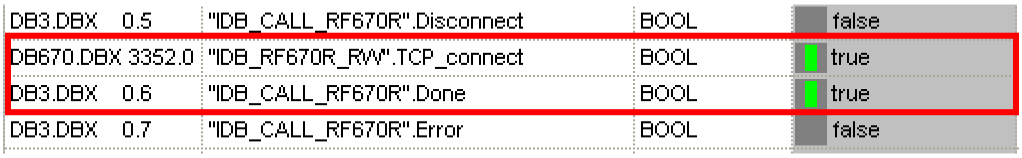
Symbol	Remark
„IDB_CALL_RF670R“.Connect	Connect
„IDB_CALL_RF670R“.Read_Tag_IDs	Read Transponder-IDs
„IDB_CALL_RF670R“.Write_Tag_id	Write Transponder-ID
„IDB_CALL_RF670R“.Read_Memory	Read RFID data
„IDB_CALL_RF670R“.Write_Memory	Write RFID data
„IDB_CALL_RF670R“.Disconnect	Disconnect
„IDB _RF670R_RW“.TCP_connect	Connection Status
„IDB_CALL_RF670R“.Done	Job has been completed without errors
„IDB_CALL_RF670R“.Error	Error
„IDB_CALL_RF670R“.Status	Error Status

5.2 Connecting to the RF670R

The table below lists instructions for establishing the connection to the RF670R reader.

Table 5-2 Connecting to the RF670R

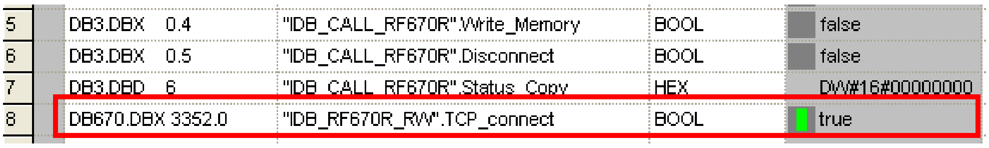
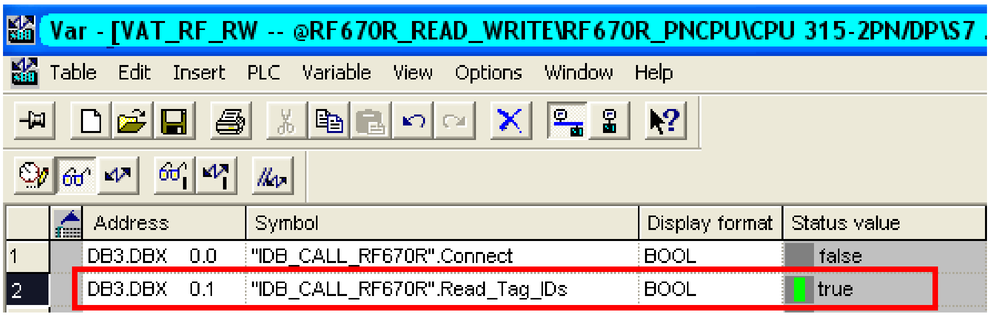
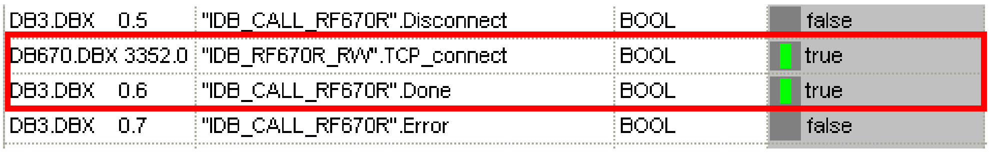
<table>
<tr><th>No.</th><th>Procedure</th></tr>
<tr><td>1</td><td>Open DB RF_PARAMETER and check if all parameters necessary for establishing the connection have been entered.
“View > Data View > Actual Value”

<table>
<tr><th>Address</th><th>Name</th><th>Type</th><th>Initial valu</th><th>Actual value</th></tr>
<tr><td>0.0</td><td>RF_VAR.IP_ADR</td><td>DWORD</td><td>DW#16#0</td><td>DW#16#C0A800FE</td></tr>
<tr><td>4.0</td><td>RF_VAR.DEVICE_ID</td><td>BYTE</td><td>B#16#2</td><td>B#16#2</td></tr>
<tr><td>6.0</td><td>RF_VAR.CON_ID</td><td>WORD</td><td>W#16#0</td><td>W#16#60</td></tr>
<tr><td>8.0</td><td>RF_VAR.LOCAL_PORT</td><td>WORD</td><td>W#16#0</td><td>W#16#7D0</td></tr>
<tr><td>10.0</td><td>RF_VAR.TAG_ID</td><td>STRING [</td><td>''</td><td>''</td></tr>
</table>
If changes are made, save and once again download the DB RF_PARAMETER data block.</td></tr>
<tr><td>2</td><td>Open the “VAT_RF_RW” variable table and enable "IDB_CALL_RF670R".Connect to establish the connection to the RF670R reader.

Var - [VAT_RF_RW -- @RF670R_READ_WRITE\RF670R_PNCPU\CPU 315-2PN/

<table>
<tr><th></th><th>Address</th><th></th><th>Symbol</th><th>Display format</th><th>Status value</th></tr>
<tr><td>1</td><td>DB3.DBX</td><td>0.0</td><td>"IDB_CALL_RF670R".Connect</td><td>BOOL</td><td>🟩 true</td></tr>
<tr><td>2</td><td>DB3.DBX</td><td>0.1</td><td>"IDB_CALL_RF670R".Read_Tag_IDs</td><td>BOOL</td><td>false</td></tr>
<tr><td>3</td><td>DB3.DBX</td><td>0.2</td><td>"IDB_CALL_RF670R".Write_Tag_Id</td><td>BOOL</td><td>false</td></tr>
</table></td></tr>
<tr><td>3</td><td>If the connection was successfully established, “IDB_RF670R_RW”.TCP_connect and "IDB_CALL_RF670R".Done will be set.

<table>
<tr><td>DB3.DBX</td><td>0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>false</td></tr>
<tr><td>DB670.DBX</td><td>3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>🟩 true</td></tr>
<tr><td>DB3.DBX</td><td>0.6</td><td>"IDB_CALL_RF670R".Done</td><td>BOOL</td><td>🟩 true</td></tr>
<tr><td>DB3.DBX</td><td>0.7</td><td>"IDB_CALL_RF670R".Error</td><td>BOOL</td><td>false</td></tr>
</table></td></tr>
</table>

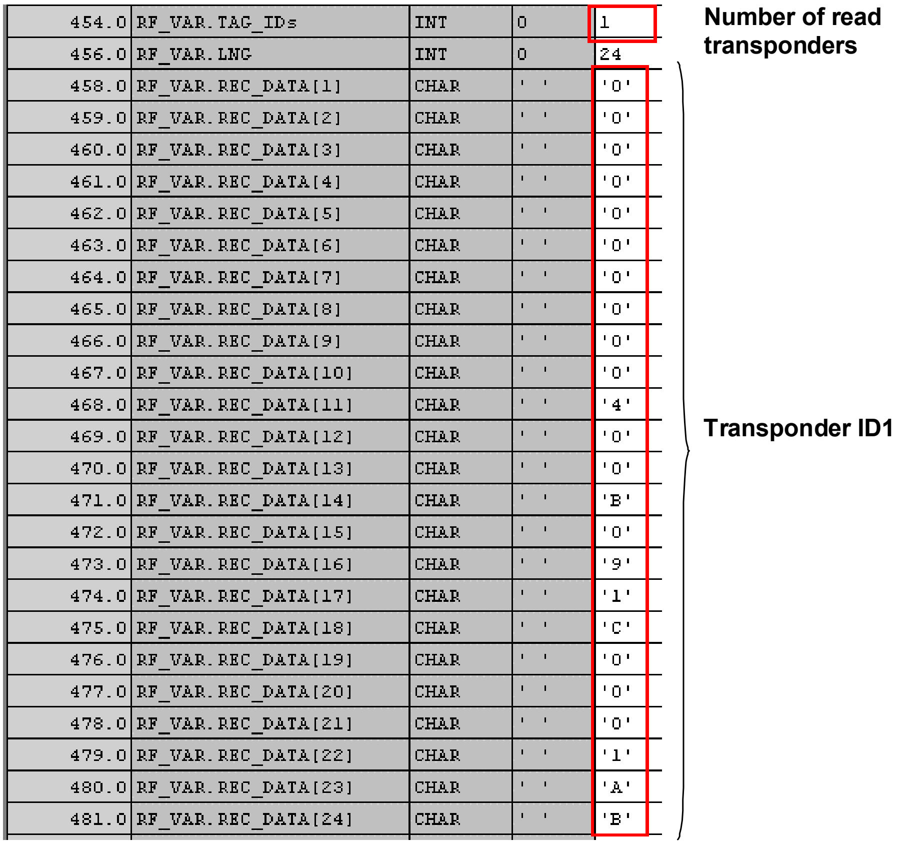
5.3 Reading the transponder ID

The table below lists instructions for reading the transponder IDs.

