PROFIBUS
and
PROFINET

Guideline

Communication Function Blocks on PROFIBUS DP and PROFINET IO

Version 2.0

November 2005

Order No: 2.182

PROFIBUS INTERNATIONAL

Document Identification: TC2-05-0001a
File name: Comm-Func-Block_2182_V20_Nov05

Prepared by the PROFIBUS Working Group 4 “Communication Function Blocks“ in the Technical Committee 2 “Communication Profiles“.

In this specification the following key words (in **bold** text) will be used:
may: indicates flexibility of choice with no implied preference.
should: indicates flexibility of choice with a strongly preferred implementation.
shall: indicates a mandatory requirement. Designers **shall** implement such mandatory requirements to ensure interoperability and to claimconformance with this specification.

Publisher:
PROFIBUS Nutzerorganisation e.V.
Haid-und-Neu-Str. 7
D-76131 Karlsruhe
Germany
Phone: +49 (0) 721 / 96 58 590
Fax: +49 (0) 721 / 96 58 589
E-mail: pi@profibus.com
Web site: www.profibus.com

CONTENTS

List of tables

Table of figures

Foreword

For PROFIBUS DP and PROFINET IO sets of communication services are defined in IEC 61158. The representation of these services in the application program is dependent of the various controllers and devices provided by different manufactures.

State of the art for the programming model and programming languages in the area of the programmable controllers (PLC) is the international standard IEC 61131-3. This standard defines a set of language elements and mechanisms (e.g. data types, function blocks) which are commonly applied in a well defined set of programming languages (e.g. Ladder Diagram, Structured Text).

This specification defines a common set of Communication Function Blocks applicable in application programs using the IEC 61131-3 languages. The application of this specification can provide benefits for the following three groups:

End users
want to implement applications (including application software and devices) using predefined solutions and having a wide choice and independent mixture of various PLC and Field Devices on PROFIBUS DP and PROFINET IO respectively.

PLC manufacturers
want to offer the PLC series with a wide choice of Field Devices from various manufactures.

Field Device manufacturers
wants to have applied his Field Devices easily with a wide choice of minimise the effort to use these Field Devices with different PLC.

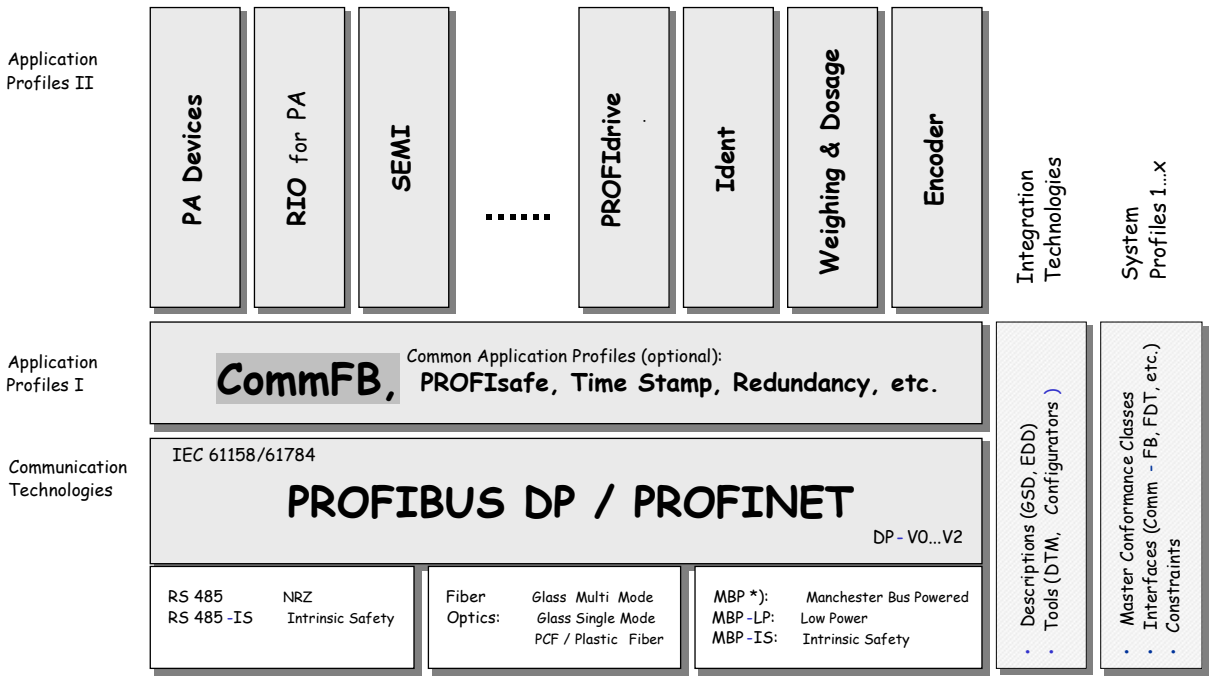
The first published version V 1.20 July 2001 of this specification defined a set of Communication Function Blocks for PROFIBUS DP. This 2nd version V 2.0 specifies a set of function blocks which is applicable for PROFIBUS DP and PROFINET IO and which are compatible to the first version to a large extent.

Following relevant changes are made in the version 2.0.

Clause	Feature	Reason
all	scope extended for PROFINET IO	relevance of PROFINET IO
all	naming conventions	made more general to conform to the extended scope of PROFIBUS DP and PROFINET IO
1.3	definitions for PROFINET IO added	extended scope of PROFINET IO
2.6	EN input and ENO output are optional and may be omitted from the textual Function Block declarations	EN and ENO are used according to IEC 61131-3, chapter 2.5.1.2
2.8.2	FC NSLOT	deleted because of irrelevance
2.8.3	unified address concept for PROFIBUS DP and PROFINET IO	extended scope of PROFINET IO
3 .. 5	UML compatible presentation of the state diagrams	IEC compatibility
3	FB SYCFR	deleted because of irrelevance
3.2	FB GETIO_PART and SETIO_PART added	requirement to access parts of the IO data of a module
3.4.2	Structure of the variable at AINFO parameter for PROFINET IO	FB RALRM extended for PROFINET IO

Clause	Feature	Reason
3.5	representation of state diagrams in UML	conforming to PNO regulations
3.5	mapping to PROFINET IO services added	extended scope of PROFINET IO
4.5	Structure of the variable at D_ADDR input for PROFINET IO	FB CNCT extended for PROFINET IO
5.3	FB RCVREC and PRVREC: F_ID parameter extended to DWORD to hold slot and subslot	different address concept of PROFINET IO
5.3	FB RCVREC and PRVREC: New output parameter SUBSLOT	different address concept of PROFINET IO
6	PLC in multiple communication roles	using all Communication Function Blocks in one application program
7.2	Communication Function Blocks and PROFINET CBA	explanation
A.1	Compliance table extended for PROFINET IO	extended scope of PROFINET IO
B	Application example „DPV1 – Parameter Channel“	deleted

The following figure gives an overview of the PNO profiles and the placement of this guideline in the context of the other PNO specifications.



1 General

1.1 Scope

This specification defines a set of Communication Function Blocks for communication among programmable controllers and Field Devices over PROFIBUS DP and PROFINET IO respectively. These Communication Function Blocks are defined according the international standard for programming languages of PLC IEC 61131-3.

The Communication Function Blocks shall have universal interfaces for as well as PROFIBUS DP and PROFINET IO. Therefore if the Communication Function Blocks are used in technology function blocks. The technology function blocks can be defined with similar interfaces for different IO systems.

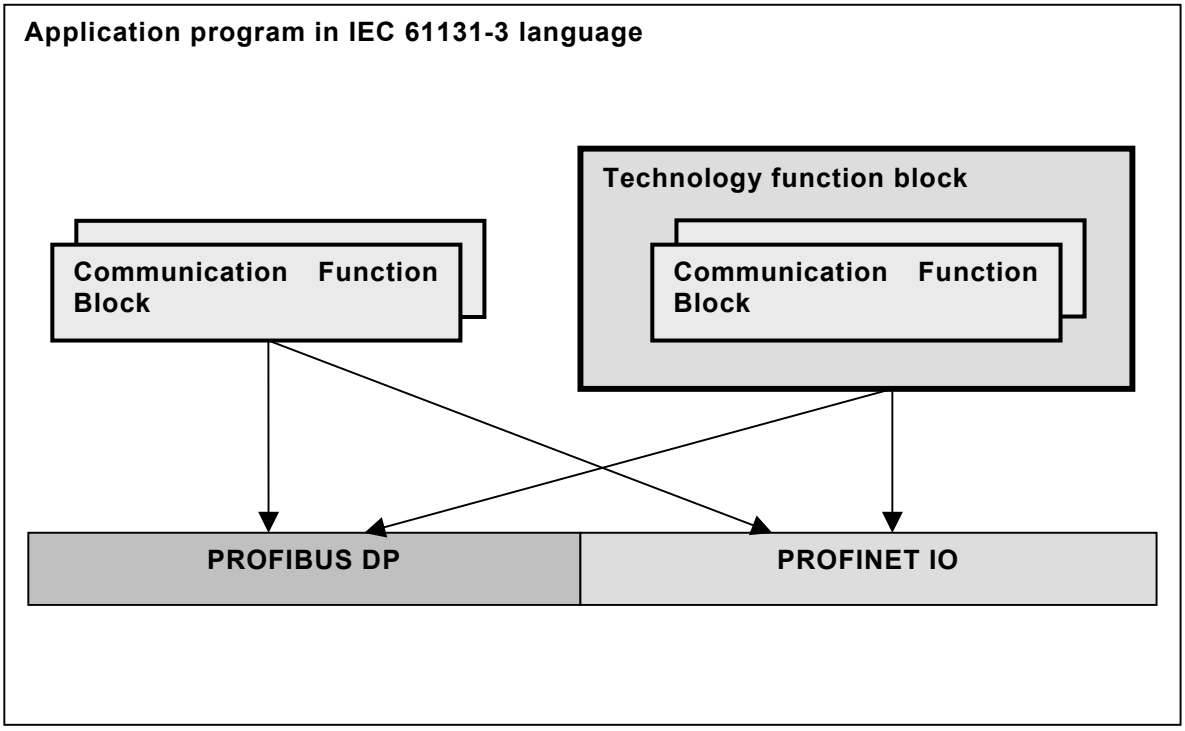


Figure 1 – Application of the Communication Function Blocks

This specification gives also some "Guidelines for the application of the *Communication Function Block*s", i.e. for implementing Field Device specific *Proxy Function Blocks* usable in the PLC. Also the relationship to PROFINET CBA is shown in the guideline.

1.2 References

The following normative documents contain provisions which constitute provisions of this specifications.

IEC 61158-5:2004, Digital data communications for measurement and control - Fieldbus for use in industrial control systems – Part 5: Application layer service definition.

IEC 61158-6:2004, Digital data communications for measurement and control - Fieldbus for use in industrial control systems – Part 6: Application layer protocol specification.

PROFINET IO, Application Layer Service Definition, Application Layer Protocol, Version 2.0, April 2005.

NOTE PROFIBUS DP is already part of IEC 61158, PROFINET IO will be part of IEC 61158 as Type 10. PROFINET IO is already published IEC PAS 62411, 2005-06.

IEC 61131-3: 2nd Edition 2003-01, Programmable Controllers, Part 3: Programming languages.

1.3 Definitions and Abbreviations

For the purpose of this specification the following definitions apply. For definitions adopted from other standards the source reference is given.

1.3.1

application program
self-contained sequence of computer instructions to solve a task

NOTE In this specification is a application program a set of function blocks and functions written in a IEC 61131-3 language.

1.3.2

array
aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting

[IEC 61131-3]

1.3.3

cyclic (exchange of data)
term used to describe events which repeat in a regular and repetitive manner

[IEC 61158-6]

1.3.4

Communication Function Block
basic function block defined in this specification and supplied by the PLC manufacturer for the access to Field Devices

1.3.5

data type
set of values together with a set of permitted operation

[IEC 61131-3]

1.3.6

declaration
mechanism for establishing the definition of a language element.

NOTE A declaration normally involves attaching an identifier to the language element, and allocating attributes such as data types and algorithms to it.

[IEC 61131-3]

1.3.7

DP-master (Class 1)
controlling device which controls several DP-slaves (Field Devices); usually a programmable controller or distributed control system

[IEC 61158-6]

1.3.8

DP-master (Class 2)
controlling device which manages configuration data (parameter sets) and diagnosis data of a DP-master (Class 1); additionally the DP-master (Class 2) can perform all communication capabilities of a DP-master (Class 1)

[IEC 61158-6]

1.3.9

DP-slave
Field Device that is assigned to one DP-master (Class 1) as a provider for cyclic IO data object exchange, in addition acyclic functions and alarms could be provided

[IEC 61158-6]

1.3.10

Field Device
device acting as a PROFIBUS DP-slave or PROFINET IO Device respectively

1.3.11

function block (FB)
programmable controller programming language element consisting of: (i) the definition of a data structure partitioned into input, output, and internal variables; and (ii) a set of operations to be performed upon the elements of the data structure when an instance of the function block type is invoked

[IEC 61131-3]

1.3.12

function
program organisation unit which, when executed, yields exactly one data element and possibly additional output variables (which may be multi-valued, e.g., an array or structure), and whose invocation can be used in textual languages as an operand in an expression

[IEC 61131-3]

1.3.13

Host Controller
controller acting as a PROFIBUS DP-master (Class 1) or PROFINET IO Controller respectively

1.3.14

Human Machine Interface (HMI)
reading and writing interface for the machine or control equipment operator or shop floor personell to the process data

1.3.15

identifier
combination of letters, numbers, and underline characters which begins with a letter or underline and which names a language element

[IEC 61131-3]

1.3.16

instance (of a function block)
individual, named copy of the data structure associated with a function block type or program type, which persists from one invocation of the associated operations to the next

[IEC 61131-3]

1.3.17

index
address of an object within an application process

[IEC 61158-6]

1.3.18

IO Controller
controlling device, which acts as client for several IO Devices (Field Devices)

NOTE This is usually a programmable controller or a distributed control system.

[PROFINET IO]

1.3.19

IO data object
object designated to be transferred cyclically for the purpose of processing and referenced by device/slot/subslot

[PROFINET IO]

1.3.20

IO Device
Field Device which acts as server for IO operation

[PROFINET IO]

1.3.21

IO Supervisor
engineering device which manages commissioning and diagnosis of an IO system

[PROFINET IO]

1.3.22

IO subsystem
subsystem composed of one Host Controller and all its associated Field Devices

[PROFINET IO]

1.3.23

IO system
system consisting of a Host Controller and Field Devices which act as server for IO operation to the Host Controller

[derived from PROFINET IO]

1.3.24

invocation
process of initiating the execution of the operations specified in a program organization unit like function block and function

[IEC 61131-3]

1.3.25

language element
item identified by a symbol on the left-hand side of a production rule in the formal specification

[IEC 61131-3]

1.3.26

library (of function blocks)
organised set of function blocks for the use in application programs

1.3.27

module
addressable unit inside the Field Device

[IEC 61158-6, IEC 61158-4]

1.3.28

parameter (input and output parameter)
variable assuming a constant used as an argument to pass in or out a function block or function

[IEC 61131-3]

1.3.29

process data
data which are already pre-processed and transferred acyclically for the purpose of information or further processing

[IEC 61158-6]

1.3.30

Proxy function block (Proxy FB)
function block used in the IEC 61131 application program representing a Field Device or a functional part of a Field Device

1.3.31

slot
address of a module within a Field Device

[IEC 61158-6, IEC 61158-4]

1.3.32

Structured Text (ST)
textual PLC programming language using (i) the same common elements as all IEC 61131-3 languages like data types, function blocks and (ii) the specific operators like +, - and (iii) language statements like IF, CASE, WHILE, which are adopted from the well know general purpose languages BASIC or PASCAL
[IEC 61131-3]

1.3.33

subslot
address of a structural unit within a slot

1.3.34

Supervisor
device or tool acting as a PROFIBUS DP-master (Class2) or PROFINET IO Supervisor respectively.

1.3.35

task
execution control element providing for periodic or triggered execution of a group of associated program organisation units like function block or funktions
[IEC 61131-3]

1.3.36

type (of a function block)
PLC languages element consisting of: (i) the definition of a data structure partitioned into input, output, and internal variables; and (ii) a set of operations to be performed upon the elements of the data structure when in instance of the function block type is invoked
[IEC 61131-3]

1.4 Compliance

This subclause defines the requirements which shall be met by programmable controller systems and Field Devices which claim compliance with this PNO specification.

The definition of the function blocks in this specification is based on the elements and rules of the "common elements" of IEC 61131-3 (2^{nd} edition). Therefore it is required that a programming system which uses the here defined function blocks is compliant to the IEC standard.

This specifications defines a set of Communication Function Blocks which have defined names, interfaces and functionality. The function block interface comprises the names, the data type and the order of the input and output parameters. The functionality is defined by the state diagram and the associated transition and action table.

A system which claims compliance with this PNO specification shall provide a subset of the here defined of Communication Function Blocks. All provided function blocks shall have the full set of parameters and functionality. The compliant function blocks shall be listed in the "*Compliance Table*" according Annex A.

In a second table also shown in Annex A the permitted "*Implementation dependant features"* shall be listed.

2 Principles for modelling Communication Function Blocks

2.1 Principles of Modelling

The following principles of modelling for the Communication Function Blocks have to be met:

- to fit into the existing PLC systems, e.g. using the existent addressing concept
- to be efficient and without overhead; that means the model shall be performance oriented
- to enable a easy application program porting between different PLC systems
- universal interface for the use with different types of IO communication subsystems especially to support the use of PROFIBUS DP and PROFINET IO
- to use directly the existing PROFIBUS DP or PROFINET IO functions, i.e. if possible one Communication Function Block shall cover one service.
- to apply good programming style is to avoid dependencies of the hardware configuration data such as addressing in the application program.

2.2 IEC 61131-3 Function Blocks for Profibus Communication and as Devices Proxies

There are various possibilities in a program of a PLC acting as a Host Controller (DP Master Class 1 and IO Controller respectively) to access to the data in remote modules and Field Devices (DP-slave and IO Device respectively):

- A typical solution in the PLC is the "cyclic access" via the so-called *process image* to the remote inputs and outputs. In the application program these remote variables are used like local I/O variables. The data exchange over the fieldbus happens cyclically. The variables are transferred independently of the execution of the application program and mapped in the process image.
- In this specification a set of *Communication Function Blocks* is defined like read and write record to achieve a data transfer which is triggered by the application program in the PLC.

The Figure 2 shows an instance of the so called *Proxy Function Block* representing the Field Device in the IEC 61131-3 application program in the PLC. This device specific *Proxy function block* exhibits the input and output parameters of the represented Field Device. Inside the Proxy FB standardised *Communication Function Blocks* provide the reading and writing access to the Field Device data using the standard Profibus protocols.

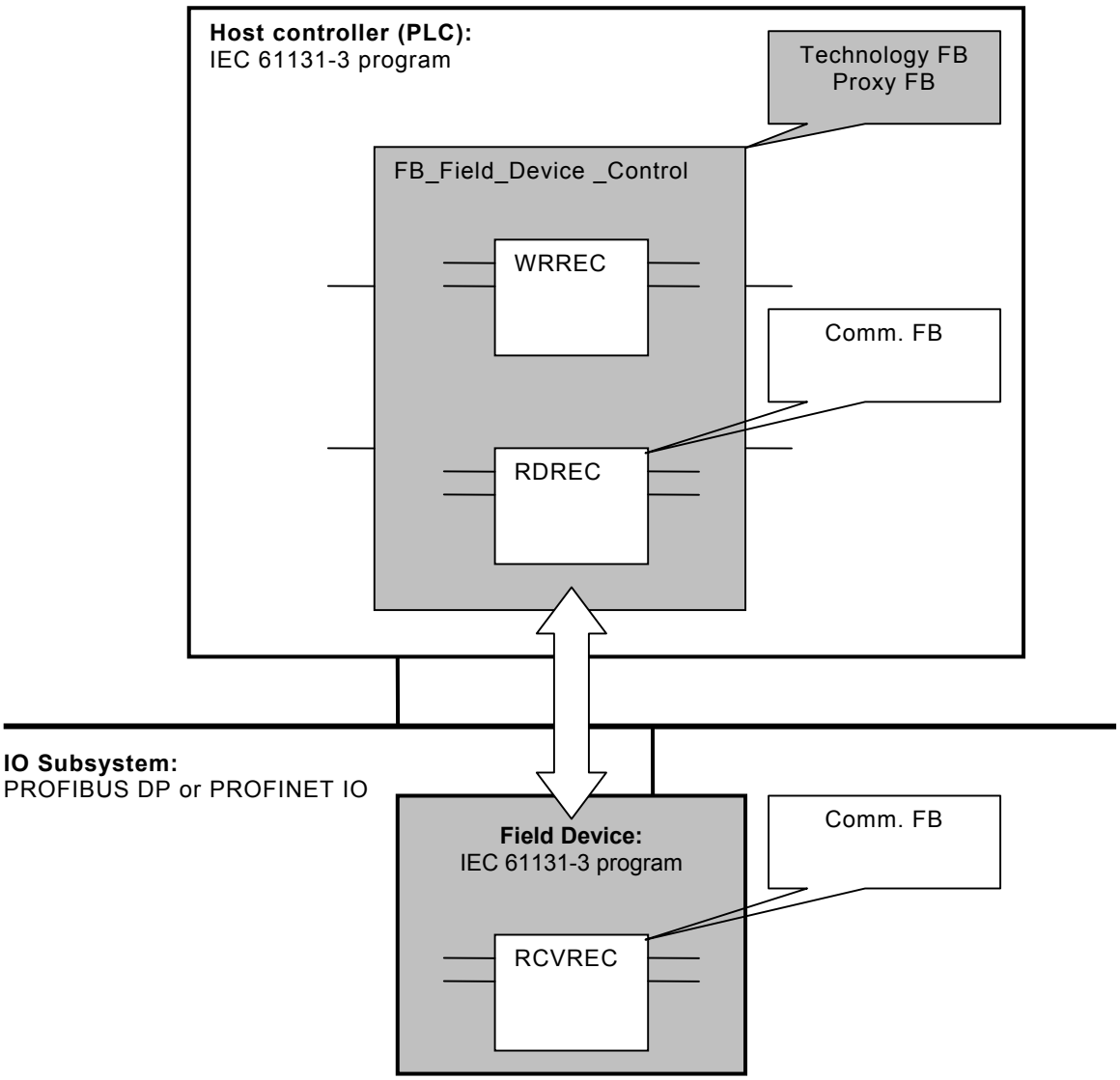


Figure 2 – Proxy FB and Communication Function Blocks

2.3 Library Concept and Program Porting

Figure 3 illustrates the usage of libraries of two PLC systems with standardised Communication Function Blocks.

The manufacturers of the PLC systems A and B provide their programming systems according IEC 61131-3 and offer both their own library with the Communication Function Blocks as defined in this PNO specification. These Communication Function Blocks have the identical interface and functionality but are specifically implemented for the different PLC systems and IO subsystems.

The Communication Function Blocks can be used by the application programmers and also by the Field Device manufacturers to build the specific Proxy FB.
PLC manufacturers may provide libraries of standardised Proxy FB based on fieldbus profiles and the Communication Function Blocks to support application development integrating Field Device functionality and support access to Field Device maintenance and diagnose features.

As shown in figure 2 two different Field Device manufacturers C and D can provide their own libraries with specific Proxy function block C and D for the application support of their Field Devices connected via Profibus to the PLC systems A and B. The device manufacturers use the standardised Communication Function Blocks for the implementation of their Proxy FB executa-

ble in different PLC systems. They can apply the same Proxy FB implementation using the standard Communication Function Blocks of the required PLC library.

The application programmers of the PLC systems A and B can use the specific Proxy FB C and D as well as the basic Communication Function Blocks.

The application programs using same the IEC 61131 programming language and the standardised Communication Function Blocks and Proxy FB can easily ported from PLC system A to B.

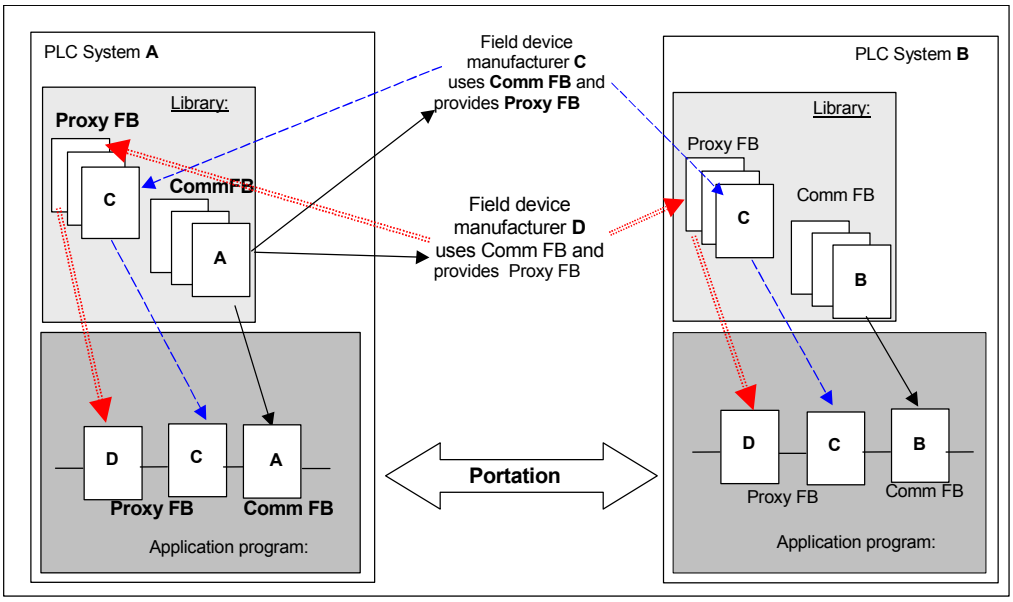


Figure 3 – Usage of function block libraries with Communication Function Blocks and Proxy FB

The guidelines in clause 4 offer additional information to implement and apply of the Communication Function Blocks and Proxy FB.

2.4 Mapping to IEC 61158

The Communication Function Blocks map onto objects and services defined in IEC 61158-6 Type 3 (Profibus DP) and IEC 61158-4 Type 10 (PROFINET IO) respectively.

2.5 Mapping to Functions and Function Blocks

The communication between the application program written in an IEC 61131-3 language and the Field Devices connected via an IO subsystem to the PLC is modelled by Communication Function Blocks. This specification defines the Communication Function Blocks for the cyclic and acyclic data access.

The representation of the interface of the function and function block types is given in graphical and textual form according IEC 61131-3.

The EN input and ENO output are given in the graphical representation of the declarations of the cyclic function blocks. These parameters are optional and may be omitted when calling the function block instance according to IEC 61131-3, chapter 2.5.1.2.

The behaviour of the function blocks is presented as a graphical state diagram with a table for the transitions and the actions.

2.6 Parameters

Parameters of different Communication Function Blocks with the same or a similar meaning shall have the same name and data type according Table 1:

Table 1 – Communication FB Parameters

Parameter	Data Type	Meaning
EN	BOOL	Enable
ENO	BOOL	Enable output
REQ	BOOL	Request function
ID	DWORD	Identification of a Field Device or a slot / subslot of it
INDEX	INT	Identifier of a process data object
OFFSET	INT	Count of the first byte within IO Data object
LEN	INT	Actual data length of a process data record
DONE	BOOL	Flag that the function has finished successfully
VALID	BOOL	Flag that the function has finished successfully and the received output data are valid
BUSY	BOOL	Flag that the function is still performing its task, its not ready to perform a new task
ERROR	BOOL	Flag that the function has finished with an error
STATUS	DWORD	Completion or error code

The Functions and Function Blocks may use the EN input and ENO output according to IEC 61131-3, chapter 2.5.1.2.

The IO data object, the process data records or the alarm and diagnosis information is passed by input-output parameters to the Communication Function Blocks. Typically these data can be described as an array of byte. The length of the byte arrays may vary from instance to instance of one Communication Function Block. It shall also be possible to use a structured data type if the used data is structured.

The INDEX input addresses a process data record, the LEN parameter contains the length of a process data record.

NOTE The data type INT is used for both the INDEX and LEN parameter though the value range of the parameters in the IO subsystems may cover 0..65535. The data type INT is supported by all PLC implementations, the better fitting UINT is not always supported. The value range of 0..32767 is sufficient for nearly all applications. The data type INT was already used by version 1.20 of this PNO guideline, and it is kept for compatibility reason. Nevertheless an INDEX > 32767 may be given on the interface using a negative number or a hexadecimal constant.

The ANY data type is used to allow the use of byte arrays of different lengths, the use of structured data types as buffers for IO data object, the process data records or the alarm and diagnosis information, or different address data structures. The user shall use variables of appropriate size which can contain the information wished. The implementer may cause an error if he can detect that a given variable does not fit to the requested service.

The DONE and the ERROR outputs pulses only from one invocation of the instance of the Communication Function Block.

The FB parameters use those data types of IEC 61131-3 which are supported in a wide range of PLC and is contained in the portability level of PLCopen.

2.7 Error Concept

Communication Function Blocks indicate if the requested function (block) was performed successfully or not. The error indication is typically used for two purposes:

1. To change the reaction to the process i.e. to implement a substitute reaction e.g. to repeat the request at another time or another place or to abort the process task.
2. To issue an alarm message to an HMI system by the application program or by the PLC system automatically.

NOTE In case 1 only very few different reactions dependent on the indicated error are typical. Detailed error information is hardly used.

If the Communication Function Block maps directly to one service primitive of IEC 61158-6 the error indications of the used service primitive is used as error indication of the function block too. The Function_Num byte**,** Error_Decode byte, Error_Code_1 byte, and Error_Code_2 byte of the DP service primitives are combined to the STATUS output.

The STATUS output has the data type DWORD which is interpreted as a packed array of four bytes as described in the following table.

Table 2 – Structure of the Output STATUS

Byte	Name	Definition	Date type
0	Function_Num	contains Function_Code / Error_Code, PDU_Identifier, and Frame_Selector	byte
1	Error_Decode	defines the meaning of Error_Code_1 and Error_Code_2, see Table 3	byte
2	Error_Code_1	see Table 4	byte
3	Error_Code_2	implementer specific	byte

NOTE The error code 2 should be used only for the purpose to detail an error defined with error decode byte and error code 1 byte e.g. for the use of an protocol analyser or an other diagnosis device.

The Function_Num byte is used as defined in IEC 61158-6 and IEC 61158-4 respectively. The value 16#40 shall be used, if no protocol element is used.

The Error_Decode byte defines the meaning of Error_Code_1 and Error_Code_2.

Table 3 – Error_Decode values

Error_Decode	Source	Meaning
16#00 .. 16#7F	PLC	No error or warning
16#80	DP V1, PNIO	Error reported according to IEC 61158
16#81	PNIO	Error reported according to IEC 61158
16#82 .. 16#8F	PLC	18#8x reports an error according the x-th parameter of the call of the Communication Function Blocks
16#FE .. 16#FF	DP Profile	profile-specific error

The Error_Code_1 defines the reason of the reported error, see Table 5.

