IEC 61131-5

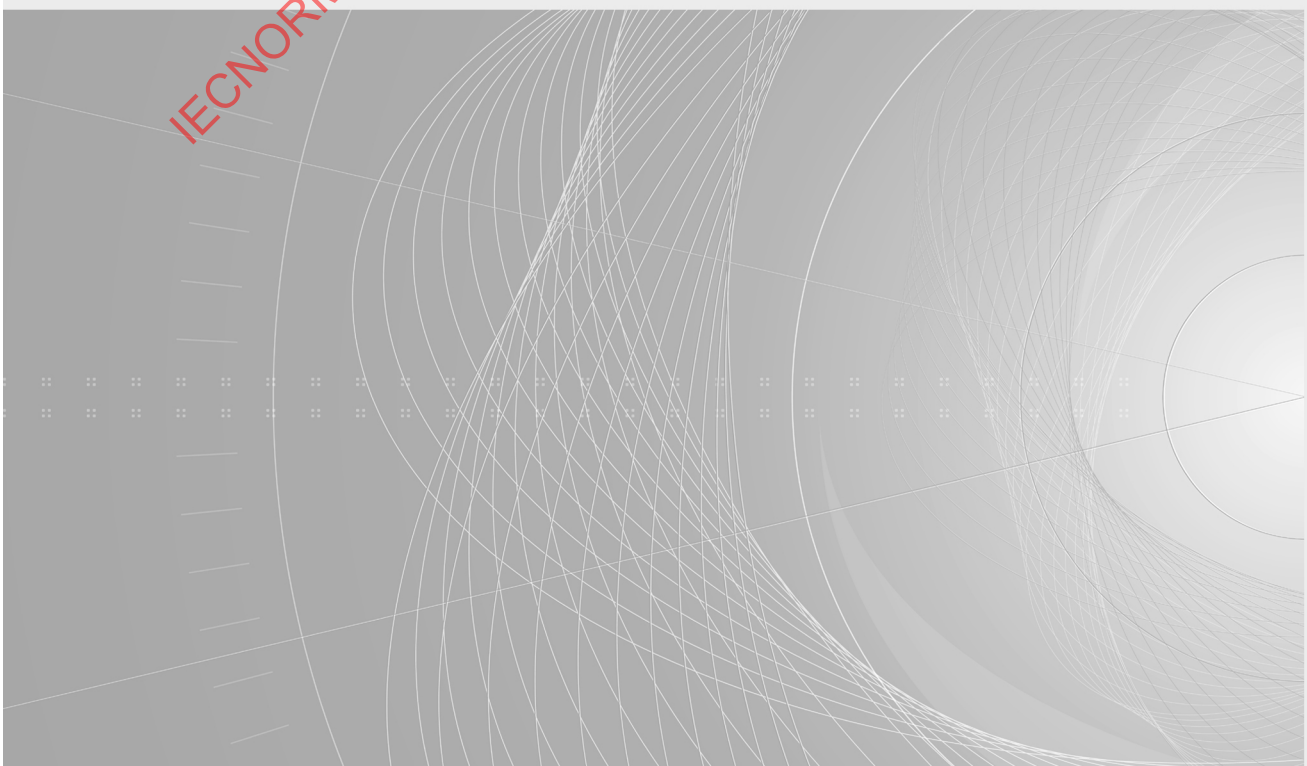
Edition 1.0 2000-11

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Programmable controllers –
Part 5: Communications

Automates programmables –
Partie 5: Communications



IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,…).
Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 61131-5

Edition 1.0 2000-11

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Programmable controllers – Part 5: Communications

Automates programmables – Partie 5: Communications

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE CODE PRIX XD

ICS 25.040.40; 35.240.50

ISBN 978-2-83220-262-3

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

PROGRAMMABLE CONTROLLERS –

Part 5: Communications

FOREWORD

1) The IEC (International Electrotechnical Commission) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of the IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, the IEC publishes International Standards. Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. The IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of the IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested National Committees.
3) The documents produced have the form of recommendations for international use and are published in the form of standards, technical specifications, technical reports or guides and they are accepted by the National Committees in that sense.
4) In order to promote international unification, IEC National Committees undertake to apply IEC International Standards transparently to the maximum extent possible in their national and regional standards. Any divergence between the IEC Standard and the corresponding national or regional standard shall be clearly indicated in the latter.
5) The IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with one of its standards.
6) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. The IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61131-5 has been prepared by subcommittee 65B: Devices, of IEC technical committee 65: Industrial-process measurement and control.

This bilingual version (2012-08) corresponds to the monolingual English version, published in 2000-11.

The text of this standard is based on the following documents:

FDIS	Report on voting
65B/411/FDIS	65B/420/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

The French version of this standard has not been voted upon.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 3.

This part should be read in conjunction with the other parts of IEC 61131. IEC 61131 consists of the following parts under the general title: *Programmable controllers*.

Part 1:1992, General information.

Part 2:1992, Equipment requirements and tests.

Part 3:1993, Programming languages.

Part 4:1994, User guidelines (published as technical report IEC TR 61131-4)

Part 5:2000, Communications

Part 8:2000, Guidelines for the application and implementation of programming languages (published as technical report IEC TR 61131-8)

Annexes A and B form an integral part of this standard.

Annex C is for information only.

Where a conflict exists between this and other IEC standards (except basic safety standards), the provisions of this standard should be considered to govern in the area of programmable controllers and their associated peripherals.

The committee has decided that the contents of this publication will remain unchanged until 2006. At this date, the publication will be

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

PROGRAMMABLE CONTROLLERS –

Part 5: Communications

1 Scope

This part of IEC 61131 specifies communication aspects of a programmable controller. It specifies from the viewpoint of a PC how any device can communicate with a PC as a server and how a PC can communicate with any device. In particular, it specifies the behavior of the PC as it provides services on behalf of other devices and the services the PC application program can request from other devices. It is not intended to specify how any device can communicate with any device using a PC as a router or gateway. The behavior of the PC as a communication client and server is specified independent of the particular communication subsystem, but the communication functionality may be dependent on the capabilities of the communication subsystem used.

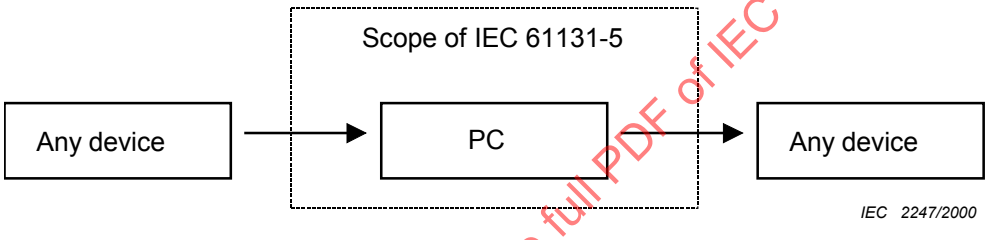


Figure 1 – Scope of this part of IEC 61131

The scope of this part is a subset of the "communication model" shown in figure 2 of IEC 61131-3; namely figures 2c and 2d are included in the scope of this part. Additionally, the means defined in this part of IEC 61131 may be used for communications within a program or between programs.

The mapping of the PC behavior to some particular communications subsystems is provided in the annexes.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of IEC 61131. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of IEC 61131 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

IEC 60050-351:1998, *International Electrotechnical Vocabulary – Part 351: Automatic control*

IEC 61131-1:1992, *Programmable controllers – Part 1: General Information*

IEC 61131-2:1992, *Programmable controllers – Part 2: Equipment requirements and tests*

IEC 61131-3:1993, *Programmable controllers – Part 3: Programming languages*

ISO/IEC 2382-1:1993, *Information technology – Vocabulary – Part 1: Fundamental terms*

ISO/IEC 9506-1:1990, *Industrial automation systems – Manufacturing Message Specification – Part 1: Service definition*

ISO/IEC 9506-2:1990, *Industrial automation systems – Manufacturing Message Specification – Part 2: Protocol specification*

3 Definitions

For the purpose of this part of IEC 61131, the following definitions apply.

This part of IEC 61131 is based on the concepts of parts 1 to 3 of IEC 61131 and makes use of the following terms defined in other international standards.

Definitions from other publications

IEC 60050-351

control

monitoring

IEC 61131-1

application program (2.1)

application program archiving (4.6.4)

cold restart (2.56)

input (2.25)

main processing unit (2.32)

modifying the application program (4.6.2.6)

output (2.40)

programmable controller (2.50)

programmable controller system (2.51)

testing the application program (4.6.2.5)

warm restart (2.56)

IEC 61131-3

access path (1.3.2)

direct representation (1.3.23)

invocation (1.3.43)

program (verb, 1.3.60)

sub-element (2.3.3.1)

ISO/IEC 2382-1

data

ISO/IEC 9506-1

client

download

event (clause 15)

server

uninterruptible variable access (12.1.1.1)

upload

variable

Definitions of this part

3.1
alarm
event which signals a specific condition

3.2
data acquisition
collection of data for the purpose of process monitoring and report generation

3.3
direct operator interface
when the client can communicate to the operator interface via the communication system with no application program interaction

3.4
device verification
allows other devices to determine if the PC is able to perform its intended function in the control system

3.5
health
the health of a PC or its subsystems is specified by returning one, and only one, of the three possible values. They are, in order of decreasing health: GOOD, WARNING and BAD

3.6
interlocked control
control through the synchronization of data exchanges between two parties. At various points in time, one party is waiting for the other party to deliver some expected data

3.7
local
internal to the PC; opposite of remote

3.8
parametric control
control by the client writing to control variables residing in the PC

3.9
processing unit
part of the main processing unit. It is the portion of a PC system which is responsible for the storage of the application program and data and the execution of the application program. A PC system has one or more processing units

3.10
program verification
testing of a PC application program to verify that it performs the function(s) it was designed to do in the process environment

3.11
recipe
description of procedures, or data for those procedures, or both, for making a product which uses the process or machinery that the controller is attached to, which is different from a previous product

3.12
remote
external to the PC; opposite of local

3.13
state
the state of the PC system is indicated by a list of attributes, each of which may be TRUE or FALSE. Zero, one, or more of these attributes may be TRUE at the same time

3.14
unsolicited
performed without an explicit request

4 Symbols and abbreviations

These are some abbreviations frequently used in this part of IEC 61131. These terms are defined or referenced in clause 3 of this part of IEC 61131.