Table 5-3 Reading the transponder IDs

<table>
<tr><th>No.</th><th>Procedure</th></tr>
<tr><td>1</td><td>Check if “IDB_RF670R_RW”.TCP_connect has the “TRUE” status <b>(*1)</b>.

<table>
<tr><td>5</td><td>DB3.DBX 0.4</td><td>"IDB_CALL_RF670R".Write_Memory</td><td>BOOL</td><td>■ false</td></tr>
<tr><td>6</td><td>DB3.DBX 0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>■ false</td></tr>
<tr><td>7</td><td>DB3.DBD 6</td><td>"IDB_CALL_RF670R".Status_Copy</td><td>HEX</td><td>DW#16#00000000</td></tr>
<tr><td>8</td><td>DB670.DBX 3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>🟩 true</td></tr>
</table></td></tr>
<tr><td>2</td><td>Enable "IDB_CALL_RF670R".Read_Tag_IDs to read the transponder IDs of the transponders in the field.

Var - [VAT_RF_RW -- @RF670R_READ_WRITE\RF670R_PNCPU\CPU 315-2PN/DP\S7
Table
Edit
Insert
PLC
Variable
View
Options
Window
Help
<table>
<tr><th></th><th>Address</th><th>Symbol</th><th>Display format</th><th>Status value</th></tr>
<tr><td>1</td><td>DB3.DBX 0.0</td><td>"IDB_CALL_RF670R".Connect</td><td>BOOL</td><td>■ false</td></tr>
<tr><td>2</td><td>DB3.DBX 0.1</td><td>"IDB_CALL_RF670R".Read_Tag_IDs</td><td>BOOL</td><td>🟩 true</td></tr>
</table></td></tr>
<tr><td>3</td><td>If the process was successfully completed, "IDB_CALL_RF670R".Done will be set.

<table>
<tr><td>DB3.DBX 0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>■ false</td></tr>
<tr><td>DB670.DBX 3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>🟩 true</td></tr>
<tr><td>DB3.DBX 0.6</td><td>"IDB_CALL_RF670R".Done</td><td>BOOL</td><td>🟩 true</td></tr>
<tr><td>DB3.DBX 0.7</td><td>"IDB_CALL_RF670R".Error</td><td>BOOL</td><td>■ false</td></tr>
</table></td></tr>
</table>

No.	Procedure
4	Open DB RF_PARAMETER. If the read operation has been successfully completed, the transponder IDs must be stored in the RF_VAR.REC_DATA field.



454.0	RF_VAR.TAG_IDs	INT	0	1
456.0	RF_VAR.LNG	INT	0	24
458.0	RF_VAR.REC_DATA[1]	CHAR	' '	'0'
459.0	RF_VAR.REC_DATA[2]	CHAR	' '	'0'
460.0	RF_VAR.REC_DATA[3]	CHAR	' '	'0'
461.0	RF_VAR.REC_DATA[4]	CHAR	' '	'0'
462.0	RF_VAR.REC_DATA[5]	CHAR	' '	'0'
463.0	RF_VAR.REC_DATA[6]	CHAR	' '	'0'
464.0	RF_VAR.REC_DATA[7]	CHAR	' '	'0'
465.0	RF_VAR.REC_DATA[8]	CHAR	' '	'0'
466.0	RF_VAR.REC_DATA[9]	CHAR	' '	'0'
467.0	RF_VAR.REC_DATA[10]	CHAR	' '	'0'
468.0	RF_VAR.REC_DATA[11]	CHAR	' '	'4'
469.0	RF_VAR.REC_DATA[12]	CHAR	' '	'0'
470.0	RF_VAR.REC_DATA[13]	CHAR	' '	'0'
471.0	RF_VAR.REC_DATA[14]	CHAR	' '	'B'
472.0	RF_VAR.REC_DATA[15]	CHAR	' '	'0'
473.0	RF_VAR.REC_DATA[16]	CHAR	' '	'9'
474.0	RF_VAR.REC_DATA[17]	CHAR	' '	'1'
475.0	RF_VAR.REC_DATA[18]	CHAR	' '	'C'
476.0	RF_VAR.REC_DATA[19]	CHAR	' '	'0'
477.0	RF_VAR.REC_DATA[20]	CHAR	' '	'0'
478.0	RF_VAR.REC_DATA[21]	CHAR	' '	'0'
479.0	RF_VAR.REC_DATA[22]	CHAR	' '	'1'
480.0	RF_VAR.REC_DATA[23]	CHAR	' '	'A'
481.0	RF_VAR.REC_DATA[24]	CHAR	' '	'B'

Number of read transponders

Transponder ID1

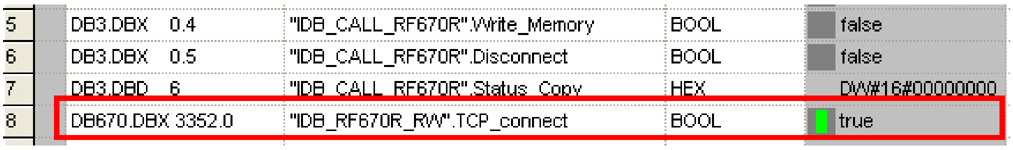
(*1): A connection to the RF670R reader must already exist.

5.4 Writing the transponder ID

The table below lists instructions for writing the transponder ID.

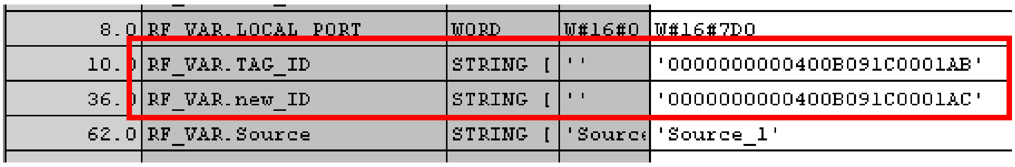
Table 5-4 Writing the transponder ID

No.	Procedure
1	Check if “IDB_RF670R_RW”.TCP_connect has the “TRUE” status **(*1)**.



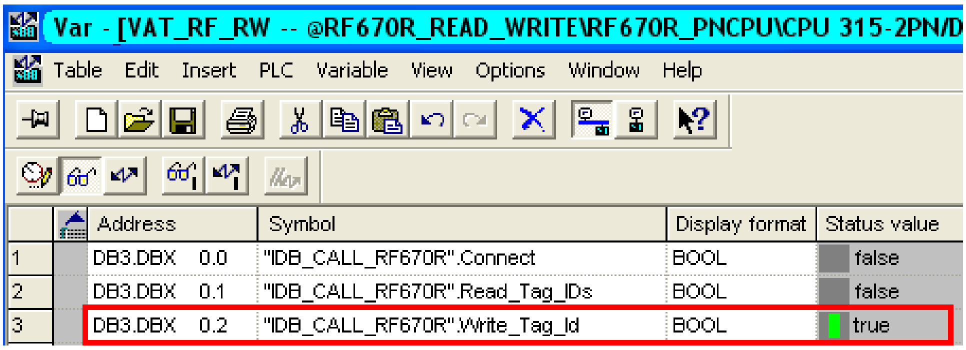
5	DB3.DBX 0.4	"IDB_CALL_RF670R".Write_Memory	BOOL	▨ false
6	DB3.DBX 0.5	"IDB_CALL_RF670R".Disconnect	BOOL	▨ false
7	DB3.DBD 6	"IDB_CALL_RF670R".Status_Copy	HEX	DW#16#00000000
8	DB670.DBX 3352.0	"IDB_RF670R_RW".TCP_connect	BOOL	🟩 true