Table 4 – Error_Code_1 values

Error_Decode	Error_Code_1	Source	Meaning
16#00	16#00	---	No error and no warning
16#00 .. 16#7F	16#00 .. 16#FF	PLC	Warning
16#80	16#00 .. 16#9F	PLC	Implementer specific
16#80	16#A0	PLC	Read error
16#80	16#A1	PLC	Write error
16#80	16#A2	PLC	Module failure
16#80	16#A3 .. 16#A6	PLC	Implementer specific
16#80	16#A7	PLC	Busy

Error_Decode	Error_Code_1	Source	Meaning
16#80	16#A8	PLC	Version conflict
16#80	16#A9	PLC	Feature not supported
16#80	16#AA .. 16#AF	PLC	DP-master specific
16#80	16#B0	Access	Invalid index
16#80	16#B1	Access	Write length error
16#80	16#B2	Access	Invalid slot
16#80	16#B3	Access	Type conflict
16#80	16#B4	Access	Invalid area
16#80	16#B5	Access	State conflict
16#80	16#B6	Access	Access denied
16#80	16#B7	Access	Invalid range
16#80	16#B8	Access	Invalid parameter
16#80	16#B9	Access	Invalid type
16#80	16#BA .. 16#BF	Access	User specific
16#80	16#C0	Resource	Read constrain conflict
16#80	16#C1	Resource	Write constrain conflict
16#80	16#C2	Resource	Resource busy
16#80	16#C3	Resource	Resource unavailable
16#80	16#C4 .. 16#C7	Resource	Implementer specific
16#80	16#C8 .. 16#CF	Resource	User specific
16#80	16#D0 .. 16#FF	User specific	User specific
16#81	16#00 .. 16#FF	PNIO	PNIO specific error, see PROFINET IO, Table 221
16#82	16#00 .. 16#FF	PLC	Error concerning the value the 2nd parameter
:	:	:	:
16#8F	16#00 .. 16#FF	PLC	Error concerning the value the 15th parameter
16#90	IOxS	PLC	Transfer of IOxs (only PROFINET IO)
16#FE .. 16#FF	16#00 .. 16#FF	Profile	Profile-specific errors

General errors coming from the IO subsystems shall use the value 16#80 or 16#81 as the value of the Error_Decode byte. Errors which can explicitly mapped to one parameter of the function block code the parameter number in the least significant nibble of the Error_Decode byte, e.g. 16#82xx means that an error was detected for parameter number 2. The parameters are counted beginning with the input parameters starting with 1 and continuing with the output and input/output parameters as defined in the function block declaration.

> NOTE The first parameter of all Communication Function Blocks is of type BOOL and cannot issue an error.

Quality information provided by the used IO system shall be evaluated and returned as an error code in the STATUS output if the STATUS output is not set to a value not equal to zero, e.g. a negative IOPS of PROFINET IO shall result in Error_Decode =16#90 and the value of the IOxS state in Error_Code_1.

The outputs at function blocks for the cyclic I/O services e.g. GETIO is set or reset at every invocation of the function block instance. In this case an error will appear for a longer time in the STATUS output if it is not only temporary. Temporary errors may be treated as irrelevant by the application program.

The outputs at function blocks for a acyclic services e.g. RDREC shows the state of the last requested service. They will stay at the same values until the service is requested with the same function block instance again.

2.8 Address Concept

2.8.1 General

The address concepts of the different IO Systems may be different. This will result in different functions or function parameters to identify a Field Device, a slot or subslot inside a Field Device.

All Communication Function Blocks hide the address concept of the different IO Systems, i.e. the address of one identified functionality can be used with all Communication Function Blocks. The application program shall be able to use the Communication Function Blocks without knowledge of the explicit hardware configuration e.g. the MAC address of a Field Device or the position of a slot or subslot in a modular Field Device. It shall be able to use symbolic addressing.

2.8.2 DP Address Concept

Addresses shall allow to identify technological functions which communicate with the PLC and its application program via Profibus DP. Typically these technological functions are represented by one slot of a DP-slave, a continuous series of slots or a whole DP-slave. Additionally the configured allocation of technological functions to slots of a DP-slave e.g. in the PA profile shall be supported.

The address concept considers existing DP addressing and existing address concepts of PLC. A PLC may communicate to DP-slaves which are connected to different DP systems, i.e. the PLC is DP-master (Class 1) to different DP systems. A DP-slave is addressed using a station number unique within the DP system. To address a slot of a DP-slave an additional slot number is used. All numbers have ranges that do not exceed 0 .. 255, i.e. one DWORD can hold this information.

If a PLC is acting as a DP-master (Class 2) it can address different DP systems using a segment number. As a DP-master (Class 1) the segment number is not relevant and set to zero.

The input parameter ID of the Communication Function Blocks addresses one slot of a DP-slave or a DP-slave. The ID contains a handle of data type DWORD, the value of it is implementer-specific.

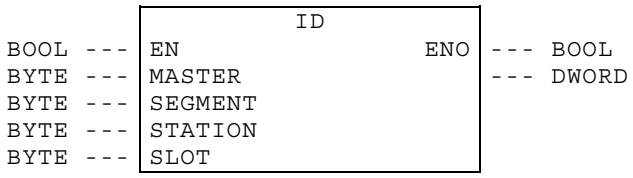
The handle may be generated by local means of the PLC or its configuration system or may be generated by using one of the following functions:

- ID: Conversion of a physical address of a DP-slave to the handle
- ADDR: Conversion of a handle to the physical address of a DP-slave
- SLOT: Addressing a slot of a DP-slave

Note IEC 61131-3 distinguishes between function block and function. The function does not have the instance construct like the function block but it can be used as a simple means to provide a value.

2.8.2.1 Function ID

The function ID converts the physical identification of a slot to a handle which can be used with the Communication Function Blocks. The slot has a unique slot number within a DP-slave, the DP-slave has a unique station number in a DP system, and a DP system is identified by an identification of its master interface. The identification of its master interface may be PLC-specific or unique in the automated system. A DP-master (Class 2) can use a DP segment number additionally.



Note: According IEC 61131-3 2nd Edition the main function output does not have a formal name in the graphical representation.

```
FUNCTION ID : DWORD    (* Generate DP slot handle *)

VAR_INPUT
  EN      : BOOL;
  MASTER  : BYTE;      (* ID of the DP system *)
  SEGMENT : BYTE;      (* Number of the DP segment *)
  STATION : BYTE;      (* Number of the DP-slave (station address) *)
  SLOT    : BYTE;      (* Number of the slot *)
END_VAR

VAR_OUT
  ENO     : BOOL;
END_VAR;
```

Figure 4 – Function ID

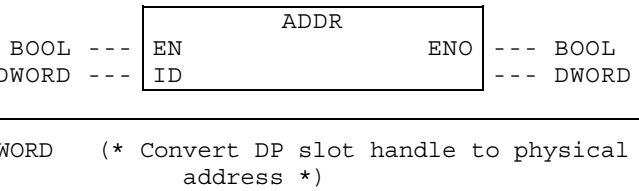
NOTE The EN and ENO parameters are optional and may be omitted when calling the function in textual languages.

The slot number 0 is used to address the DP-slave interface.

If no slot exists at the given physical address 16#FFFF_FFFF is returned as an error indication. The output ENO shall be false.

2.8.2.2 Function ADDR

The function ADDR converts a handle which addresses a slot or a DP-slave into its physical address.




```
FUNCTION ADDR : DWORD   (* Convert DP slot handle to physical
                           address *)
VAR_INPUT
  EN      : BOOL;
  ID : DWORD;           (* Slot or device handle *)
END_VAR

VAR_OUT
  ENO     : BOOL;
END_VAR;
```

Figure 5 – Function ADDR

NOTE The EN and ENO parameters are optional and may be omitted when calling the function in textual languages.

The result of the function ADDR is a DWORD which is interpreted as a packed array of four bytes as described in the following table.

NOTE: An output of data type DWORD is used because it is not allowed to use a structured variable as a function result with IEC 61131-3.

Table 5 – Structure of the Output of ADDR

Byte	Name	Definition	Date type
0	master	DP master interface identification	byte
1	segment	segment number	byte
2	station	station number	byte
3	slot	slot number	byte

If the handle of a DP-slave interface is given, the returned slot number shall be 0.

2.8.2.3 Function SLOT

The function SLOT provides the handle of a slot identified by its number to a given DP-slave.

```
                                  SLOT
                    BOOL ---  EN                  ENO  --- BOOL
                   DWORD ---  ID                       --- DWORD
                    BYTE ---  SLOT
```

```
 FUNCTION SLOT : DWORD    (* Provides the slot to a given DP-slave
                             handle *)
 VAR_INPUT
   EN       : BOOL;
   ID        : DWORD;      (* Handle to a DP-slave or a slot *)
   SLOT      : BYTE;       (* Slot number *)
 END_VAR

VAR_OUT
   ENO      : BOOL;
 END_VAR;
```

Figure 6 – SLOT

NOTE The EN and ENO parameters are optional and may be omitted when calling the function in textual languages.

If the handle of a DP-slave is given, the returned handle addresses the slot identified by the number in the SLOT input of this DP-slave. If the handle at input ID addresses a slot of a DP-slave the same handle is returned as if the DP-slave is addressed at the input ID. If a slot with this number does not exist 16#FFFF_FFFF is returned as an error indication. The output ENO shall be false.

NOTE If an implementer uses the physical address in its handle, these functions may easily be implemented using logic and arithmetic expressions.

2.8.3 Address Conversion

Addresses shall allow to identify technological functions which communicate with the PLC and its application program. Typically these technological functions are represented by one slot or subslot of a Field Device.

The address conversion considers existing PROFIBUS DP and PROFINET IO addressing and existing address concepts of PLC. A PLC may communicate to Field Devices which are connected to different IO subsystems.

The input parameter ID of the Communication Function Blocks addresses one slot of a PROFIBUS DP-slave or a subslot of a PROFINET IO Device. The ID contains a handle of data type DWORD, the value of it is implementer-specific.

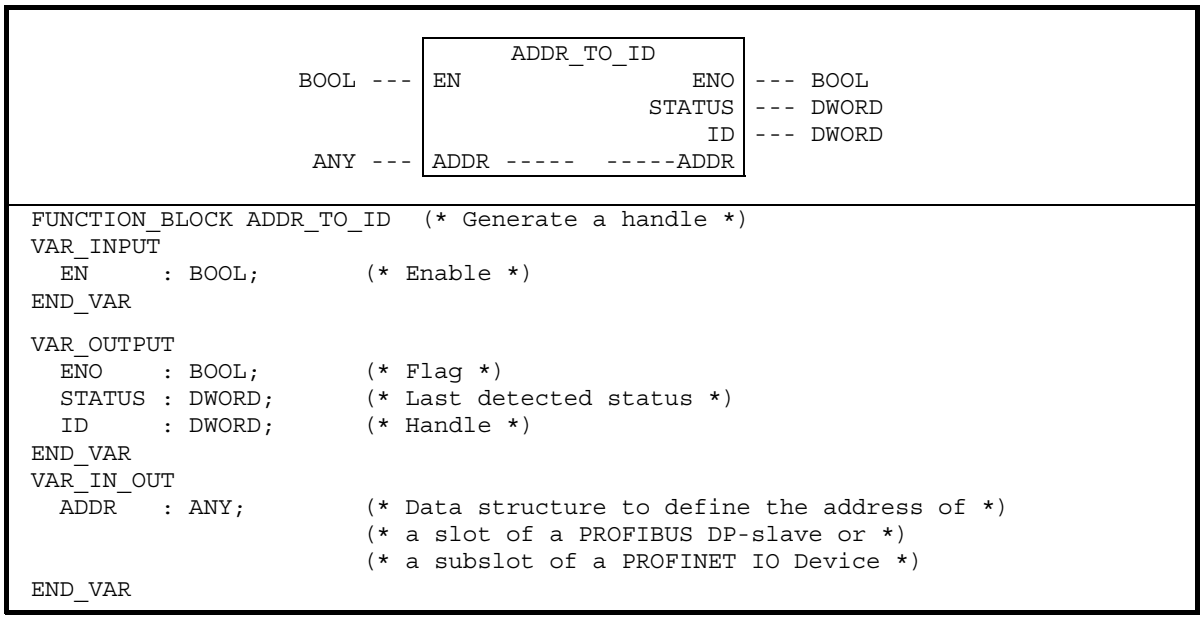
The handle may be generated by local means of the PLC or its configuration system or may be generated by using one of the following functions:

- ADDR_TO_ID: Conversion of a address of a PROFINET IO Device to the handle

- ID_TO_ADDR: Conversion of a handle to the address of a PROFINET IO Device

2.8.3.1 Function Block ADDR_TO_ID

The function block ADDR_TO_ID converts the physical identification of a slot of a PROFIBUS DP-slave or subslot of a PROFINET IO Device to a handle which can be used with the Communication Function Blocks.




```
FUNCTION_BLOCK ADDR_TO_ID  (* Generate a handle *)
VAR_INPUT
  EN     : BOOL;        (* Enable *)
END_VAR

VAR_OUTPUT
  ENO    : BOOL;        (* Flag *)
  STATUS : DWORD;       (* Last detected status *)
  ID     : DWORD;       (* Handle *)
END_VAR
VAR_IN_OUT
  ADDR   : ANY;         (* Data structure to define the address of *)
                        (* a slot of a PROFIBUS DP-slave or *)
                        (* a subslot of a PROFINET IO Device *)
END_VAR
```

Figure 7 – Function Block ADDR_TO_ID

NOTE The EN and ENO parameters are optional and may be omitted when calling the function in textual languages.

The physical address is given in a data structure which contains the address components as shown in the following tables:

Table 6 – Structure of the ADDR data structure of PROFIBUS DP

Name	Date type	Definition
SYSTEM	BYTE	Type of the IO Subsystem (1= PROFIBUS DP)
VERSION	BYTE	Version of the data structure (1= Version 1)
D	STRUCT	Identification of the Field Device
MASTER	BYTE	DP master interface identification
SEGMENT	BYTE	segment number
STATION	BYTE	station number
	END_STRUCT	
SLOT	BYTE	slot number

Table 7 – Structure of the ADDR data structure of PROFINET IO

Name	Date Type	Definition
SYSTEM	BYTE	Type of the IO Subsystem (2= PROFINET IO)
VERSION	BYTE	Version of the data structure (1= Version 1)
D	STRUCT	Identification of the Field Device
STATIONNAME	STRING	Station name
INSTANCE	WORD	Instance ID
DEVICE	WORD	Device ID
VENDOR	WORD	Vendor ID
	END_STRUCT	
API	DWORD	application process identifier
SLOT	WORD	Slot Number (identification of a slot)
SUBSLOT	WORD	Subslot Number (identification of a subslot)

NOTE The string containing the station name shall be of appropiate size.

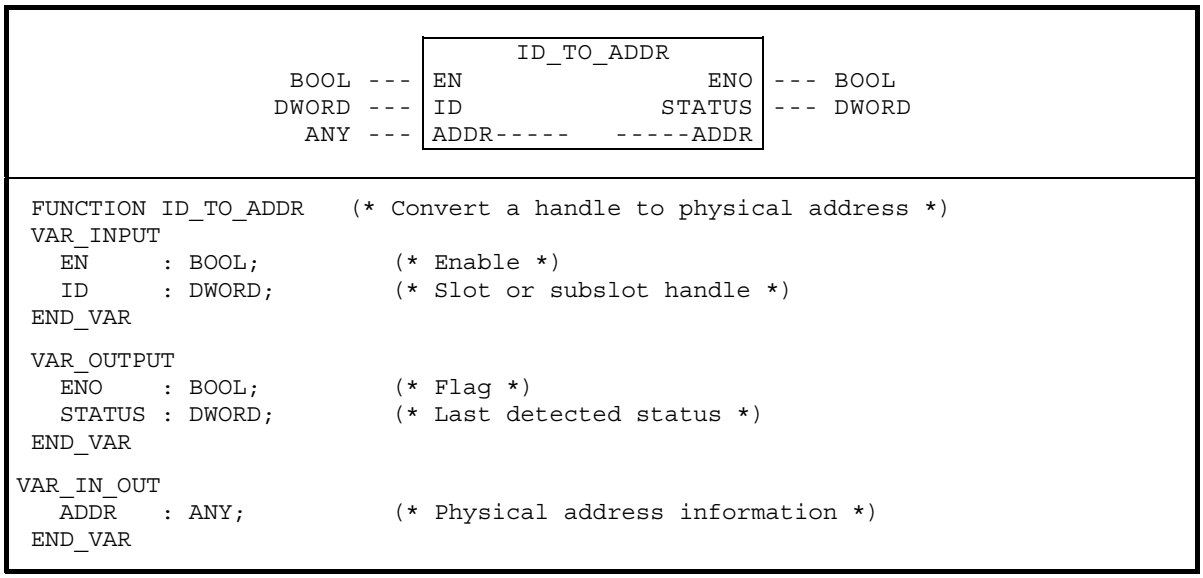
NOTE The structures of Table 6 and Table 7 are equivalent to the address information in Table 23 and Table 24

For PROFIBUS DP the slot number 0 is used to address the DP-slave interface. For PROFINET IO the subslot number 0 is used to address a slot of a PROFINET IO Device.

If the IO Device, the slot or subslot does not exist at the given physical address an error is given at the STATUS output. The output ENO shall be false.

2.8.3.2 Function Block ID_TO_ADDR

The function block ID_TO_ADDR converts a handle which addresses a slot of a PROFIBUS DP-slave or a subslot of a PROFINET IO Device into its physical address.




```
FUNCTION ID_TO_ADDR   (* Convert a handle to physical address *)
VAR_INPUT
  EN     : BOOL;          (* Enable *)
  ID     : DWORD;         (* Slot or subslot handle *)
END_VAR

VAR_OUTPUT
  ENO    : BOOL;          (* Flag *)
  STATUS : DWORD;         (* Last detected status *)
END_VAR

VAR_IN_OUT
  ADDR   : ANY;           (* Physical address information *)
END_VAR
```

Figure 8 – Function Block ID_TO_ADDR

NOTE The EN and ENO parameters are optional and may be omitted when calling the function in textual languages.

The result of the function block ID_TO_ADDR is a data structure which contains the physical address as shown in Table 6 – Structure of the ADDR data structure of PROFIBUS DP and Table 7 – Structure of the ADDR data structure of PROFINET IO.

If the given handle at the ID input is not valid, an error is given at the STATUS output. The output ENO shall be false.

3 Communication Function Blocks for Host Controller

3.1 General Information

This clause defines the Communication Function Blocks for a PLC acting as a Host Controller (DP Master (Class 1) or PROFINET IO Controller) programmed in IEC 61131-3.

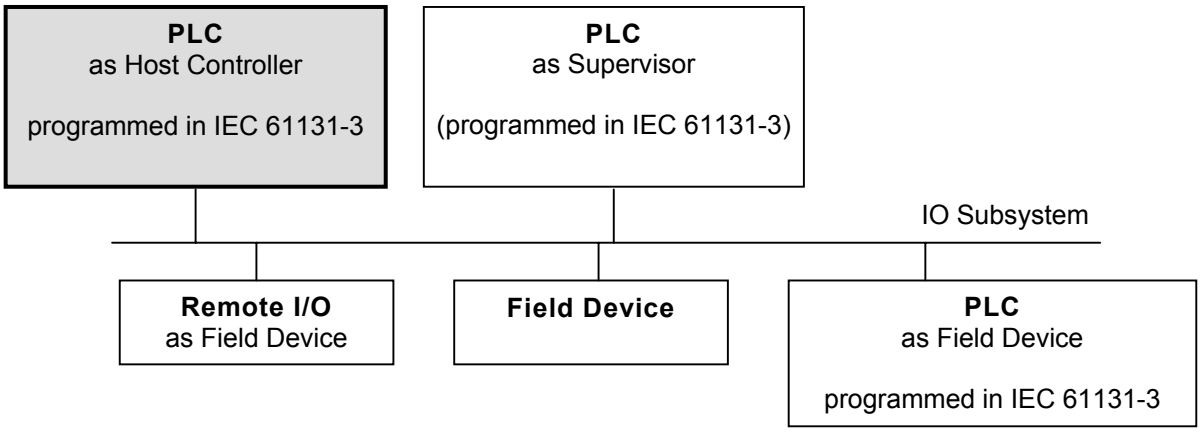


Figure 9 – System with a PLC as Host Controller

The following function blocks define the application program interface to the basic services for a PLC acting as a Host Controller:

- GETIO: Get input data of a Field Device
- SETIO: Set output data of a Field Device
- GETIO_PART: Get a part of input data of a Field Device
- SETIO_PART: Set a part of output data of a Field Device
- RDREC: Read a process data record from a Field Device
- WRREC: Write a process data record to a Field Device
- RALRM: Receive an alarm from a Field Device
- RDIAG: Read diagnosis information from a Field Device

The following function blocks define a application program interface with higher communication functions using the basic services for a PLC acting as a Host Controller:

- ICRTL: Request a interlocked control function from a Field Device

3.2 Cyclic Exchange of IO data object

3.2.1 General

The figure below shows the two FBs for cyclic exchange of IO data object.

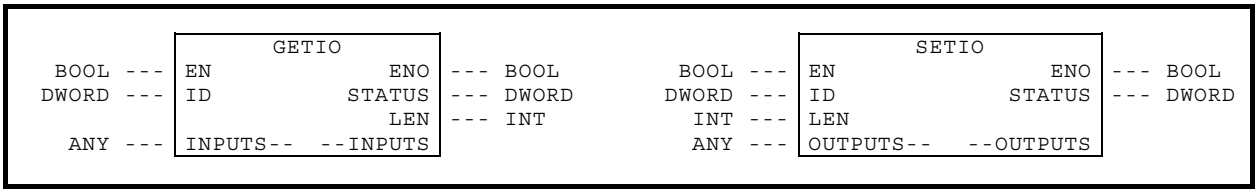


Figure 10 – Communication Function Blocks for cyclic exchange of data

The function blocks for cyclic exchange of IO data object are used when a PLC is acting as a Host Controller.

A Host Controller transfers output data cyclically to its Field Device, in return it gets the input data from the Field Device. The IO data object may be accessed by the application program via the %I or %Q areas or be read or written using the function blocks GETIO and SETIO as defined below. Enabling a Communication Function Block for cyclic exchange of IO data object means, that the IO data object is transferred to or from the Host Controller interface.

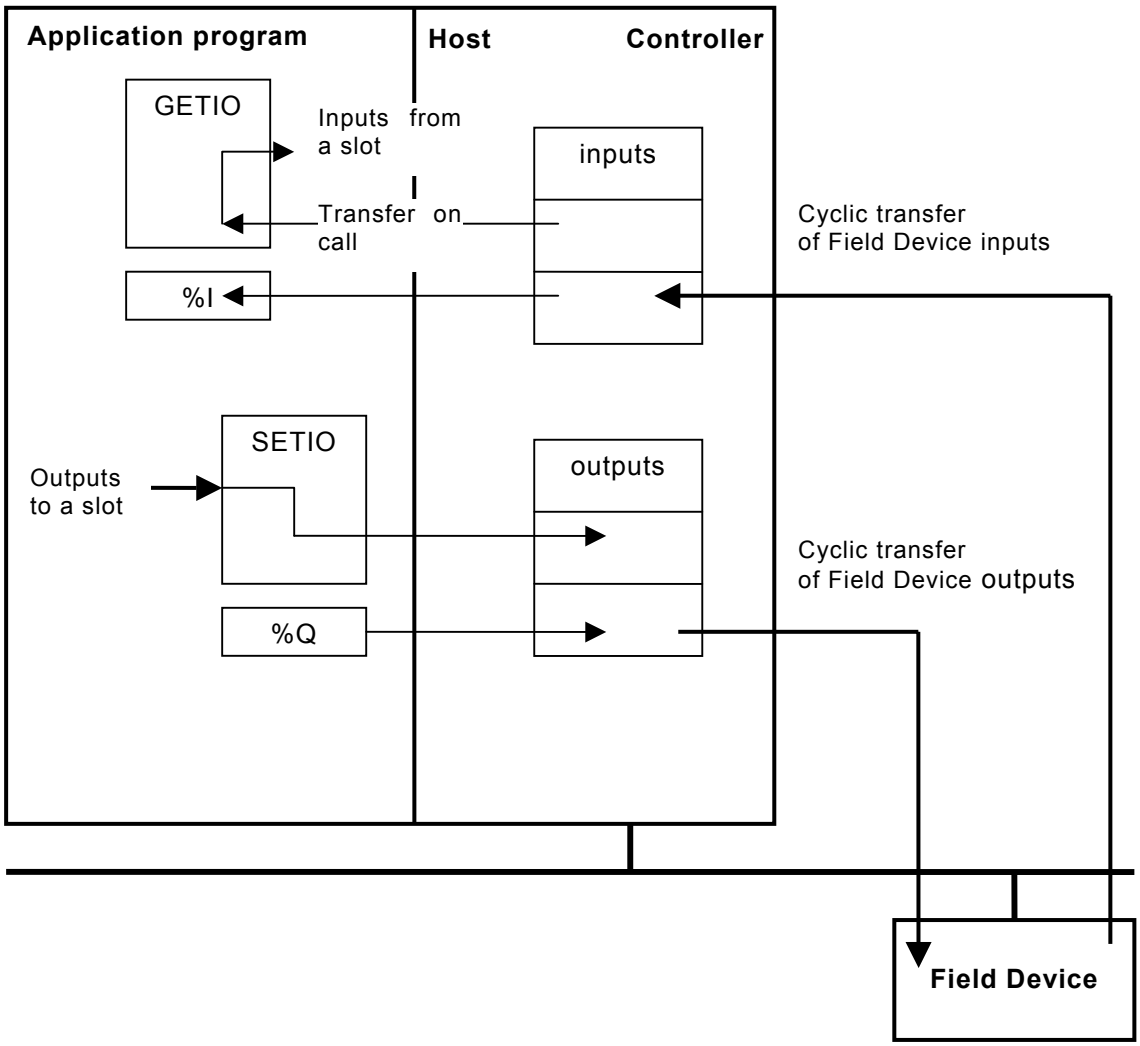


Figure 11 – Communication path for cyclic IO data object

NOTE 1 The same output data should not be written by different function block instances or be written via the %Q interface, because which values are transferred to the slave may be unpredictable.

NOTE 2 Process image vs. direct access, manufacturer dependent

The GETIO function block gets the input data of the addressed slot or subslot of a Field Device from the Host Controller interface out of the cyclically read input data of the Field Device.

The GETIO_PART function block gets only a part of the input data of the addressed slot or sub-slot of a Field Device from the Host Controller interface out of the cyclically read input data of the Field Device.

The SETIO function block transfers the output data of a slot or subslot to the Host Controller interface. The Host Controller collects the data of the Field Device and cyclically transfers these outputs to a slot or subslot of the Field Device.

The SETIO_PART function block transfers only a part of the output data of a slot or subslot to the Host Controller interface. The Host Controller collects the data of the Field Device and cyclically transfers these outputs to a slot or subslot of the Field Device.

3.2.2 Get IO data object (GETIO)

The communication function Get IO data object for a Host Controller uses the GETIO function block defined in this clause. One instance of a GETIO function block provides one instance of the PLC function Get IO data object. The function is invoked by a 1 of the EN input.

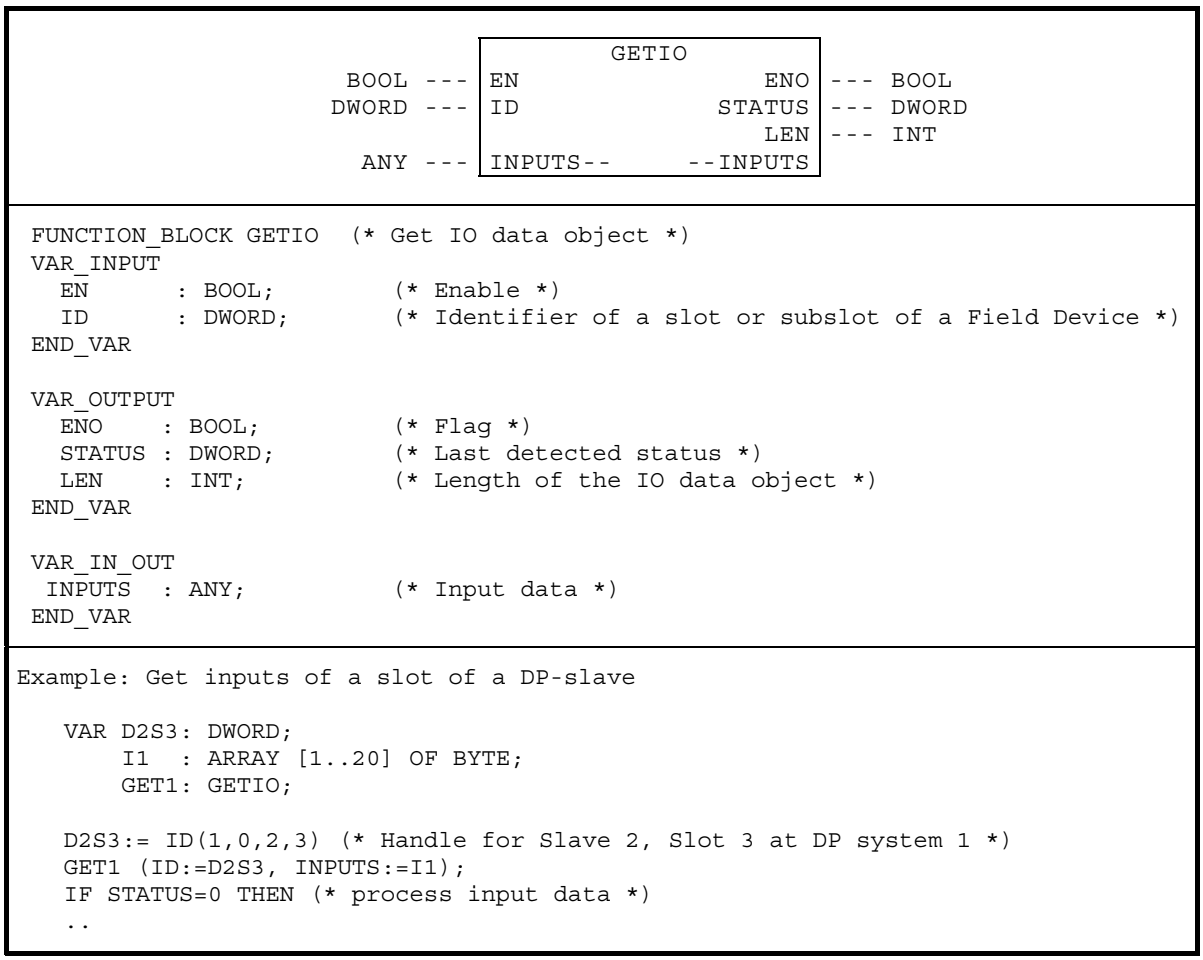
The ID parameter identifies the Field Device or the slot or subslot of a Field Device the IO data object is read from.

NOTE An array declaration with zero elements is not supported in IEC 61131-3, therefore the minimum length shall be 1 byte even if the record length is zero. The actual length is given with the LEN parameter.

If the input data are valid, the ENO output is set to 1 and the Input data are stored in the variable given at the INPUTS parameter. The variable passed to the INPUTS parameter shall be of appropriate size to receive the input data. The LEN output contains the length of the read input data in byte. The output parameters of this FB are set synchronously.

If a variable at the %I area is referenced at the INPUTS parameter the implementer shall specify the rules using this variable with this parameter.

If an error occurred, the ENO output is set to 0 and the STATUS output contains the error code. The STATUS values are defined in Table 2.




```
FUNCTION_BLOCK GETIO  (* Get IO data object *)
VAR_INPUT
  EN      : BOOL;         (* Enable *)
  ID      : DWORD;        (* Identifier of a slot or subslot of a Field Device *)
END_VAR

VAR_OUTPUT
  ENO     : BOOL;         (* Flag *)
  STATUS  : DWORD;        (* Last detected status *)
  LEN     : INT;          (* Length of the IO data object *)
END_VAR

VAR_IN_OUT
 INPUTS   : ANY;          (* Input data *)
END_VAR
```

```
Example: Get inputs of a slot of a DP-slave

   VAR D2S3: DWORD;
       I1  : ARRAY [1..20] OF BYTE;
       GET1: GETIO;

   D2S3:= ID(1,0,2,3) (* Handle for Slave 2, Slot 3 at DP system 1 *)
   GET1 (ID:=D2S3, INPUTS:=I1);
   IF STATUS=0 THEN (* process input data *)
   ..
```

Figure 12 – GETIO function block

NOTE The EN and ENO parameters are optional and may be omitted when calling an instance of the function block in textual languages.

The following state diagram describes the algorithm of the GETIO function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the GETIO function block outputs.