CFB	Communication function block
FB	Function block
I/O	Input and output
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
MMS	Manufacturing Message Specification, ISO/IEC 9506-1 and ISO/IEC 9506-2
OSI	Open Systems Interconnection
PADT	Programming and debugging tool
PC	Programmable controller
PU	Processing unit

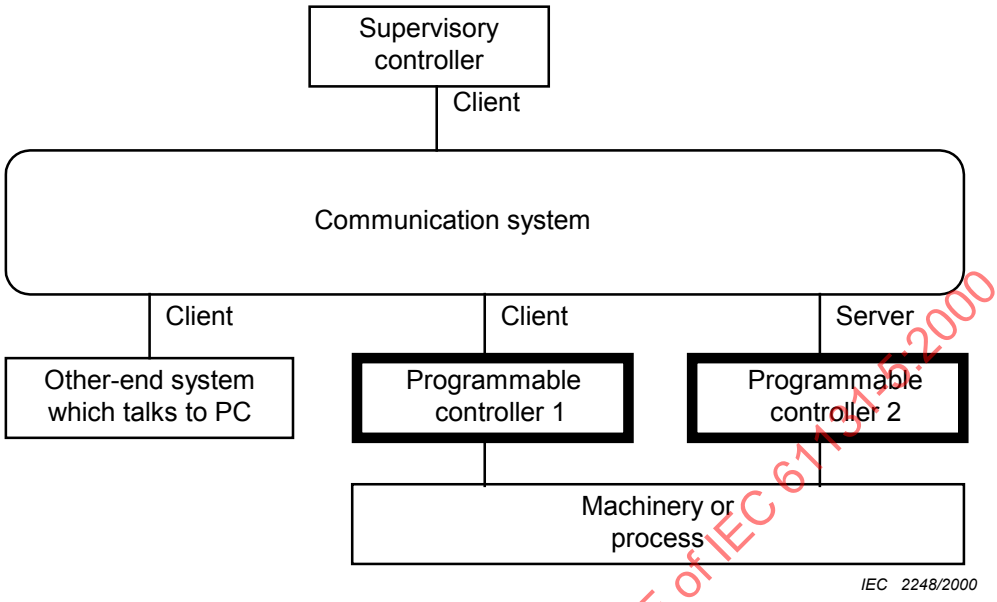
5 Models

This clause specifies the models which are used in the remainder of this part of IEC 61131.

5.1 PC network communication model

A programmable controller supplies some specific application functions to the rest of the control system. It may also request functions from other programmable controllers. The communication functions defined in this part of IEC 61131 are based on a communication subsystem that can report communication errors to the signal processing function of the PC (see 5.2).

The following diagram illustrates the devices in a communication network, showing three possible devices that request PC functions (clients) from PC 2. The two highlighted PCs are in the scope of this part of IEC 61131.



NOTE From the communication viewpoint the 'supervisory controller' and the 'other-end system which talks to PC' mentioned in this figure exhibit the same behavior to a PC communication server, i.e., they submit requests to the PC2.

Figure 2 – PC communication model

A PC may use its client function to communicate with any device if it behaves like a PC.

5.2 PC functional model

A PC consists of several functions (see figure 3). For a PC within the scope of this part of IEC 61131, at least one communication function is present.

The following diagram is taken from IEC 61131-1, figure 1. It is designed to illustrate some of the subsystems of a typical PC.

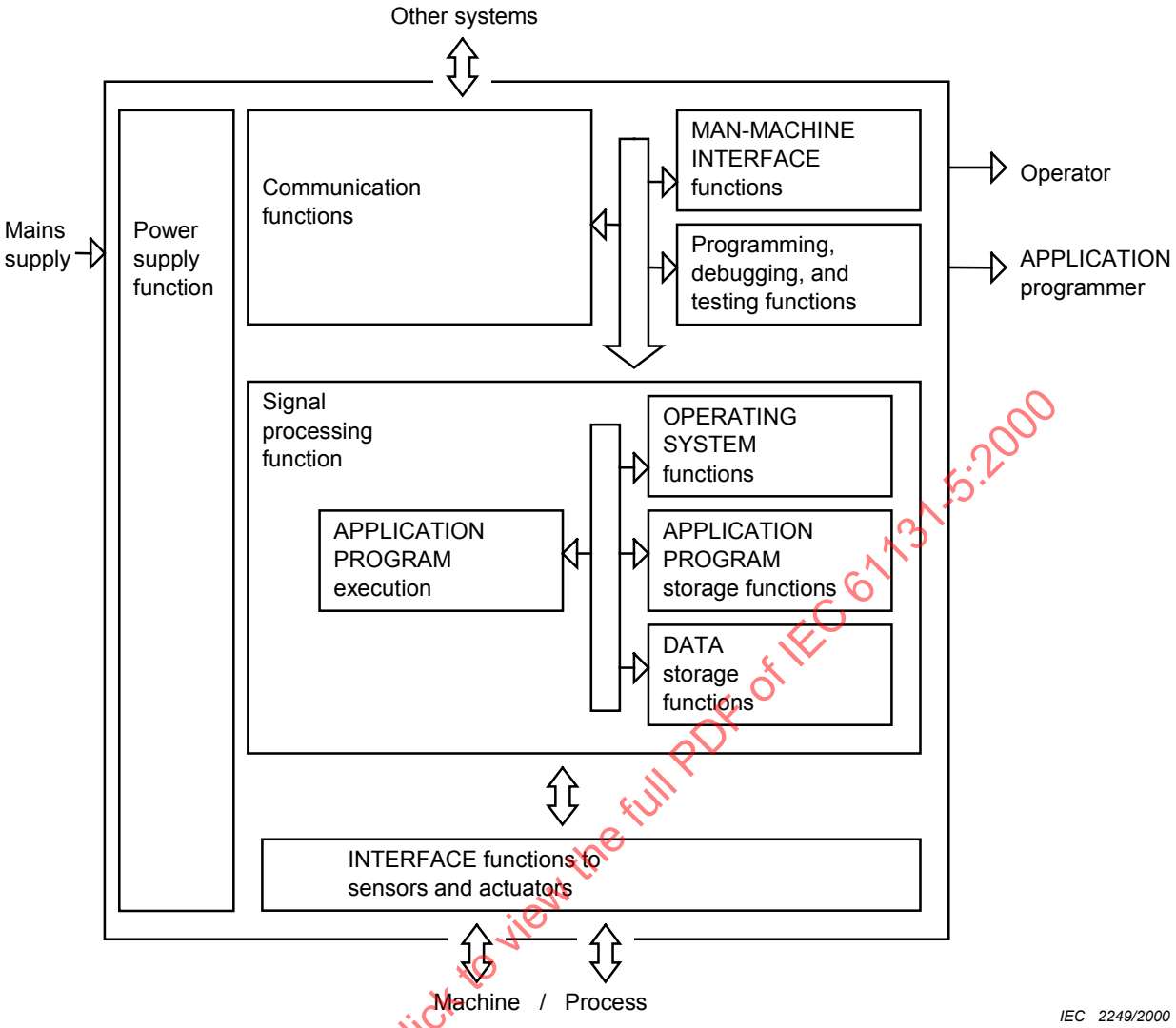


Figure 3 – Programmable controller functional model

There is a function that is part of the PC system, but usually external to the PC itself, known as the programming and debugging tool (PADT). The PADT is modelled as interacting with the PC via the communications function.

The Interface Function to Sensors and Actuators can have I/O which are local or remote to the Main Processing Unit (see 5.3 for the hardware model). The Interface Function to Sensors and Actuators has two attributes for each Application Program which defines how the PC is monitoring and controlling the machine/process. The input attribute has the following states:

- inputs provided to the Application Program are being supplied by the sensors,
- inputs provided to the Application Program are being held in the current state.

The output attribute has the following states:

- the actuators are being controlled by the Application Program,
- the actuators are being held in the current state.

5.3 PC hardware model

The following figure shows the PC hardware model. It shows the modules that make up a PC. A PC subsystem consists of one or more modules. The following figure corresponds to figure B.1 of IEC 61131-1 and figure 1 of IEC 61131-2.