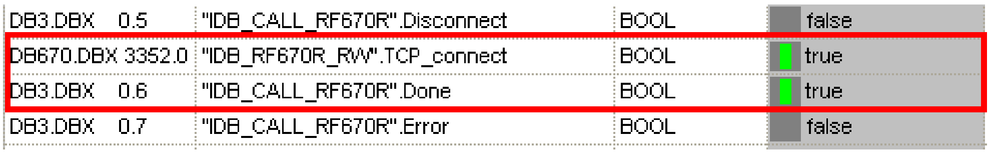
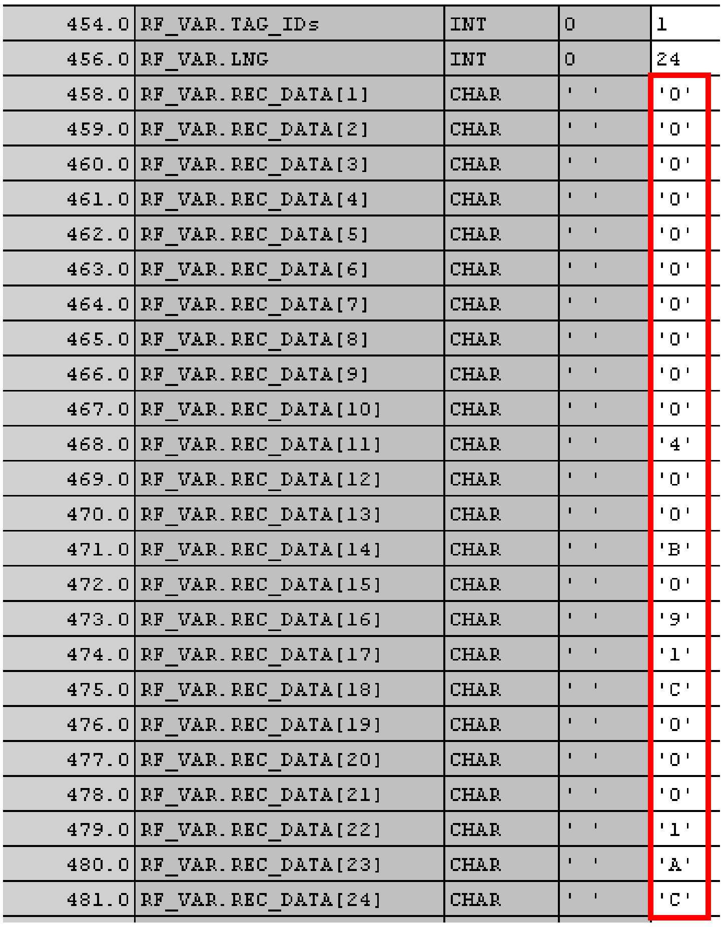
No.	Procedure
2	Enter the current and the new transponder ID in RF_PARAMETER to write the transponder ID of a transponder.



8.0	RF_VAR.LOCAL_PORT	WORD	W#16#0	W#16#7D0
10.0	RF_VAR.TAG_ID	STRING [	''	'0000000000400B091C0001AB'
36.0	RF_VAR.new_ID	STRING [	''	'0000000000400B091C0001AC'
62.0	RF_VAR.Source	STRING [	'Source	'Source_1'

Save and once again download the DB RF_PARAMETER data block.

<table>
<tr><th>No.</th><th>Procedure</th></tr>
<tr><td>3</td><td>Enable "IDB_CALL_RF670R".Write_Tag_ID.


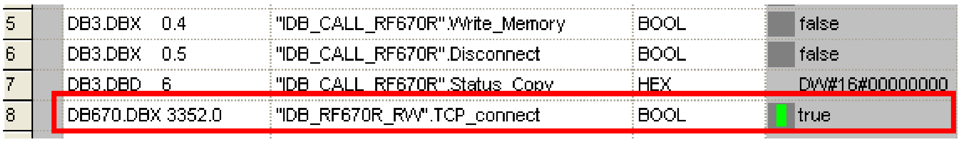
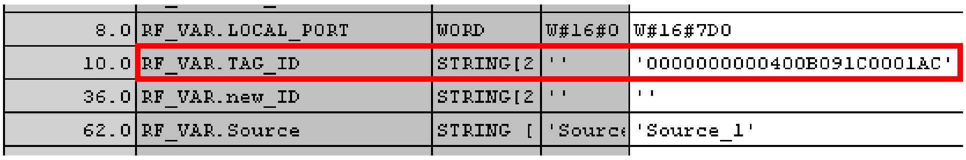
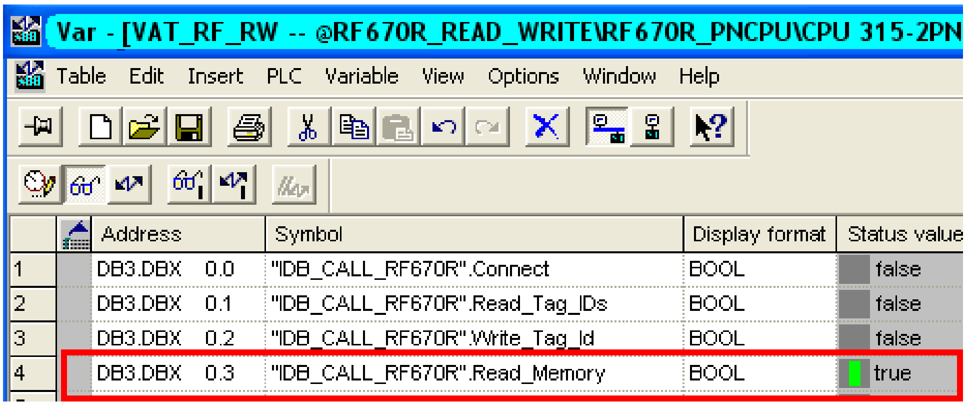
Var - [VAT_RF_RW -- @RF670R_READ_WRITE\RF670R_PNCPU\CPU 315-2PN/D
<table>
<tr><th></th><th>Address</th><th></th><th>Symbol</th><th>Display format</th><th>Status value</th></tr>
<tr><td>1</td><td>DB3.DBX</td><td>0.0</td><td>"IDB_CALL_RF670R".Connect</td><td>BOOL</td><td>false</td></tr>
<tr><td>2</td><td>DB3.DBX</td><td>0.1</td><td>"IDB_CALL_RF670R".Read_Tag_IDs</td><td>BOOL</td><td>false</td></tr>
<tr><td>3</td><td>DB3.DBX</td><td>0.2</td><td>"IDB_CALL_RF670R".Write_Tag_Id</td><td>BOOL</td><td>🟩 true</td></tr>
</table></td></tr>
<tr><td>4</td><td>If the process was successfully completed, "IDB_CALL_RF670R".Done will be set.

<table>
<tr><td>DB3.DBX</td><td>0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>false</td></tr>
<tr><td>DB670.DBX</td><td>3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>🟩 true</td></tr>
<tr><td>DB3.DBX</td><td>0.6</td><td>"IDB_CALL_RF670R".Done</td><td>BOOL</td><td>🟩 true</td></tr>
<tr><td>DB3.DBX</td><td>0.7</td><td>"IDB_CALL_RF670R".Error</td><td>BOOL</td><td>false</td></tr>
</table></td></tr>
<tr><td>5</td><td>Follow steps 1-2 of Table 5-2 to check if the new transponder ID was successfully written to the transponder.
Result in DB RF_PARAMETER:

<table>
<tr><td>454.0</td><td>RF_VAR.TAG_IDs</td><td>INT</td><td>0</td><td>1</td></tr>
<tr><td>456.0</td><td>RF_VAR.LNG</td><td>INT</td><td>0</td><td>24</td></tr>
<tr><td>458.0</td><td>RF_VAR.REC_DATA[1]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>459.0</td><td>RF_VAR.REC_DATA[2]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>460.0</td><td>RF_VAR.REC_DATA[3]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>461.0</td><td>RF_VAR.REC_DATA[4]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>462.0</td><td>RF_VAR.REC_DATA[5]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>463.0</td><td>RF_VAR.REC_DATA[6]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>464.0</td><td>RF_VAR.REC_DATA[7]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>465.0</td><td>RF_VAR.REC_DATA[8]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>466.0</td><td>RF_VAR.REC_DATA[9]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>467.0</td><td>RF_VAR.REC_DATA[10]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>468.0</td><td>RF_VAR.REC_DATA[11]</td><td>CHAR</td><td>' '</td><td>'4'</td></tr>
<tr><td>469.0</td><td>RF_VAR.REC_DATA[12]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>470.0</td><td>RF_VAR.REC_DATA[13]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>471.0</td><td>RF_VAR.REC_DATA[14]</td><td>CHAR</td><td>' '</td><td>'B'</td></tr>
<tr><td>472.0</td><td>RF_VAR.REC_DATA[15]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>473.0</td><td>RF_VAR.REC_DATA[16]</td><td>CHAR</td><td>' '</td><td>'9'</td></tr>
<tr><td>474.0</td><td>RF_VAR.REC_DATA[17]</td><td>CHAR</td><td>' '</td><td>'1'</td></tr>
<tr><td>475.0</td><td>RF_VAR.REC_DATA[18]</td><td>CHAR</td><td>' '</td><td>'C'</td></tr>
<tr><td>476.0</td><td>RF_VAR.REC_DATA[19]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>477.0</td><td>RF_VAR.REC_DATA[20]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>478.0</td><td>RF_VAR.REC_DATA[21]</td><td>CHAR</td><td>' '</td><td>'0'</td></tr>
<tr><td>479.0</td><td>RF_VAR.REC_DATA[22]</td><td>CHAR</td><td>' '</td><td>'1'</td></tr>
<tr><td>480.0</td><td>RF_VAR.REC_DATA[23]</td><td>CHAR</td><td>' '</td><td>'A'</td></tr>
<tr><td>481.0</td><td>RF_VAR.REC_DATA[24]</td><td>CHAR</td><td>' '</td><td>'C'</td></tr>
</table></td></tr>
</table>