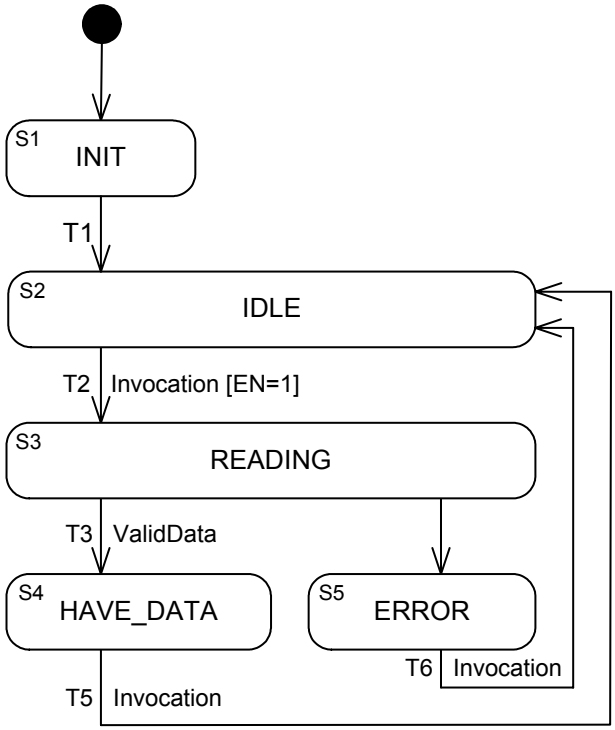


Figure 13 – State diagram of GETIO function block

The following table defines the transitions and actions given in the state diagram above.

Table 8 - Transitions and actions for GETIO state diagram

STATE NAME	STATE DESCRIPTION
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, no actions
S3 READING	read device data Evaluate FB input ID. Get IO data object from Host Controller with Rem Add= out of ID input, Inp Data= INPUTS parameter
S4 HAVE_ DATA	indicate valid data Deposit IO data object in INPUTS parameter and set LEN output set FB outputs
S5 ERROR	indicate error set FB outputs

TRAN-SITION	SOURCE STATE	TARGET STATE	EVENT	CONDITION	ACTION
T1	S1	S2		Initialisation done	ENO := 0 STATUS := 0 INPUTS, LEN := System null
T2	S2	S3	Invocation (Next invocation)	EN=1	ENO := --- STATUS := --- INPUTS, LEN := ---
T3	S3	S4	ValidData (Valid IO data object)		ENO := 1 STATUS := 0 INPUTS, LEN := New data

T4	S3	S5	Error (No valid IO data object)		ENO := 0 STATUS := error code INPUTS, LEN := ---
T5	S5	S2	Invocation (Next invocation)		ENO := --- STATUS := --- INPUTS, LEN := ---
T6	S5	S2	Invocation (Next invocation)		ENO := --- STATUS := --- INPUTS, LEN := ---
--- indicates “unchanged” FB outputs					

3.2.3 Set IO data object (SETIO)

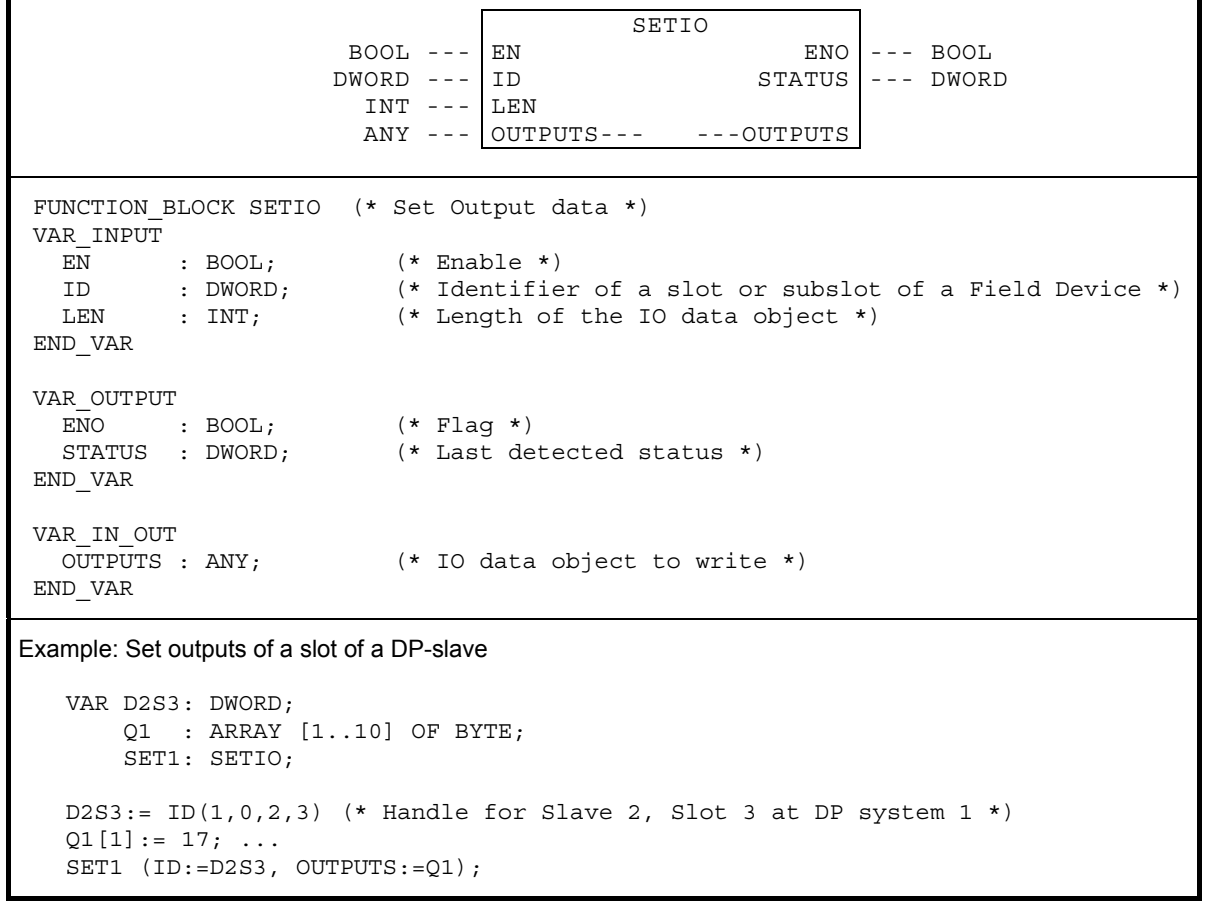
The communication function Set IO data object for a Host Controller uses the SETIO function block defined in this clause. One instance of a SETIO function block provides one instance of the PLC function Set IO data object. The function is invoked by a 1 of the EN input.

The ID parameter identifies the slot or subslot of the Field Device the IO data object is set for. The IO input contains the IO data object that shall be written to the slot or subslot of the Field Device. The variable passed to the OUTPUTS parameter shall be of appropriate size to provide the output data. The LEN input contains the length of the Output data in byte.

NOTE An array declaration with zero elements is not supported in IEC 61131-3, therefore the minimum length shall be 1 byte even if the record length is zero. The actual length is given with the LEN parameter.

If the Output data are stored successfully and the Field Device is still cyclically communicating, the ENO output is set to 1. The output parameters of this FB are set synchronously.

If an error occurred, the ENO output is set to 0 and the STATUS output contains the error code.




```
FUNCTION_BLOCK SETIO  (* Set Output data *)
VAR_INPUT
  EN      : BOOL;         (* Enable *)
  ID      : DWORD;        (* Identifier of a slot or subslot of a Field Device *)
  LEN     : INT;          (* Length of the IO data object *)
END_VAR

VAR_OUTPUT
  ENO     : BOOL;         (* Flag *)
  STATUS  : DWORD;        (* Last detected status *)
END_VAR

VAR_IN_OUT
  OUTPUTS : ANY;          (* IO data object to write *)
END_VAR
```

Example: Set outputs of a slot of a DP-slave

```
VAR D2S3: DWORD;
    Q1  : ARRAY [1..10] OF BYTE;
    SET1: SETIO;

D2S3:= ID(1,0,2,3) (* Handle for Slave 2, Slot 3 at DP system 1 *)
Q1[1]:= 17; ...
SET1 (ID:=D2S3, OUTPUTS:=Q1);
```

Figure 14 – SETIO function block

NOTE The EN and ENO parameters are optional and may be omitted when calling an instance of the function block in textual languages.

The following state diagram describes the algorithm of the SETIO function block. The following table describes the transitions and actions of this state diagram and the actions to be performed within the states and the settings of the SETIO function block outputs.

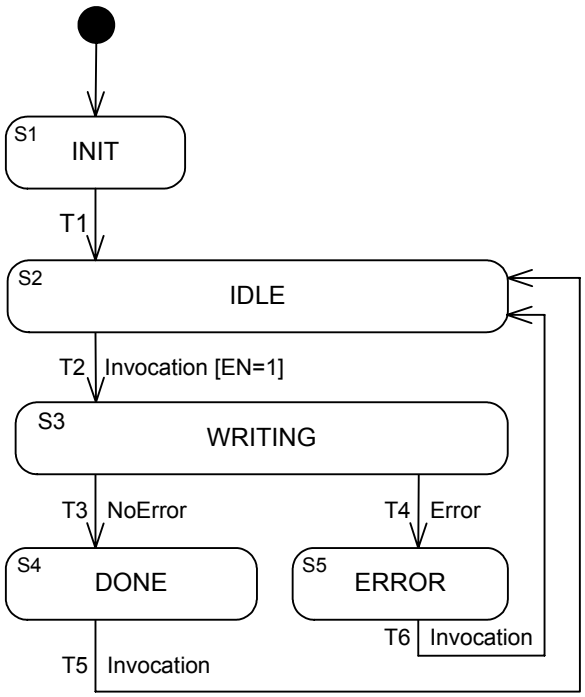


Figure 15 – State diagram of SETIO function block

The following table defines the transitions and actions given in the state diagram above.

Table 9 - Transitions and actions for SETIO state diagram

STATE NAME	**STATE DESCRIPTION**
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, no actions
S3 WRITING	read device data Transfer IO data object to DP-master with Rem Add= out of ID input Out Data= OUTPUTS parameter
S4 DONE	indicate valid data Deposit data in parameter LEN
S5 ERROR	indicate error set FB outputs

TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	ENO := 0 STATUS := 0
T2	S2	S3	Invocation (Next invocation)	EN=1	ENO := --- STATUS := ---
T3	S3	S4	NoError (No communication problems detected)		ENO := 1 STATUS := 0
T4	S3	S5	Error (Communication prob-lems detected)		ENO := 0 STATUS := error code
T5	S5	S2	Invocation (Next invocation)		ENO := --- STATUS := ---
T6	S5	S2	Invocation (Next invocation)		ENO := --- STATUS := ---
--- indicates “unchanged” FB outputs					

3.2.4 Get a Part of IO data object (GETIO_PART)

The GETIO_PART function block gets a subset of the input data associated to a slot or subslot of a Field Device.

The input data are addressed within the slot or subslot through OFFSET and LEN parameters. The GETIO_PART function block gets the input data from the Host Controller interface out of the cyclically read input data of the Field Device. The function block is invoked by a 1 of the EN input.

The ID parameter identifies the Field Device or the slot or subslot of a Field Device the IO data object is read from. The parameter OFFSET and LEN addresses an individual subset within these IO data object. The OFFSET parameter contains the number of the first byte to be read. The input data of the slot or subslot are counted beginning with 0. The LEN input contains the length of the input data to read in byte.

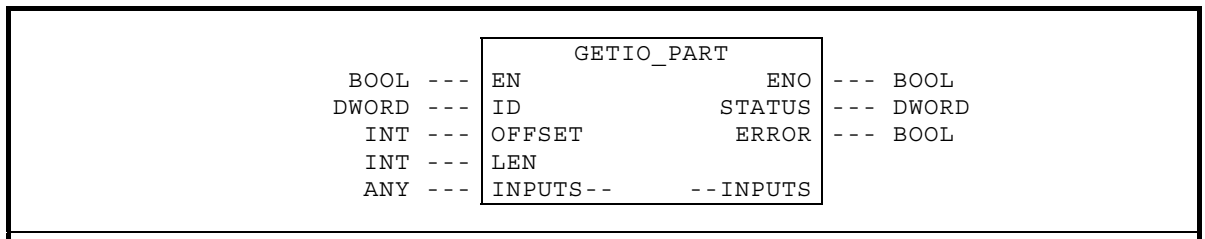
If the input data are valid, the ENO output is set to 1, the ERROR output is set to 0, and the input data are stored in the variable given at the INPUTS parameter. The variable passed to the INPUTS parameter shall be of appropriate size to receive the input data. The output parameters of this FB are set synchronously.

NOTE For PROFIBUS DP an ARRAY[1..244] OF BYTE can hold the data in all cases.

If a variable at the %I area is referenced at the INPUTS parameter the implementer shall specify the rules using this variable with this parameter.

If an error occurred, the ENO output is set to 0, the ERROR output is set to 1, and the STATUS output contains the error code. The STATUS values are defined in [3]. If the OFFSET and LEN parameters address a range out of the scope of the input data of the slot the STATUS parameter is set with STATUS[3]=16#B7.

NOTE On EN=0 the ENO output is set to 0, and the function block may be not invoked at all, and all other outputs of the function block may be frozen.




```
 FUNCTION GETIO_PART  (* Get a part of slot or subslot related IO data object *)
 VAR_INPUT
   EN      : BOOL;         (* Enable *)
   ID      : DWORD;        (* Identifier of a slot or subslot of a Field Device *)
   OFFSET  : INT ;         (* Count of the first byte within the IO data object *)
   LEN     : INT;          (* Length of the data to read *)
END_VAR

 VAR_OUTPUT
   ENO    : BOOL;          (* Flag *)
   STATUS : DWORD;         (* Last detected status *)
   ERROR  : BOOL;          (* Flag indication an error *)
 END_VAR

 VAR_IN_OUT
  INPUTS  : ANY;           (* Requested subset of the input data *)
 END_VAR
```

```
Example: Get inputs of a part of the slot data of a DP-slave

   VAR D2S3: DWORD;
       I1  : ARRAY [1..20] OF BYTE;
       CH2 : INT; (* offset of the channel within the slot *)
       L2  : INT: (* length of the channel data *)
       GET1: GETIO_PART;
   END_VAR


   D2S3:= ID(1,0,2,3) (* Handle for Slave 2, Slot 3 at DP system 1 *)
   GET1 (ID:=D2S3, OFFSET:=CH2, LEN:=L2, INPUTS:=I1);
   IF ERROR=0 THEN (* process input data *)
   ..
```

Figure 16 – GETIO_PART function block

NOTE The EN and ENO parameters are optional and may be omitted when calling an instance of the function block in textual languages.

The following state diagram describes the algorithm of the GETIO_PART function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the GETIO_PART function block outputs.

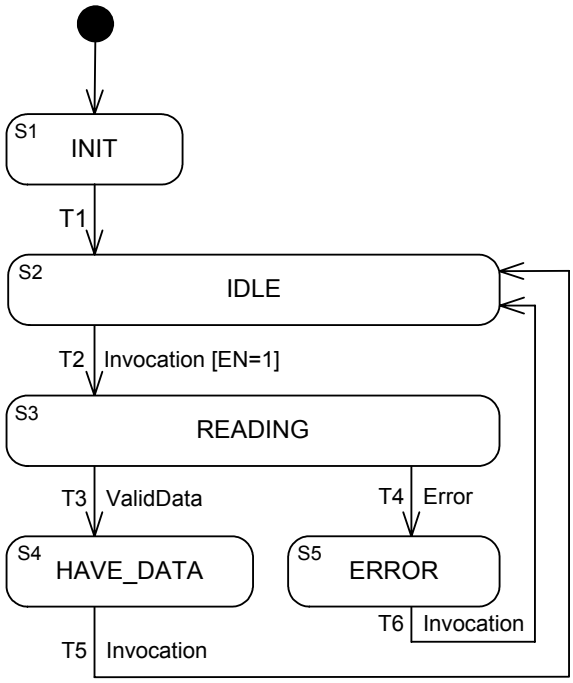


Figure 17 – State diagram of GETIO_PART function block

The following table defines the transitions and actions given in the state diagram above.

Table 10 - Transitions and actions for GETIO_PART state diagram

STATE NAME	**STATE DESCRIPTION**
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, no actions
S3 READING	read device data Evaluate input ID, OFFSET and LEN. Get IO data object from Host Controller with Rem Add= out of ID, OFFSET and LEN input Inp Data= INPUTS parameter
S4 HAVE_ DATA	indicate valid data Deposit subset of the input data of the slot, addressed by OFFSET and LEN in INPUTS parameter
S5 ERROR	indicate error set FB outputs

TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	ENO := 0 ERROR :=0 STATUS := 0 INPUTS := system null
T2	S2	S3	Invocation (Next invocation)	EN=1	ENO := --- ERROR := ---- STATUS := --- INPUTS := ---
T3	S3	S4	ValidData (Valid IO data object, OFFSET and LEN address data inside the input data of the slot)		ENO := 1 ERROR :=0 STATUS := 0 INPUTS := New data

T4	S3	S5	Error (No valid IO data object or OFFSET and LEN address data outside the input data of the slot)		ENO := 0 ERROR := 1 STATUS := New Error code INPUTS := ---
T5	S5	S2	Invocation (Next invocation)		ENO := --- ERROR := ---- STATUS := --- INPUTS := ---
T6	S5	S2	Invocation (Next invocation)		ENO := --- ERROR := ---- STATUS := --- INPUTS := ---
--- indicates “unchanged” FB outputs					

3.2.5 Set IO data object Related to a Part of a Slot (SETIO_PART)

The SETIO_PART function block sets the output data for a subset of the output data associated to a slot or subslot of a Field Device. The output data are addressed within the slot or subslot through the OFFSET and LEN parameters. The SETIO_PART function block sets the output data to the Host Controller interface into the cyclically written output data of the Field Device. The function block is invoked by a 1 of the EN input.

The ID parameter identifies the slot or subslot of the Field Device the output data are set for. The OFFSET and LEN parameters identifies a subset of data within a slot or subslot of the Field Device the output data are set for. The OFFSET parameter contains the number of the first byte to be written, the output data of the slot or subslot are counted beginning with 0. The LEN input contains the length of the output data in byte. The OUTPUTS parameter contains the subset of the output data that shall be written to the slot or subslot of the Field Device. The variable passed to the OUTPUTS parameter shall be of appropriate size to provide the output data.

NOTE For PROFIBUS DP an ARRAY[1..244] OF BYTE can hold the data in all cases.

If the Output data are stored successfully and the Field Device is still cyclically communicating, the ENO output is set to 1. The output parameters of this function block are set synchronously.

If an error occurred, the ENO output is set to 0, the ERROR output is set to 1, and the STATUS output contains the error code. Additionally if the OFFSET and LEN parameters address a range out of the scope of the data of the slot a STATUS parameters is set with STATUS[3]=16#B7.

NOTE On EN=0 the ENO output is set to 0, and the function block may be not invoked at all, and all other outputs of the function block may be frozen.


```
                          SETIO_PART
                  BOOL ---|EN                  ENO|--- BOOL
                 DWORD ---|ID                STATUS|--- DWORD
                   INT ---|OFFSET             ERROR|--- BOOL
                   INT ---|LEN
                   ANY ---|OUTPUTS---    ---OUTPUTS
```

```
FUNCTION SETIO_PART   (* Set Output data to a part of a slot or subslot*)
VAR_INPUT
  EN      : BOOL;        (* Enable *)
  ID      : DWORD;       (* Identifier of a slot / subslot of a Field Device *)
  OFFSET  : INT ;        (* Offset of the subset of output data *)
  LEN     : INT;         (* Length of the output data to write *)
END_VAR

VAR_OUTPUT
  ENO    : BOOL;         (* Flag *)
  STATUS : DWORD;        (* Last detected status *)
  ERROR  : BOOL;         (* Flag indication an error *)
END_VAR

VAR_IN_OUT
  OUTPUTS : ANY;         (* IO data object to write *)
END_VAR
```

Example: Set a part of the outputs of a slot of a DP-slave

```
VAR D2S3: DWORD;
    CH2 : INT; (* offset of the subset of data within the slot *)
    L2  : INT; (* Length of the output data to write *)
    Q1  : ARRAY [1..10] OF BYTE;
    SET1: SETIO_PART;
END_VAR

D2S3:= ID(1,0,2,3) (* Handle for Slave 2, Slot 3 at DP system 1 *)
Q1[1]:= 17; ...
SET1 (ID:=D2S3, OFFSET:=CH2, LEN:=L2, OUTPUTS:=Q1);
```

Figure 18 – SETIO_PART function block

NOTE The EN and ENO parameters are optional and may be omitted when calling an instance of the function block in textual languages.

The following state diagram describes the algorithm of the SETIO_PART function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the SETIO_PART function block outputs.

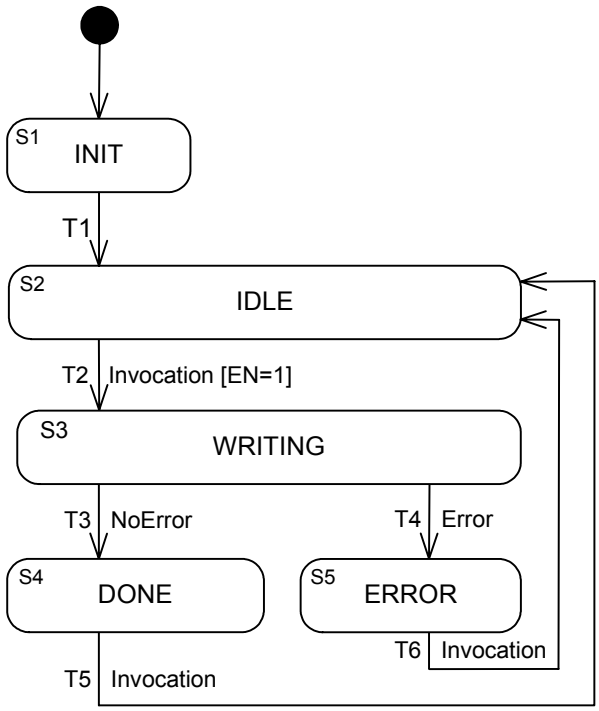


Figure 19 – State diagram of SETIO_PART function block

The following table defines the transitions and actions given in the state diagram above.

Table 11 - Transitions of the SETIO_PART state diagram

STATE NAME	**STATE DESCRIPTION**
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, no actions
S3 WRITING	Evaluate input ID, OFFSET and LEN Transfer IO data object to Field Device with Rem Add= out of ID, OFFSET and LEN input Out Data= OUTPUTS parameter
S4 DONE	No actions
S5 ERROR	indicate error set FB outputs

TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	ENO := 0 ERROR := 0 STATUS := 0
T2	S2	S3	Invocation (Next invocation)	EN=1	ENO := --- ERROR := --- STATUS := ---
T3	S3	S4	NoError (No communication problems detected, OFFSET and LEN address data inside the output data of the slot)		ENO := 1 ERROR := 0 STATUS := 0

T4	S3	S5	Error (Communication problems detected or OFFSET and LEN address data outside the output data of the slot)		ENO := 0 ERROR := 1 STATUS := New error code
T5	S5	S2	Invocation (Next invocation)		ENO := --- ERROR := --- STATUS := ---
T6	S5	S2	Invocation (Next invocation)		ENO := --- ERROR := --- STATUS := ---
--- indicates “unchanged” FB outputs					

3.3 Exchange of Process Data Records

3.3.1 General

The figure below shows the Communication Function Blocks for acyclic exchange of process data records.

The function blocks for acyclic exchange of process data records are used when a PLC is acting as a Host Controller.

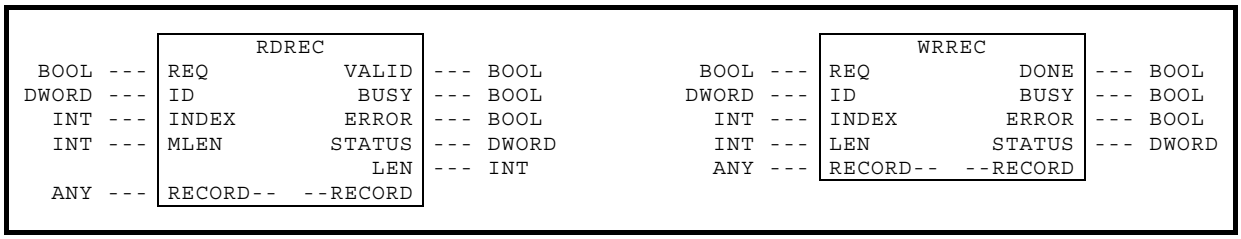


Figure 20 – Communication Function Blocks for acyclic exchange of data records

The function blocks for writing and reading have the similar parameter set except the different association of the send/read data as input or output parameters.

3.3.2 Read Process Data Record (RDREC)

The communication function Read Process Data Record for a Host Controller uses the RDREC function block defined in this clause. One instance of a RDREC function block provides one instance of the PLC function Read Process Data Record. The function is invoked when REQ input is equal to 1.

The ID parameter identifies the slot or subslot of the Field Device the data record is read from. The INDEX input of the READ function block contains an integer which identifies the data record to be read.

The MLEN parameter specifies the count of bytes which shall be read as an maximum. The variable given as RECORD parameter shall be at least of MLEN byte.

If the data record is read successfully, the VALID output indicates that the read data record is stored in the RECORD parameter. The LEN output contains the length of the data record in byte.

If an error occurred, the ERROR output indicates an error and the STATUS output contains the error code. The STATUS values are defined in Table 2.

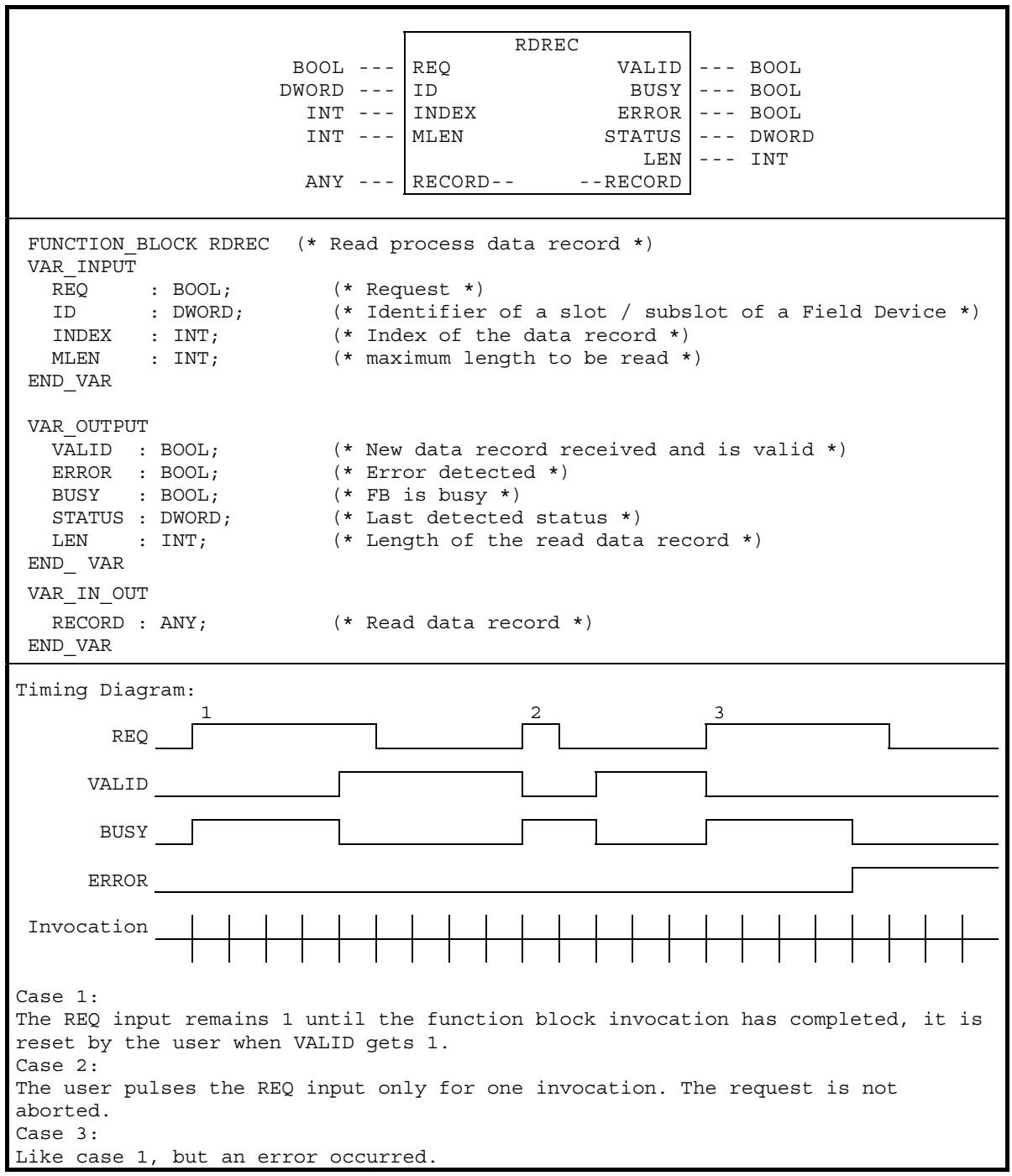


Figure 21 – RDREC function block

The following state diagram describes the algorithm of the RDREC function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the RDREC function block outputs.

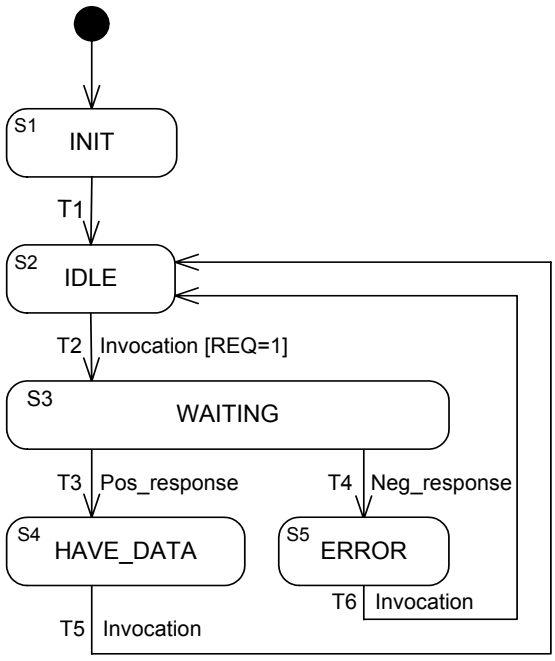


Figure 22 – State diagram of RDREC function block

The following table defines the transitions and actions given in the state diagram above.

Table 12 - Transitions and actions for RDREC state diagram

STATE NAME	**STATE DESCRIPTION**
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 WAITING	read device data Evaluate FB inputs ID and INDEX. Request variables from remote communication partner: ProcessData.Read.Req or Read.Req addressing the Field Device as specified with the ID parameter Index= INDEX Length= MLEN
S4 HAVE_ DATA	indicate valid data Deposit data in parameter LEN and RECORD
S5 ERROR	indicate error set FB outputs

TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	VALID := 0 BUSY := 0 ERROR := 0 STATUS := 0 LEN := System null
T2	S2	S3	Invocation (Next invocation)	REQ=1	Evaluate inputs VALID := 0 BUSY := 1 ERROR := 0 STATUS := -1 (is busy) LEN := System null

T3	S3	S4	Pos_response (Positive response from remote communication partner: ProcessData.Read.Cnf(+) for PROFIBUS DP or Read.Cnf(+) for PROFINET IO)		VALID := 1 BUSY := 0 ERROR := 0 STATUS := 0 LEN := New data
T4	S3	S5	Neg_response (Negative response from remote communication partner or other communication problems detected: ProcessData.Read.Cnf(-) for PROFIBUS DP or Read.Cnf(-) for PROFINET IO or Abort.Ind or local problems)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := New errorCode LEN := System null
T5	S4	S2	Invocation (Next invocation)		VALID := 1 BUSY := 0 ERROR := 0 STATUS := 0 LEN := datalen
T6	S5	S2	Invocation (Next invocation)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := errorCode LEN := System null
--- indicates “unchanged” FB outputs					

3.3.3 Write Data Record (WRREC)

The communication function Write Process Data Record for a Host Controller uses the WRREC function block defined in this clause. One instance of a WRREC function block provides one instance of the PLC function Write Process Data Record. The function is invoked when the REQ input is equal to 1.

The ID parameter identifies the slot or subslot of the Field Device the process data record is written to. The INDEX input of the WRREC function block contains an integer which identifies the data record to be written. The data record shall be stored in the variable given at the RECORD parameter. The LEN input contains the length of the data record to be written in byte. The variable given as RECORD parameter shall be at least of LEN byte.

NOTE An array declaration with zero elements is not supported in IEC 61131-3, therefore the minimum length shall be 1 byte even if the record length is zero. The actual length is given with the LEN parameter.

The values of the RECORD and LEN parameters shall not be changed as long as the BUSY output is true.