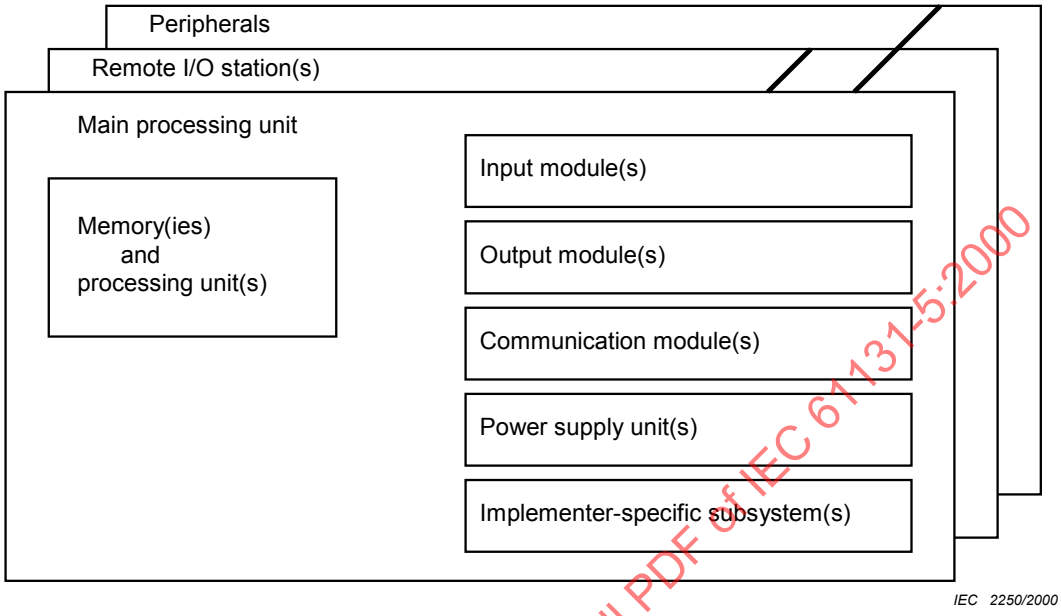


Figure 4 – Programmable controller hardware model

5.4 Software model

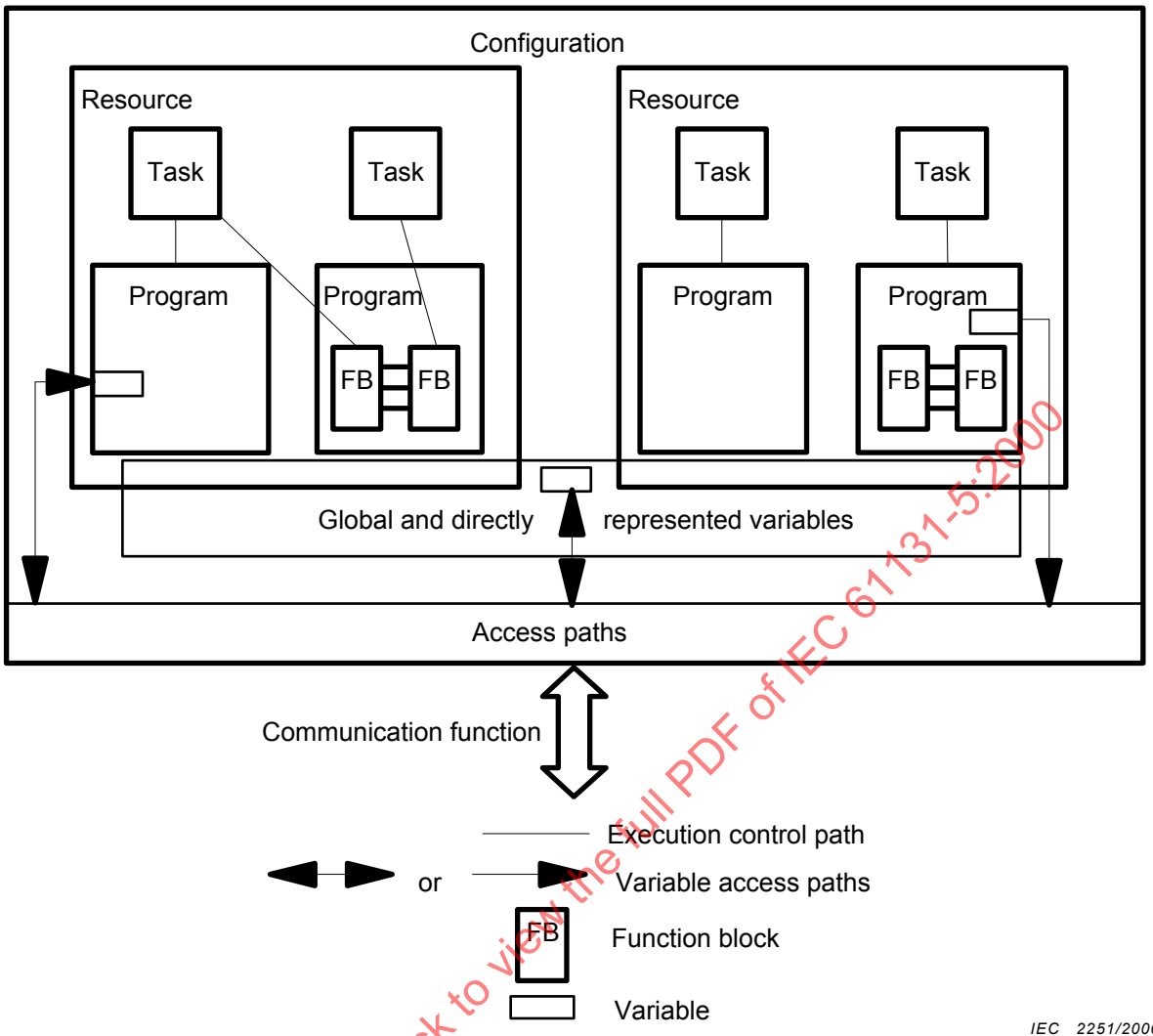
Figure 5 shows the PC software model defined in IEC 61131-3, figure 1. It illustrates the basic high-level language elements of the PC programming languages and their interrelationships. These consist of elements which are programmed using the languages defined in IEC 61131-3, i.e. programs and function blocks; and configuration elements, namely, configurations, resources, tasks, global variables, and access paths, which support the installation of programmable controller programs into programmable controller systems.

A configuration is the language element which corresponds to a programmable controller system as defined in IEC 61131-1. A resource corresponds to a "signal processing function" and its "man-machine interface" and "sensor and actuator interface" functions (if any) as defined in IEC 61131-1. A configuration contains one or more resources, each of which contains one or more programs executed under the control of zero or more tasks. A program may contain zero or more function blocks or other language elements as defined in IEC 61131-3.

Configurations and resources can be started and stopped via the "operator interface", "programming, testing, and monitoring", or "operating system" functions defined in IEC 61131-1. The mechanisms for the starting and stopping of configurations and resources via communication services are defined in this part of IEC 61131.

Programs, resources, global variables, access paths (and their corresponding access privileges), and configurations can be loaded or deleted by the "communication function" defined in IEC 61131-1. The loading or deletion of a configuration or resource shall be equivalent to the loading or deletion of all the elements it contains.

Access paths and their corresponding access privileges allow to access variables of a PC via communication services.



NOTE 1 This figure is illustrative only. The graphical representation is not normative.

NOTE 2 In a configuration with a single resource, the resource need not be explicitly represented.

Figure 5 – PC software model

6 PC communication services

This clause describes the concept of status information of a PC and provides a specification of the services the PC provides to the control system via the communication subsystem. (The next clause specifies how the PC application program can use the communication subsystem to interact with other devices.)

6.1 PC subsystems and their status

A PC can provide status, which includes state information and fault indications.

Status can be reported on some of the subsystems identified in the following figure. In addition, there is a summary status that provides general information about the PC.

Table 1 – Status presenting entities

No.	Status presenting entities
1	PC (as a whole)
2	I/O subsystem (includes Input and Output modules and other intelligent I/O devices)
3	Processing unit
4	Power supply subsystem
5	Memory subsystem
6	Communication subsystem
7	Implementer specific subsystems
NOTE The status is intended to provide information about the controller including its hardware and firmware subsystems, not considering configuration information. It is not intended to provide information about the controlled process nor the PC application program. The status data contains information concerning the state and the health of the PC and its subsystems.	

There are two concepts used in this part of IEC 61131 related to status: health and state. The "health" of a PC or its subsystems is specified by returning one and only one of the three possible values. The semantics associated with each value is specified below. They are, in order of decreasing health:

a) GOOD – If TRUE, the PC (or the specified subsystem) has not detected any problems which would prohibit it from performing the intended function;

b) WARNING – If TRUE, the PC (or the specified subsystem) has not detected any problems which would prohibit it from performing the intended function, but it has detected at least one problem which could place some limits on its abilities. The limit may be time, performance, etc. (see the following statements for further definition of these limits);

c) BAD – If TRUE, the PC (or the specified subsystem) has detected at least one problem which could prohibit it from performing the intended function.

The "state" of the PC system is indicated by a list of attributes, each of which may be TRUE or FALSE. Zero, one, or more of these attributes may be TRUE at the same time. The semantics associated with each attribute is specified in the remainder of this clause.

Each of the status information can also have implementer specified attributes. Some examples of implementer specified attributes are:

a) additional error diagnostics (e.g. EEPROM write cycles exceeded);

b) additional operational states (e.g. auto-calibrate enabled);

c) local key status (e.g. auto-restart required).

Implementations are not required to provide subsystem status. All instances of similar types of subsystems present in a system are reported separately. The name of the subsystem can be provided to allow differentiating subsystems of the same type.

6.1.1 PC summary status

The PC provides the following summary status information.