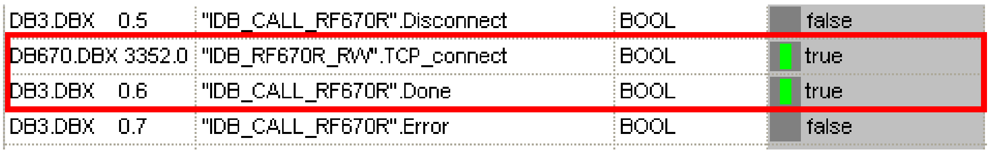
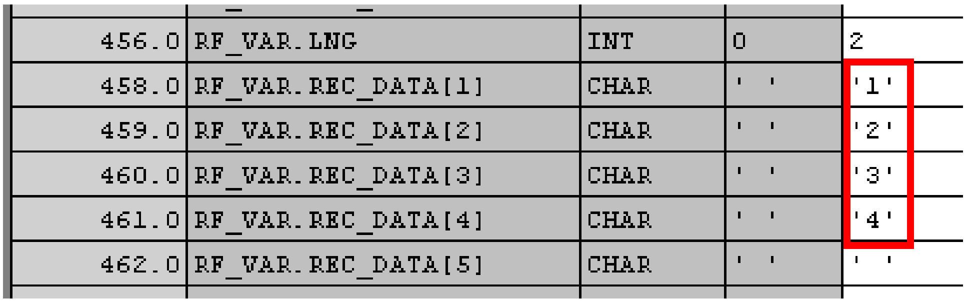
(*1): A connection to the RF670R reader must already exist.

5.5 Reading RFID data from a transponder

The table below lists instructions for reading RFID data from a transponder.

Table 5-5 Reading RFID data from a transponder

<table>
<tr><th>No.</th><th>Procedure</th></tr>
<tr><td>1</td><td>Check if “IDB_RF670R_RW”.TCP_connect has the “TRUE” status <b>(*1)</b>.

<table>
<tr><td>5</td><td>DB3.DBX 0.4</td><td>"IDB_CALL_RF670R".Write_Memory</td><td>BOOL</td><td>▨ false</td></tr>
<tr><td>6</td><td>DB3.DBX 0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>▨ false</td></tr>
<tr><td>7</td><td>DB3.DBD 6</td><td>"IDB_CALL_RF670R".Status_Copy</td><td>HEX</td><td>DW#16#00000000</td></tr>
<tr><td>8</td><td>DB670.DBX 3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>🟩 true</td></tr>
</table></td></tr>
<tr><td>2</td><td>Enter the tag ID of the transponder whose memory you want to read in RF_PARAMETER.

<table>
<tr><td>8.0</td><td>RF_VAR.LOCAL_PORT</td><td>WORD</td><td>W#16#0</td><td>W#16#7D0</td></tr>
<tr><td>10.0</td><td>RF_VAR.TAG_ID</td><td>STRING[2</td><td>''</td><td>'00000000000400B091C0001AC'</td></tr>
<tr><td>36.0</td><td>RF_VAR.new_ID</td><td>STRING[2</td><td>''</td><td>''</td></tr>
<tr><td>62.0</td><td>RF_VAR.Source</td><td>STRING [</td><td>'Source</td><td>'Source_1'</td></tr>
</table>
Save and once again download the DB RF_PARAMETER data block.</td></tr>
<tr><td>3</td><td>Enable "IDB_CALL_RF670R".Read_Memory.


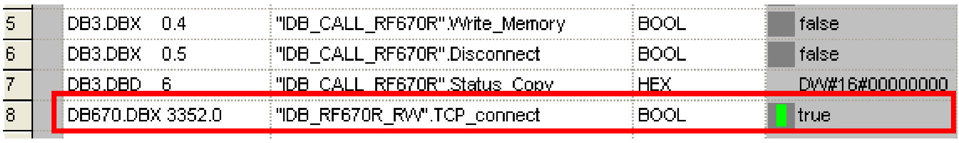
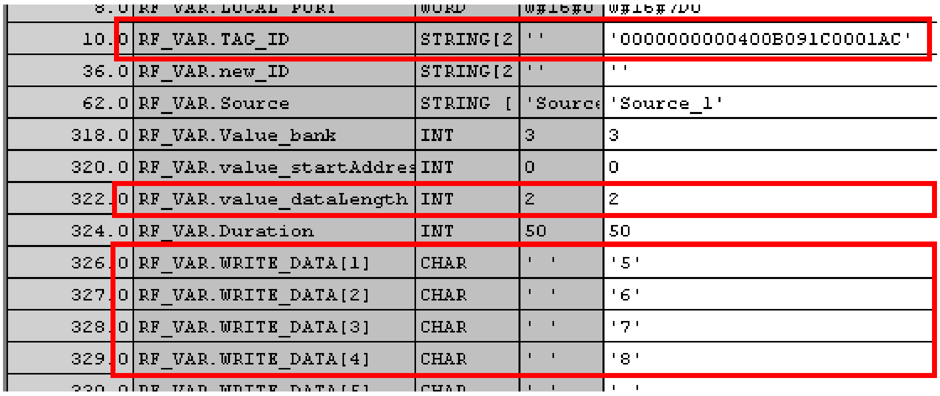
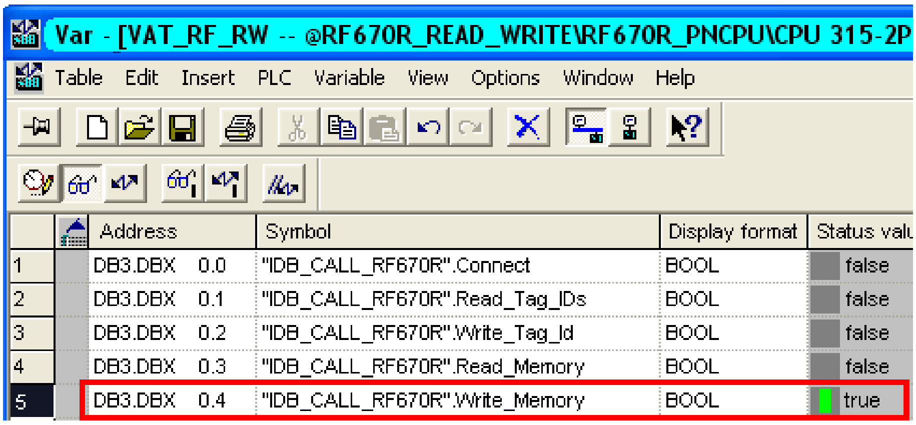
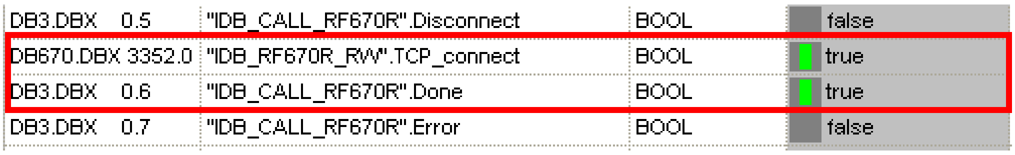
Var - [VAT_RF_RW -- @RF670R_READ_WRITE\RF670R_PNCPU\CPU 315-2PN
<table>
<tr><td></td><td>Address</td><td>Symbol</td><td>Display format</td><td>Status value</td></tr>
<tr><td>1</td><td>DB3.DBX 0.0</td><td>"IDB_CALL_RF670R".Connect</td><td>BOOL</td><td>▨ false</td></tr>
<tr><td>2</td><td>DB3.DBX 0.1</td><td>"IDB_CALL_RF670R".Read_Tag_IDs</td><td>BOOL</td><td>▨ false</td></tr>
<tr><td>3</td><td>DB3.DBX 0.2</td><td>"IDB_CALL_RF670R".Write_Tag_Id</td><td>BOOL</td><td>▨ false</td></tr>
<tr><td>4</td><td>DB3.DBX 0.3</td><td>"IDB_CALL_RF670R".Read_Memory</td><td>BOOL</td><td>🟩 true</td></tr>
</table></td></tr>
<tr><td>4</td><td>If the process was successfully completed, "IDB_CALL_RF670R".Done will be set.