If the data record is written successfully, the DONE output indicates that the read data record is written to the Field Device.

If an error occurred, the ERROR output indicates an error and the STATUS output contains the error code. The STATUS values are defined in Table 2.

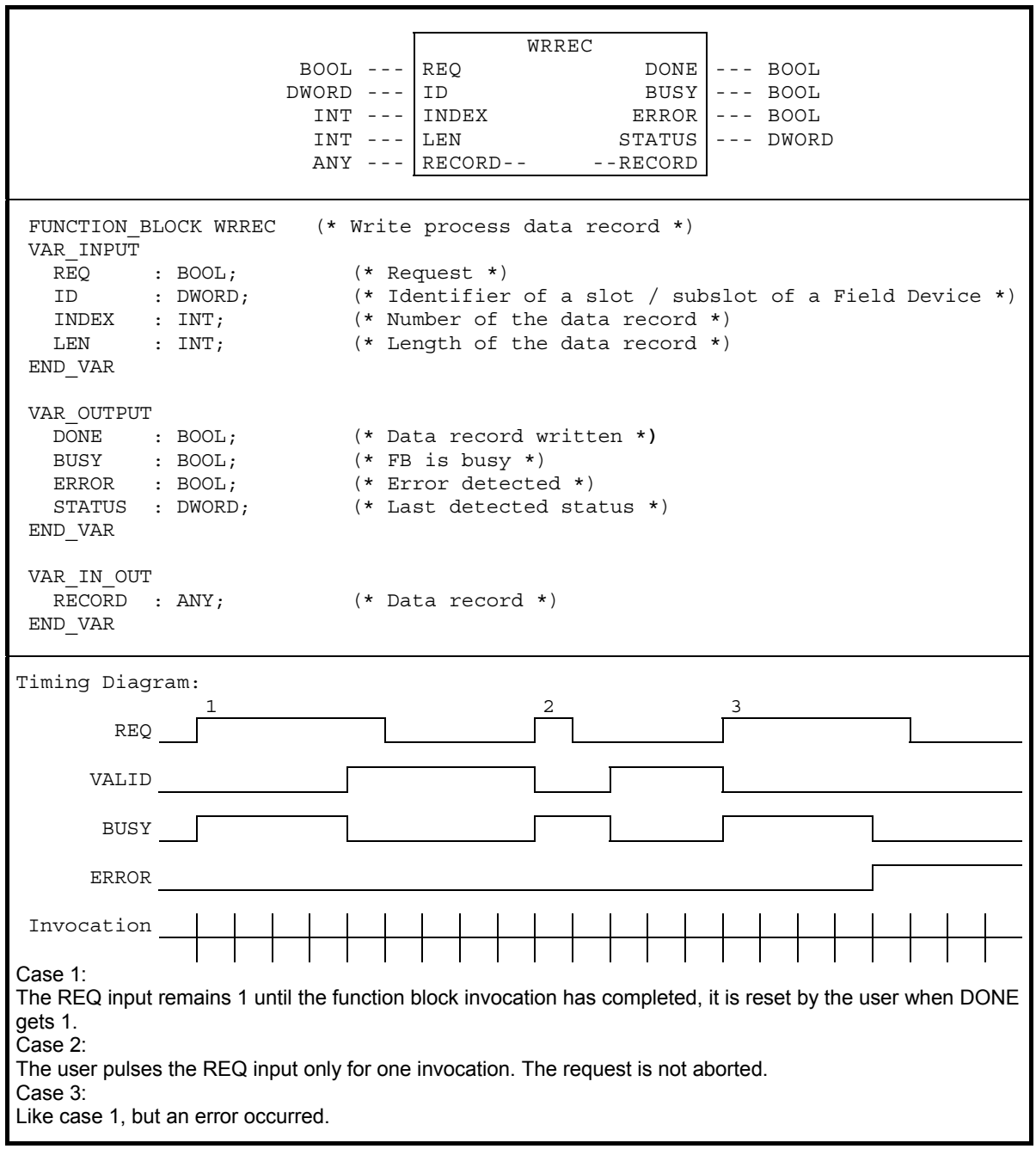


Figure 23 – WRREC function block

The following state diagram describes the algorithm of the WRREC function block. The following table describes the transitions and actions of this state diagram and the actions to be performed within the states and the settings of the WRREC function block outputs.

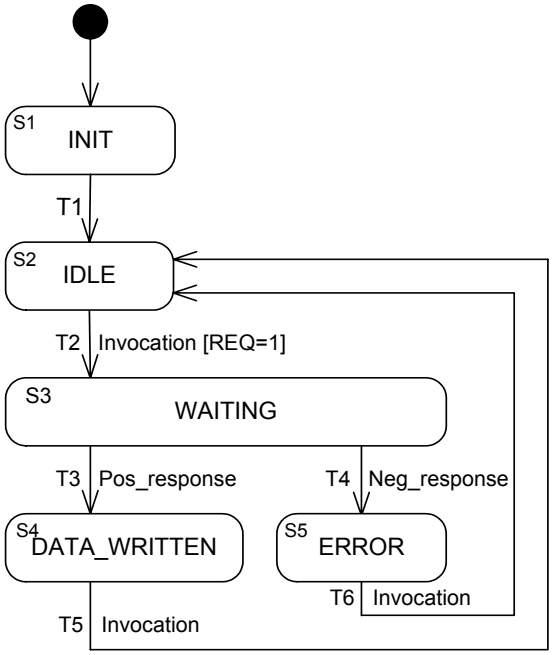


Figure 24 – State diagram of WRREC function block

The following table defines the transitions given in the state diagram above.

Table 13 - Transitions of the WRREC state diagram

STATE NAME	**STATE DESCRIPTION**
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 WAITING	Evaluate FB parameters ID, INDEX, LEN and RECORD Request variables from remote communication partner: ProcessData.Write.Req or Write.Req addressing the Field Device as specified with the ID parameter Index= INDEX Length= LEN Data= RECORD
S4 DATA_WRITTEN	Indicate done
S5 ERROR	indicate error

TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	VALID := 0 BUSY := 0 ERROR := 0 STATUS := 0
T2	S2	S3	invocation (Next invocation)	REQ=1	VALID := 0 BUSY := 1 ERROR := 0 STATUS := -1 (is busy)

T3	S3	S4	Pos_response (Positive response from remote communication partner: ProcessData.Write.Cnf(+)for PROFIBUS DP or Write.Cnf(+) for PROFINET IO)		VALID := 1 BUSY := 0 ERROR := 0 STATUS := 0
T4	S3	S5	Neg_response (Negative response from remote communication partner or other communication problems detected: ProcessData.Write.Cnf(-) for PROFIBUS DP or Write.Cnf(-) for PROFINET IO or Abort.Ind or local problems)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := New errorCode
T5	S4	S2	Invocation (Next invocation)		VALID := 1 BUSY := 0 ERROR := 0 STATUS := 0
T6	S5	S2	Invocation (Next invocation)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := errorCode
--- indicates “unchanged” FB outputs					

3.4 Alarms and Diagnosis

3.4.1 General

A Field Device may send alarms to a Host Controller. The PLC operating system may react on an incoming alarm by activating a task (see IEC 61131-3) of the application program. Or the application program may poll for alarm cyclically. Using the function block RALRM the application program can poll for the alarm or get more information about the alarm when activated by a task.

If the alarm activates a task the PLC operating system shall acknowledge the alarm automatically when the task is terminated. If the alarm is polled by the application program, the application program is responsible to acknowledge the alarm by using the Communication Function Block RALRM.

Additionally the PROFIBUS DP systems provides diagnosis status information about the DP-slaves associated to a DP-master (Class 1). This information can be retrieved using the function block RDIAG.

3.4.2 Receiving Alarms (RALRM)

The communication function Receive Alarm for a Host Controller uses the RALRM function block defined in this clause. One instance of a RALRM function block provides one instance of the PLC function Receive Alarm.

The function is invoked by EN=1. The MODE input controls the functionality of the RALRM function block.

This function blocks contains the methods to receive and acknowledge an alarm. All aspects of receiving an alarm shall use one function block instance, the different methods are distinguished using the MODE input.

MODE	Meaning
1	Receive all alarms: If the Host Controller interface has received an alarm, all function block outputs are updated. The alarm is acknowledged.
2	Receive alarms from one slot or subslot: If the Host Controller interface has received an alarm for the slot or subslot the identification of which is given in F_ID input, all function block outputs are updated. The alarm is acknowledged.

The MLEN parameter specifies the count of bytes which shall be received as a maximum of the alarm information. The byte array given as AINFO parameter shall be at least of MLEN byte.

NOTE An array declaration with zero elements is not supported in IEC 61131-3, therefore the minimum length shall be 1 byte even if the record length is zero. The actual length is given with the LEN parameter.

If an alarm is received (with MODE=1 or MODE=2), the NEW output indicates that the alarm information is stored in the outputs. The LEN output contains the length in byte of the additional alarm information stored in the AINFO parameter.

The AINFO parameter contains alarm information of the following structure depending on the used IO subsystem type:

Table 14 - Structure of the variable at AINFO parameter for PROFIBUS DP

Offset	Name	Data Type	Meaning
0	D_LEN	BYTE	Length of the alarm information: 4 .. 63
1	ATYPE	BYTE	Alarm Type: 1 Diagnosis Alarm 2 Process Alarm 3 Pull Alarm 4 Plug Alarm 5 Status Alarm 6 Update Alarm 32-126 Manufacturer Specific
2	SLOT	BYTE	Slot number
3	ASPEC	BYTE	Alarm Specifier: 0 no further differentiation 1 Alarm appears and the related module is disturbed 2 Alarm disappears and the related module has no further errors 3 Alarm disappears and the related module is still disturbed
4 - 62	ADD_INFO	ARRAY [1..x] OF BYTE	Additional alarm information with a maximum of 59 bytes

Table 15 - Structure of the variable at AINFO parameter for PROFINET IO

Offset	Name	Description	
0-1	BlockType	reserved	BlockType
2-3	BlockLength	BlockLength	
4-5	Version	VersionHigh	VersionLow
6-7	Typ	AlarmType	

Offset	Name	Description					
8-11	API	Application Process Identifier					
12-13	Slot	Slot Number					
14-15	Subslot	Subslot Number					
16-19	Source ID	Modul Identification					
20-23		Submodul Identification					
24-25	Alarm Specifier	ARDiagnosisState Bit 15	Reserved Bit 14	Submodule Diagnosis-State Bit13	Manufacturer Diagnosis Bit12	Channel Diagnosis Bit11	Sequence Nummer Bit 0 .. 10
26-27	User structure identifier	User structure identifier					
ab 28	User	Alarm-Info (0...xx Byte)					

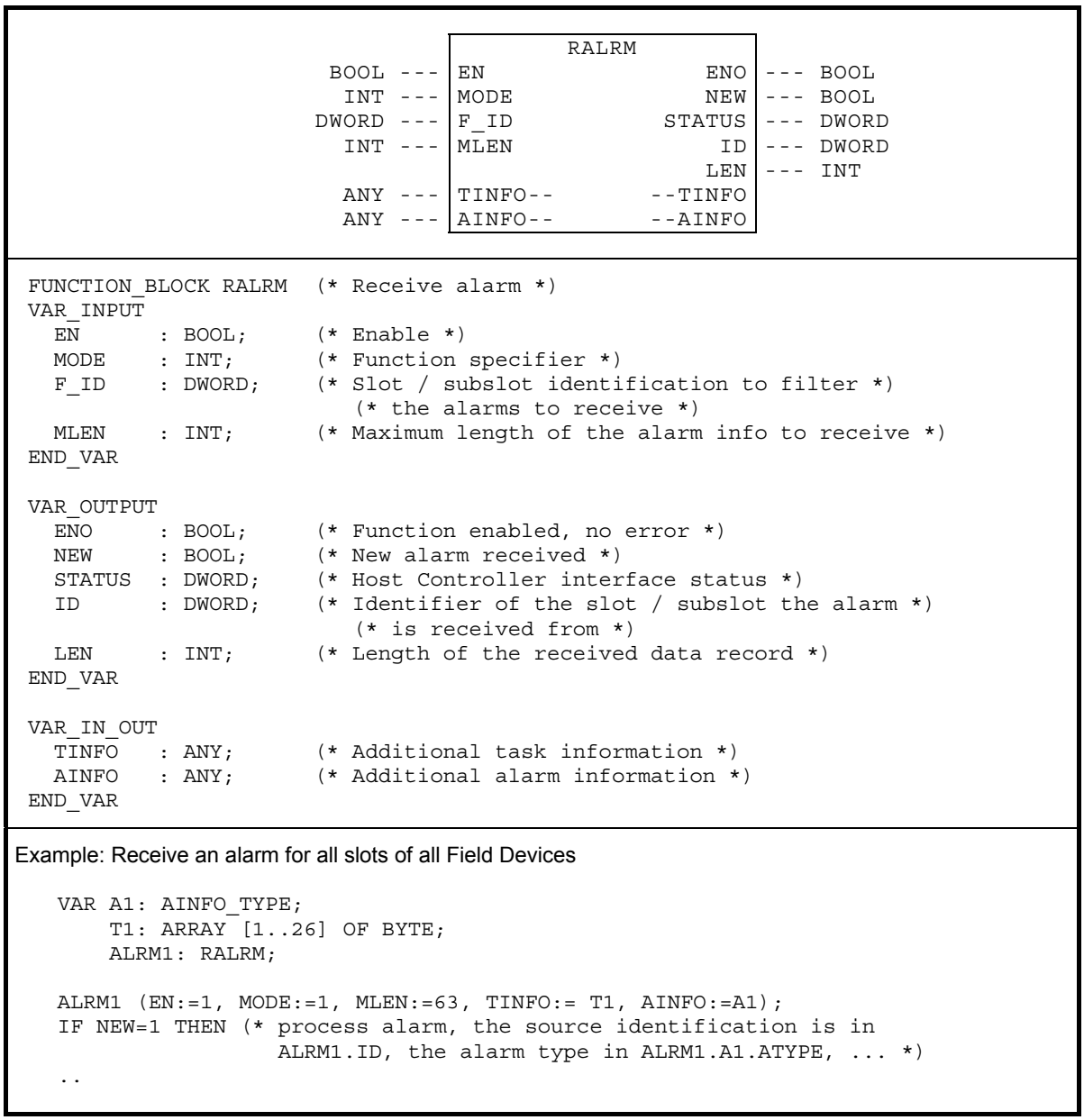
See PROFINET IO V2.00 for detailed information.

The variable passed to the AINFO parameter shall be of appropriate size to receive the additional alarm information.

NOTE The RALRM function block is used to receive alarm information of alarms from a PROFIBUS DP or a PROFINET IO subsystem. It may also come from other IO subsystem types. In this case the AINFO parameter may be of different length and structure. The implementer shall specify the content of the alarm information in these cases.

If a task is started when an alarm is received the variable given at the TINFO parameter may contain additional task information. The implementer shall specify the content of the task information TINFO.

NOTE The structure of additional alarm information given at ´the AINFO parameter is defined by the used IO subsystem. The information how to interpret this information should be given in the information given via the TINFO parameter.




```
FUNCTION_BLOCK RALRM  (* Receive alarm *)
VAR_INPUT
  EN      : BOOL;     (* Enable *)
  MODE    : INT;      (* Function specifier *)
  F_ID    : DWORD;    (* Slot / subslot identification to filter *)
                         (* the alarms to receive *)
  MLEN    : INT;      (* Maximum length of the alarm info to receive *)
END_VAR

VAR_OUTPUT
  ENO     : BOOL;     (* Function enabled, no error *)
  NEW     : BOOL;     (* New alarm received *)
  STATUS  : DWORD;    (* Host Controller interface status *)
  ID      : DWORD;    (* Identifier of the slot / subslot the alarm *)
                         (* is received from *)
  LEN     : INT;      (* Length of the received data record *)
END_VAR

VAR_IN_OUT
  TINFO   : ANY;      (* Additional task information *)
  AINFO   : ANY;      (* Additional alarm information *)
END_VAR
```

Example: Receive an alarm for all slots of all Field Devices

```
VAR A1: AINFO_TYPE;
    T1: ARRAY [1..26] OF BYTE;
    ALRM1: RALRM;

ALRM1 (EN:=1, MODE:=1, MLEN:=63, TINFO:= T1, AINFO:=A1);
IF NEW=1 THEN (* process alarm, the source identification is in
                 ALRM1.ID, the alarm type in ALRM1.A1.ATYPE, ... *)
..
```

Figure 25 – RALRM function block

The following state diagram describes the algorithm of the RALRM function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the RALRM function block outputs.

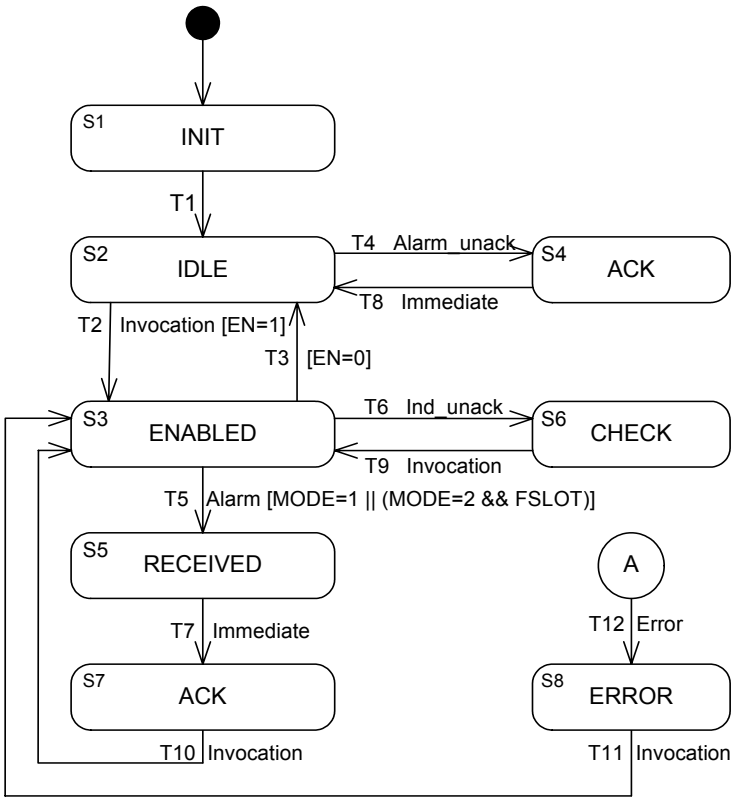


Figure 26 – State diagram of RALRM function block

The ERROR state may be entered from the states ENABLED, CHECK, RECEIVED, POS_ACK or NEG_ACK if a communication error is detected.

The following table defines the transitions and actions given in the state diagram above.

Table 16 - Transitions and actions for RALRM state diagram

STATE NAME	**STATE DESCRIPTION**
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 ENABLE	No actions
S4 / S7 ACK	Positive response to DP-Master: Alarm.rsp(+) with parameters taken from the indication
S5 RECEIVED	Deposit data in parameter ID, LEN, TINFO, and AINFO
S6 CHECK	Update outputs ID and LEN
S8 ERROR	indicate error

TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	ENO := 0 NEW := 0 ID, LEN := System null TINFO, AINFO := --- STATUS := 0

T2	S2	S3	Invocation (Next invocation)	EN=1	ENO := 1 NEW := 0 ID, LEN := 0 TINFO, AINFO := --- STATUS := ---
T3	S3	S2		EN=0	ENO := 0 NEW := 0 ID, LEN := System null TINFO, AINFO := --- STATUS := 0
T4	S2	S4	Alarm_unack (Unacknowledged alarm from Field Device: Alarm.Ind for PROFIBUS DP or AlarmNotification.Ind for PROFIBUS IO)		ENO := 1 NEW := --- ID, LEN := --- TINFO, AINFO := --- STATUS := ---
T5	S3	S5	Alarm (indication from Field Device: Alarm.Ind for PROFIBUS DP or AlarmNotification.Ind for PROFIBUS IO)	MODE=1 or (MODE=2 and FSLOT=slot identification of the alarm)	ENO := 1 NEW := 1 ID, LEN := New data TINFO, AINFO := New info STATUS := ---
T6	S3	S6	Ind_unack (unacknowledged indication from DP-slave)	MODE=0	ENO := 1 NEW := 1 ID, LEN := New data TINFO, AINFO := --- STATUS := ---
T7	S5	S7	Immediate		ENO := 1 NEW := --- ID, LEN := --- TINFO, AINFO := --- STATUS := ---
T8	S4	S2	Immediate		ENO := 0 NEW := 0 ID, LEN := System null TINFO, AINFO := --- STATUS := 0
T9	S6	S3	Invocation (nest invocation)		ENO := 1 NEW := 0 ID, LEN := 0 TINFO, AINFO := --- STATUS := ---
T10	S7	S3	Invocation (nest invocation)		ENO := 1 NEW := 0 ID, LEN := 0 TINFO, AINFO := --- STATUS := ---
T11	S8	S3	Invocation (nest invocation)		ENO := 1 NEW := 0 ID, LEN := 0 TINFO, AINFO := --- STATUS := ---
T12	S3, S4, S5, S6, S10	S8	Error (Communikation error detected)		ENO := 0 NEW := 0 ID, LEN := --- TINFO, AINFO := --- STATUS := New status
--- indicates “unchanged” FB outputs					

3.4.3 Read Diagnosis (RDIAG)

This function is provided only for PROFIBUS DP.

The communication function Read Diagnosis for a DP-master (Class 1) uses the RDIAG function block defined in this clause. The DP system provides diagnosis status information about the DP-slave to a DP-Master. One instance of a RDIAG function block provides one instance of the PLC function Read Diagnosis. The function is invoked when the REQ input is equal to 1.

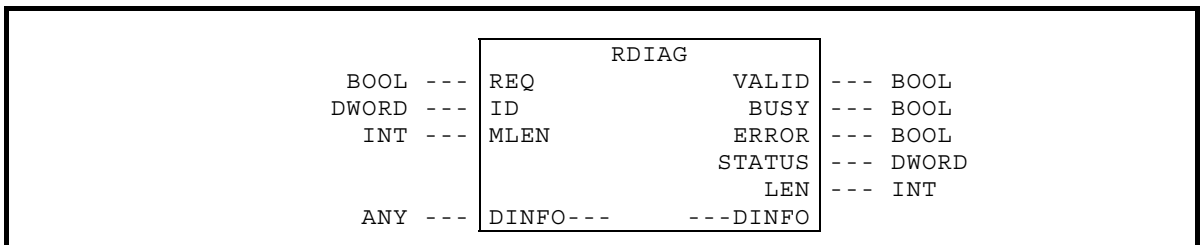
The ID parameter identifies the slot of the Field Device the diagnosis is read from.

The MLEN parameter specifies the count of bytes which shall be read as an maximum. The variable given at the DINFO parameter shall be at least of MLEN byte. Possible value range of the MLEN input is 0 .. 238.

If the diagnosis information is read successfully, the VALID output indicates that the data is stored in the DINFO parameter. The variable passed to the DINFO parameter shall be of appropriate size to receive the diagnosis data. An ARRAY[1..238] OF BYTE can hold the data in all cases. The LEN output contains the length of the data in byte.

> NOTE If the interface to the DP-master can provide the diagnosis information synchronously e.g. at the time requested, the BUSY output is never seen to be 1, and the other outputs are valid.

If an error occurred, the ERROR output indicates an error and the STATUS output contains the error code. The STATUS values are defined in table 3.




```
FUNCTION_BLOCK RDIAG      (* Read diagnosis *)
VAR_INPUT
  REQ     : BOOL;         (* Request *)
  ID      : DWORD;        (* Identifier of a DP-slave *)
  MLEN    : INT;          (* maximum length to be read *)
END_VAR

VAR_OUTPUT
  VALID  : BOOL;          (* New diagnosis data received and is valid *)
  BUSY   : BOOL;          (* FB is busy *)
  ERROR  : BOOL;          (* Error detected *)
  STATUS : DWORD;         (* Last detected status *)
  LEN    : INT;           (* Length of the diagnosis data *)
END_ VAR
VAR_IN_OUT
  DINFO  : ANY;           (* Read diagnosis data *)
END_VAR
```

Figure 27 – RDIAG function block

The following state diagram describes the algorithm of the RDIAG function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the RDIAG function block outputs.

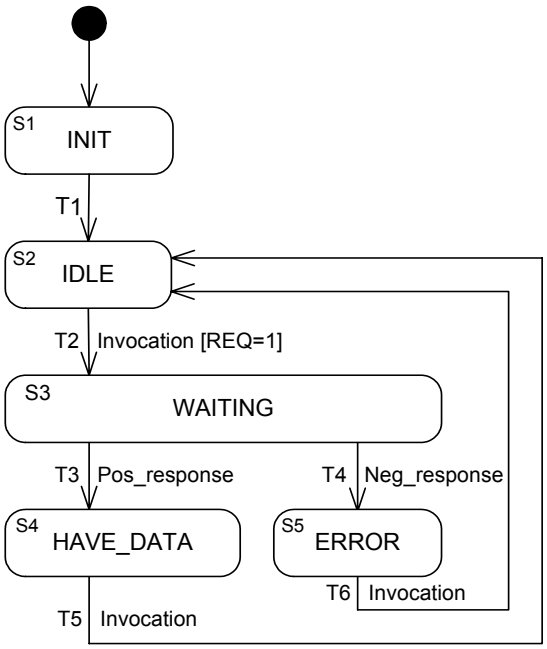


Figure 28 – State diagram of RDIAG function block

The following table defines the transitions and actions given in the state diagram above.

Table 17 - Transitions and actions for RDIAG state diagram

STATE NAME	**STATE DESCRIPTION**
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 WAITING	Evaluate FB inputs ID. Get diagnosis GetSlaveDiag.Req with AREP= slave id out of ID
S4 HAVE_DATA	Deposit data in parameter LEN and DINFO
S5 ERROR	indicate error

TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	VALID := 0 LEN, DINFO := Sys-tem null ERROR, STATUS := 0
T2	S2	S3	Invocation (Next invocation)	REQ=1	VALID := 0 LEN, DINFO := --- ERROR, STATUS := ---
T3	S3	S4	Pos_response (Positive response from DP-master interface: GetSlaveDiag.Cnf(+))		VALID := 1 LEN, DINFO := New data ERROR, STATUS := 0

T4	S3	S5	Neg_response (Negative response from remote communication partner or other communication problems detected: GetSlaveDiag.Cnf(-) or other communication problems)		VALID := 0 LEN, DINFO := --- ERROR, STATUS := New error code
T5	S4	S2	Invocation (Next invocation)		VALID := --- LEN, DINFO := --- ERROR, STATUS := ---
T6	S5	S2	Invocation (Next invocation)		VALID := --- LEN, DINFO := --- ERROR, STATUS := ---
--- indicates “unchanged” FB outputs					

3.5 Higher Communication Functions

3.5.1 Interlocked Control (ICTRL)

For the purpose to achieve interlocked control including an order from the Host Controller to the slot or subslot of the Field Device accompanied by a set of data a standard function block can be used. The following timeline illustrates the sequence of this function.

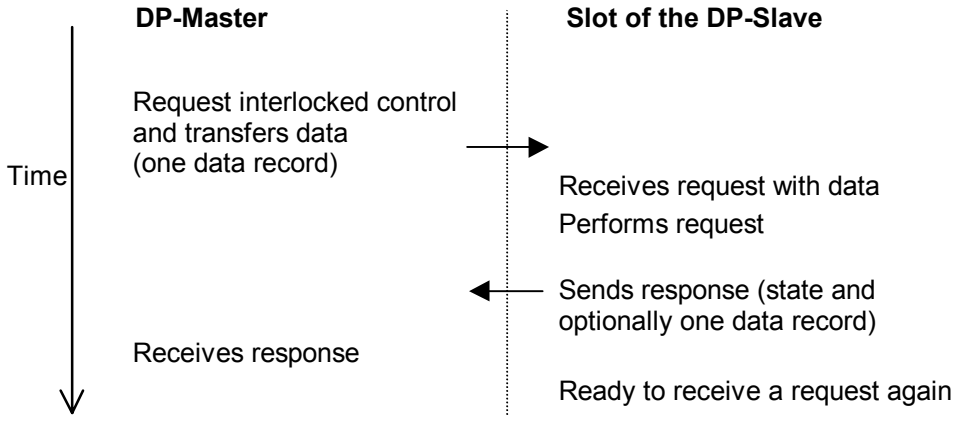


Figure 29 – Interlocked Control Timeline

The interface of this communication function "Interlocked Control" for a Host Controller uses the ICTRL function block defined in this clause. When interlocked control is requested the function block writes a data record to a slot or subslot of a Field Device. The Field Device performs the request and transfers the state of its execution to the function block. If the request is done and a result data record is available at the Field Device, the function block reads this data record.

The maximum amount of data for the request and the maximum amount of data which can be received as a result is one data record.

One instance of a ICTRL function block provides one instance of the PLC function Interlocked Control. The function is invoked when the REQ input is equal to 1.

The ID parameter identifies the slot of the Field Device the request shall be executed. The I_REQ input of the ICTRL function block contains an integer which identifies the data record of the request. The I_RES input of the ICTRL function block contains an integer which identifies the

data record of the result. This feature is optional: If no result is available or no result data record shall be transferred, the value –1 shall be given. The R_STATE input of the ICTRL function block gives the remote state byte in the user data of the slot which will be used to get the state of the request execution.

The DATA_REQ input of the ICTRL function block contains the data of the request. The first byte of these data may be used as a method identifier. The first byte shall not have the value 255.

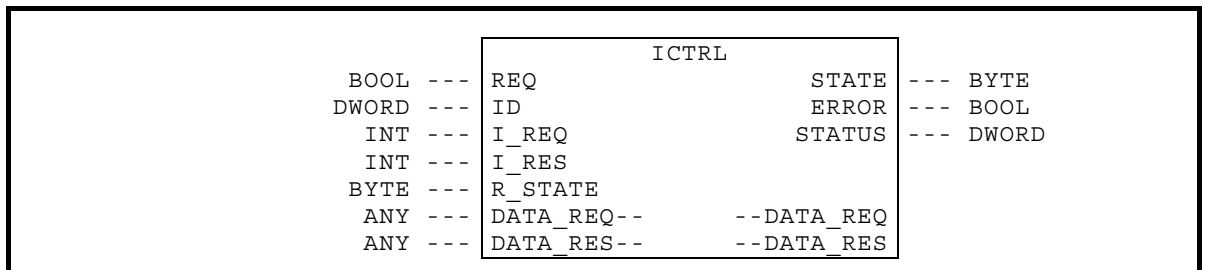
If the function block is called with REQ input = 0 a still executing request is cancelled.

The state of the request execution is given at the STATE output. It contains one of the following values:

Table 18 – States of interlocked control execution

Value of STATE output	**Meaning**
0	NOT_READY: Not ready to receive a new request
1	READY: Ready to receive a new request
2	REQUESTING: Request received and transferring
3	EXECUTING: Request executing
4	READY_WITHOUT_DATA: Request done at Field Device without result data record
5	READY_WITH_DATA: Request done at Field Device, result data record available
6	READING_RESULT: Reading result data record
7	READY: Function block completed
254	ABORTED: Request aborted by Field Device
255	CANCELLED: Request cancelled by function block

The DATA_RES output contains the result data record of the request if a result data record is available.




```
FUNCTION_BLOCK ICTRL  (* Start interlocked control *)
VAR_INPUT
  REQ     : BOOL;        (* Request *)
  ID      : DWORD;       (* Identifier of a slot of a Field Device*)
  I_REQ   : INT;         (* Index of the data record to request
                            interlocked control *)
  I_RES   : INT;         (* Index of the data record
                            for the result *)
  R_STATE : BYTE;        (* Remote job state byte out of the
                            user data of the slot *)
END_VAR

VAR_OUTPUT
  STATE   : BYTE;        (* State of the execution *)
  ERROR   : BOOL;        (* Error detected *)
  STATUS  : DWORD;       (* Last detected status *)
END_VAR

VAR_IN_OUT
  DATA_REQ: ANY;         (* Data record to request the job *)
  DATA_RES: ANY;         (* Data record for the result *)
END_VAR
```

Example:

```
VAR JOB1: ICRTL;
    START: BOOL;
    D3S4: DWORD;          (* Identifier of the slot *)
    REQ1: ARRAY[1..5] OF BYTE := [1, 4(0)]; (* Request buffer *)
    RSP1: ARRAY[1..3] OF BYTE               (* Response buffer *)

JOB1 (REQ:= START, ID:= D3S4,       (* Start interlocked control *)
      I_REQ:= 34, I_RES:= 34,       (* Use records #34 for data transfer *)
      R_STATE:= %IB16,              (* Use Input byte 16 for state *)
      DATA_REQ:= REQ1, DATA_RES:= RSP1);
```

Figure 30 – ICTRL function block

The following state diagram in the figure below describes the algorithm of the ICTRL function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the ICTRL function block outputs.