Table 2 – PC summary status

No.	Item	Description	
1	Health	GOOD	All subsystems in the PC indicate a GOOD health condition
2		WARNING	At least one subsystem indicates a WARNING health condition and no subsystem indicates a BAD health condition
3		BAD	At least one subsystem indicates a BAD health condition
4	Running	If TRUE, this attribute indicates if at least one part of the user application has been loaded and is under control of the PC	
5	Local control	If TRUE, this attribute indicates if local override control is active. If active, the ability to control a PC and its subsystems from the network may be limited. For example, this could be closely tied to the use of a local key switch	
6	No outputs disabled	If TRUE, this attribute indicates that the PC can change the physical state of all outputs as a result of application program execution or other means. If not TRUE, the physical state of some of the outputs are not affected (logical state may be affected). This is typically used in the testing and modifying of application programs in the PC	
7	No inputs disabled	If TRUE, this attribute indicates that the PC can access the physical state of all inputs as a result of application program execution or other means. If not TRUE, the physical state of some inputs cannot be accessed. This is typically used in the testing and modifying of application programs where the inputs can be simulated	
8	Forced	If TRUE, this attribute indicates that at least one I/O point associated with the PC has been forced. When an Input is forced, the application program will receive the value specified by the PADT instead of the actual value from the machine or process. When an output is forced, the machine or process will receive the value specified by the PADT instead of the value generated by execution of the application program. When a variable is forced, the application program will use the value specified by the PADT instead of that generated by the normal program execution	
9	User application present	If TRUE, this attribute indicates that the Processing Unit has at least one user application present	
10	I/O subsystem	If TRUE, this attribute indicates "WARNING" or "BAD" which is caused by an I/O subsystem	
11	Processing unit subsystem	If TRUE, this attribute indicates "WARNING" or "BAD" which is caused by a processing unit subsystem	
12	Power supply subsystem	If TRUE, this attribute indicates "WARNING" or "BAD" which is caused by a power supply subsystem	
13	Memory subsystem	If TRUE, this attribute indicates "WARNING" or "BAD" which is caused by a memory subsystem	
14	Communication subsystem	If TRUE, this attribute indicates "WARNING" or "BAD" which is caused by a communication subsystem	
15	Implementer specified subsystem	If TRUE, this attribute indicates "WARNING" or "BAD" which is caused by an implementer specified subsystem	

6.1.2 I/O subsystem

The PC provides the following status information of its I/O subsystem.

Table 3 – Status of I/O subsystem

No.	Item	Description	
1	Health	GOOD	indicates that there have been no errors detected in this I/O subsystem
2		WARNING	indicates that a minor fault has been detected in the I/O subsystem. An example of a minor fault is the occurrence of recoverable errors in the communication with a remote I/O station
3		BAD	indicates that a major fault has been detected in the I/O subsystem. An example of a major fault is losing communication with a remote I/O station
4	No outputs disabled	If TRUE, this attribute indicates that the PC can change the physical state of all outputs associated with the specified I/O subsystem as a result of application program execution or other means. If not TRUE, the physical state of some of the outputs is not affected (logical state may be affected). This is typically used in the testing and modifying of application programs in the PC	
5	No inputs disabled	If TRUE, this attribute indicates that the PC can access the physical state of all inputs associated with the specified I/O subsystem as a result of application program execution or other means. If not TRUE, the physical state some inputs cannot be accessed. This is typically used in the testing and modifying of application programs where the inputs can be simulated	
6	I/O forced	If TRUE, this attribute indicates that at least one I/O point associated with this subsystem has been forced. When an Input is forced, the application program will receive the value specified by the PADT instead of the actual value from the machine or process. When an output is forced, the machine or process will receive the value specified by the PADT instead of the value generated by execution of the application program	
NOTE The definition of "major fault" and "minor fault" shall be provided by the implementer.			

6.1.3 Processing unit

The PC provides the following status information of its processing unit.

Table 4 – Status of processing unit

No.	Item	Description
1 2 3	Health	This attribute identifies the health of the processing unit. The implementer shall specify the conditions when GOOD, WARNING or BAD are valid
4	Running	If TRUE, this attribute indicates if at least one part of the user application has been loaded and is under control of the processing unit
5	Local control	If TRUE, this attribute indicates if local override control is active. If active, the ability to control the processing unit from the network may be limited. For example, this could be closely tied to the use of a local key switch
6	No outputs disabled	If TRUE, this attribute indicates that the processing unit can change the physical state of all outputs controlled by this processing unit as a result of application program execution or other means. If not TRUE, the physical state of some of the outputs are not affected (logical state may be affected). This is typically used in the testing and modifying of application programs in the PU
7	No inputs disabled	If TRUE, this attribute indicates that the processing unit can access the physical state of all inputs accessible from this processing unit as a result of application program execution or other means. If not TRUE, the physical state of some inputs cannot be accessed. This is typically used in the testing and modifying of application programs where the inputs can be simulated
8	User application present	If TRUE, this attribute indicates that the Processing Unit has at least one User Application present
9	Forced	If TRUE, this attribute indicates that at least one variable associated with this Processing Unit has been forced. When a variable is forced, the application program will use the value specified by the PADT instead of that generated by the normal program execution.

6.1.4 Power supply subsystem

The PC can provide status information about any of the power supply subsystems; see figure 6 for the assumed configuration of a PC power supply. The requirements on power supplies of PC systems and their behavior is described in IEC 61131-1 and IEC 61131-2.

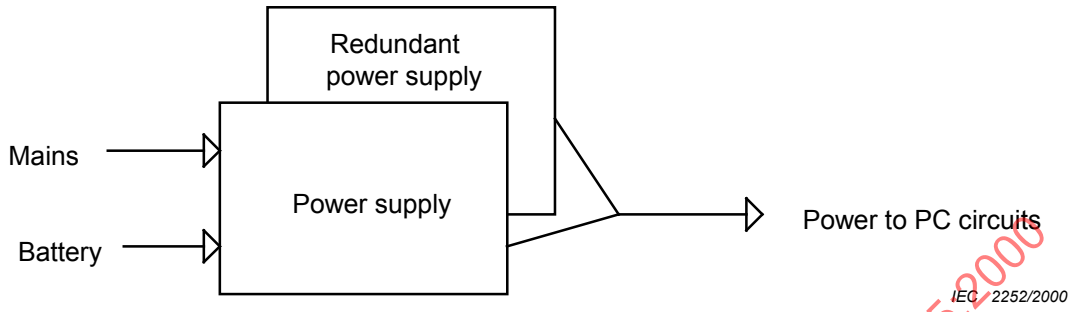


Figure 6 – Programmable controller power supply

Table 5 – Status of power supply

No.	Item	Description	
1	Health	GOOD	indicates that there have been no problems detected in the power supply to prevent it from remaining operable for an indefinite time
2		WARNING	indicates that a problem has been detected in the power supply which may cause to become inoperable in a limited time
3		BAD	indicates that the power supply is not operable
4	In use	If TRUE, this attribute indicates that the power supply subsystem is in use, i.e. it supplies power to the PC	
5	Mains operating	If TRUE, this attribute indicates that the mains are supplying power within the range specified for the power supply	
6	Mains low	If TRUE, this attribute indicates that the mains are not supplying power within the range specified for the power supply	
7	Battery operating	If TRUE, this attribute indicates that the battery is supplying power within the range specified for the power supply	
8	Battery low	If TRUE, this attribute indicates that the battery is not able to supply power within the range specified for the power supply	
9	Protection tripped	If TRUE, this attribute indicates that a protection device within the power supply has removed a portion of the power to the PC	

6.1.5 Memory subsystem

The PC provides the following status information of its memory subsystem.

Table 6 – Status of memory

No.	Item	Description	
1	Health	GOOD	No errors have been found in the memory associated with this subsystem
2		WARNING	At least one correctable error has been detected and no uncorrectable errors have been detected
3		BAD	At least one uncorrectable error has been detected
4	Protected[1)]	If TRUE, this attribute indicates that the memory in this memory subsystem has been protected in that it cannot be modified. This generally indicates that the application program located in this memory subsystem cannot be altered. [1)] This attribute models a logical state not physical characteristics of the subsystem. If some portions of the memory are protected and some are not, these shall be reported as multiple subsystems.	

6.1.6 Communication subsystem

The PC provides the following status information of its communication subsystem.

Table 7 – Status of communication subsystem

No.	Item	Description	
1	Health	GOOD	indicates that either no errors or an acceptable number of recoverable errors has occurred
2		WARNING	indicates that more than an acceptable number of recoverable errors has occurred
3		BAD	indicates that the communication subsystem is not able to communicate with all devices as intended
4	In use	If TRUE, this attribute indicates that the communication subsystem is currently operating. For example in the case of an MMS communication interface this means that at least one application association is established. Otherwise, the implementer shall define the semantic of this attribute	
5	Local error	If TRUE, this attribute indicates that there are some errors, internal to the communication subsystem, that inhibit operation	
6	Remote error	If TRUE, this attribute indicates that there are some errors, at devices being communicated with, that inhibit operation	
NOTE 1 The communication subsystem reporting its state may not be able to report its own bad state in the way defined in this clause. But, within a PC system, several independent communication subsystems may operate, and all of them may provide status information. NOTE 2 It is intended that the implementer specific information will provide additional information about each particular interface. ISO network interfaces also provide additional information via network management functions.			

6.1.7 Implementer specific subsystems

Other subsystems of a PC system shall be modelled as implementer specific subsystems. Some examples of these subsystems are:

a) PID controller;

b) motion controller;

c) other auxiliary processors.

Table 8 – Status of implementer specific subsystem

No.	Item	Description	
1	Health	GOOD	indicates that there have been no errors detected in this subsystem
2		WARNING	indicates that a minor fault has been detected in this subsystem
3		BAD	indicates that a major fault has been detected in this subsystem
NOTE The definition of "major fault" and "minor fault" shall be provided by the implementer.			