<table>
<tr><td>DB3.DBX 0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>▨ false</td></tr>
<tr><td>DB670.DBX 3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>🟩 true</td></tr>
<tr><td>DB3.DBX 0.6</td><td>"IDB_CALL_RF670R".Done</td><td>BOOL</td><td>🟩 true</td></tr>
<tr><td>DB3.DBX 0.7</td><td>"IDB_CALL_RF670R".Error</td><td>BOOL</td><td>▨ false</td></tr>
</table></td></tr>
<tr><td>5</td><td>Open DB RF_PARAMETER. If the read operation has been successfully completed, the RFID data must be stored in the RF_VAR.REC_DATA field.

<table>
<tr><td>456.0</td><td>RF_VAR.LNG</td><td>INT</td><td>0</td><td>2</td></tr>
<tr><td>458.0</td><td>RF_VAR.REC_DATA[1]</td><td>CHAR</td><td>' '</td><td>'1'</td></tr>
<tr><td>459.0</td><td>RF_VAR.REC_DATA[2]</td><td>CHAR</td><td>' '</td><td>'2'</td></tr>
<tr><td>460.0</td><td>RF_VAR.REC_DATA[3]</td><td>CHAR</td><td>' '</td><td>'3'</td></tr>
<tr><td>461.0</td><td>RF_VAR.REC_DATA[4]</td><td>CHAR</td><td>' '</td><td>'4'</td></tr>
<tr><td>462.0</td><td>RF_VAR.REC_DATA[5]</td><td>CHAR</td><td>' '</td><td>' '</td></tr>
</table></td></tr>
</table>

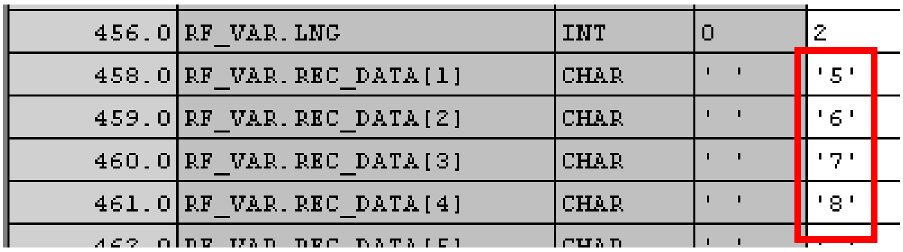
(*1): A connection to the RF670R reader must already exist.

5.6 Writing RFID data to a transponder

The table below lists instructions for writing RFID data to a transponder.

Table 5-6 Writing RFID data to a transponder

<table>
<tr><th>No.</th><th>Procedure</th></tr>
<tr><td>1</td><td>Check if “IDB_RF670R_RW”.TCP_connect has the “TRUE” status <b>(*1)</b>.

<table>
<tr><td>5</td><td>DB3.DBX 0.4</td><td>"IDB_CALL_RF670R".Write_Memory</td><td>BOOL</td><td>false</td></tr>
<tr><td>6</td><td>DB3.DBX 0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>false</td></tr>
<tr><td>7</td><td>DB3.DBD 6</td><td>"IDB_CALL_RF670R".Status_Copy</td><td>HEX</td><td>DW#16#00000000</td></tr>
<tr><td>8</td><td>DB670.DBX 3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>true</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>Enter the transponder ID of the transponder whose memory you want to write, the length of the RFID data and the RFID data in RF_PARAMETER.

<table>
<tr><td>8.0</td><td>RF_VAR.LOCAL_PORT</td><td>WORD</td><td>W#16#0</td><td>W#16#7D0</td></tr>
<tr><td>10.0</td><td>RF_VAR.TAG_ID</td><td>STRING[2</td><td>''</td><td>'0000000000400B091C0001AC'</td></tr>
<tr><td>36.0</td><td>RF_VAR.new_ID</td><td>STRING[2</td><td>''</td><td>''</td></tr>
<tr><td>62.0</td><td>RF_VAR.Source</td><td>STRING [</td><td>'Source</td><td>'Source_1'</td></tr>
<tr><td>318.0</td><td>RF_VAR.Value_bank</td><td>INT</td><td>3</td><td>3</td></tr>
<tr><td>320.0</td><td>RF_VAR.value_startAddres</td><td>INT</td><td>0</td><td>0</td></tr>
<tr><td>322.0</td><td>RF_VAR.value_dataLength</td><td>INT</td><td>2</td><td>2</td></tr>
<tr><td>324.0</td><td>RF_VAR.Duration</td><td>INT</td><td>50</td><td>50</td></tr>
<tr><td>326.0</td><td>RF_VAR.WRITE_DATA[1]</td><td>CHAR</td><td>' '</td><td>'5'</td></tr>
<tr><td>327.0</td><td>RF_VAR.WRITE_DATA[2]</td><td>CHAR</td><td>' '</td><td>'6'</td></tr>
<tr><td>328.0</td><td>RF_VAR.WRITE_DATA[3]</td><td>CHAR</td><td>' '</td><td>'7'</td></tr>
<tr><td>329.0</td><td>RF_VAR.WRITE_DATA[4]</td><td>CHAR</td><td>' '</td><td>'8'</td></tr>
<tr><td>330.0</td><td>RF_VAR.WRITE_DATA[5]</td><td>CHAR</td><td>' '</td><td>' '</td></tr>
</table>
Save and once again download the DB RF_PARAMETER data block.</td></tr>
<tr><td>3</td><td>Enable "IDB_CALL_RF670R".Write_Memory.

Var - [VAT_RF_RW -- @RF670R_READ_WRITE\RF670R_PNCPU\CPU 315-2P
Table Edit Insert PLC Variable View Options Window Help
<table>
<tr><th></th><th>Address</th><th>Symbol</th><th>Display format</th><th>Status valu</th></tr>
<tr><td>1</td><td>DB3.DBX 0.0</td><td>"IDB_CALL_RF670R".Connect</td><td>BOOL</td><td>false</td></tr>
<tr><td>2</td><td>DB3.DBX 0.1</td><td>"IDB_CALL_RF670R".Read_Tag_IDs</td><td>BOOL</td><td>false</td></tr>
<tr><td>3</td><td>DB3.DBX 0.2</td><td>"IDB_CALL_RF670R".Write_Tag_Id</td><td>BOOL</td><td>false</td></tr>
<tr><td>4</td><td>DB3.DBX 0.3</td><td>"IDB_CALL_RF670R".Read_Memory</td><td>BOOL</td><td>false</td></tr>
<tr><td>5</td><td>DB3.DBX 0.4</td><td>"IDB_CALL_RF670R".Write_Memory</td><td>BOOL</td><td>true</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>If the process was successfully completed, "IDB_CALL_RF670R".Done will be set.

<table>
<tr><td>DB3.DBX 0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>false</td></tr>
<tr><td>DB670.DBX 3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>true</td></tr>
<tr><td>DB3.DBX 0.6</td><td>"IDB_CALL_RF670R".Done</td><td>BOOL</td><td>true</td></tr>
<tr><td>DB3.DBX 0.7</td><td>"IDB_CALL_RF670R".Error</td><td>BOOL</td><td>false</td></tr>
</table>
</td></tr>
</table>

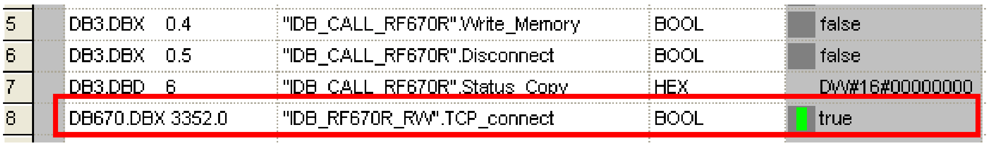
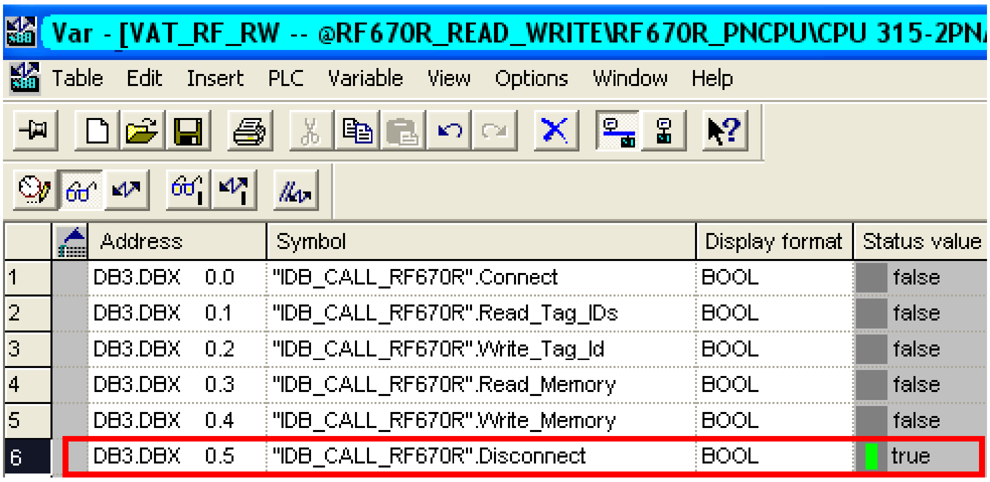
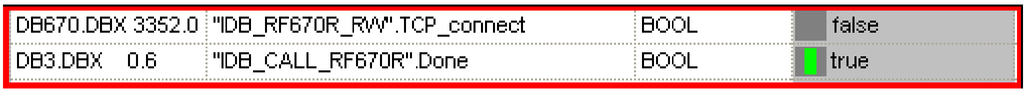
<table>
<tr><th>No.</th><th>Procedure</th></tr>
<tr><td>5</td><td>Follow steps 1-3 of Table 5-4 to check if the new RFID data was successfully written to the transponder.
Result in DB RF_PARAMETER:

<table>
<tr><td>456.0</td><td>RF_VAR.LNG</td><td>INT</td><td>0</td><td>2</td></tr>
<tr><td>458.0</td><td>RF_VAR.REC_DATA[1]</td><td>CHAR</td><td>' '</td><td>'5'</td></tr>
<tr><td>459.0</td><td>RF_VAR.REC_DATA[2]</td><td>CHAR</td><td>' '</td><td>'6'</td></tr>
<tr><td>460.0</td><td>RF_VAR.REC_DATA[3]</td><td>CHAR</td><td>' '</td><td>'7'</td></tr>
<tr><td>461.0</td><td>RF_VAR.REC_DATA[4]</td><td>CHAR</td><td>' '</td><td>'8'</td></tr>
<tr><td>462.0</td><td>RF_VAR.REC_DATA[5]</td><td>CHAR</td><td>' '</td><td></td></tr>
</table></td></tr>
</table>

(*1): A connection to the RF670R reader must already exist.

5.7 Disconnecting

The table below provides instructions for terminating the connection.

Table 5-7 Disconnecting

<table>
<tr><th>No.</th><th>Procedure</th></tr>
<tr><td>1</td><td>Check if “IDB_RF670R_RW”.TCP_connect has the “TRUE” status <b>(*1)</b>.

<table>
<tr><td>5</td><td>DB3.DBX 0.4</td><td>"IDB_CALL_RF670R".Write_Memory</td><td>BOOL</td><td>false</td></tr>
<tr><td>6</td><td>DB3.DBX 0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>false</td></tr>
<tr><td>7</td><td>DB3.DBD 6</td><td>"IDB_CALL_RF670R".Status_Copy</td><td>HEX</td><td>DW#16#00000000</td></tr>
<tr><td>8</td><td>DB670.DBX 3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>true</td></tr>
</table></td></tr>
<tr><td>2</td><td>Enable "IDB_CALL_RF670R".Disconnect to disconnect from the RF670R.

Var - [VAT_RF_RW -- @RF670R_READ_WRITE\RF670R_PNCPU\CPU 315-2PN/
Table Edit Insert PLC Variable View Options Window Help
<table>
<tr><th></th><th>Address</th><th>Symbol</th><th>Display format</th><th>Status value</th></tr>
<tr><td>1</td><td>DB3.DBX 0.0</td><td>"IDB_CALL_RF670R".Connect</td><td>BOOL</td><td>false</td></tr>
<tr><td>2</td><td>DB3.DBX 0.1</td><td>"IDB_CALL_RF670R".Read_Tag_IDs</td><td>BOOL</td><td>false</td></tr>
<tr><td>3</td><td>DB3.DBX 0.2</td><td>"IDB_CALL_RF670R".Write_Tag_Id</td><td>BOOL</td><td>false</td></tr>
<tr><td>4</td><td>DB3.DBX 0.3</td><td>"IDB_CALL_RF670R".Read_Memory</td><td>BOOL</td><td>false</td></tr>
<tr><td>5</td><td>DB3.DBX 0.4</td><td>"IDB_CALL_RF670R".Write_Memory</td><td>BOOL</td><td>false</td></tr>
<tr><td>6</td><td>DB3.DBX 0.5</td><td>"IDB_CALL_RF670R".Disconnect</td><td>BOOL</td><td>true</td></tr>
</table></td></tr>
<tr><td>3</td><td>If the connection was successfully terminated, “IDB_RF670R_RW”.TCP_connect will be reset and "IDB_CALL_RF670R".Done will be set .

<table>
<tr><td>DB670.DBX 3352.0</td><td>"IDB_RF670R_RW".TCP_connect</td><td>BOOL</td><td>false</td></tr>
<tr><td>DB3.DBX 0.6</td><td>"IDB_CALL_RF670R".Done</td><td>BOOL</td><td>true</td></tr>
</table></td></tr>
</table>

(*1): A connection to the RF670R reader must already exist.

6 Further Information

This chapter shows you modifications to the program (e.g., changing the receive buffer length or modifications to the structure of DB RF_PARAMETER (UDT RF_VAR)).

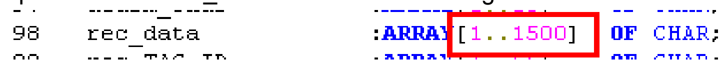
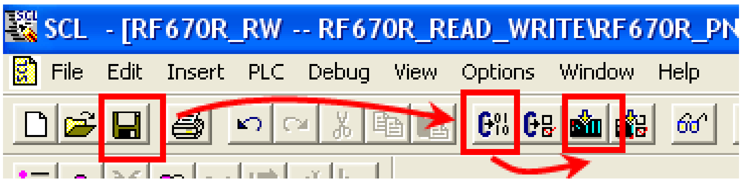
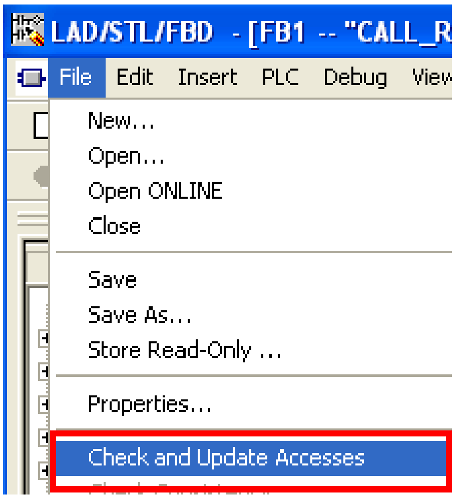
In addition, we explain the function of the conversion blocks FB INT_TO_HEX (FB2) and FB HEX_TO_INT (FB3).

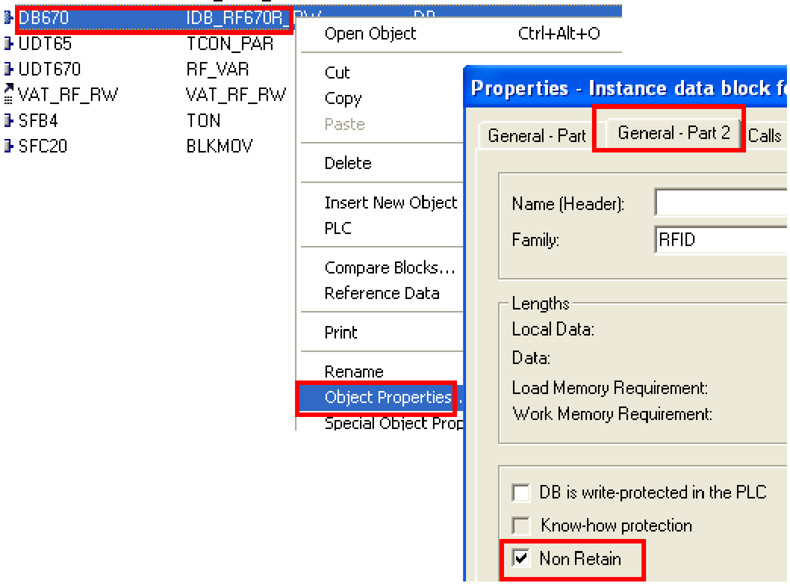
6.1 Modifications to the program

Changing the receive buffer length

If the total length of the response message is longer than the length of the receive buffer, you have to change the length of the receive buffer. The following table lists the steps necessary to make this change.