One byte of the user input data of the device is used for synchronisation of remote jobs as the remote job state. It is typically read cyclic by the device FB and given to the FB ICTRL via its input parameter R_STATE.

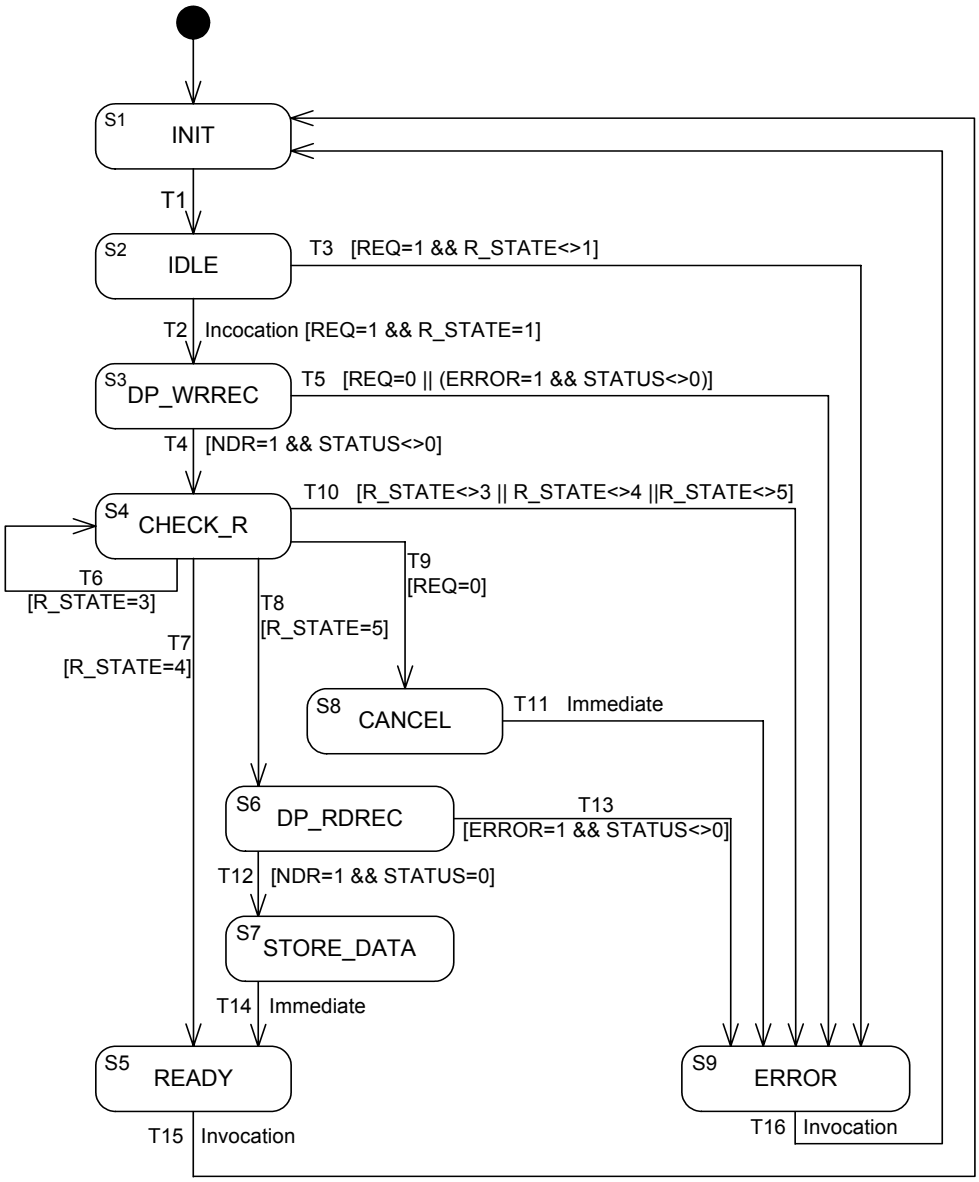


Figure 31 – State diagram of ICTRL function block

Table 19 - Transitions and actions for ICTRL state diagram

STATE NAME	STATE DESCRIPTION
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 DP_WRREC	Request new interlocked control from remote communication partner: VAR IC1: WRREC; IC1(REQ:= REQ, ID:= ID INDEX:= I_REQ; RECORD:= DATA_REQ)
S4 CHECK_R	Check input parameter R_STATE
S5 READY	
S6 DP_RDREC	Read result from remote communication partner: VAR IC2: RDREC; IC2 (REQ:= REQ,

	ID:= ID INDEX:= I_RES);
S7 STORE_DATA	Deposit data
S8 CANCEL	Request new interlocked control from remote communication partner: DATA_REQ [1]:= 16#FF; IC1(REQ:= REQ, ID:= ID INDEX:= I_REQ; RECORD:= DATA_REQ)
S9 ERROR	indicate error

TRAN-SITION	SOURCE STATE	TARGET STATE	EVENT	CONDITION	ACTION
T1	S1	S2		Initialisation done	STATE := R_STATE ERROR := 0 STATUS := 0 DATA_RES := System null
T2	S2	S3	Invocation (Next invocation)	REQ = 1 and R_STATE = 1 (Ready)	STATE := 2 (REQUESTING) ERROR := --- STATUS := --- DATA_RES := ---
T3	S2	S9		REQ = 1 and R_STATE <> 1 (READY)	STATE := 0 ERROR := 1 STATUS := Error code DATA_RES := ---
T4	S3	S4		Positive result of FB WRREC from remote communication partner: NDR=1and STATUS=0	STATE := R_STATE ERROR := --- STATUS := --- DATA_RES := ---
T5	S3	S9		REQ=0 or negative result of FB WRREC from remote communication partner: ERROR=1; STATUS<>0	STATE := 0 ERROR := 1 STATUS := Error code DATA_RES := ---
T6	S4	S4		Remote job state of input parameter R_STATE = 3 (EXECUTING)	STATE := R_STATE ERROR := --- STATUS := --- DATA_RES := ---
T7	S4	S5		Remote job state of input parameter R_STATE = 4 (READY_WITHOUT_DATA)	STATE := 7 (READY) ERROR := 0 STATUS := 0 DATA_RES := ---
T8	S4	S6		Remote job state of input parameter R_STATE = 5 (READY_WITH_DATA)	STATE := 6 (READING_RESULT) ERROR := --- STATUS := --- DATA_RES := System null
T9	S4	S8		REQ=0	STATE := 255 (CANCELLED) ERROR := --- STATUS := --- DATA_RES := ---
T10	S4	S9		Remote job state of input parameter R_STATE <> 3, 4 or 5	STATE := 0 ERROR := 1 STATUS := Error code DATA_RES := ---

T11	S8	S9	Immediate		STATE := 0 ERROR := 1 STATUS := Error code DATA_RES := ---
T12	S6	S7		Positive result of FB RDREC from remote communication partner: NDR=1; STATUS=0 and result data are ready	STATE := --- ERROR := --- STATUS := --- DATA_RES := IC2.RECORD
T13	S6	S9		Negative result of FB RDREC from remote communication partner: ERROR=1; STATUS<>0	STATE := 0 ERROR := 1 STATUS := Error code DATA_RES := ---
T14	S7	S5	Immediate		STATE := 7 (READY) ERROR := 0 STATUS := 0 DATA_RES := ---
T15	S5	S1	Invocation (next Invocation)		STATE := 0 ERROR := 0 STATUS := 0 DATA_RES := System null
T16	S9	S1	Invocation (next Invocation)		STATE := 0 ERROR := 0 STATUS := 0 DATA_RES := System null
--- indicates “unchanged” FB outputs					

4 Communication Function Blocks for Supervisor

4.1 General

A PLC may act as a Supervisor.

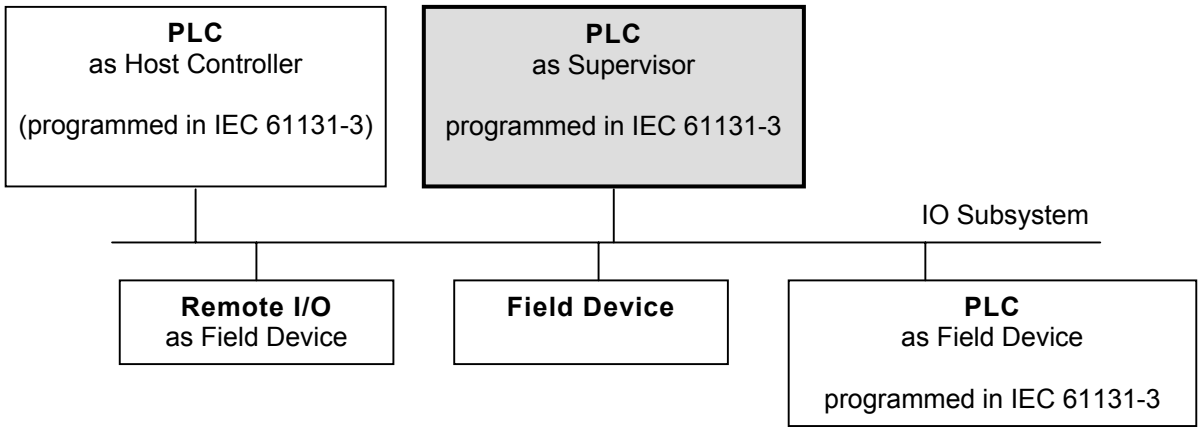


Figure 32 – Profibus system with a PLC as Supervisor

The following function blocks define the application program interface for a PLC acting as a Supervisor:

- RDIN: Read input data of a Field Device
- RDOUT: Read output data of a Field Device
- RDREC: Read a process data record from a slot of a Field Device
- WRREC: Write a process data record to a slot of a Field Device
- RDIAG: Read diagnosis information from a Field Device
- CNCT: Manage a connection to a Field Device

4.2 Reading IO data object

4.2.1 Read Input Data (RDIN)

The communication function Read Input Data Record for a Supervisor uses the RDIN function block defined in this clause. One instance of a RDIN function block provides one instance of the PLC function Read Input Data. The function is invoked when the REQ input is equal to 1.

The ID parameter identifies the slot or subslot of the Field Device the input data is read from.

The MLEN parameter specifies the count of bytes which shall be read as a maximum. The variable given as INPUTS parameter shall be at least of MLEN byte.

If the input data are read successfully, the VALID output indicates that the read data are stored in the IO output. The LEN output contains the length of the read Input data in byte.

If an error occurred, the ERROR output indicates an error and the STATUS output contains the error code. The STATUS values are defined in table 3.

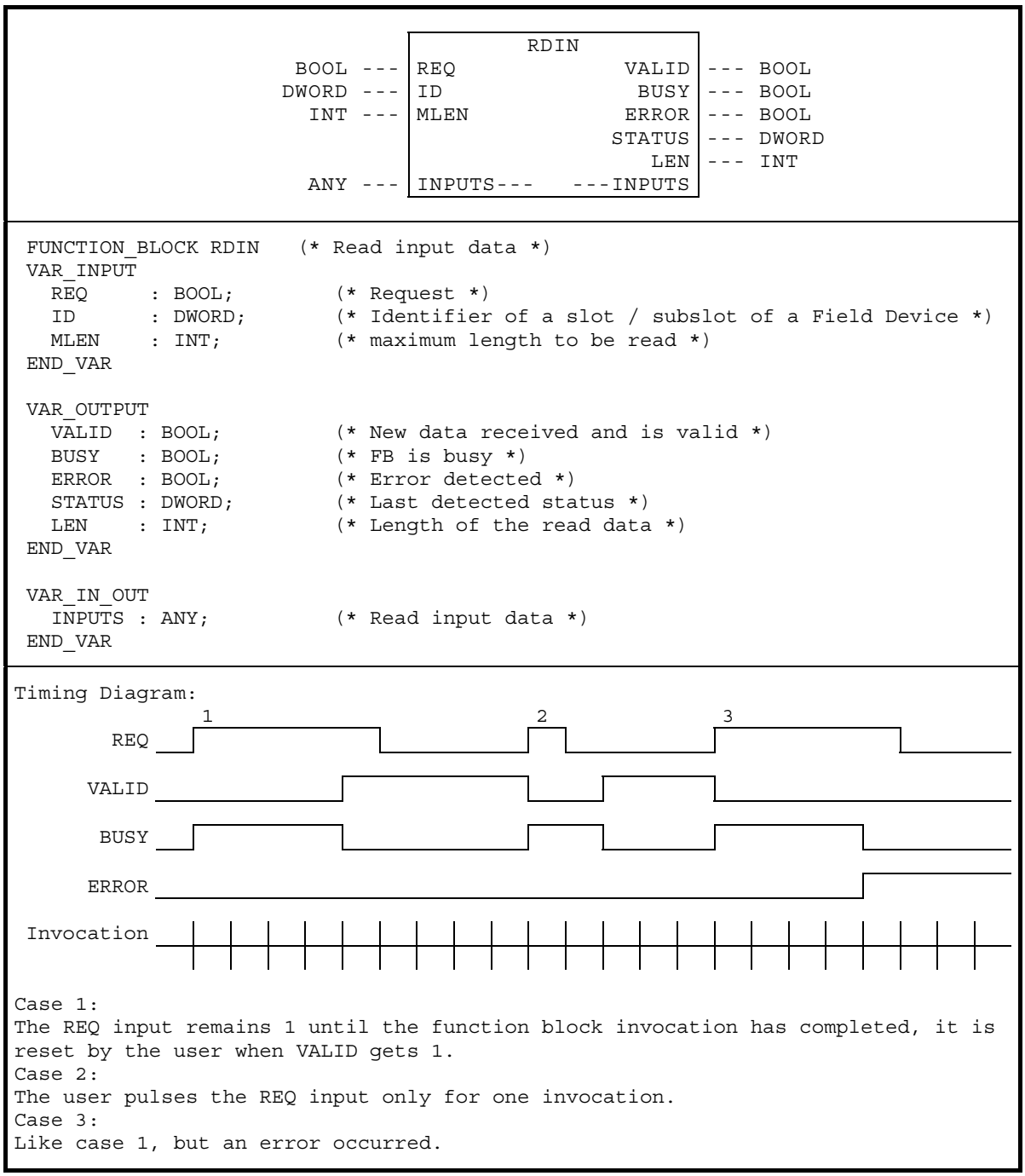


Figure 33 – RDIN function block

The following state diagram describes the algorithm of the RDIN function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the RDIN function block outputs.

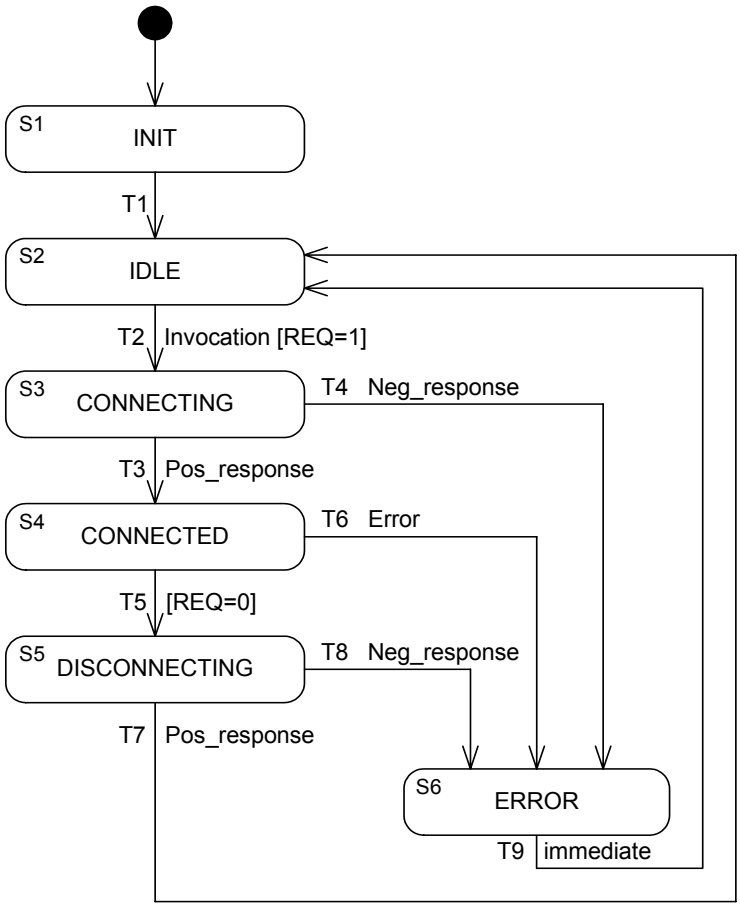


Figure 34 – State diagram of RDIN function block

The following table defines the transitions given in the state diagram above.

Table 20 - Transitions and actions for RDIN state diagram

STATE NAME	STATE DESCRIPTION
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 CONNECTING	Evaluate FB input ID. Request to establish a point-to-point connection to the remote communication partner: Connect.Req with AREP= device id out of ID AddAddrParam.D-Len= D_LEN out of D_ADDR AddAddrParam.D-Addr.D-NetworkAddress = D
S4 CONNECTED	No actions
S5 DISCONNECTING	Evaluate FB input ID. Request to close the connection to the remote communication partner: Disconnect.Req with AREP= device id out of ID
S6 ERROR	indicate error

TRAN-SITION	SOURCE STATE	TARGET STATE	EVENT	CONDITION	ACTION
T1	S1	S2		Initialisation done	VALID := 0 BUSY := 0 ERROR := 0 STATUS := 0
T2	S2	S3	Invocation (Next invocation)	REQ=1	VALID := 0 BUSY := 1 ERROR := 0 STATUS := -1
T3	S3	S4	Pos_response (Positive response from remote communication partner: Connect.Cnf(+)for PROFIBUS DP and for PROFINET IO)		VALID := 1 BUSY := 0 ERROR := 0 STATUS := 0
T4	S3	S6	Neg_response (Negative response from remote communication partner or other communi-cation problems detected: Connect.Cnf(-)for PROFIBUS DP and for PROFINET IO or Abort.Ind or local problems)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := New error code
T5	S4	S5		REQ=0	VALID := 0 BUSY := 1 ERROR := 0 STATUS := 0
T6	S4	S6	Error (Communication problems detected, connection aborted: Abort.Ind or local problems)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := New error code
T7	S5	S2	Pos_response (Positive response from remote communication partner: Disconnect.Cnf(+) for PROFIBUS DP or Re-lease.Cnf(+) for PROFINET IO)		VALID := --- BUSY := --- ERROR := --- STATUS := ---
T8	S5	S6	Neg_response (Negative response from remote communication partner or other communi-cation problems detected: Disconnect.Cnf(-) for PROFIBUS DP or Re-lease.Cnf(+) for PROFINET IO or Abort.Ind or local problems)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := New error code
T9	S6	S2	Immediate		
--- indicates “unchanged” FB outputs					

4.2.2 Read Output Data (RDOUT)

The communication function Read Output Data for a Supervisor uses the RDOUT function block defined in this clause. One instance of a RDOUT function block provides one instance of the PLC function Read Output Data. The function is invoked when the REQ input is equal to 1.

The ID parameter identifies the slot of the Field Device the output data is read from.

The MLEN parameter specifies the count of bytes which shall be read as an maximum. The byte array given as OUTPUTS parameter shall be at least of MLEN byte.

If the output data are read successfully, the VALID output indicates that the read data are stored in the OUTPUTS parameter. The variable passed to the OUTPUTS parameter shall be of appropriate size to receive the output data. The LEN output contains the length of the read Output data in byte.

If an error occurred, the ERROR output indicates an error and the STATUS output contains the error code. The STATUS values are defined in Table 2.

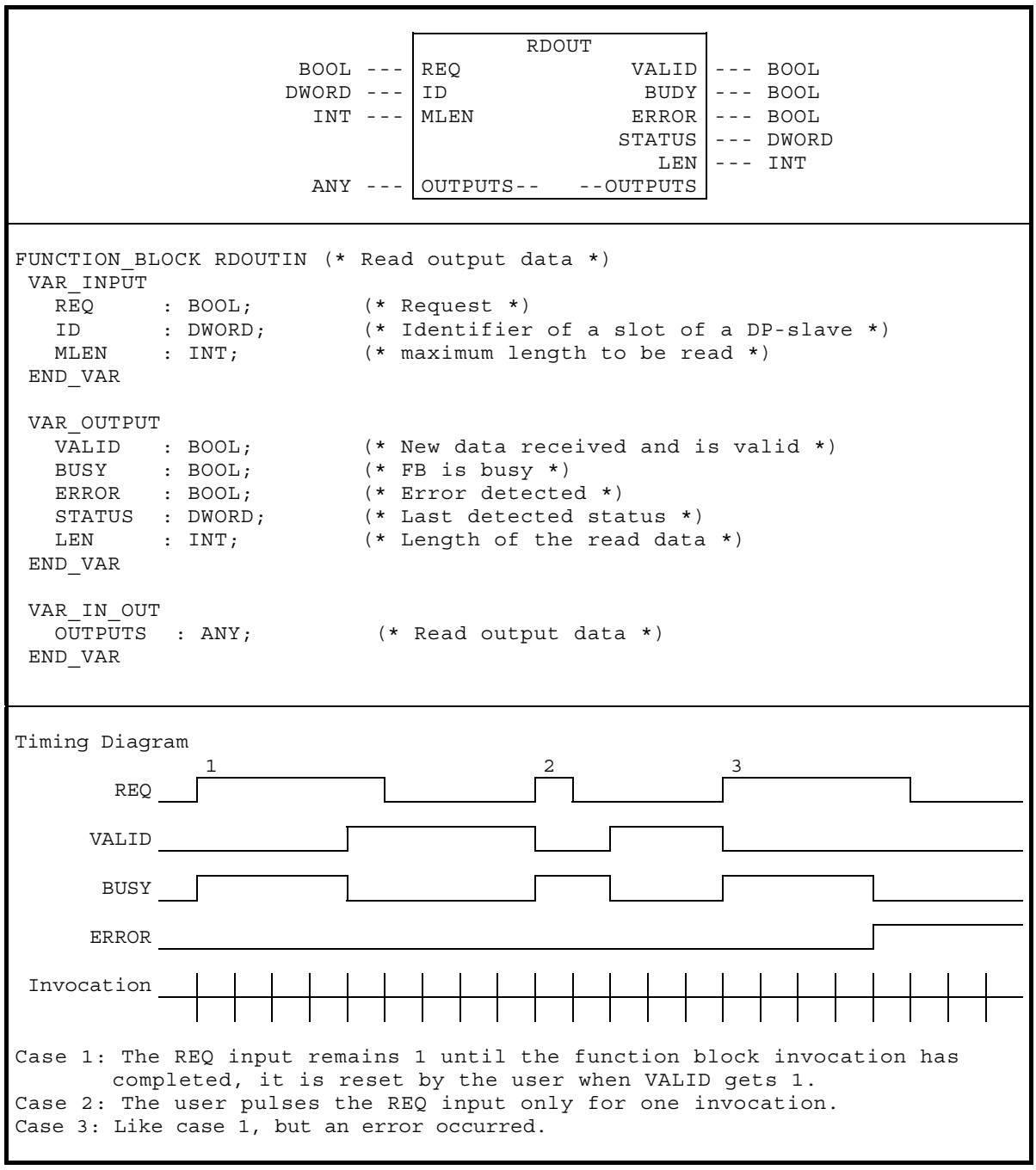


Figure 35 – RDOUT function block

The following state diagram describes the algorithm of the RDOUT function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the RDOUT function block outputs.

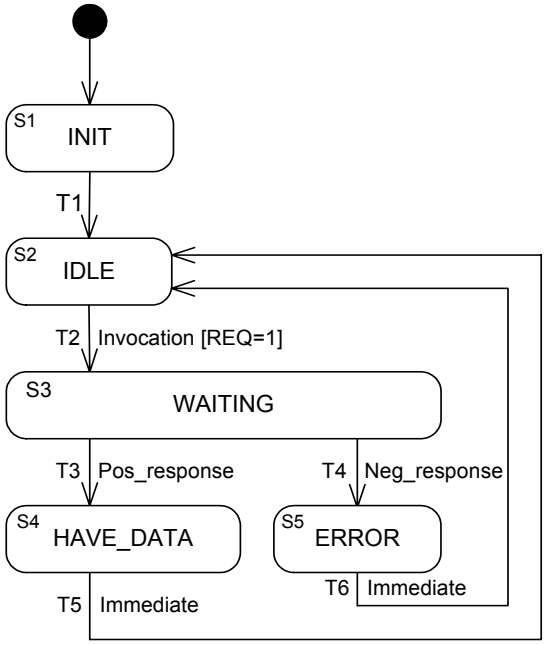


Figure 36 – State diagram of RDOUT function block

The following table defines the transitions and actions given in the state diagram above.

Table 21 - Transitions and actions for RDOUT state diagram

STATE NAME	STATE DESCRIPTION
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 WAITING	Evaluate FB input ID. Request variables from remote communication partner: ReadOutput.Req with AREP= device id out of ID
S4 HAVE_DATA	Deposit data in parameter LEN and IO
S5 ERROR	indicate error

TRAN-SITION	SOURCE STATE	TARGET STATE	EVENT	CONDITION	ACTION
T1	S1	S2		Initialisation done	VALID := 0 BUSY := 0 ERROR := 0 STATUS := 0 IO, LEN := System null
T2	S2	S3	Invocation (Next invocation)	REQ=1	VALID := 0 BUSY := 1 ERROR := 0 STATUS := -1 IO, LEN := ---
T3	S3	S4	Pos_response (Positive response from remote communication partner: ReadOutput.Cnf(+))		VALID := 1 BUSY := 0 ERROR := 0 STATUS := 0 IO, LEN := New data

T4	S3	S5	Neg_response (Negative response from remote communication partner or other communication problems detected: ReadOutput.Cnf(-) or Abort.Ind or local problems)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := New error code IO, LEN := ---
T5	S4	S2	Immediate		VALID := --- BUSY := --- ERROR := --- STATUS := --- IO, LEN := ---
T6	S5	S2	Immediate		VALID := --- BUSY := --- ERROR := --- STATUS := --- IO, LEN := ---
--- indicates “unchanged” FB outputs					

4.3 Exchange of Process Data Records

The function blocks RDREC and WRREC defined in the previous chapters are defined for a PLC acting as a Host Controller (Class 1). These function blocks can also be used for communication as a Supervisor.

4.4 Diagnosis

4.4.1 Read Diagnosis (RDIAG)

This function is provided only for PROFIBUS DP because the service used for the function block is not provided by PROFINET IO. Diagnosis is realized using the alarm services and data records.

The communication function Read Diagnosis for a DP-master (Class 2) uses the RDIAG function block as defined in clause 3.4.3. One instance of a RDIAG function block provides one instance of the PLC function Read Diagnosis.

The ID parameter identifies the slot of the Field Device the diagnosis is read from. A Connection to this Field Device shall be established before.

The following table defines the transitions given in the state diagram defined in clause 3.4.3, if this Communication Function Block is acting in the context of a Supervisor.

Table 22 - Transitions and actions for RDIAG state diagram

STATE NAME	STATE DESCRIPTION
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 WAITING	Evaluate FB inputs ID. Get diagnosis ReadSlaveDiag.Req with AREP= slave id out of ID
S4 HAVE_DATA	Deposit data in parameter LEN and DINFO
S5 ERROR	indicate error

TRAN-SITION	SOURCE STATE	TARGET STATE	EVENT	CONDITION	ACTION
T1	S1	S2		Initialisation done	VALID := 0 LEN, DINFO := System null ERROR, STATUS := 0
T2	S2	S3	Invocation (Next invocation)	REQ=1	VALID := 0 LEN, DINFO := --- ERROR, STATUS := ---
T3	S3	S4	Pos_response (Positive response from DP-master interface: ReadSlaveDiag.Cnf(+))		VALID := 1 LEN, DINFO := New data ERROR, STATUS := 0
T4	S3	S5	Neg_response (Negative response from remote communication partner or other communication problems detected: ReadSlaveDiag.Cnf(-) or other communication problems)		VALID := 0 LEN, DINFO := --- ERROR, STATUS := New error code
T5	S4	S2	Invocation (Next invocation)		VALID := --- LEN, DINFO := --- ERROR, STATUS := ---
T6	S5	S2	Invocation (Next invocation)		VALID := --- LEN, DINFO := --- ERROR, STATUS := ---
--- indicates “unchanged” FB outputs					

4.5 Connection Management (CNCT)

Supervisors need connections to access a Field Device.

NOTE A PLC system may manage connections by local means. In this case the following function blocks for the connection management have no functionality and always indicate success.

The communication function Connection Management for a Supervisor uses the CNCT function block defined in this clause. One instance of a CNCT function block provides one instance of the PLC function Connection Management. A connection shall be established to a Field Device which is connected to an IO system. The function is invoked when the REQ input is equal to 1.

The variable given D_ADDR input shall identify the destination Field Device. The ID parameter identifies the Field Device the connection shall be established to.

NOTE The structure D_ADDR is changed compared with the version 1.20 of this document.

The variable at the D_ADDR input shall be structured as defined in the following tables depending on the used IO Subsystem:

Table 23 - Structure of the variable at D_ADDR input for PROFIBUS DP

Component name	Data type	Meaning
D_TYPE	BYTE	= 1: PROFIBUS DP
D_LEN	BYTE	length of the substructure D
D	STRUCT	Destination address of the Field Device
API	BYTE	AP
SCL	BYTE	Access level
N_ADDR	ARRAY [1..6] OF BYTE	only if D_TYPE=1: Network address
MAC	ARRAY [1..x] OF BYTE	only if D_TYPE=1: MAC address where x = D_LEN-8
	END_STRUCT	
SLOT	BYTE	Slot identification

Table 24 - Structure of the variable at D_ADDR input for PROFINET IO

Component name	Data type	Meaning
D_TYPE	BYTE	= 2: PROFINET IO
D_VERSION	BYTE	= 1: first version of D structure
D	STRUCT	Destination address of the Field Device
STATIONNAME	STRING	Station name
INSTANCE	WORD	Instance ID
DEVICE	WORD	Device ID
VENDOR	WORD	Vendor ID
	END_STRUCT	
API	DWORD	Application Process Identifier
SLOT	WORD	Slot Number (identification of the slot)
SUBSLOT	WORD	Subslot Number (identification of the subslot)

If the station name is given the IP address is don't care.

NOTE A length parameter is not given for the D structure for PROFINET IO because the lengths may differ because of different alignment rules.

The connection is a peer-to-peer connection. One PLC can only establish one connection to the same Field Device.

NOTE The CNCT function block defines a connection to a slot or subslot of a Field Device and returns a handle via the ID output. This connection establishes an Application Relationship (AR) to the Field Device. If another connection and handle is needed to the same Field Device the existing AR may be used.

If the connection is established successfully, the VALID output indicates that the connection can be used.

The connection remains connected as long as the function block is called with REQ=0 or an error is indicated. If a connection is established and the function block is called with REQ input = 0 the connection is disconnected.

If an error occurred, the ERROR output indicates an error and the STATUS output contains the error code. The STATUS values are defined in table 3.