6.1.8 Presentation of status information

The status information shall be presented using variables with a pre-defined access path in the configuration declaration of the PC application program or shall be presented as a variable with direct representation to a remote communication partner.

Table 9 – Presentation of status information

No.	Presentation of status information
1	PC summary status as variable with pre-defined access path P_PCSTATE
2	PC summary status as variable with direct representation %S
3	PC summary status and status of all subsystems as variable with pre-defined access path P_PCSTATUS
4	Status information of each subsystem as a set of variables with direct representation %SC<n>
5	Type of each subsystem as a set of variables with direct representation %SU<n>
6	Name of each subsystem as a set of variables with direct representation %SN<n>
7	State of each subsystem as a set of variables with direct representation %SS<n>
8	Implementer specific status of each subsystem as a set of variables with direct representation %SI<n>

If the PC summary status shall be presented in a variable, it shall have the access path P_PCSTATE which shall be pre-defined in the configuration declaration. The variable shall be of type WORD and shall contain the PC summary status beginning with item number 1 at the least significant bit upwards.

If the PC summary status shall be presented as a variable with direct representation, the direct representation shall be %S and shall be of type WORD. It shall contain the PC summary status beginning with item number 1 at the least significant bit upwards.

If the complete status information shall be presented as a variable, it shall have the access path P_PCSTATUS pre-defined in the configuration section. This variable shall have a structured type as follows:

```
ARRAY [0..p_NOS] OF
    STRUCT
        SUBSYSTEM  : (SUMMARY, IO, PU, POWER, MEMORY, COMMUNICATION,
                IMPLEMENTER);
        NAME    : STRING[<Max_Name_Len>];
        STATE   : ARRAY[0..15] OF BOOL;
        SPECIFIC    : ARRAY[0..p_BIT] OF BOOL;
    END_STRUCT;
```

IEC 2253/2000

Figure 7 – Type description of status information

The array element with the number 0 shall contain the PC summary status, each element with a higher number shall contain the status of one subsystem. The sub-element SUBSYSTEM shall contain the type of the PC or of a subsystem. The sub-element NAME shall contain the name of the PC or of a subsystem. The implementer shall specify the supported maximum

length for name strings, i.e. the value of Max_Name_Len. The sub-element STATE shall contain the state information of the PC or of a subsystem as an array of BOOL in the same order as specified in tables 2 to 8. The implementer shall specify the number of elements of the array P_PCSTATUS i.e. the value of p_NOS, the supported types of subsystems, the semantic of the values in the sub-element STATE for the implementer specific subsystem, the size of the sub-element SPECIFIC, i.e. the value of p_BIT, and the semantic of the sub-element SPECIFIC.

The status information of each subsystem may be presented as a variable with direct representation %SC<n>, where <n> stands for a number between 0 (representing the PC summary status) and the number of subsystems p_NOS. The variable shall have the same internal representation as a variable with the type of the structure part of the type described in the figure above.

Additionally there may be a set of variables with direct representation %SU<n>, %SN<n>, %SS<n>, and %SI<n>. The <n> stands for a number between 0 (representing the PC summary status) and the number of subsystems p_NOS. The variables shall have the same internal representation as a variable with the type of one of the structure sub-elements of the type described in the figure above. In detail, the %SU<n> shall correspond to the sub-element SUBSYSTEM, %SN<n> to the sub-element NAME, %SS<n> to the sub-element STATE, and %SI<n> to the sub-element SPECIFIC.

6.2 Application specific functions

The remainder of this clause describes the functions which a PC provides to a control system, using the communication subsystem, as illustrated in figure 2.

PC communication function	PC as requester	PC as responder	Function block available
Device verification	yes	yes	yes
Data acquisition	yes	yes	yes
Control	yes	yes	yes
Synchronization between user applications	yes	yes	yes
Alarm reporting	yes	no	yes
Program execution and I/O control	no	yes	no
Application program transfer	no	yes	no
Connection management	yes	yes	yes

Each of these is treated separately in the remainder of this clause. Not all functions are available in all PCs. See clause 7 for the function block definitions.

There are some applications which combine the application categories defined below, for example, supervisory control and data acquisition.

The following elements, while usually provided by PCs, are outside the scope of this part of IEC 61131:

a) operator interface;

b) programming, testing, and modifying the application program, and program verification.

PCs have the ability to use operator interface devices. These devices are used by an operator to monitor or modify the controlled process or both. They may also be used by a client system to communicate with the operator.

Direct operator interface is when the client can communicate to the operator interface via the communication system with no application program interaction.

Programming is the process of creating a PC application program on an instruction by instruction or a function block by function block basis. Testing and modifying is the process of finding and removing errors ("bugs") in an existing application program by making changes to it. Program verification is the testing of a PC application program to verify that it performs the function(s) it was designed to do in the process environment.

6.2.1 Device verification

This function is provided to allow other devices to determine if the PC is able to perform its intended function in the automated system. A PC can provide status of itself and its subsystems. Status includes health and state information. A device may explicitly request status from the PC or the PC may initiate an unsolicited status report using services provided by the communication interface. See 6.1 for the definition of health and state information of a PC system and of its subsystems.

Table 10 – Device verification features

No.	Device verification
1	Provide status information
2	Initiate unsolicited status reports

6.2.2 Data acquisition

Data contained in a PC is presented as variables. This data may come from a variety of sources and may have a wide range of meanings. It can be obtained by the client through one of several methods.

a) Polled – The client reads the value of one or more variables at a time or condition determined by the client. The access to the variables may be controlled by the PC. Only selected variables are accessible over the network.

b) Programmed – The data is provided by the PC to the client at a time or condition determined by the PC application program.

c) Configured – The communications interface to the PC can be configured by a client to initiate a data transfer to the client.

The kinds of variables in the PC which are visible to the communication system are:

a) variables with direct representation;

b) other variables which have access paths (see IEC 61131-3 for the definition of access paths).

If the directly represented variables are accessible for communication these variables shall use the direct representation as an identifier. The PC server (i.e. the PC which owns the variables) can interpret the identifier using an implementer defined algorithm.

NOTE Variables with direct representation can be used like "normal" variables while programming an application program. An additional symbolic name may be assigned to a directly represented variable using the AT construct in the variable declaration (see IEC 61131-3).

Typically there are thousands of these variables with direct representation even in a smaller PC. It is not reasonable to hold the name and the address of all these variables in an object dictionary of a PC.

The PC system may restrict access to variables with direct representation. The conditions (size, location, etc.) under which each data type supported by the PC can be uninterruptedly accessed shall be specified by the implementer.

Table 11 – Data acquisition features

No.	Data acquisition
1	Variables with direct representation are accessible
2	Access paths on configuration level
3	Access paths on program level
4	Means to restrict access to variables with direct representation
5	Conditions for uninterruptible access to variables

6.2.3 Control

A PC may support two methods of control: parametric and interlocked.

Parametric control is when the operation of the PC is directed by writing values to variables residing in the PC. This change in operation is determined by either the application program or other local mechanisms.

The access to the variables is controlled by the PC which holds the variables. Only those variables which have the READ_WRITE qualifier selected in the access path declaration are accessible over the network for parametric control.

Interlocked control is when the client requests the server to execute an application operation and to inform the client of the result of the operation. There are two aspects of this service, the synchronization of the client and server, and the exchange of data between them.

In interlocked control, this data exchange occurs at synchronization points in the application program. This service can be used to have the effect of a remote procedure call from one application program to another. The timeline shown in figure 8 illustrates this.

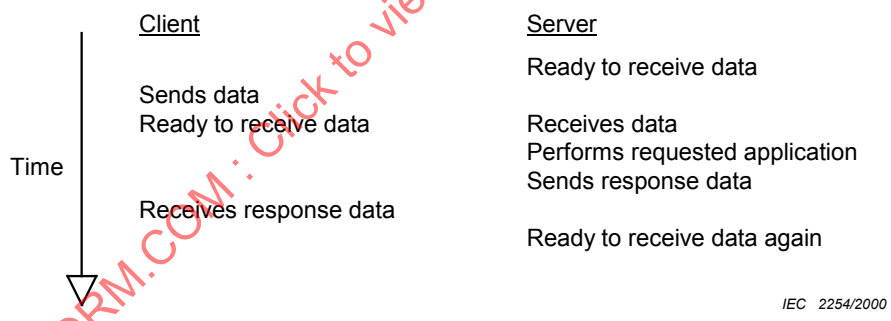


Figure 8 – Interlocked control timeline

The PC implements interlocked control using the SEND (client) and RCV (server) function blocks. Other devices may use other means to emulate the behavior of these function blocks to access this PC communication function.

Table 12 – Control features

No.	Control
1	Variables with direct representation are accessible
2	Access paths on configuration level
3	Access paths on program level
4	Means to restrict access to variables with direct representation
5	Conditions for uninterruptible access to variables
6	Interlocked control

6.2.4 Synchronization between user applications

User applications may need a synchronization service. For example, a user application may start the execution of another application after completion of an algorithm. The synchronization service is provided by the interlocked control mechanism (see 6.2.3).

6.2.5 Alarm reporting

The PC can have the ability to signal alarm messages to a client when a predetermined condition occurs. The client may indicate an acknowledgement of these alarms to the PC. This differs from normal data acquisition in that the state of an alarm point is remembered by the PC until it is acknowledged by the client. A summary of the unacknowledged alarms can be generated by the PC at the request of the client.