Table 6-1

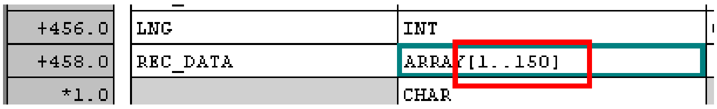
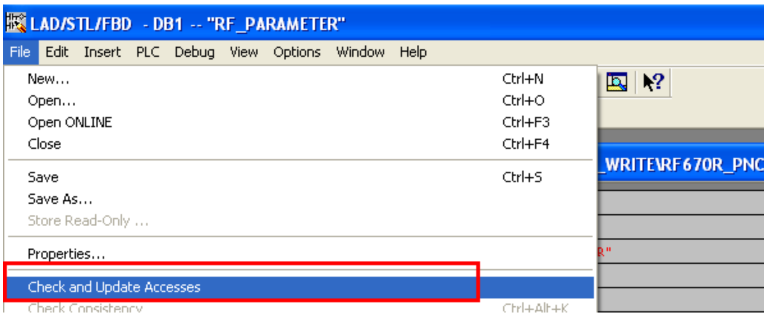
No.	Procedure
1	Open the "RF670R_RW" SCL code and change the receive buffer length.  `98    rec_data                    :ARRAY[1..1500] OF CHAR;`
2	Once again, save, compile and download the SCL code.  SCL - [RF670R_RW -- RF670R_READ_WRITE\RF670R_PN File Edit Insert PLC Debug View Options Window Help
3	Open FB CALL_RF670R (FB1) and update the instances.  LAD/STL/FBD - [FB1 -- "CALL_R File Edit Insert PLC Debug View New... Open... Open ONLINE Close Save Save As... Store Read-Only ... Properties... Check and Update Accesses Once again, save the FB.

No.	Procedure
4	Set the recompiled IDB_RF670R_RW instance data block to “Non Retain”.  DB670 IDB_RF670R_ UDT65 TCON_PAR UDT670 RF_VAR VAT_RF_RW VAT_RF_RW SFB4 TON SFC20 BLKMOV Properties - Instance data block f Name (Header): Family: RFID Lengths Local Data: Data: Load Memory Requirement: Work Memory Requirement: ☐ DB is write-protected in the PLC ☐ Know-how protection ☑ Non Retain Open Object Ctrl+Alt+O Cut Copy Paste Delete Insert New Object PLC Compare Blocks... Reference Data Print Rename Object Properties Special Object Prop General - Part General - Part 2 Calls
5	Once again, download the entire project to your controller.

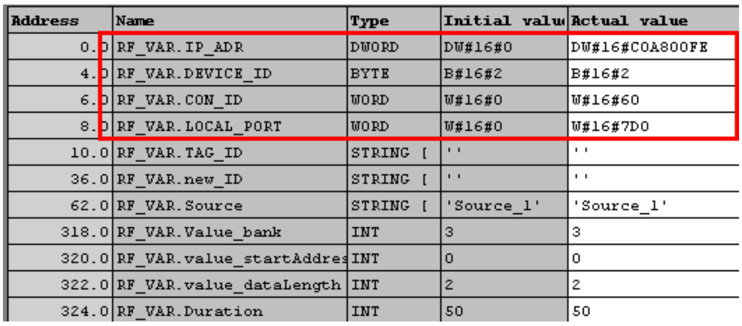
Modifications to the RF_VAR structure

If you want to edit UDT variables, e.g. the length of the “RF_VAR.REC_DATA” field, you have to follow the steps listed in the following table.

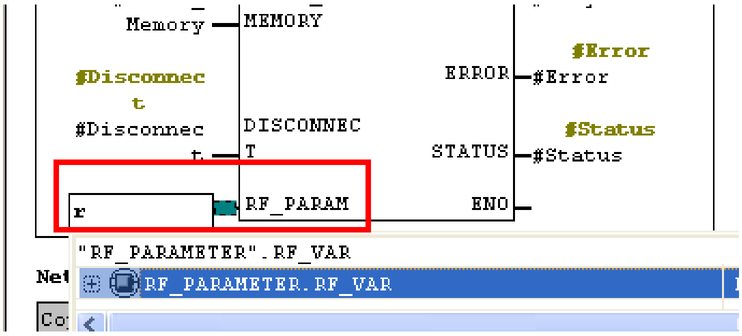
Table 6-2

No.	Procedure
1	Open UDT RF_VAR.
2	Edit the UDT variable, for example the length of the “RF_VAR.REC_DATA” field.  +456.0 LNG INT +458.0 REC_DATA ARRAY[1..150] *1.0 CHAR Save the UDT.
3	Open DB RF_PARAMETER in which the UDT is declared and update the instances.  LAD/STL/FBD - DB1 -- "RF_PARAMETER" File Edit Insert PLC Debug View Options Window Help New... Ctrl+N Open... Ctrl+O Open ONLINE Ctrl+F3 Close Ctrl+F4 Save Ctrl+S Save As... Store Read-Only ... Properties... Check and Update Accesses Check Consistency Ctrl+Alt+K WRITE\RF670R_PNC

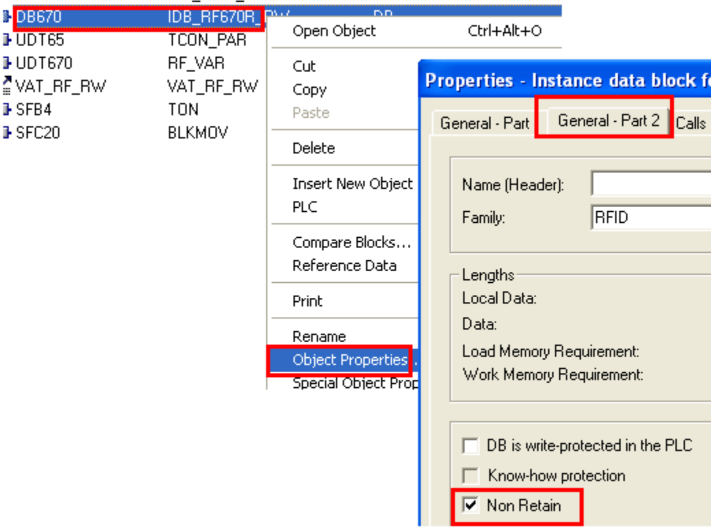
<table>
<tr><th>No.</th><th>Procedure</th></tr>
<tr><td>4</td><td>
In DB RF_PARAMETER, once again enter the parameters of the TCP connection and all parameters necessary for RFID data processing and save DB RF_PARAMETER.



<table>
<tr><th>Address</th><th>Name</th><th>Type</th><th>Initial valu</th><th>Actual value</th></tr>
<tr><td>0.0</td><td>RF_VAR.IP_ADR</td><td>DWORD</td><td>DW#16#0</td><td>DW#16#C0A800FE</td></tr>
<tr><td>4.0</td><td>RF_VAR.DEVICE_ID</td><td>BYTE</td><td>B#16#2</td><td>B#16#2</td></tr>
<tr><td>6.0</td><td>RF_VAR.CON_ID</td><td>WORD</td><td>W#16#0</td><td>W#16#60</td></tr>
<tr><td>8.0</td><td>RF_VAR.LOCAL_PORT</td><td>WORD</td><td>W#16#0</td><td>W#16#7D0</td></tr>
<tr><td>10.0</td><td>RF_VAR.TAG_ID</td><td>STRING [</td><td>''</td><td>''</td></tr>
<tr><td>36.0</td><td>RF_VAR.new_ID</td><td>STRING [</td><td>''</td><td>''</td></tr>
<tr><td>62.0</td><td>RF_VAR.Source</td><td>STRING [</td><td>'Source_1'</td><td>'Source_1'</td></tr>
<tr><td>318.0</td><td>RF_VAR.Value_bank</td><td>INT</td><td>3</td><td>3</td></tr>
<tr><td>320.0</td><td>RF_VAR.value_startAddre</td><td>INT</td><td>0</td><td>0</td></tr>
<tr><td>322.0</td><td>RF_VAR.value_dataLength</td><td>INT</td><td>2</td><td>2</td></tr>
<tr><td>324.0</td><td>RF_VAR.Duration</td><td>INT</td><td>50</td><td>50</td></tr>
</table>
</td></tr>
<tr><td>5</td><td>
Once again, save and compile the “RF670R_RW” SCL code and update the instances in FB CALL_RF670R.



Save the FB.
</td></tr>
<tr><td>6</td><td>
Set the recompiled IDB_RF670R_RW instance data block to “Non Retain”.



Name (Header):
Family: RFID
□ DB is write-protected in the PLC
□ Know-how protection
☑ Non Retain
</td></tr>
<tr><td>7</td><td>Once again, download the entire project to your controller.</td></tr>
</table>

6.2 Auxiliary blocks for converting a SIMATIC variable type for the RFID reader

All data that is written to or read from a transponder is available in hexadecimal character format.

If your raw data is available in a common SIMATIC numerical format, you first have to convert it to the format of the RF670R (hexadecimal character format).