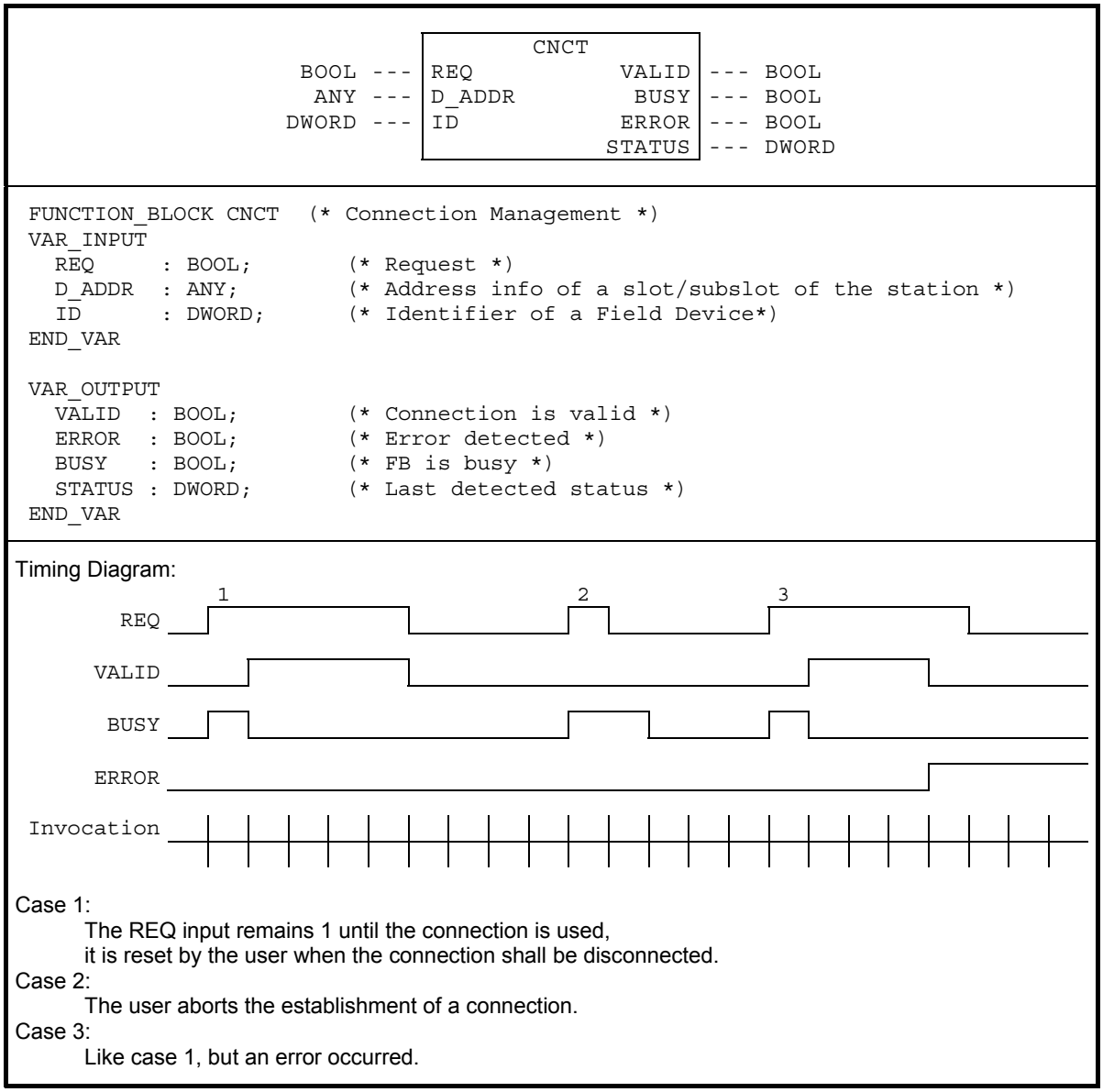


Figure 37 – CNCT function block

The following state diagram describes the algorithm of the CNCT function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the CNCT function block outputs.

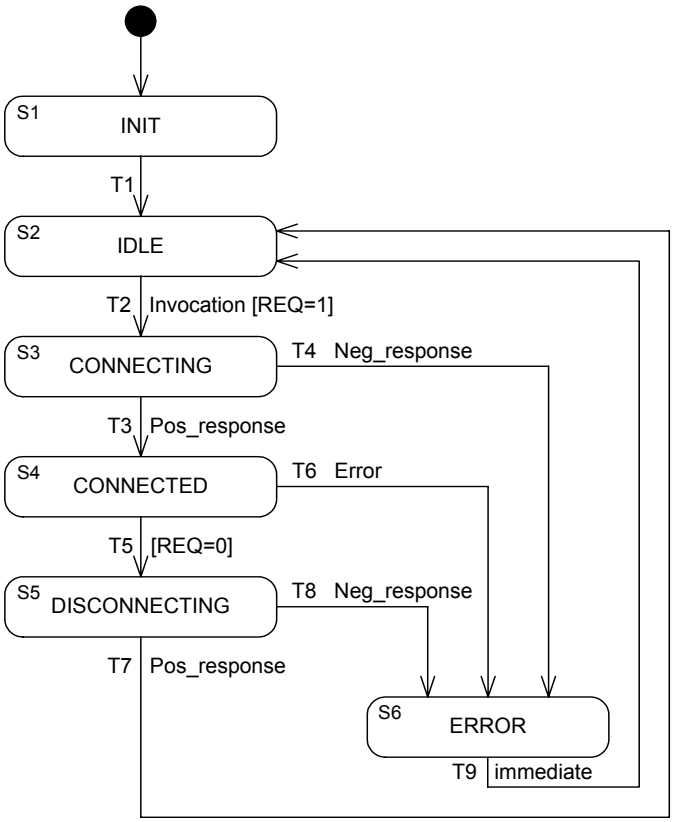


Figure 38 – State diagram of CNCT function block

The following table defines the transitions and actions given in the state diagram above.

Table 25 - Transitions and actions for CNCT state diagram

STATE NAME	STATE DESCRIPTION
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 CONNECTING	Evaluate FB input ID. Request to establish a point-to-point connection to the remote communication partner: Connect.Req with AREP= device id out of ID AddAddrParam.D-Len= D_LEN out of D_ADDR AddAddrParam.D-Addr.D-NetworkAddress = D
S4 CONNECTED	No actions
S5 DISCONNECTING	Evaluate FB input ID. Request to close the connection to the remote communication partner: Disconnect.Req with AREP= device id out of ID
S6 ERROR	indicate error

TRAN-SITION	SOURCE STATE	TARGET STATE	EVENT	CONDITION	ACTION
T1	S1	S2		Initialisation done	VALID := 0 BUSY := 0 ERROR := 0 STATUS := 0
T2	S2	S3	Invocation (Next invocation)	REQ=1	VALID := 0 BUSY := 1 ERROR := 0 STATUS := -1
T3	S3	S4	Pos_response (Positive response from remote communication partner: Connect.Cnf(+)for PROFIBUS DP and for PROFINET IO)		VALID := 1 BUSY := 0 ERROR := 0 STATUS := 0
T4	S3	S6	Neg_response (Negative response from remote communication partner or other communi-cation problems detected: Connect.Cnf(-)for PROFIBUS DP and for PROFINET IO or Abort.Ind or local problems)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := New error code
T5	S4	S5		REQ=0	VALID := 0 BUSY := 1 ERROR := 0 STATUS := 0
T6	S4	S6	Error (Communication problems detected, connection aborted: Abort.Ind or local problems)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := New error code
T7	S5	S2	Pos_response (Positive response from remote communication partner: Disconnect.Cnf(+) for PROFIBUS DP or Re-lease.Cnf(+) for PROFINET IO)		VALID := --- BUSY := --- ERROR := --- STATUS := ---
T8	S5	S6	Neg_response (Negative response from remote communication partner or other communi-cation problems detected: Disconnect.Cnf(-) for PROFIBUS DP or Re-lease.Cnf(+) for PROFINET IO or Abort.Ind or local problems)		VALID := 0 BUSY := 0 ERROR := 1 STATUS := New error code
T9	S6	S2	Immediate		
--- indicates “unchanged” FB outputs					

5 Communication Function Blocks for Field Devices

5.1 Model of a PLC as a Field Device

The Communication Function Blocks defined in the previous clauses are defined for PLC which is acting as a Host Controller or as a Supervisor. A PLC may also be used acting as Field Device. The following Communication Function Blocks define the application interface when the PLC is acting as a Field Device.

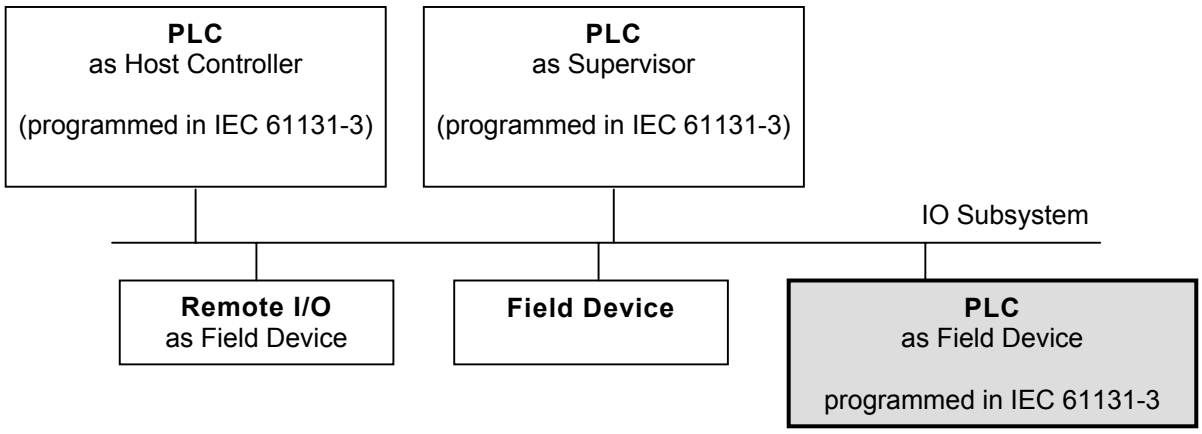


Figure 39 – Profibus system with a PLC as Field Device

A PLC as a Field Device may contain process control functions for one (or for more than one) parts of a plant or machinery. Each of these process control functions are typically implemented by function block instances or function calls using the programming languages of IEC 61131-3. In a Field Device one process control function shall be modelled as one (or more than one) slot or subslot.

It shall be possible, that the application program parts which implement one process control function are programmed independently from each other and from other program parts, e.g. one process control function only knows which slots or subslot it uses, and there shall be no knowledge necessary which slots / subslots are used by other process control functions. The function blocks defined as an application interface for Field Device in this chapter shall support this.

The application program interface to the IO system are Communication Function Blocks. The following function blocks provide this application program interface:

- RCVCO: Receives the (cyclic) output data of a Host Controller
- SBCCI: Subscribe (cyclic) input data of another Field Device
- PRVCI: Provides (publishes) the (cyclic) input data of the Field Device
- RCVREC: Receives a process data record from a Host Controller or a Supervisor
- PRVREC: Receives a request and provides a process data record to a Host Controller or a Supervisor
- SALRM: Request to send an alarm from the Field Device to the Host Controller
- SDIAG: Request to send diagnosis from a DP-slave to a DP-Master

5.2 IO Data Object Interface

5.2.1 General

The output data of a Host Controller to a Field Device are received by a RCVCO function block or may be mapped into the %I area of the application program of the Field Device, e.g. these Host Controller outputs are treated as inputs of the Field Device PLC.

The input data of a Host Controller from this Field Device are provided by a PRVCI function block or may be mapped into the %Q area of the application program of the Field Device, e.g. these Host Controller inputs are treated as outputs of the Field Device PLC.

Output data of another (publishing) Field Device to this Field Device may be subscribed and are received by a SBCCI function block or may be mapped into the %I area of the application program of the subscribing Field Device, e.g. outputs of a publishing Field Device are treated as inputs of the Field Device PLC.

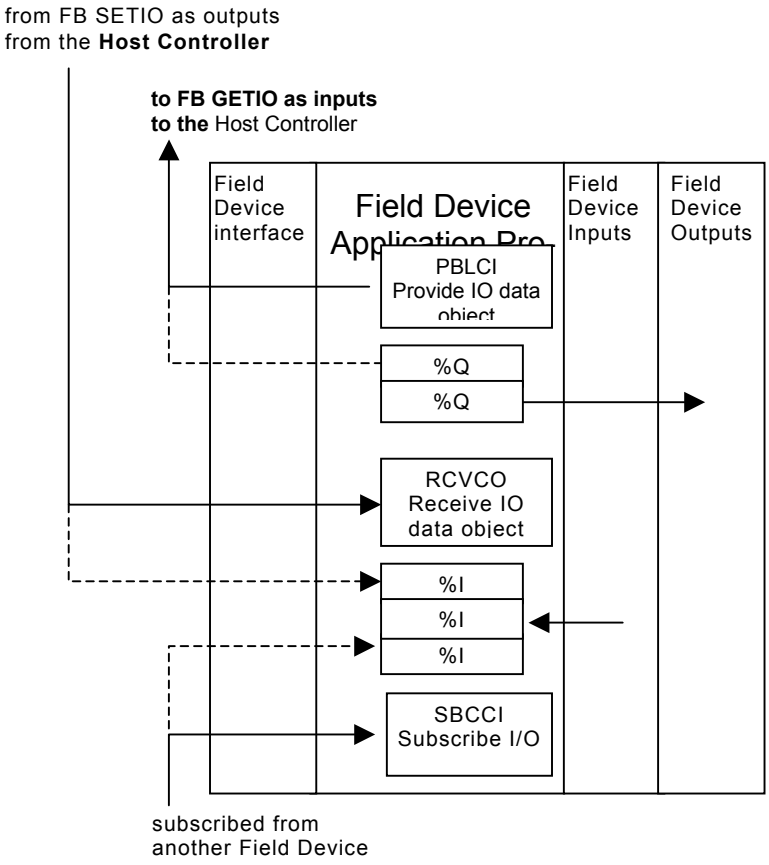


Figure 40 – PLC as a Field Device Using IO data object

Enabling a Communication Function Block for receiving, subscribing and providing IO data object means, that the IO data object are transferred to or from the Field Device interface to the application program of the Field Device CPU. The RCVCO function block gets the output data of the addressed slot from the Host Controller out of the input data interface of the Field Device. The SBCCI function block subscribes and gets input data of the addressed Field Device from the data interface of the other Field Device. The PRVCI function block provides the data of a slot or subslot to the output data interface of the Field Device to the Host Controller as inputs.

NOTE The same output data of the Field Device CPU should not be written by different function block instances or be written via the %Q interface, because which values are transferred to the slave may be unpredictable.

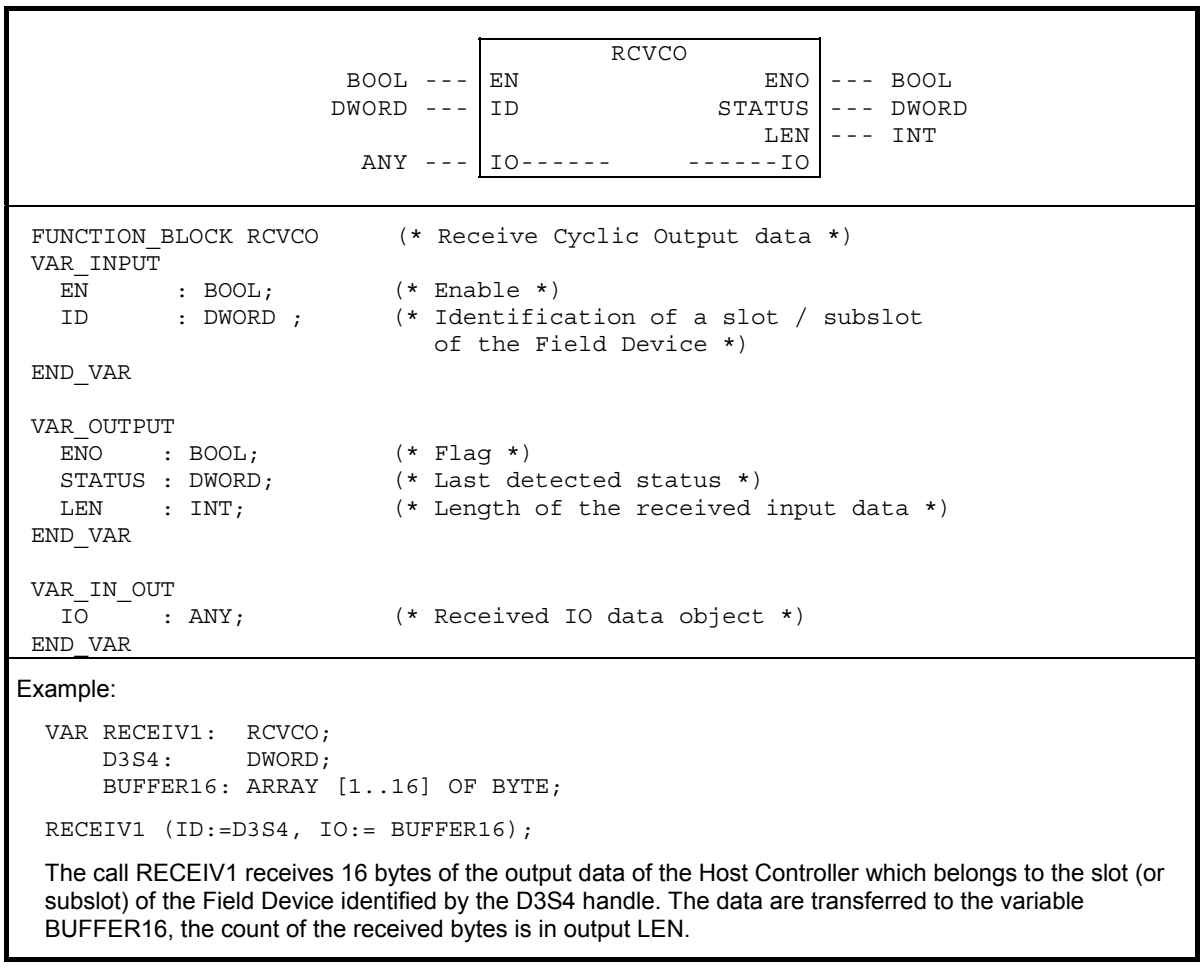
5.2.2 Receive Cyclic Output Data (RCVCO)

The communication function Receive Cyclic Output Data for a Field Device uses the RCVCO function block defined in this clause. One instance of a RCVCO function block provides one instance of the PLC function Receive Cyclic Output Data. The function is invoked by a 1 at the EN input.

The ID parameter identifies the slot of the Field Device the output data shall be received. The data are given by the Host Controller as its cyclic output data for this Field Device.

If the data are received successfully, the ENO output is set to 1 and the received IO data object are stored in the variable at the IO parameter. The variable passed to the IO parameter shall be of appropriate size to receive the diagnosis data. The LEN output contains the length of the received IO data object in byte.

If an error occurred, the ENO output is set to 0 and the STATUS output contains the error code. The STATUS values are defined in Table 2.




```
FUNCTION_BLOCK RCVCO      (* Receive Cyclic Output data *)
VAR_INPUT
  EN      : BOOL;         (* Enable *)
  ID      : DWORD ;       (* Identification of a slot / subslot
                             of the Field Device *)
END_VAR

VAR_OUTPUT
  ENO     : BOOL;         (* Flag *)
  STATUS  : DWORD;        (* Last detected status *)
  LEN     : INT;          (* Length of the received input data *)
END_VAR

VAR_IN_OUT
  IO      : ANY;          (* Received IO data object *)
END_VAR
```

Example:

```
VAR RECEIV1:  RCVCO;
    D3S4:     DWORD;
    BUFFER16: ARRAY [1..16] OF BYTE;

RECEIV1 (ID:=D3S4, IO:= BUFFER16);
```

The call RECEIV1 receives 16 bytes of the output data of the Host Controller which belongs to the slot (or subslot) of the Field Device identified by the D3S4 handle. The data are transferred to the variable BUFFER16, the count of the received bytes is in output LEN.

Figure 41 – RCVCO function block

NOTE The EN and ENO parameters are optional and may be omitted when calling an instance of the function block in textual languages.

The following state diagram describes the algorithm of the RCVCO function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the RCVCO function block outputs.

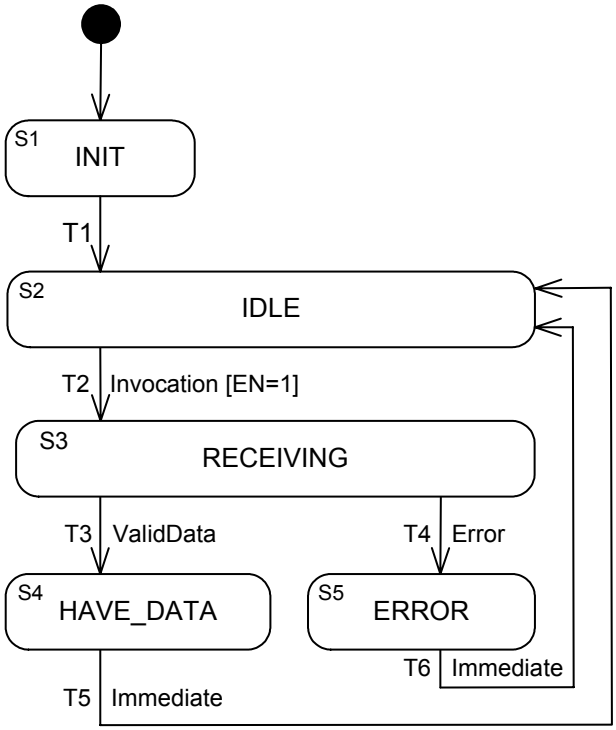


Figure 42 – State diagram of RCVCO function block

The following table defines the transitions and actions given in the state diagram above.

Table 26 – Transitions and actions for RCVCO state diagram

STATE NAME		**STATE DESCRIPTION**			
S1 INIT		cold start state, initialise outputs			
S2 IDLE		idle state, No actions			
S3 RECEIVING		Evaluate FB input SLOT. Get IO data object for slot from DP-slave interface			
S4 HAVE_ DATA		Deposit received IO data object of the slot in IO output and set LEN output			
S5 ERROR		indicate error set FB outputs			
TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	EN0 := 0 STATUS := 0 IO, LEN := 0
T2	S2	S3	Invocation (Next invocation)	EN=1	EN0 := --- STATUS := --- IO, LEN := ---
T3	S3	S4	ValidData (Next invocation of this instance and valid IO data object)		EN0 := 1 STATUS := 0 IO, LEN := New data
T4	S3	S5	Error (Next invocation of this instance and no valid IO data object)		EN0 := 0 STATUS := New error code IO, LEN := ---
T5	S4	S2	Immediate		EN0 := --- STATUS := --- IO, LEN := ---

T6	S5	S2	Immediate		EN0 := --- STATUS := --- IO, LEN := ---
--- indicates “unchanged” FB outputs					

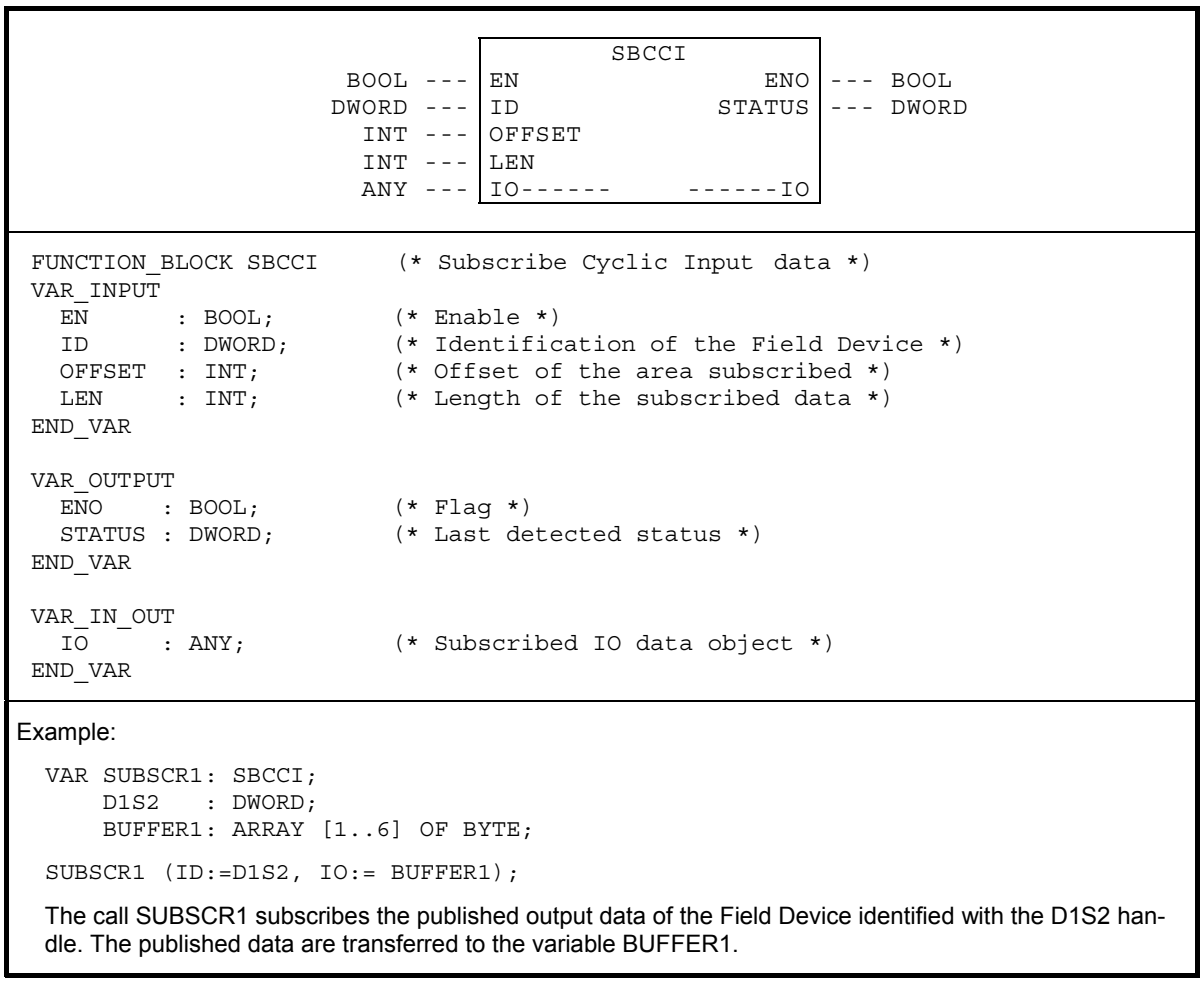
5.2.3 Subscribe Cyclic Input Data (SBCCI)

The communication function Subscribe Cyclic Input Data for a Field Device uses the SBCCI function block defined in this clause. One instance of a SBCCI function block provides one instance of the PLC function Subscribe Cyclic Input Data. The function is invoked by a 1 of the EN input.

This Communication Function Block subscribes and gets the input data of the slot of the Field Device identified with the ID input. The OFFSET and LEN inputs specify the data area inside the cyclic input data the other Field Device sends to the Host Controller and are subscribed by this Field Device using the SBCCI function block. The OFFSET input counts from 0.

If the data are received successfully, the ENO output is set to 1 and the subscribed IO data object are stored in the variable given at the IO parameter. The variable passed to the IO parameter shall be of appropriate size to receive the data.

If an error occurred, the ENO output is set to 0 and the STATUS output contains the error code. The STATUS values are defined in table 3.




```
FUNCTION_BLOCK SBCCI      (* Subscribe Cyclic Input data *)
VAR_INPUT
  EN      : BOOL;         (* Enable *)
  ID      : DWORD;        (* Identification of the Field Device *)
  OFFSET  : INT;          (* Offset of the area subscribed *)
  LEN     : INT;          (* Length of the subscribed data *)
END_VAR

VAR_OUTPUT
  ENO     : BOOL;         (* Flag *)
  STATUS : DWORD;         (* Last detected status *)
END_VAR

VAR_IN_OUT
  IO      : ANY;          (* Subscribed IO data object *)
END_VAR
```

Example:

```
VAR SUBSCR1: SBCCI;
    D1S2   : DWORD;
    BUFFER1: ARRAY [1..6] OF BYTE;

SUBSCR1 (ID:=D1S2, IO:= BUFFER1);
```

The call SUBSCR1 subscribes the published output data of the Field Device identified with the D1S2 handle. The published data are transferred to the variable BUFFER1.

Figure 43 – SBCCI function block

NOTE The EN and ENO parameters are optional and may be omitted when calling an instance of the function block in textual languages.

The following state diagram describes the algorithm of the SBCCI function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the SBCCI function block outputs.

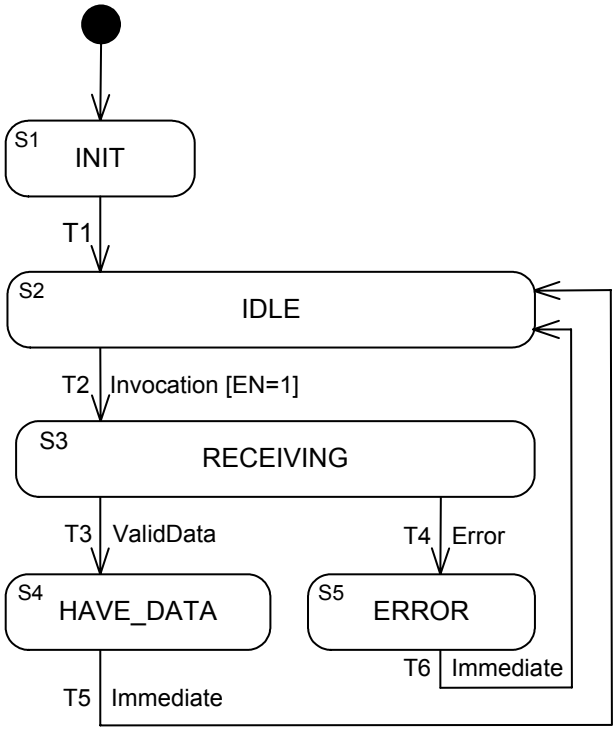


Figure 44 – State diagram of SBCCI function block

The following table defines the transitions and actions given in the state diagram above.

Table 27 - Transitions and actions for SBCCI state diagram

STATE NAME		**STATE DESCRIPTION**			
S1 INIT		cold start state, initialise outputs			
S2 IDLE		idle state, No actions			
S3 RECEIVING		Evaluate FB input ID. Get subscribed IO data object from DPSlave interface			
S4 HAVE_ DATA		Deposit received IO data object of the publisher in IO parameter, set LEN output			
S5 ERROR		indicate error set FB outputs			
TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	EN0 := 0 STATUS := 0 IO := System null
T2	S2	S3	Invocation (Next invocation)	EN=1	EN0 := --- STATUS := --- IO := ---
T3	S3	S4	ValidData (Valid IO data object)		EN0 := 1 STATUS := 0 IO := New data

T4	S3	S5	Error (No valid IO data object)		EN0 := 0 STATUS := New error code IO := ---
T5	S4	S2	Immediate		EN0 := --- STATUS := --- IO := ---
T6	S5	S2	Immediate		EN0 := --- STATUS := --- IO := ---
--- indicates “unchanged” FB outputs					

5.2.4 Provide Cyclic Input Data (PRVCI)

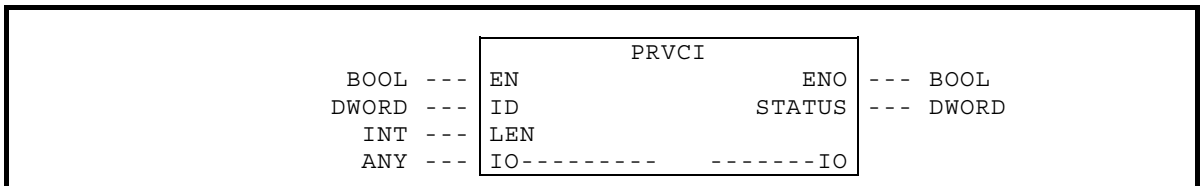
The communication function Provide Cyclic Input Data of a Field Device uses the PRVCI function block defined in this clause. One instance of a PRVCI function block provides one instance of the PLC function Provide Cyclic Data. The function is invoked by a 1 of the EN input.

The ID parameter identifies the slot of the slave the IO data object is provided for. The variable given at the IO input shall contain the IO data object that shall be provided as the input data of the slot of the Field Device to the Host Controller. The variable passed to the IO parameter shall be of appropriate size to contain the output data. The LEN input contains the length of the IO data in byte.

If the LEN input is set to 0, no valid data are available. For PROFINET IO the IOPS shall be set to bad data detected by subslot.

If the IO data object are provides successfully, the ENO output is set to 1.