Table 13 – Alarm reporting features

No.	Alarm reporting
1	Signal messages
2	Receive acknowledgements
3	Generate summary of unacknowledged alarms

6.2.6 Application program execution and I/O control

Execution of a PC Application Program is managed by the Application Program Execution function (see 5.2). PC Application Programs can be started and stopped. PC Application Programs can be started either from an initial state or from the state they were in at the time they were stopped.

The PC application program in a PC system consists of one configuration and zero, one or more resources (see IEC 61131-3). Configurations and resources may be started and stopped. The resources are started and stopped when the configuration is started and stopped and they can be started or stopped independently of the configuration.

The Interface Function to Actuators (outputs) associated with a running Application Program can be directed to either use the values supplied by the Application Program or held in a known state. This Interface state is specified at the time the Application Program state is changed. The outputs can be directed to either be set to implementer specified states, hold the outputs in the current state, set all outputs to zero, or change some outputs to user specified states (on or off, with those not specified holding the last state) through an implementer specified mechanism (for example, tables, PC procedure, etc.).

The Interface Function to Sensors (inputs) associated with a running Application Program can be directed to either provide the actual data from the sensors or to continue to use previously supplied values. The input state is specified at the time the Application Program state is changed.

Table 14 – Startable and stoppable units

No.	Startable and stoppable unit
1	Configuration
2	Resource

The I/O (inputs and outputs) associated with a running resource shall either be controlled by the program or held in a known state, which is determined when the resource is started. Resources shall be able to be started either from an initial state (cold restart using START) or from the state they were in at the time they were stopped (warm restart using RESUME). The desired state of the outputs shall be able to be specified as part of the stopping process.

The state of the I/O can be set to the following values when a configuration or resource is started or stopped:

Table 15 – Meaning of I/O State

Value of I/O State	Meaning	Set by
Controlled	Actuators are being controlled by the application program or the part of the application program being started. Inputs are being supplied to the application program or the part of the application program being started by the interface function to sensors and actuators.	Starting
Hold outputs	Actuators are not being controlled by the application program or the part of the application program being started, they are held in the current state. Sensors are being supplied to the application program or the part of the application program being started by the interface function to sensors and actuators.	Starting
Hold current state	Actuators are not being controlled by the application program or the part of the application program being started or stopped, they are held in the current state. Sensors are not being supplied to the application program or the part of the application program being started, they are held in the current state.	Starting, stopping
Implementer state	Actuators are not being controlled by the application program or the part of the application program being stopped, they are held in a state, which was specified by the implementer. The application program is not running, therefore the state of the Sensor Interface is not specified.	Stopping
Zero outputs	Actuators are not being controlled by the application program or the part of the application program being stopped, they are held in the zero state. The application program is not running, therefore the state of the Sensor Interface is not specified.	Stopping
User specified	Actuators are not being controlled by the application program or the part of the application program being stopped, they are held in a state which was specified by the user. The application program is not running, therefore the state of the Sensor Interface is not specified.	Stopping

Table 16 – I/O state

No.	I/O state
1	Controlled
2	Hold outputs
3	Hold current state
4	Implementer specified
5	Zero outputs
6	User specified
NOTE The communication subsystem should not be depended upon as a replacement for hardwired emergency stop switches. Normal safety practices should be followed.	

Table 17 – Execution and I/O control features

No.	Execution and I/O control
1	Receive requests to start a startable and stoppable unit
2	Receive requests to stop a startable and stoppable unit
3	Receive requests to resume a startable and stoppable unit

6.2.7 Application program transfer

The application program is transferred using the application program storage and data storage function of a PC (see 5.2). Application program transfer allows the client to upload the complete contents of the programmable memory or portions thereof for archiving or verification or to download it for restoring the PC to a known state. This function also provides the ability to place the PC in a safe state before modifying the contents of its programmable memory, and restarting it in a safe manner when the application program transfer is completed.

The initiation of the program transfer is typically done by a device that is not a PC. The services include:

a) upload for archive;

b) upload for verification;

c) download for restore to previously known good system; and

d) download an off-line developed system.

The portions of the programmable memory which can be uploaded or downloaded are given in table 18.

NOTE A load unit contains the variable P_DDATE, their value is the date of the last modification of the load unit.

Table 18 – Loadable units

No.	Loadable units
1	Configuration
2	Resource
3	Programs
4	Global variables
5	Access paths on configuration level
6	Access paths on program level
7	Load units contain the variable P_DDATE

The implementer shall specify if other language elements, for example, function types or function block types are loadable and the conditions and restrictions for downloading and uploading these.

This part of IEC 61131 defines a means to perform uploads and downloads on the whole PC, a subsystem of the PC and a portion of a subsystem. The whole or the portion of the PC to be downloaded shall be explicitly stopped before the download. The whole or the portion of the system being downloaded shall not be available for other uses until the download is completed. The implementer shall specify what other clients can do with a PC when one client is downloading it.

Table 19 – Application program transfer features

No.	Application program transfer
1	Receive requests to download a loadable unit
2	Receive requests to upload a loadable unit

6.2.8 Connection management

The Connection Management provides the means to install, to maintain and close communications connections to a remote communication partner. If multiple requests are initiated to operate with a device, they may all work over the same connection. Not all communications subsystems require connections, for example, point-to-point communication.

Connections are controlled explicitly by the application program using the CONNECT function block or are provided by the communication subsystem if and when needed.

Table 20 – Connection management features

No.	Connection management
1	Install connections
2	Close connections
3	Using one connection for multiple requests

7 PC communication function blocks

7.1 Overview of the communication function blocks

The following PC communication functions and their corresponding function blocks are described below.

Table 21 – Overview of the communication function blocks

No.	Subclause	Name of communication function block or function
1	7.2 Semantic of communication FB parameters (addressing of remote variables)	REMOTE_VAR
2 3	7.3 Device verification	STATUS, USTATUS
4	7.4 Polled data acquisition	READ,
5 6 7 8	7.5 Programmed data acquisition	USEND, URCV, BSEND, BRCV
9	7.6 Parametric control	WRITE,
10 11	7.7 Interlocked control	SEND, RCV
12 13	7.8 Programmed alarm report	NOTIFY, ALARM
14	7.9 Connection management	CONNECT

The numbers given in the above table shall be used to state compliance to these communication function blocks (CFB).

7.1.1 Device verification

The STATUS and USTATUS function blocks are provided so that the PC can collect status from other devices. These are provided to allow the PC to determine if the other devices are able to perform their intended function in the automated system.

7.1.2 Data acquisition

Data contained in other devices may be presented as variables. This data may come from a variety of sources and may have a wide range of meanings. It can be obtained by the PC through one of two methods using communication function blocks.

a) Polled – The PC uses the READ function block to obtain the value of one or more variables at a time or condition determined by the PC application program. The access to the variables may be controlled by the device being read.

b) Programmed – The data is provided to the PC at a time or condition determined by the other device. The PC uses the URCV function block to provide the data to the PC application program. The PC uses the USEND to provide unsolicited data to other devices.

The conditions (size, location, etc.) under which each data type supported by the other device can be uninterruptedly accessed is determined by the other device.

7.1.3 Control

Two methods of control shall be supported by PCs: parametric and interlocked.

Parametric control is when the operation of the other device is directed by writing values to variables residing in it. The access to the variables may be controlled by the PC which holds the variables. The PC uses the WRITE function block to perform this action from the PC application program.

Interlocked control is when the client requests the server to execute an application operation and to inform the client of the result of the operation. The PC uses the SEND and RCV function blocks to implement the client and server roles, respectively.

7.1.4 Alarm reporting

The PC can have the ability to signal alarm messages to a client when a predetermined condition occurs. The client may indicate an acknowledgement of these alarms to the PC. The ALARM and NOTIFY function blocks are used by the PC application program to generate acknowledged and unacknowledged alarms, respectively.

7.1.5 Connection management

The PC application program uses the CONNECT function block to manage connections.

7.2 Semantic of communication FB parameters

The communication function blocks use a common semantic of their function block inputs and outputs. The meaning of these inputs and outputs is described below. Some communication function blocks have special input or output parameters, they are described where the communication function blocks themselves are described.

Table 22 – Semantic of communication FB parameters

Parameter name	Data type of the parameter	Interpretation
EN_R	BOOL	Enabled to receive data
REQ/RESP	BOOL	Perform function on raising edge
ID	COMM_CHANNEL	Identification of the communication channel
R_ID	STRING	Identification of the remote FB inside the channel
SD_i	ANY	User data to send
VAR_i	STRING or data type of the output of the function REMOTE_VAR	Identification of a variable of the remote communication partner
DONE	BOOL	Requested function performed (good and valid)
NDR	BOOL	New user data received (good and valid)
ERROR	BOOL	New non-zero status received
STATUS	INT	Last detected status (error or good)
RD_i	ANY	Last received user data

The ID input references the communication channel used by the instance of the communication function block, i.e. it determines the remote communication partner. The ID input is of COMM_CHANNEL type which shall be implementer defined.