The following blocks show this using the example of an integer.

6.2.1 FB INT_TO_HEX and FB HEX_TO_INT parameters

The following figure and table show the FB INT_TO_HEX call interface.

Figure 6-1

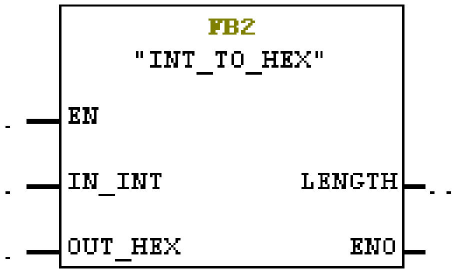


Table 6-3

Symbol	Data type	Explanation
EN	BOOL	Enable input. Relevant only in FBD and LAD representation.
IN_INT	INT	Data that must be converted to hexadecimal characters. The data is fetched from DB In_Out_Data (see Figure 6-3).
OUT_HEX	ANY	Converted data to be written to the transponder.
LENGTH	INT	Byte data length. In this case: 2.
ENO	BOOL	Enable output. Relevant only in FBD and LAD representation.

The following figure and table show the FB HEX_TO_INT call interface.

Figure 6-2

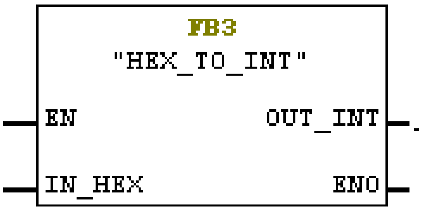
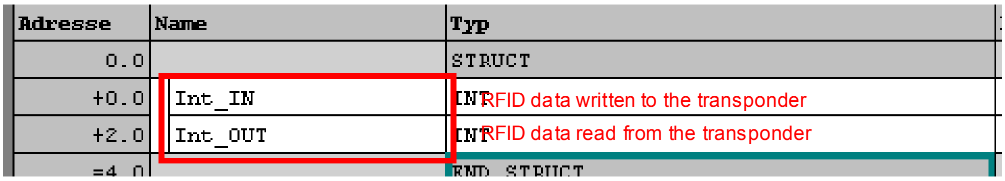


Table 6-4

Symbol	Data type	Explanation
EN	BOOL	Enable input. Relevant only in FBD and LAD representation.
IN_HEX	ARRAY [1..128] OF CHAR	Received data that must be converted to INT.
OUT_INT	INT	Converted data that is stored in DB In_Out_Data (see Figure 6-3).
ENO	BOOL	Enable output. Relevant only in FBD and LAD representation.

Figure 6-3 DB In_Out_Data



Adresse	Name	Typ
0.0		STRUCT
+0.0	Int_IN	INT
+2.0	Int_OUT	INT
=4.0		END_STRUCT

RFID data written to the transponder

RFID data read from the transponder

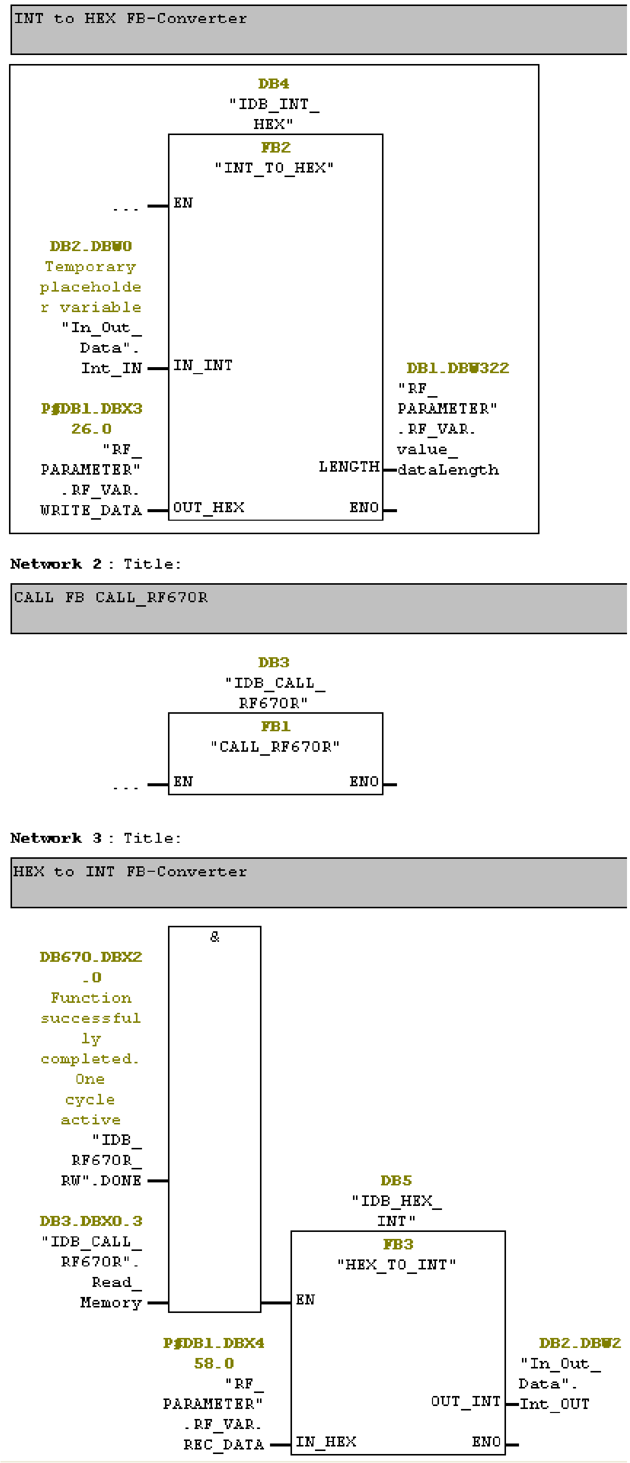
Note

For more program details, the documented FB INT_TO_HEX (FB2) and FB HEX_TO_INT (FB3) SCL codes are available to you in the project (see \1\).

6.2.2 Call of FB INT_TO_HEX and FB HEX_TO_INT in OB1

If you write data of the INT type from a data source to a transponder or if you want to store the data read from a transponder in this data source as INT, you have to call FB INT_TO_HEX and FB HEX_TO_INT.

Figure 6-4



INT to HEX FB-Converter

Network 2 : Title:

CALL FB CALL_RF670R

Network 3 : Title:

HEX to INT FB-Converter

7 References

7.1 References

This list is by no means complete and only presents a selection of related references.

Table 7-1

	Topic	Title
/1/	STEP7 SIMATIC S7-300/400	Automating with STEP7 in STL and SCL Author: Hans Berger Publicis Corporate Publishing ISBN: 978-3-89578-412-5
/2/	STEP7 SIMATIC S7-300/400	Automating with STEP 7 in LAD and FBD Author: Hans Berger Publicis Corporate Publishing ISBN: 978-3-89578-410-1
/3/	STEP7 SIMATIC S7-300	Automating with SIMATIC S7-300 inside TIA Portal Author: Hans Berger Publicis Corporate Publishing ISBN: 978-3-89578-382-1
/4/	STEP7 SIMATIC S7-400	Automatisieren mit SIMATIC S7-400 im TIA Portal Author: Hans Berger Publicis Corporate Publishing ISBN: 978-3-89578-372-2
/5/	STEP7 SIMATIC S7-1200	Automating with SIMATIC S7-1200 Author: Hans Berger Publicis Corporate Publishing ISBN: 978-3-89578-356-2

7.2 Internet links

This list is by no means complete and only provides a selection of useful information.

Table 7-2

	Topic	Title
\1\	Reference to the document	http://support.automation.siemens.com/WW/view/en/23626344
\2\	Siemens Industry Online Support	http://support.automation.siemens.com
\3\	Overview of communication services supported by PN-CPUs	http://support.automation.siemens.com/WW/view/en/18909487
\4\	SIMATIC Sensors RFID Systems SIMATIC RF600	http://support.automation.siemens.com/WW/view/en/22437600
\5\	SIMATIC RF670R	http://support.automation.siemens.com/WW/view/en/44661579

	Topic	Title
	Function Manual	
\6\	local_device_id	http://support.automation.siemens.com/WW/view/en/51339682
\7\	S7-300, CPU 31xC and CPU 31x: Installation Operating Instructions	http://support.automation.siemens.com/WW/view/en/13008499
\8\	System and Standard Functions for S7-300/400 Volume 1/2	http://support.automation.siemens.com/WW/view/en/44240604

8 History

Table 8-1

Version	Date	Modification
V1.0	06/2012	First edition
	08/2012	Note in chapter 2.4 has been changed
V1.1	04/2013	Note in chapter 2.1 has been inserted and chapter 3.3 has been changed