If an error occurred, the ENO output is set to 0 and the STATUS output contains the error code. The STATUS values are defined in Table 2.




```
FUNCTION_BLOCK PRVCI  (* Provide Cyclic Input data *)
 VAR_INPUT
   EN      : BOOL;         (* Enable *)
   ID      : INT;          (* Identifier of a slot/subslot of the Field Device *)
   LEN     : INT;          (* Length of the IO data object *)
END_VAR

 VAR_OUTPUT
   ENO    : BOOL;          (* Flag *)
   STATUS : DWORD;         (* Last detected status *)
 END_VAR

 VAR_IN_OUT
   IO     : ANY;           (* IO data object to provide *)
 END_VAR
```

Figure 45 – PRVCI function block

NOTE The EN and ENO parameters are optional and may be omitted when calling an instance of the function block in textual languages.

The following state diagram describes the algorithm of the PRVCI function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the PRVCI function block outputs.

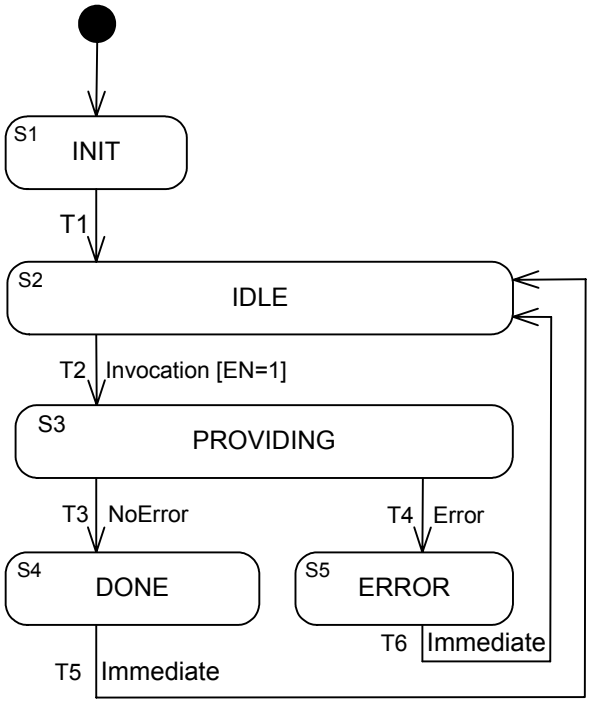


Figure 46 – State diagram of PRVCI function block

The following table defines the transitions and actions given in the state diagram above.

Table 28 - Transitions and actions for PRVCI state diagram

STATE NAME		**STATE DESCRIPTION**			
S1 INIT		cold start state, initialise outputs			
S2 IDLE		idle state, No actions			
S3 PROVIDING		Evaluate FB input ID. Transfer IO data object to DP-slave interface as output data of the slot			
S4 DONE		No actions			
S5 ERROR		indicate error			
TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	EN0 := 0 STATUS := 0
T2	S2	S3	Invocation (Next invocation)	EN=1	EN0 := --- STATUS := ---
T3	S3	S4	NoError (No communication problems detected)		EN0 := 1 STATUS := 0
T4	S3	S5	Error (Communication problems detected)		EN0 := 0 STATUS := New error code
T5	S4	S2	Invocation (Next invocation)		EN0 := --- STATUS := ---

T6	S5	S2	Invocation (Next invocation)		EN0 := --- STATUS := ---
--- indicates “unchanged” FB outputs					

5.3 Process Data Record Interface

5.3.1 General

A Field Device can receive a process data record from a Host Controller or a Supervisor. The Host Controller or Supervisor may use (if its a PLC) the WRREC function block. The PLC application program is informed about this using the RCVREC function block and can process the process data record.

A Field Device can receive a request to provide a process data record to a Host Controller or Supervisor. The Host Controller or Supervisor may use (if its a PLC) the RDREC function block. The PLC application program is informed about this request and can provide the requested process data record using the PRVREC function block.

5.3.2 Receive Process Data Record (RCVREC)

The communication function Receive Process Data Record for a Field Device uses the RCVREC function block defined in this clause. One instance of a RCVREC function block provides one instance of the PLC function Receive Process Data Record.

The function is invoked by EN=1. The MODE input controls the functionality of the RCVREC function block.

MODE	Meaning
0	Check for request: If the Field Device interface has received a process data record, only the outputs NEW, SLOT, INDEX and RLEN are set. Multiple calls of this function block with MODE=0 returns the outputs for the same request.
1	Receive all process data records: If the Field Device interface has received a process data record, the function block outputs are updated and the data record is transferred to the RECORD parameter. The service is responded positively.
2	Receive process data records for one slot or subslot: If the Field Device interface has received a process data record for the slot or subslot the numbers of which are given in input F_ID, the function block outputs are updated and the data record is transferred to the RECORD parameter. The service is responded positively.
4	Negative response: After checking the request to receive a process data record, this function block refuses to accept this record and sends a negative response to the Host Controller. The error reason is given with the inputs CODE1 and CODE2.

NOTE 1 This function blocks contains the methods to check, receive and acknowledge a process data record. All aspects of receiving a process data record may use one function block instance, the different methods are distinguished using the MODE input.

For PROFIBUS DP the subslot number shall always contain 0.

The MLEN parameter specifies the count of bytes which shall be received as a maximum. The byte array given as RECORD parameter shall be at least of MLEN byte.

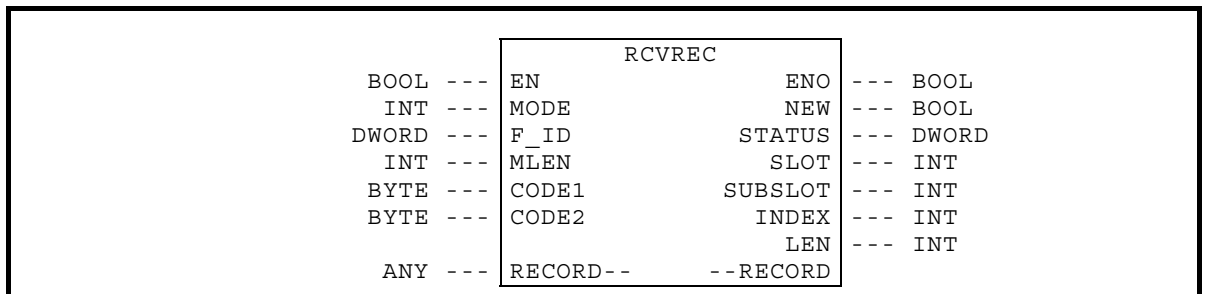
NOTE 2 An array declaration with zero elements is not supported in IEC 61131-3, therefore the minimum length shall be 1 byte even if the record length is zero. The actual length is given with the LEN parameter.

If a data record is received (with MODE=1 or MODE=2), the NEW output indicates that the data record is stored in the variable given at the RECORD parameter. The variable passed to the RECORD parameter shall be of appropriate size to receive the process data record. The LEN output contains the length of the data record in byte.

If the function block refuses to accept the data record, the CODE1 input sets the Error Code 1, and the CODE2 input sets the Error Code 2 of the negative response.

NOTE 3 The application program of the Field Device shall acknowledge the received request, otherwise the Host Controller will get a timeout error and will deactivate the DP interface of the Field Device.

If an error occurred, the ENO=0 indicates an error and the STATUS output contains the error code. The STATUS values are defined in Table 2.




```
FUNCTION_BLOCK RCVREC  (* Receive process data record *)
VAR_INPUT
  EN      : BOOL;     (* Enable *)
  MODE    : INT;      (* Function specifier *)
  F_ID    : DWORD;    (* Slot / subslot to filter *)
                         (* the process data records to receive *)
  MLEN    : INT;      (* Maximum length of a data record to receive *)
  CODE1   : BYTE;     (* Reason for negative response *)
  CODE2   : BYTE;     (* Reason for negative response *)
END_VAR

VAR_OUTPUT
  ENO     : BOOL;     (* Function enabled *)
  NEW     : BOOL;     (* New data record received *)
  STATUS  : DWORD;    (* Field Device interface status *)
  SLOT    : INT;      (* Slot the record is received for *)
  SUBSLOT : INT;      (* Subslot the record is received for *)
  INDEX   : BOOL;     (* Index of the received process data record *)
  LEN     : INT;      (* Length of the received data record *)
END_VAR

VAR_IN_OUT
  RECORD : ANY;       (* Received data record *)
END_VAR
```


Example 1: Receive a process data record for all slots / subslots of the Field Device

```
VAR R1: ARRAY [1..240] OF BYTE;
    RREC1: RCVREC;

RREC1 (MODE:=1, MLEN:=240, RECORD:=R1);
IF NEW=1 THEN (* process record, the index is in RREC1.INDEX *)
..
```

Example 2: Check the received process data records, process and acknowledge conditionally

```
VAR R2: ARRAY [1..240] OF BYTE;
    RREC2: RCVREC;

RREC2 (MODE:=0); (* Check for a new record *)
IF NEW=1 THEN (* new process data record available *)
   IF RREC2.SLOT=12 and RREC2.INDEX=1 THEN
      RREC2 (MODE:=1, RECORD:=R2); (* get this record *)
         (* process data record for slot 12 and index 1 *)
   ELSEIF RREC2.SLOT=13 and RREC2.INDEX=2 THEN
      RREC2 (MODE:=1, RECORD:=R2); (* get this record *)
         (* process data record for slot 13 and index 2 *)
   :
   ELSE RREC2 (MODE:=4); (* give negative response *)
   END_IF;
```

Figure 47 – RCVREC function block

NOTE The EN and ENO parameters are optional and may be omitted when calling an instance of the function block in textual languages.

The following state diagram describes the algorithm of the RCVREC function block. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the RCVREC function block outputs.

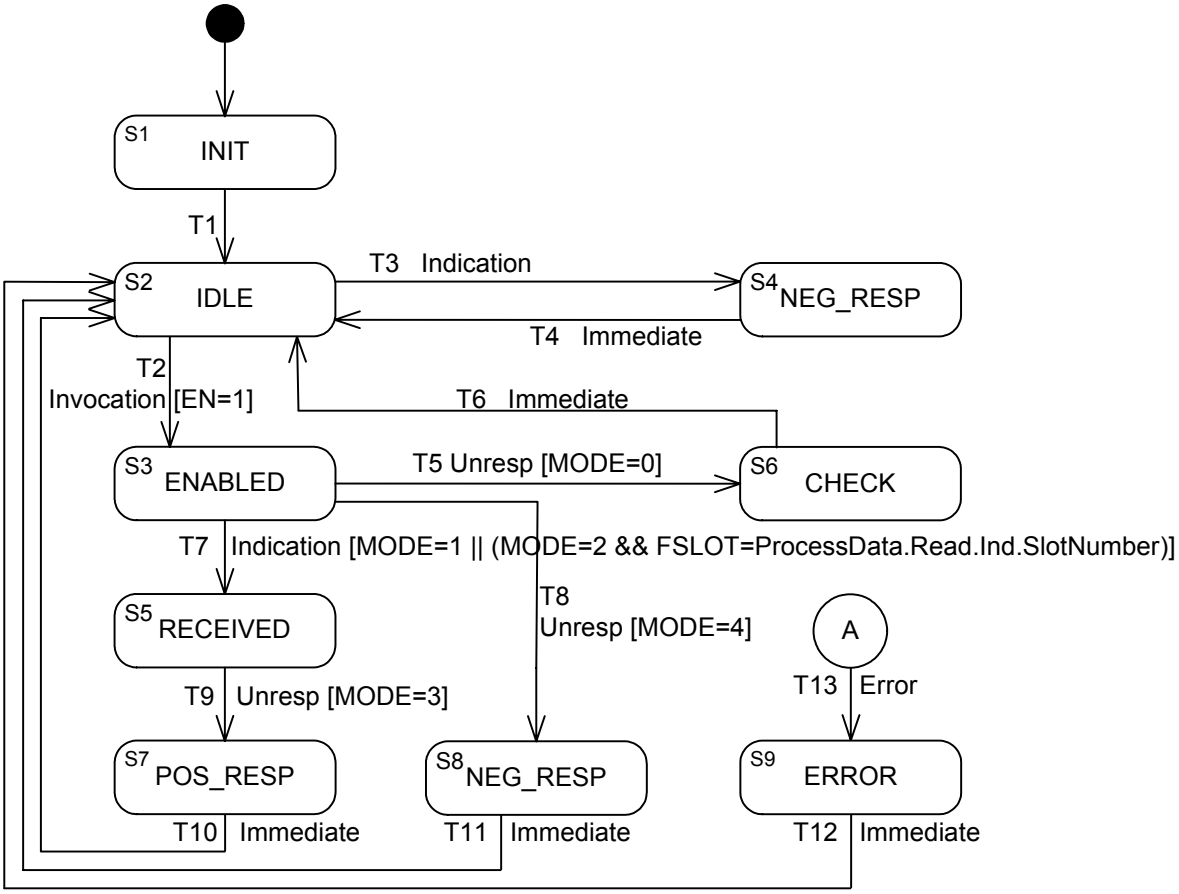


Figure 48 – State diagram of RCVREC function block

The ERROR state may be entered by the states ENABLED, CHECK, RECEIVED, POS_ACK or NEG_ACK if a communication error is detected.

The following table defines the transitions and actions given in the state diagram above.

Table 29 - Transitions and actions for RCVREC state diagram

STATE NAME		**STATE DESCRIPTION**			
S1	INIT	cold start state, initialise outputs			
S2	IDLE	idle state, No actions			
S3	ENABLE	No actions			
S4 / S8	NEG_ACK	Negative response to DP-Master: ProcessData.Write.rsp(-)with Error Decode= 16#80 Error Code 1= CODE1 Error Code 2= CODE2			
S5	RECEIVED	Deposit data in parameter SLOT, INDEX, LEN and RECORD			
S6	CHECK	Update outputs			
S7	POS_ACK	Positive response to DP-Master: ProcessData.Write.rsp(+) with Length= MIN(MLEN,LEN)			
S8	ERROR	indicate error			
TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**

T1	S1	S2		Initialisation done	ENO := 0 NEW := 0 SLOT, INDEX, LEN := System null RECORD := --- STATUS := 0
T2	S2	S3	Invocation (Next invocation)	EN=1	ENO := 1 NEW := 0 SLOT, INDEX, LEN := 0 RECORD := --- STATUS := ---
T3	S2	S4	Unack_ind (Unacknowledged indication from Host Controller or Supervisor: ProcessData.Write.Ind for PROFIBUS DP or Write.Ind for PROFINET IO)		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, LEN := --- RECORD := --- STATUS := ---
T4	S4	S2	Immediate		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, LEN := --- RECORD := --- STATUS := ---
T5	S3	S6	Unack_ind (Unacknowledged indication from DP-master (Class 1) or (Class 2))	MODE=0	ENO := NEW := SLOT, SUBSLOT, INDEX, LEN := New data RECORD := --- STATUS := ---
T6	S6	S2	Immediate		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, LEN := --- RECORD := --- STATUS := ---
T7	S3	S5	Indication (Indication from Host Controller or Supervisor: ProcessData.Write.Ind for PROFIBUS DP or Write.Ind for PROFINET IO)	MODE=1 or (MODE=2 and FSLOT= ProcessData.Write.Ind.SlotNumber)	ENO := 1 NEW := 1 SLOT, SUBSLOT, INDEX, LEN := New data RECORD := New record STATUS := ---
T8	S3	S8	Unack_ind (Unacknowledged indication from DP-master (Class 1) or (Class 2))	MODE=4	ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, LEN := --- RECORD := --- STATUS := ---
T9	S5	S7	Immediate		ENO := 0 NEW := --- SLOT, SUBSLOT, INDEX, LEN := --- RECORD := --- STATUS := 0
T10	S7	S2	Immediate		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, LEN := 0 RECORD := --- STATUS := ---

T11	S8	S2	Immediate		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, LEN := 0 RECORD := --- STATUS := ---
T12	S9	S2	Immediate		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, LEN := 0 RECORD := --- STATUS := ---
T13	S3, S4, S5, S6, S7, S9	S9	Error (Communikation error detected)		ENO := 0 NEW := 0 SLOT, SUBSLOT, INDEX, LEN := --- RECORD := --- STATUS := New status
--- indicates “unchanged” FB outputs					

5.3.3 Provide Process Data Record (PRVREC)

The communication function Provide Process Data Record for a Field Device uses the PRVREC function block shown in figure below.

One instance of a PRVREC function block provides one instance of the PLC function Provide Process Data Record.

The function is invoked by EN=1. The MODE input controls the functionality of the PRVREC function block.

MODE	Meaning
0	Check for request: If the Field Device interface has received a request to provide a process data record, only the outputs NEW, SLOT, INDEX and RLEN are set. Multiple calls of this function block with MODE=0 returns the outputs for the same request.
1	Receive all requests: If the Field Device interface has received a request, the function block outputs are updated.
2	Receive requests for one slot or subslot: If the Field Device interface has received a request for the slot and subslot the numbers of which are given in F_ID input, the function block outputs are updated.
3	Positive response: After checking or receiving the request to provide a process data record, this function block provides the requested process data record with its RECORD parameter and sends a positive response to the Host Controller.
4	Negative response: After checking or receiving the request to provide a process data record, this function block refuses to provide this record and sends a negative response to the Host Controller. The error reason is given with the inputs CODE1 and CODE2.

NOTE 1 This function blocks contains the methods to check, receive and respond a request for a process data record. All aspects of providing a process data record may use one function block instance, the different methods are distinguished using the MODE input.

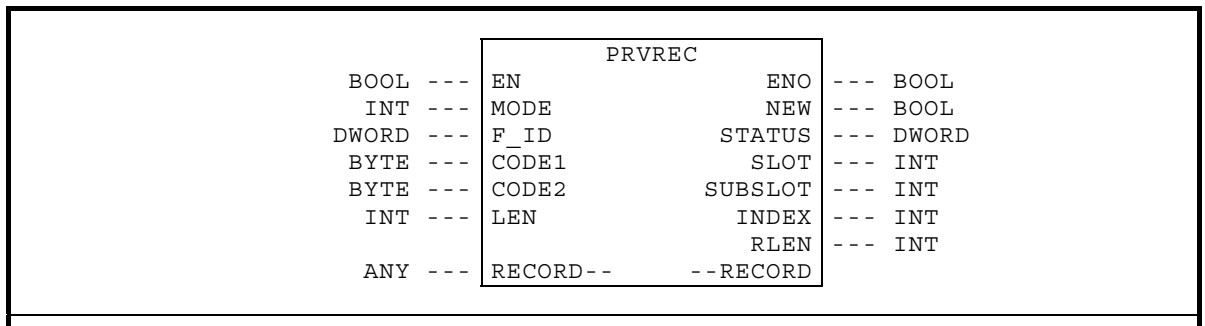
The MLEN parameter specifies the count of bytes which shall be provided as a maximum. The byte array given as RECORD parameter shall be at least of MLEN byte.

NOTE 2 An array declaration with zero elements is not supported in IEC 61131-3, therefor the minimum length shall be 1 byte even if the record length is zero. The actual length is given with the LEN parameter.

If a data record is received (with MODE=1 or MODE=2), the NEW output indicates that the data record is stored in the variable given at the RECORD parameter. The variable passed to the RECORD parameter shall be of appropriate size to contain the process data record. The LEN output contains the length of the data record in byte.

If the function block refuses to accept the data record, the CODE1 input sets the Error Code1, and the CODE2 input sets the Error Code 2 of the negative response.

If an error occurred, the ENO=0 indicates an error and the STATUS output contains the error code. The STATUS values are defined in table 3.




```
FUNCTION_BLOCK PRVREC (* Provide process data record *)
VAR_INPUT
  EN      : BOOL;     (* Enable *)
  MODE    : INT;      (* Function specifier *)
  F_ID    : DWORD;    (* Slot / subslot to filter the requests *)
                         (* to provide process data records *)
  CODE1   : BYTE;     (* Reason for negative response *)
  CODE2   : BYTE;     (* Reason for negative response *)
  LEN     : INT;      (* Length of a data record to provide *)
END_VAR

VAR_OUTPUT
  ENO     : BOOL;     (* Function enabled *)
  NEW     : BOOL;     (* New data record requested *)
  STATUS  : DWORD;    (* Field Device interface status *)
  SLOT    : INT;      (* Slot the record is requested for *)
  SUBSLOT : INT;      (* Subslot the record is requested for *)
  INDEX   : INT;      (* Index of the requested process data record *)
  RLEN    : INT;      (* Length of the requested data record *)
END_VAR

VAR_IN_OUT
  RECORD : ANY;       (* Provided data record *)
END_VAR
```


Example 1: Provide a process data record

```
VAR P1: ARRAY [1..240] OF BYTE;
    L1: INT;
    PREC1: PRVREC;

PREC1 (MODE:=1);
IF NEW=1 THEN
  (* provide record, the slot number is in PREC1.SLOT,
     the index is in PREC1.INDEX, store the record in P1
     and its length in L1 *)
  PREC1 (MODE:=3, LEN:=L1; RECORD:=P1);
..
```

Example 2: Check the received requests, process and acknowledge conditionally

```
VAR P2: ARRAY [1..240] OF BYTE;
    L2: INT;
    PREC2: PRVREC;

PREC2 (MODE:=0);            (* check for a new request *)
IF NEW=1 THEN (* new request for a process data record available *)
   IF PREC2.SLOT=12 and PREC2.INDEX=1 THEN
     (* build record for slot 12 and index 1 and store in P2 *)
      PREC2 (MODE:=3, RECORD:=P2);     (* provide this record *)

   ELSEIF PREC2.SLOT=13 and PREC2.INDEX=2 THEN
     (* build record for slot 13 and index 2 and store in P2 *)
      PREC2 (MODE:=3, RECORD:=R2);     (* get this record *)
   :
   ELSE PREC2 (MODE:=4);               (* give negative response *)
   END_IF;
```

Figure 49 – PRVREC function block

NOTE The EN and ENO parameters are optional and may be omitted when calling an instance of the function block in textual languages.

The following state diagram describes the algorithm of the PRVREC function blocks. The following tables describe the transitions of this state diagram and the actions to be performed within the states and the settings of the PRVREC function block outputs.

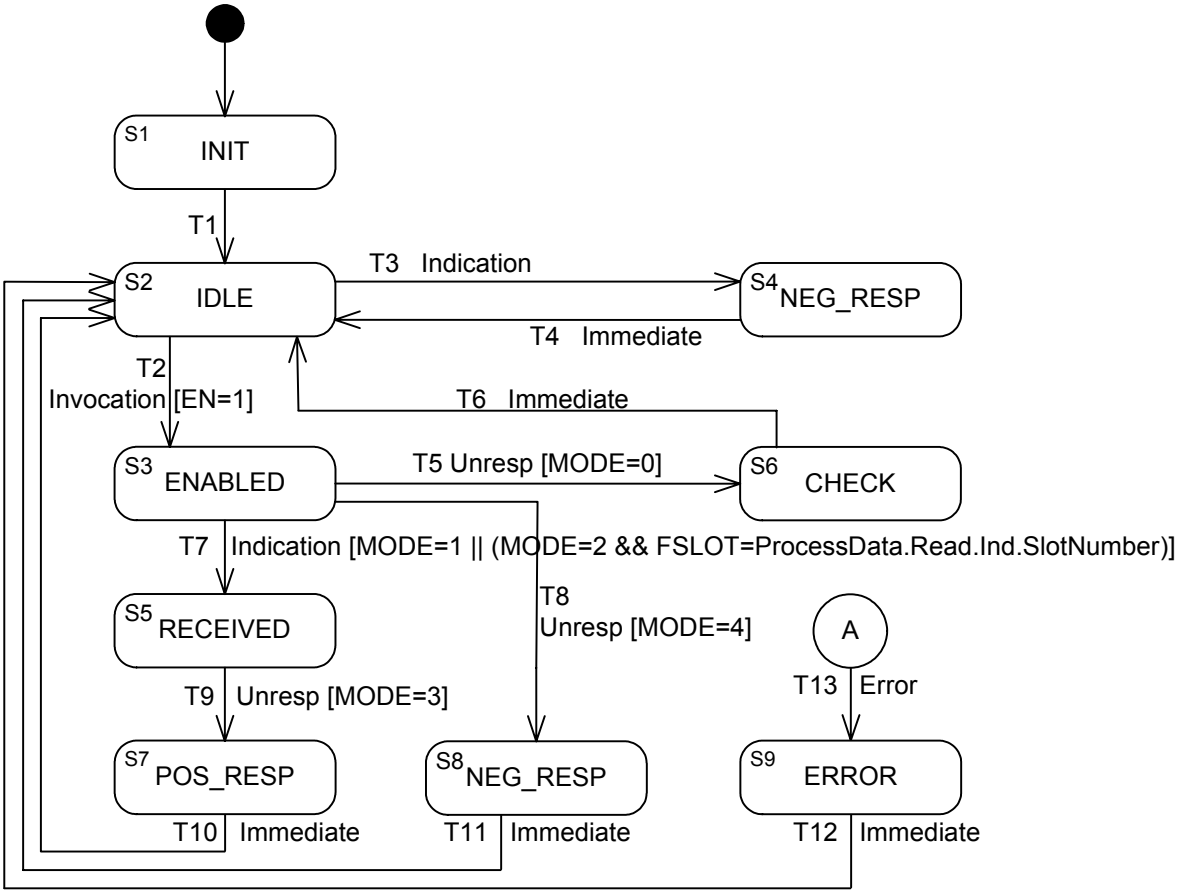


Figure 50 – State diagram of PRVREC function block

The ERROR state may be entered by the states ENABLED, CHECK, RECEIVED, POS_RESP or NEG_RESP if a communication error is detected.

The following table defines the transitions and actions given in the state diagram above.

Table 30 - Transitions and actions for PRVREC state diagram

STATE NAME	STATE DESCRIPTION
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 ENABLE	No actions
S4 / S8 NEG_RESP	Negative response to DP-Master: ProcessData.Read.rsp(-) with Error Decode= 16#80 Error Code 1=CODE1 Error Code 2= CODE2
S5 RECEIVED	Deposit data in parameter SLOT, INDEX, and RLEN
S6 CHECK	No actions
S7 POS_ACK	Positive response to DP-Master: ProcessData.Read.rsp(+) with Length= LEN Data= RECORD
S8 ERROR	Update status

TRAN-SITION	SOURCE STATE	TARGET STATE	EVENT	CONDITION	ACTION
T1	S1	S2		Initialisation done	ENO := 0 NEW := 0 SLOT, SUBSLOT, INDEX, RLEN := System null STATUS := ---
T2	S2	S3	Invocation (Next invocation)	EN=1	ENO := 1 NEW := --- SLOT, SUBSLOT, INDEX, RLEN := --- STATUS := ---
T3	S2	S4	Indication (Indication from Host Controller or Supervisor: ProcessData.Read.Ind for PROFIBUS DP or Read.Ind for PROFINET IO)		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, RLEN := --- STATUS := ---
T4	S4	S2	Immediate		ENO := 1 NEW := --- SLOT, SUBSLOT, INDEX, RLEN := --- STATUS := ---
T5	S3	S6	Unresp (Unresponded indication from Host Controller or Supervisor: ProcessData.Read.Ind for PROFIBUS DP or Read.Ind for PROFINET IO)	MODE=0	ENO := 1 NEW := 1 SLOT, INDEX, RLEN := New data STATUS := ---
T6	S6	S2	Immediate		ENO := 1 NEW := --- SLOT, SUBSLOT, INDEX, RLEN := --- STATUS := ---
T7	S3	S5	Indication (Indication from Host Controller or Supervisor: ProcessData.Read.Ind for PROFIBUS DP or Read.Ind for PROFINET IO)	MODE=1 or (MODE=2 and FSLOT= ProcessData.Read.Ind.SlotNumber)	ENO := 1 NEW := 1 SLOT, SUBSLOT, INDEX, RLEN := New data STATUS := ---
T8	S3	S8	Unresp (Unresponded indication from Host Controller or Supervisor: ProcessData.Read.Ind for PROFIBUS DP or Read.Ind for PROFINET IO)	MODE=4	ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, RLEN := --- STATUS := ---
T9	S5	S7	Unresp (Unresponded indication from Host Controller or Supervisor: ProcessData.Read.Ind for PROFIBUS DP or Read.Ind for PROFINET IO)	MODE=3	ENO := 1 NEW := --- SLOT, SUBSLOT, INDEX, RLEN := --- RECORD := --- STATUS := ---
T10	S7	S2	Immediate		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, LEN := 0 STATUS := ---

T11	S8	S2	Immediate		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, RLEN := 0 STATUS := ---
T12	S9	S2	Immediate		ENO := 1 NEW := 0 SLOT, SUBSLOT, INDEX, RLEN := 0 STATUS := ---
T13	S3, S4, S5, S6, S7, S9	S9	Error		ENO := 0 NEW := 0 SLOT, SUBSLOT, INDEX, RLEN := --- STATUS := New status
--- indicates “unchanged” FB outputs					

5.4 Alarm Handling and Diagnosis

A Field Device can generate alarms to its associated Host Controller to inform it e.g. about certain process events or other events to state the some limitations of the capabilities of the Field Device for diagnostic reasons.

The Field Device shall inform its Host Controller when it is in a state which prevents it to perform the intended process control function. A Supervisor shall be able to read this information. The PLC application program of the Field Device shall provide adequate diagnosis information when it recognises such a state.

5.4.1 Send Alarm (SALRM)

The communication function Send Alarm for a Field Device uses the SALRM function block defined in this clause. One instance of a SALRM function block provides one instance of the PLC function Send Alarm. The function is invoked when the REQ input is equal to 1.

The ID parameter identifies the slot of the Field Device the alarm is generated for. The ATYPE input shall contain the alarm type. The ASPEC input shall contain the alarm specifier. The LEN input contains the length in byte of the additional alarm information stored in the AINFO parameter.

The Variable given as AINFO parameter shall be at least of LEN byte.

If the alarm is transmitted successfully, the VALID output indicates that the alarm was received by the Host Controller.

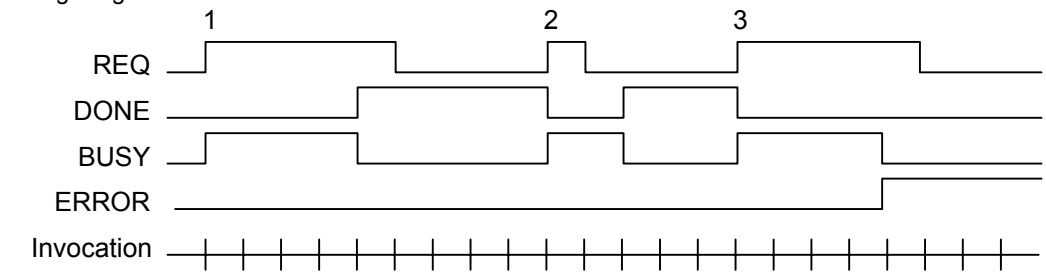
If an error occurred, the ERROR output indicates an error and the STATUS output contains the error code. The STATUS values are defined in Table 2.


```
                          SALRM
          BOOL ---  REQ              DONE  --- BOOL
         DWORD ---  ID               BUSY  --- BOOL
           INT ---  ATYPE           ERROR  --- BOOL
           INT ---  ASPEC          STATUS  --- DWORD
           INT ---  LEN
           ANY ---  AINFO---    ---AINFO
```

```
FUNCTION_BLOCK SALRM  (* Generate alarm *)
VAR_INPUT
  REQ     : BOOL;         (* Request *)
  ID      : INT;          (* Handle of the slot of the alarm source *)
  ATYPE   : INT;          (* Type of the alarm *)
  ASPEC   : INT;          (* Specifier of the alarm *)
  LEN     : INT;          (* Length to be read *)
END_VAR

VAR_OUTPUT
  DONE   : BOOL;          (* Function done *)
  ERROR  : BOOL;          (* Error detected *)
  BUSY   : BOOL;          (* FB is busy *)
  STATUS : DWORD;         (* Last detected status *)
END_ VAR
VAR_IN_OUT
  AINFO  : ANY            (* Additional alarm information *)
END_VAR
```

Timing Diagram:



Case 1:
The REQ input remains 1 until the function block invocation has completed, it is reset by the user when DONE gets 1.

Case 2:
The user pulses the REQ input only for one invocation. The request is not aborted.

Case 3:
Like case 1, but an error occurred.

Figure 51 – SALRM function block

The following state diagram describes the algorithm of the SALRM function block.