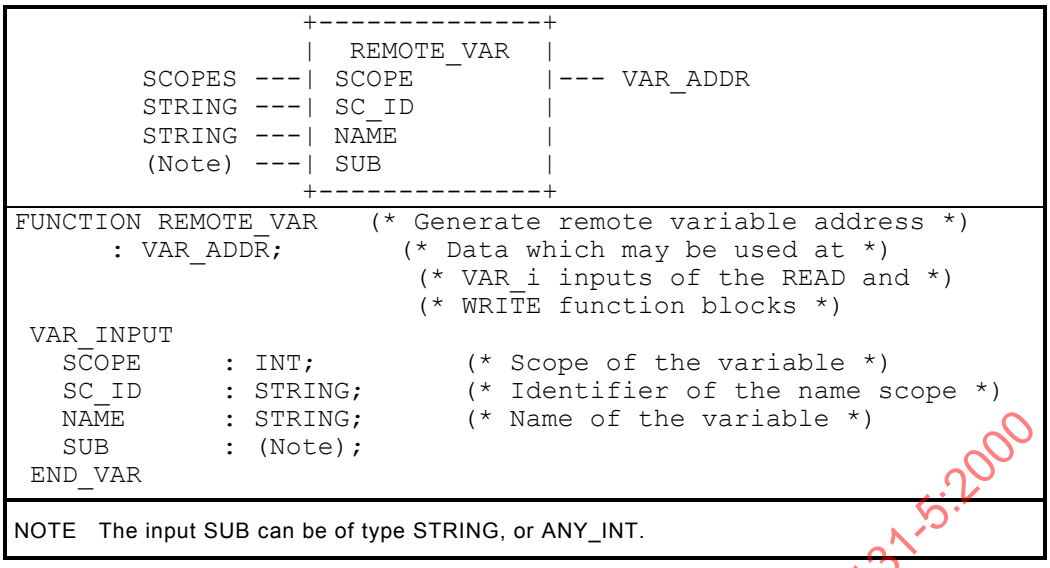
NOTE The value given at the ID input of a communication function block instance is intended to hold or reference the information which is necessary to manage the communication to the remote communication partner. This information may be dependent on the implementation and the communication subsystem used.

The R_ID input is used to identify the corresponding instance of the communication function block at the remote partner, if the PC communication function is provided by a corresponding pair of function block instances.

One instance of a communication function block shall use the same communication channel and communicate to the same corresponding remote function block instance throughout its whole lifetime.

The variables to be read or written are identified using the VAR_i inputs of the READ and WRITE function blocks. The actual parameter is typically a string which contains the name of the remote variable.

Additionally the VAR_i parameter may also have an implementer defined data type named VAR_ADDR. A function REMOTE_VAR is defined to generate the access information for nested variables.




```
FUNCTION REMOTE_VAR    (* Generate remote variable address *)
      : VAR_ADDR;        (* Data which may be used at *)
                          (* VAR_i inputs of the READ and *)
                          (* WRITE function blocks *)
 VAR_INPUT
   SCOPE      : INT;          (* Scope of the variable *)
   SC_ID      : STRING;       (* Identifier of the name scope *)
   NAME       : STRING;       (* Name of the variable *)
   SUB        : (Note);
 END_VAR
```

NOTE The input SUB can be of type STRING, or ANY_INT.

IEC 2255/2000

Figure 9 – Function REMOTE_VAR

SCOPE is an integer which identifies the name scopes of the programming languages of IEC 61131-3, the communication system, or the implementer supports as scope of a remote variable.

Table 23 – Values of the SCOPE parameter

Name scope of IEC 61131-3	Value
Configuration	0
Resource	1
Program	2
Function block instance	3
Reserved for future standardization	4 .. 9
Reserved for name scopes specific to communication subsystems	10 .. 99
Implementer specific	<0, > 99

If the SCOPE of the variable requires an identifier, the SC_ID is used to supply that identifier. NAME contains the name of the remote variable. If SUB is of string type, the value of the string is interpreted as a sub-element name. If SUB is an ANY_INT, the value is interpreted as an index. If SUB is an empty string, the complete variable is addressed.

The data type of the function output is implementer defined. The result of the function may be passed to the VAR_i inputs of the READ and WRITE function block or may be stored in a variable of the same data type. The use of the REMOTE_VAR function may be restricted only to produce valid values for the VAR_i parameters of the READ and WRITE function blocks.

There may be additional functions to support the communication subsystem specific or implementer specific addressing schemes which produce an output of VAR_ADDR type. The communication subsystem specific functions are specified in the mapping annexes of this part of IEC 61131.

All parameters of the communication function blocks are mandatory except the extensible parameters SD_i, RD_i, and VAR_i depending on the function block type. It is not requested

that the SD_i or RD_i parameters have the same data type if more than one SD_i parameter is used at one function block instance or if an SD_i parameter is used with different function block instances. The implementer shall specify the number of SD_i, RD_i, and VAR_i parameters which are supported with one invocation of a communication function block. Additionally, he shall describe if there are restrictions with the use of these parameters, for example data type or size of the actual parameters.

If a communication function block requests that data types are compatible, the compatibility check shall always be true if the same IEC 61131-3 data types are used at a client PC and a server PC. Additional communication subsystem specific compatibility rules may be defined.

The outputs are initialized with system zero. NDR, DONE, and ERROR pulse true until the next invocation of this instance. That is, each of the communication function blocks implies the following structure, which is not shown in the state diagrams of those function blocks.

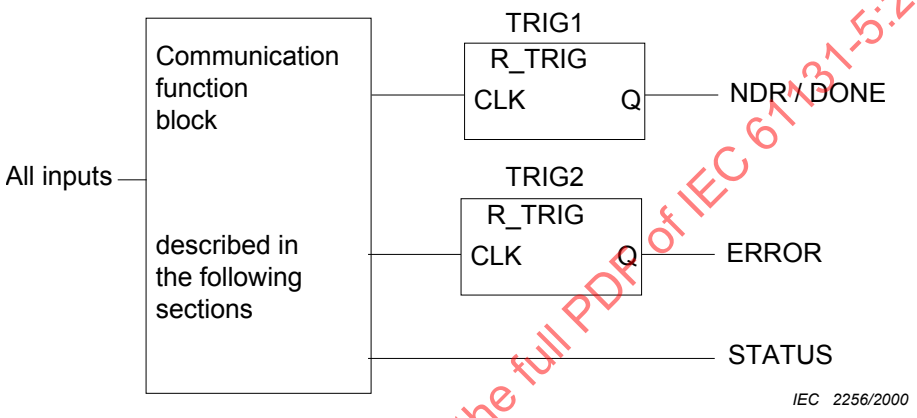


Figure 10 – Principle of status signalling

If a communication error of an instance of a communication function block is detected by the PC system or by the algorithm of the communication function block, the ERROR and the STATUS output of the function block instance are set. The ERROR output remains true during the time between two invocations of this instance (see figure 11).

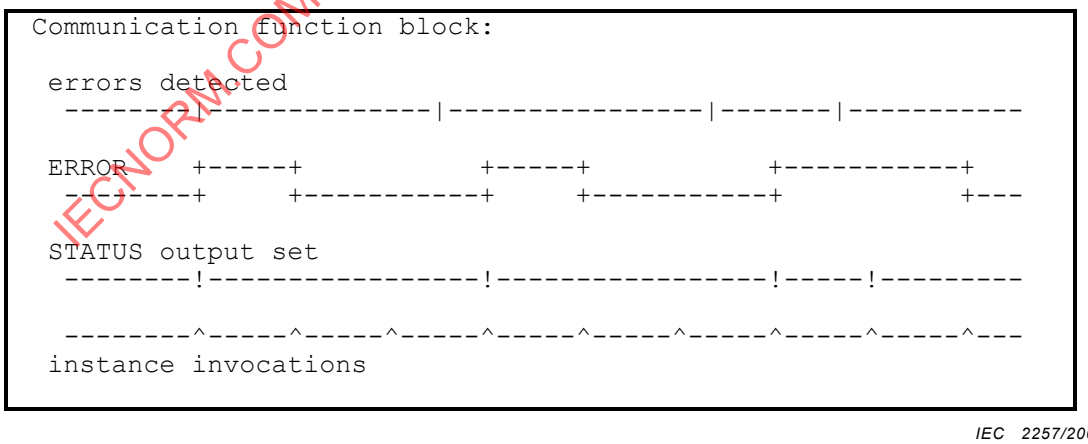


Figure 11 – Timing diagram of the ERROR and STATUS outputs

The NDR, DONE, ERROR and the STATUS output shall be set to new values only synchronously to the instance invocations of the communication function blocks, even if an error is detected in between.

The following values for the STATUS output of the various function blocks have been defined. Not all values are used by all function blocks.

Table 24 – Value and interpretation of the STATUS output

STATUS value	Interpretation
0	No error
1	Error of lower layers, no communication possible
2	Other negative response from remote communication partner
3	R_ID does not exist in the communication channel
4	Data type mismatch
5	Reset received
6	Receiver not enabled
7	Remote communication partner in wrong state
8	Access denied to remote object
9	Receiver overrun (user data are new)
10	Access to local object rejected
11	Requested service exceeds local resources
12 .. 20	Reserved for future standardization
–1	Instance of this function is busy and cannot provide additional services at this time
< –1 or > 20	Codes less than –1 or greater than 20 are to be specified by the implementer

If errors are received from one or more communication partners in the case of one-to-many or one-to-all communication, the STATUS parameter is set according to the first error received.

The RD_i parameters shall contain the received data. These parameters shall be input/output parameters.

The following subclauses contain a description of the communication function blocks. The representation of the function blocks is given in a graphical and a textual way, and the types and the meanings of the associated inputs and outputs are shown. The STATUS output shall take on the appropriate value as defined.

The normal operation is illustrated by a timing diagram.

The operation of the function blocks is described based on state diagrams. The transitions depending on the application program are mapped onto conditions of the function block inputs. The transitions depending on the communication system are described independent of the communication system used. The explicit mappings to certain communication systems is described in the annexes. The value of the function block outputs are given for each state of the state diagram.