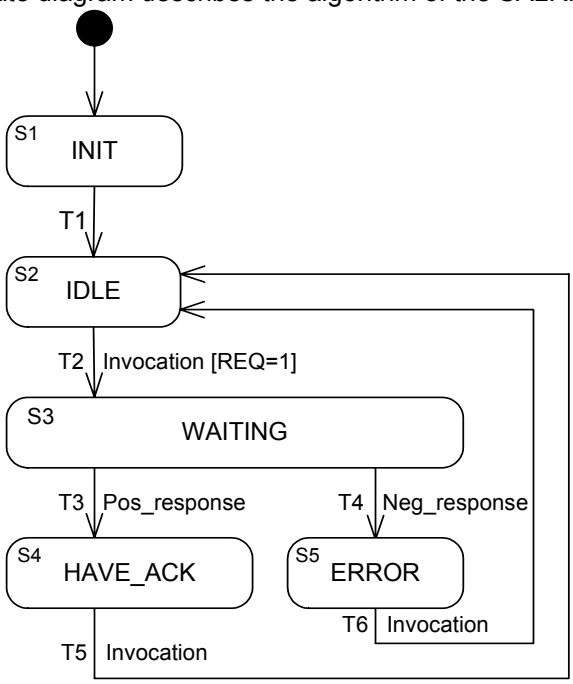


Figure 52 – State diagram of SALRM function block

The following table defines the transitions and actions given in the state diagram above.

Table 31 - Transitions and actions for SALRM state diagram

STATE NAME	**STATE DESCRIPTION**
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 WAITING	Evaluate FB inputs. Request alarm notification: AlarmNotification.Req with AREP= ... Slot number= SLOT Alarm Type= ATYPE Seq Nr= increment last Seq Nr specific to the DP-slave Alarm Specifier= ASPEC Add Ack= false Alarm Data= AINFO
S4 HAVE_ ACK	No actions
S5 ERROR	indicate error

TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	DONE := 0 BUSY := 0 ERROR := 0 STATUS := 0
T2	S2	S3	Invocation (Next invocation)	REQ=1	DONE := 0 BUSY := 1 ERROR := 0 STATUS := -1 (is busy)

T3	S3	S4	Pos_response (Positive response from remote communication partner: AlarmNotification.Cnf(+))		DONE := 1 BUSY := 0 ERROR := 0 STATUS := 0
T4	S3	S5	Neg_response (Negative response from remote communication partner or other communication problems detected: AlarmNotification.Cnf(-) or Abort.Ind or local problems)		DONE := 0 BUSY := 0 ERROR := 1 STATUS := New error code
T5	S4	S2	Invocation (Next invocation of this instance)		DONE := --- BUSY := --- ERROR := --- STATUS := ---
T6	S5	S2	Invocation (Next invocation of this instance)		DONE := --- BUSY := --- ERROR := --- STATUS := ---
--- indicates “unchanged” FB outputs					

5.4.2 Generate Diagnosis Information (SDIAG)

This function is provided only for PROFIBUS DP because the service used for the function block is not provided by PROFINET IO. Diagnosis is realized using the alarm services and data records.

The communication function Generate Diagnosis Information for a DP-slave uses the SDIAG function block defined in this clause. One instance of a SDIAG function block provides one instance of the PLC function Generate Diagnosis Information. The function is invoked when the REQ input is equal to 1.

The ED_FLAG input provides the ext_diag_flag information, the ED_OV provides the ext_diag_overflow information. The LEN input contains the length in byte of the additional diagnosis information stored in the ED_DATA parameter. The Variable given as ED_DATA parameter shall be at least of LEN byte. Possible value range of the LEN input is 0 .. 59.

If an error occurred, the ERROR output indicates an error and the STATUS output contains the error code. The STATUS values are defined in table 3.


```
                          SDIAG
BOOL --- REQ              ERROR --- BOOL
BOOL --- ED_FLAG        STATUS --- DWORD
BOOL --- ED_OV
 INT --- LEN
 ANY --- ED_DATA--   --ED_DATA
```

```
FUNCTION_BLOCK SDIAG  (* Set diagnosis information *)
VAR_INPUT
  REQ     : BOOL;        (* Request *)
  ED_FLAG : BOOL;        (* ext_diag_flag *)
  ED_OV   : BOOL;        (* ext_diag_overflow *)
  LEN     : INT;         (* Length of the data in ED_DATA *)
END_VAR

VAR_OUTPUT
  ERROR  : BOOL;         (* Error detected *)
  STATUS : DWORD;        (* Last detected status *)
END_ VAR
VAR_IN_OUT
  ED_DATA : ANY           (* Additional diagnosis information *)
END_VAR
```

Figure 53 – SDIAG function block

The following state diagram describes the algorithm of the SDIAG function block.

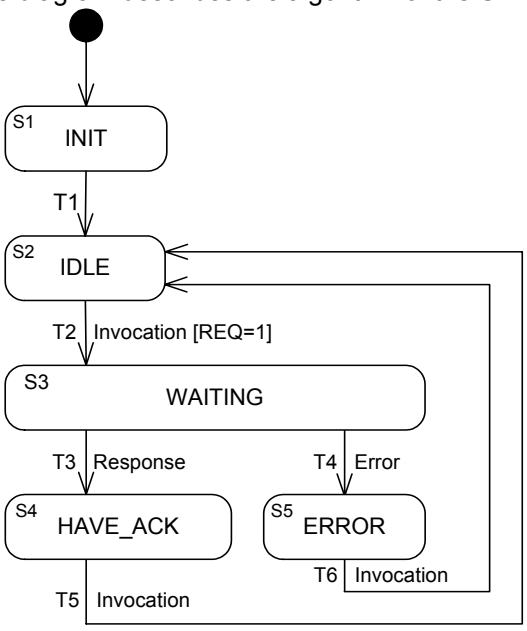


Figure 54 – State diagram of SDIAG function block

The following table defines the transitions and actions given in the state diagram above.

Table 32 - Transitions and actions for SDIAG state diagram

STATE NAME	STATE DESCRIPTION
S1 INIT	cold start state, initialise outputs
S2 IDLE	idle state, No actions
S3 WAITING	Evaluate FB inputs.

		Set diagnosis information: SetSlaveDiag.Req with AREP= ... Ext_Diag_Flag= ED_FLAG Ext_Diag_Overflow= ED_OV Length of Ext_Diag_Data= LEN Ext_Diag_Data= ED_DATA			
S4 HAVE_ ACK		No actions			
S5 ERROR		indicate error			
TRAN-SITION	**SOURCE STATE**	**TARGET STATE**	**EVENT**	**CONDITION**	**ACTION**
T1	S1	S2		Initialisation done	ERROR := 0 STATUS := 0
T2	S2	S3	Invocation (Next invocation)	REQ=1	ERROR := 0 STATUS := ---
T3	S3	S4	Response (Response from remote communication partner: SetSlaveDiag.Cnf)		ERROR := 0 STATUS := 0
T4	S3	S5	Error (Communication problems detected: Abort.Ind or local problems)		ERROR := 1 STATUS := New error code
T5	S4	S2	Invocation (Next invocation of this instance)		ERROR := --- STATUS := ---
T6	S5	S2	Invocation (Next invocation of this instance)		ERROR := --- STATUS := ---
--- indicates “unchanged” FB outputs					

6 PLC in Multiple Communication Roles

A real PLC may implement multiple communication roles i.e. it may act as a Host Controller, a Supervisor, and a Field Device at the same time.

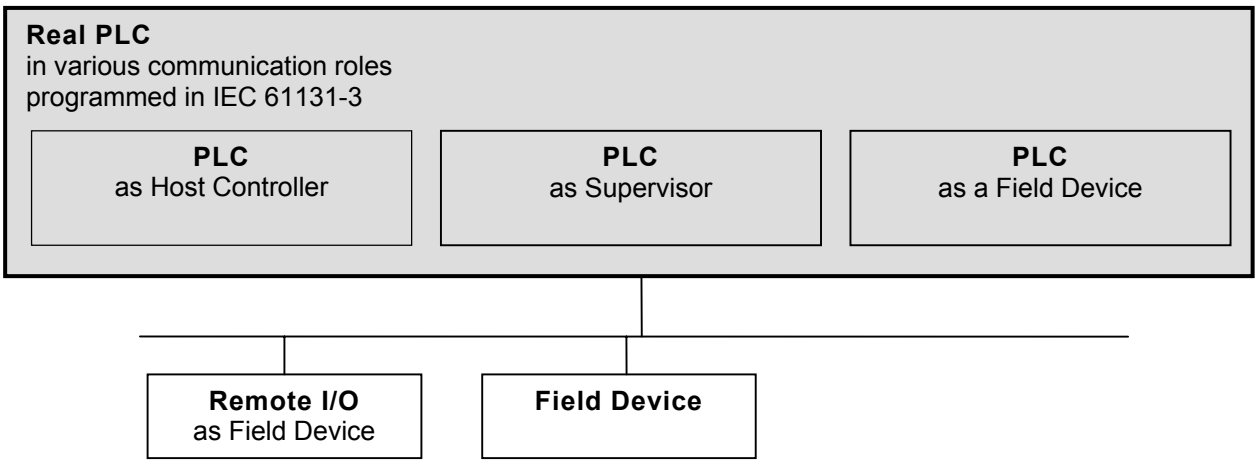


Figure 55 – PLC in multiple communication roles

All Communication Function Blocks defined in the previous clauses may be used in the application program of this PLC.

This allows to use e.g. a technological function block which was implemented to run on a Field Device also on a PLC which acts additionally as a Host Controller.

7 Guidelines for application of Communication Function Blocks

7.1 Communication Function Blocks and Proxy Function Blocks

This clause provides some tutorial information for the user of the Communication Function Blocks defined in the previous clauses. The Communication Function Blocks can be used in two levels of PLC applications written in IEC 61131-3 programming languages as shown in the following figure:

1. In a Host Controller the Communication FB can be used in application programs to communicate ***directly*** with Field Device. This use is mainly offered to experts which are familiar with the communication specific functionality.

2. The Communication FB can also applied ***inside*** a so-called Proxy FB to achieve for its user a "hidden" communication with the Field Device. The Proxy FB can represent the technological functionality of the Field Device completely or partly. In this case an expert for the communication and for the specific Field Device functionality provides a predefined standardised interface to the Field Device to the user of the Proxy FB.

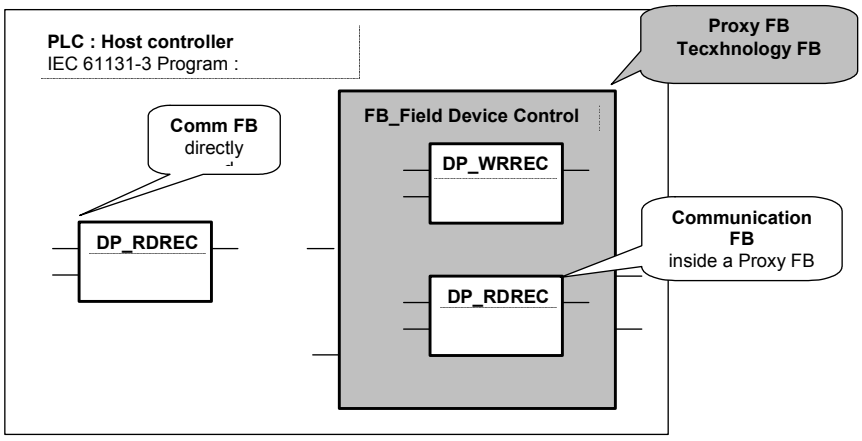


Figure 56 – Usage of Communication FB and Proxy FB in the PLC program (Host Controller)

7.2 Communication Function Blocks and PROFINET CBA

The user is able to create a PROFINET CBA component out of a technology function block (also called proxy function block) or a user written application program (or part of it), which uses the above defined communication function blocks. Thus he will be able to use the PROFINET CBA component from the context of other PROFINET CBA devices.

The term “proxy” is also used in the context of PROFINET CBA as transparent communication link, thus the meaning is quite different from the meaning defined within this specification

7.3 Mapping Technological Functionality to Proxy FB

There are two main concepts for mapping technological functionality to Proxy FB as illustrated in the following figure:

1. In the typical usage *one **single*** function block instance represents the device or the logical object in the application program. This is called here a *Proxy FB*.

   In this usage the invocation (call) of the FB passes *all* FB inputs via the actual parameters

into the function block instance. This kind of FB has only one "method", which is performed by the single FB algorithm dependant on the history of the last FB invocation and the new actual parameter set.
However various input data can cause different behaviour of the function block. For example one input can determine a specific "mode" or "method" of the FB execution. In the following figure the so-called Proxy FB illustrates this usage.

2. In another usage a *set of **different*** function block types and instances represent the various "methods" applied to one device or logical object.
This device or object needs not to be represented by a user accessible FB but may be a virtual object managing the internal device data, e.g. a separate instance or a common data structure. Each invocation of these various "method"-FB may have their specific set of parameters and may causes a different method on the device. Each of these invocations references the object representing the device for co-ordination and synchronisation. In the following figure the function blocks Methods1_FB, Methods2_ illustrate this usage.

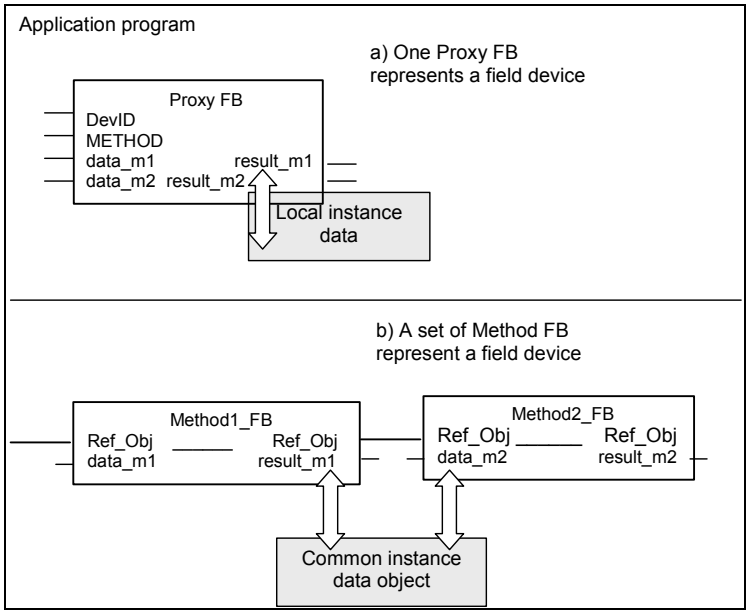


Figure 57 – Concepts of FB application
a) One function block as proxy b) Set of function blocks as methods

7.4 Using Device IO

7.4.1 Integrated and External Device IO

The following example of a minimal PID controller (MiniPID Field Device) illustrates the two possible attachments of IO to a Field Device. The PID controller receives cyclically the set point SP value via the Field Device interface from the Host Controller and calculates the OUT value depending on the actual process value PV. The process data OUT and PV are *locally* attached the Field Device and presented to the interface of the Field Device.

The parameters PAR of the PID controller can be set and modified via a process data record as a data structure PAR. The current version of the parameters is for example contained in the data record PAR as byte 0..1 and is managed only in the host.

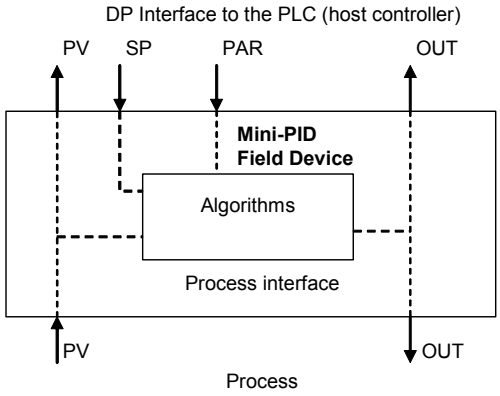


Figure 58 – Field Device with local IO

Another possible application is shown in the following figure. The Mini-PID Field Device puts and gets the process data PV and OUT by other Field Devices AI and AO via the PLC (Host Controller). In this case the following interface structure is applied.

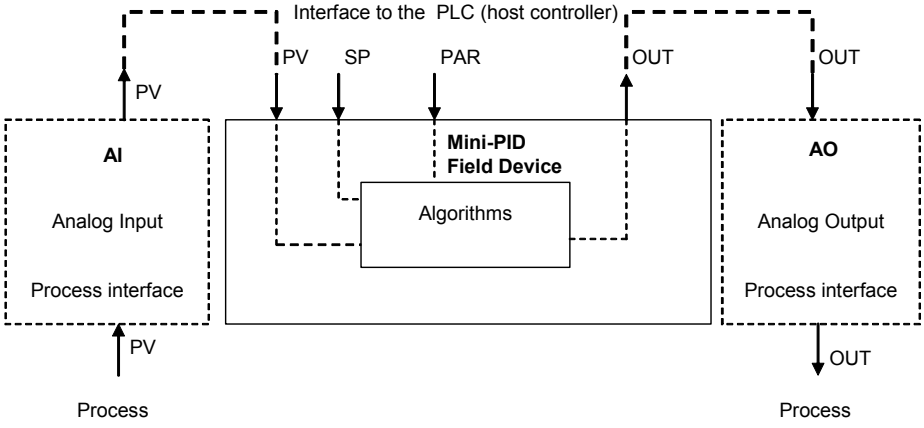


Figure 59 – Field Device with external IO

7.4.2 Proxy FB for a Device with Local IO

For the Field Device above shown with the local IO a simplified Proxy FB usable in the PLC (Host Controller) program is given in the next figure. The parameter ID identifies the Field Device, SP and PAR are input parameters for the PID, PV and OUT output the actual process values to the PLC program. REQ_PAR and VPAR serve for the actualisation of the data record PAR. Details are shown in the example program below in ST language

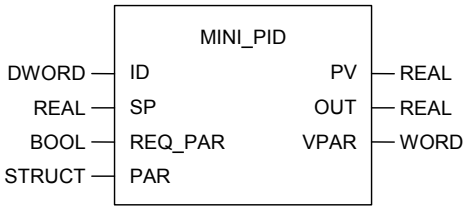


Figure 60 – Proxy FB MINI_PID with local IO

For each PID Field Device a Proxy FB MINI-PID is instanciated and the corresponding parameter ID is given to identify the device.

REQ_PAR=1 initiates the parameterisation, i.e. the data of PAR are sent to the device as data record 1. In VPAR the acknowledgement of the successful parameterisation is given as a version number in the VPAR variable.

The example program below shows the algorithm of the Proxy FB in IEC 61131 language Structured Text (ST).

```
FUNCTION_BLOCK MINI_PID;
VAR_IN
     ID: DWORD;
     SP: REAL;
     REQ_PAR: BOOL;
     PAR: STRUCT;
          VPAR:     WORD;
          PARS:     ARRAY [1..20] OF BYTE;
          END_STRUCT;
END_VAR;
VAR_OUT;
     PV, OUT: REAL;
     VPAR:     WORD;
     END_VAR;
VAR
     OLD_REQ: BOOL;
     INS: STRUCT;
          PV, OUT: REAL;
          VPAR:WORD;
          END_STRUCT;
     RD_INS: GETIO;(* Instance for reading the input data of Field Device  *)
     WR_OUTS:SETIO;(* Instance for writing the output data of Field Device *)
     WRPAR:      WRREC;(* Instance for writing parameters of Field Device *)
END_VAR;

BEGIN
RD_INS(ID:= ID);
INS := RD_INS.INPUTS; (* transfers an array of bytes to a structure *)
WRPAR(REQ:= REQ_PAR; ID:= ID; MLEN:= 20; RECORD:= PAR.PARS);

IF   WRPAR.DONE THEN     (* parameterisation successfully finished *)
     VPAR:= PAR.VPAR;
     :
     END_IF;
IF   WRPAR.ERROR THEN    (*parameterisation failed *)
     :
     END_IF;
WR_OUTS(ID:= ID; OUTPUTS:= SP; LEN := 10);
END;
```

Alternatively the algorithm shown in the following figure may be used to parameterise the Field Device. This algorithm uses the BUSY output to avoid to call the WRPAR instance and to poll the result of the parameterisation.

```
IF (REQ = true) AND (OLD_REQ = false)
THEN IF NOT WRPAR.BUSY
     THEN  (* first call *)
          WRPAR(REQ:= true; ID:= ID; LEN:= 20; RECORD:= PAR.PARS);
     ELSE
          :    (* error handler: REQ_PAR on busy parameterisation *)
     END_IF;
END_IF;
IF WRPAR.BUSY THEN WRPAR (); (* next invocations with the same parameters *)
END_IF;
IF WRPAR.DONE THEN        (*parameterisation successfully finished *)
     VPAR:= PAR.VPAR;
     :
     END_IF;
IF WRPAR.ERROR THEN           (*parameterisation failed *)
     :
```



```
     END_IF;
OLD_REQ:= REQ;
```

The output BUSY of the WRREC can be used for reducing the invocations and to achieve a simple error handling upon REQ_PAR=1 during running parameterising.

7.4.3 Proxy FB for a Field Device with IO via the Process Image

If the cyclic data from and to the Field Device is mapped via the *process image* the actual process variable PV is used as an *input* of the Proxy FB. The setpoint OUT has to be written in the process output image.

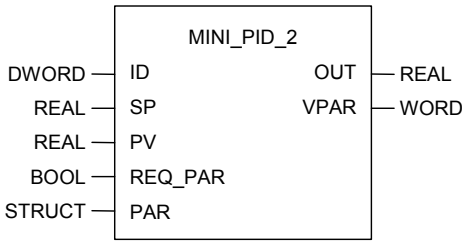


Figure 61 – Proxy FB MINI_PID_2 with IO via the process image

In this case the FB input SP and the FB output OUT have to be the corresponding addresses in the process image.

Using this FB the inputs and /or outputs of the PID functionality may be integrated into a PID Field Device or the PID functionnality may get its input and outputs from an other Field Device.

The example program below shows the algorithm of the Proxy FB MINI_PID_2 in IEC 61131 language Structured Text (ST).

```
FUNCTION_BLOCK MINI_PID_2;
VAR_IN
     ID: DWORD;
     SP, PV: REAL;
     REQ_PAR: BOOL;
     PAR: STRUCT;
          VPAR: WORD;
          PARS: ARRAY [1..20] OF BYTE;
          END_STRUCT;
END_VAR;
VAR_OUT;
     OUT: REAL;
     VPAR: WORD;
END_VAR;
VAR
     OLD_REQ: BOOL;
     INS: STRUCT;
          PV, OUT: REAL;
          VPAR:WORD;
          END_STRUCT;
     OUTS: REAL;
     RD_INS:   GETIO;
     WR_OUTS:  SETIO;
     WRPAR:    WRREC;
     :
END_VAR;

BEGIN
WRPAR(REQ:= REQ_PAR; ID:= ID; LEN:= 20; RECORD:= PAR.PARS);
     (* permanent invocations for updating of outputs *)
IF WRPAR.DONE THEN VPAR:= PAR.VPAR;
                           (* parameterisation successfully finished *)
     END_IF;
IF WRPAR.ERROR THEN ...(* parameterisation failed *)
END_IF;
END
```

7.4.4 Some Recommendations

The usage of the BUSY output is useful for the application of the asynchronuously executed function blocks.

Comparing Proxy FB using an integrated IO interface or using IO via process image shows some differences:

Proxy FB with integrated IO interface (e.g. FB MINI_PID):

- The interface of the Proxy FB represents the complete Field Device interface of its technological functionality.
- The Field Device can be represented by one Proxy FB.
- The Field Device is identified only by one parameter ID.

Proxy FB with IO via process image (e.g. FB MINI_PID_2):

- The same FB interface may be used with different configurations:

  The IO are integrated in the Field Device the Proxy FB represents.

The IO come from a different device.
The FB itself may contain the technological functionality or parts of it.

- In the case of a Field Device with integrated IO the interface of this Field Device is splitted to the Proxy FB and the process image.

7.5 Scheduling of Function Blocks

Accordíng IEC 61131-3 the function block instance can be

- *invoked* (called) in the body of another Program Organisation Unit (POU) like a program or a function block which is in the next (higher) level of the hierarchy and/or

- *scheduled* by an associated Task which is defined according IEC 61131-3 on a periodic basis or upon the occurence of specified "events" and conditions.

That means if a funktion block instance does not have an association to an IEC 61131-3 task the execution of its algorithm is a part of the execution of the invoking POU.

In the figure below PROGRAM_1 is scheduled periodically with a task (100 ms cycle) and it invokes the included function block instances M_2 like subroutines.

Otherwise if a function block instance is associated with a task it shall be under the *exclusive* control of the task, independant of the rules of evaluation of the program organisation unit (FB or program) in which the function block instance is declared.

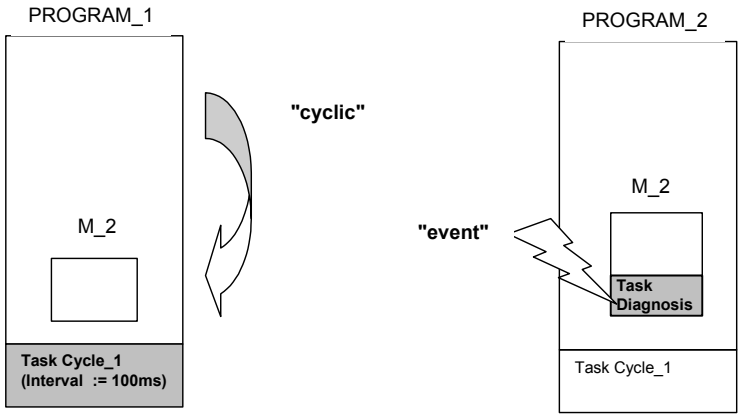


Figure 62 – Scheduling of a function block
a) in a task scheduled program (cyclic) - b) by a direct task association (event)

Proxy Function Blocks representing a Field Device may need for performance reason like response time to be *scheduled* in multiple tasks. Therefore the PLC Programming System and the PLC Runtime System may support one of the following solutions:

- The graphical and/or textual representation of the *same* function block instance invoked in *multiple* programs as illustrated in the figure below.
  In this case the function block instance M_2 is scheduled as well as cyclically by the invocation in PROGRAM_1 and event driven upon an occurrence of an diagnosis event in PROGRAM_2.

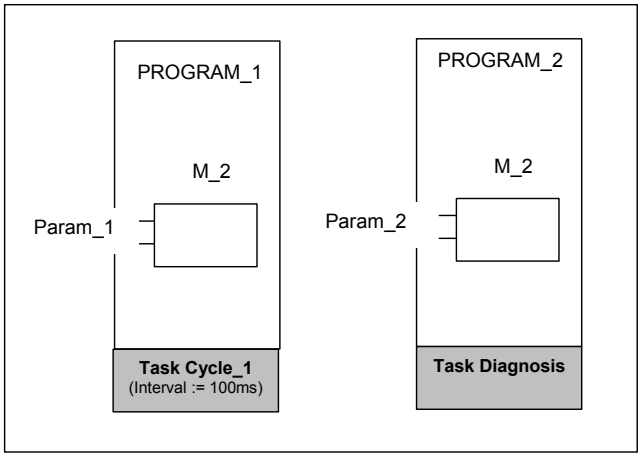


Figure 63 – Multiple scheduling of a FB instance by invokations in different programs

- The graphical and/or textual representation of *multiple* invokations of the same function block instance with different explicit task associations in the *same* program.
  In this case the function block instance M_2 is scheduled as well as by a cyclic task and by a diagnosis task.

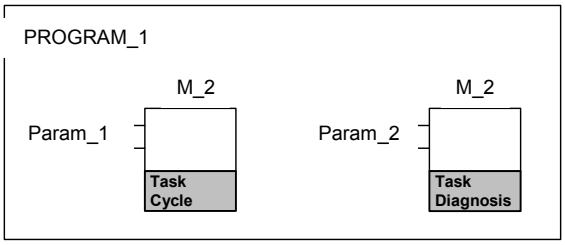


Figure 64 – Multiple scheduling of a FB instance by different invokations in the same programs

Some provision may be necessary for both concepts of multiple scheduling of function blocks:

- Specific invokation parameters can be used for the distinction which task has actually scheduled the function block; i.e. the indication of the actual invokation by a specific parameter can be used from inside the function block to apply the suitable "method".

- Since the multiple scheduling may cause an interrupt of the current function block execution the so-called reentrant programming technique provides the necessary data consistency; i.e. the concurrent access to common instance data has to be mutually exclusive using e.g. semaphores.

Annex A - Compliance Table

The following table lists all communication functions and Communication Function Blocks defined in this specification.
A manufacturer which claim to be compliant with this PNO specification shall provide a list in the format of this table and shall identify all compliant communication functions and function blocks.

Table A.1 - Compliance table

No	Communication function blocs and function	Explanation	Implementation specific additional information	Compliant (Y/N) for DP	Compliant (Y/N) for PN IO
Address Functions					
1	ID	Function for conversion of a physical address to the handle			n.a.
2	ADDR	Function for conversion of a handle to the physical address			n.a.
3	SLOT	Function for addressing a slot of a DP-slave			n.a.
4	ADDR_TO_ID	Function for conversion of a physical address to the handle			
5	ID_TO_ADDR	Function for conversion of a handle to the physical address			
Communication Function Blocks Host Controller					
6	GETIO	Get IO data object			
7	SETIO	Set IO data object			
8	GETIO_PART	Get a Part of IO data object			
9	SETIO_PART	Set IO data object Related to a Part of a Slot			
10	RDREC	Read Process Data Record			
11	WRREC	Write Process Data Record			
12	RALRM	Receive Alarm			
13	RDIAG	Read Diagnosis			n.a.
14	ICTRL	"Interlocked Control"			
Communication Function Blocks Supervisor					
15	RDIN	Read input data			
16	RDOUT	Read output data			
17	RDREC	Read process data record from a slot			
18	WRREC	Write process data record to a slot			
19	RDIAG	Read diagnosis information			n.a.
20	CNCT	Manage a connection			
Communication Function Blocks for Field Device					
21	RCVCO	Receives output data			
22	SBCCI	Subscribe input data			
23	PRVCI	Provide (publish) input data			
24	RCVREC	Receive process data record			
25	PRVREC	Receive request and provides a process data record			
26	SALRM	Request to send an alarm			
27	SDIAG	Request to send diagnosis			n.a.
Communication Function Blocks for Host Controller Serving Field Device Applications					
28	RCVCO	Receive output data			
29	PRVCI	Provide (publish) input data			
30	RCVREC	Receive process data record			
31	PRVREC	Receive request and provides a process data record			

In the following tables the permitted "*Implementation dependant features"* shall be listed.

Table A.2 - Implementation dependant features

Clause	Feature	Implementation chosen
2.7	Use and meaning of Error_Code_2	
2.7	Use and meaning of implementer-specific values in Error_Code_1	
3.2	Use of variables of the %I area at the INPUTS parameter of FB GETIO	
3.2	Use of variables of the %I area at the INPUTS parameter of FB GETIO_PART	
3.4	Content of the task information TINFO	

Table A.3 - Implementation dependant features for PROFIBUS DP

Clause	Feature	Implementation chosen
3.2	Maximum length of IO data object of one DP-slave supported by a DP-master (Class 1)	
3.2	Maximum length of IO data object of one slot of a DP-slave supported by a DP-master (Class 1)	
3.2	Maximum length of consistent IO data object of one slot of a DP-slave supported by a DP-master (Class 1)	
4.2	Maximum length of IO data object of one DP-slave supported by a DP-master (Class 2)	
4.2	Maximum length of IO data object of one slot of a DP-slave supported by a DP-master (Class 2)	
4.2	Maximum length of consistent IO data object of one slot of a DP-slave supported by a DP-master (Class 2)	

Table A.4 - Implementation dependant features for PROFINET IO

Clause	Feature	Implementation chosen
3.2	Maximum length of IO data object of one IO Device supported by an IO Controller	
3.2	Maximum length of IO data object of one subslot of an IO Device supported by an IO Controller	
3.2	Maximum length of consistent IO data object of one slot of an IO Device supported by an IO Controller	
4.2	Maximum length of IO data object of one IO Device supported by a Supervisor	
4.2	Maximum length of IO data object of one slot of a IO Device supported by a Supervisor	
4.2	Maximum length of consistent IO data object of one slot of a IO Device supported by a Supervisor	

PROFIBUS Nutzerorganisation e.V.
Haid-und-Neu-Str. 7
76131 Karlsruhe
Germany
Phone: +49 (0) 721 / 96 58 590
Fax: +49 (0) 721 / 96 58 589
info@profibus.com
www.profibus.com