Errors caused by the communication system or by local problems may occur asynchronously in all states of a communication function block. Only those errors are explicitly described in the state diagrams which cause state transitions or require actions. Otherwise, the errors shall only be signalled to the application program using the ERROR and STATUS outputs as shown in figures 10 and 11.

The communication function blocks need to be initialized. The initialization state INIT contained in all state diagrams shall be left at least before the return of the first invocation of the function block instance. In this state, all actions shall be done to enable communication. The communication channel to the remote communication partner shall be established, if the connection management of the channel is not explicitly programmed.

7.3 Device verification

The PC communication function device verification uses the STATUS and the USTATUS function blocks.

One instance of a STATUS or USTATUS function block provides one instance of the PC function device verification.

A PC can request a remote communication partner to send back to it its status information using the STATUS function block.

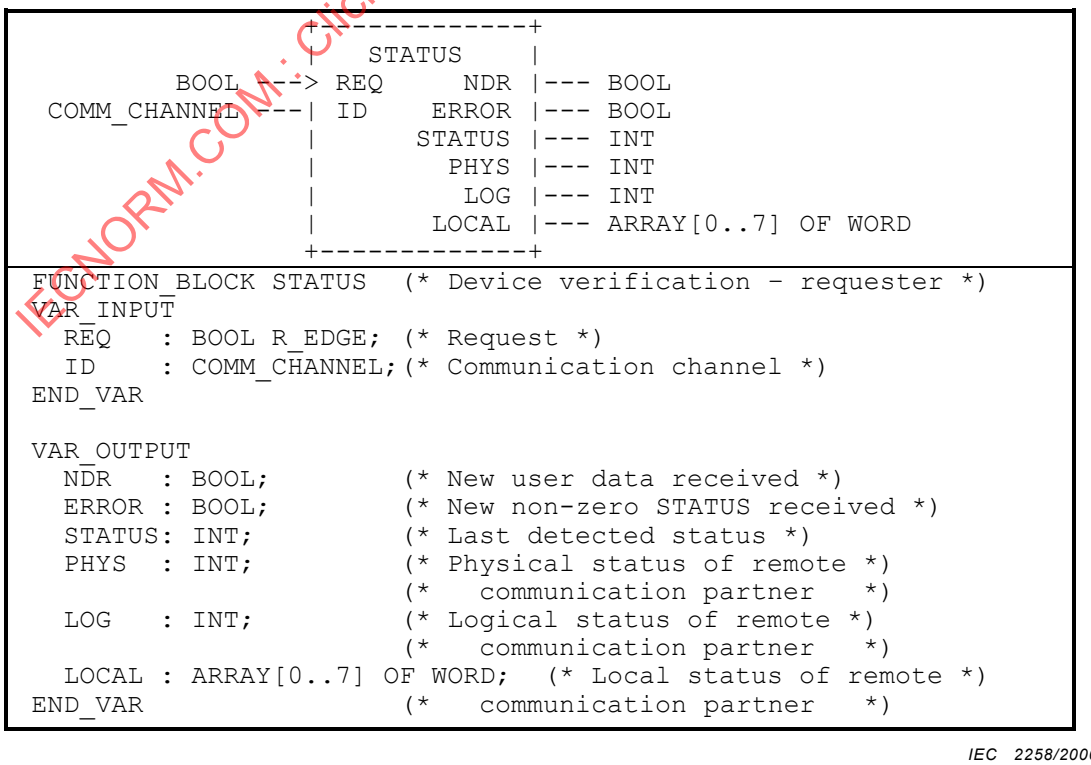
A PC can itself enable to receive status information of a remote communication partner using the USTATUS function block. The remote communication partner shall at least inform the USTATUS instance whenever its status information presented in the PHYS and LOG output changes.

The FB output PHYS contains the physical status of the remote device, the FB output LOG contains its logical communication status. The FB output LOCAL may contain additional status information up to 128 bits. The implementer shall specify the used length of this additional status information and shall define the semantics of it.

NOTE The READ function block can be used to obtain additional status information.

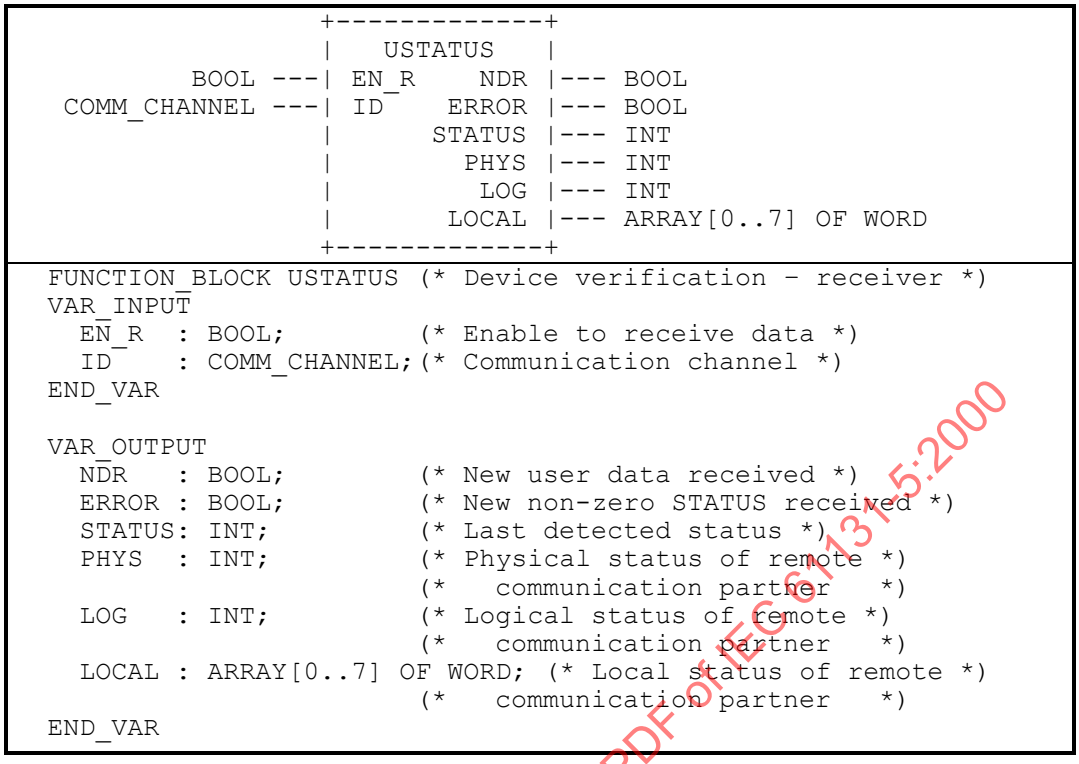
The ID parameter identifies the communication channel to the remote communication partner.

If an error occurred, the ERROR output pulses one cycle to indicate an error and the STATUS output contains the error code.



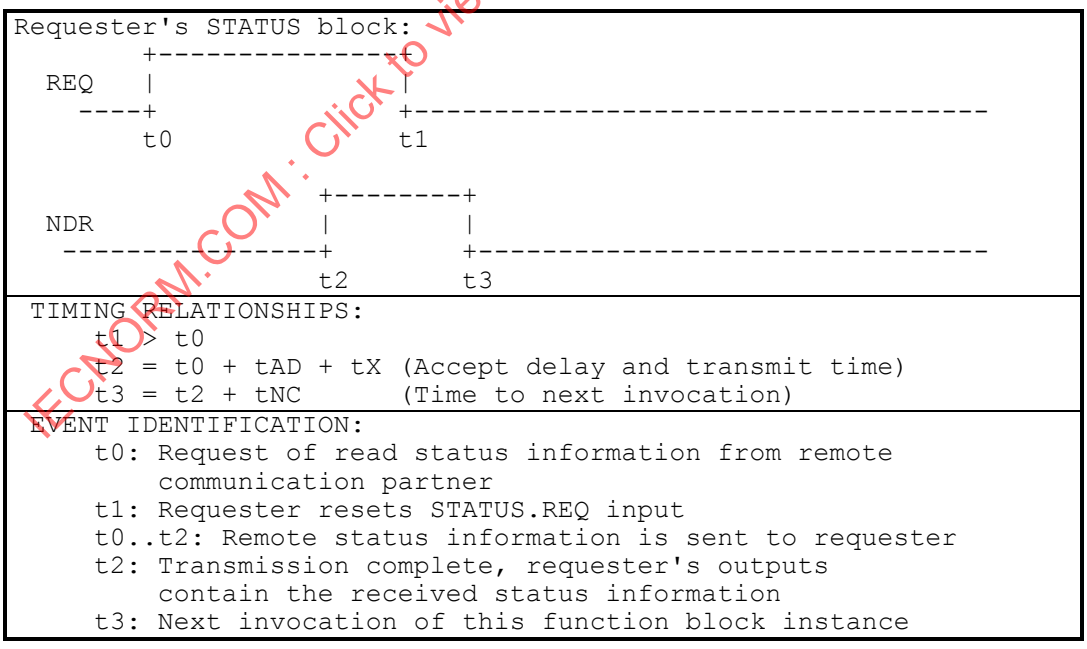
IEC 2258/2000

Figure 12 – STATUS function block



IEC 2259/2000

Figure 13 – USTATUS function block



IEC 2260/2000

Figure 14 – Timing diagram of the STATUS function block

The state diagram shown in figure 15 describes the algorithm of the STATUS function block. Tables 25 and 26 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the STATUS function block outputs.

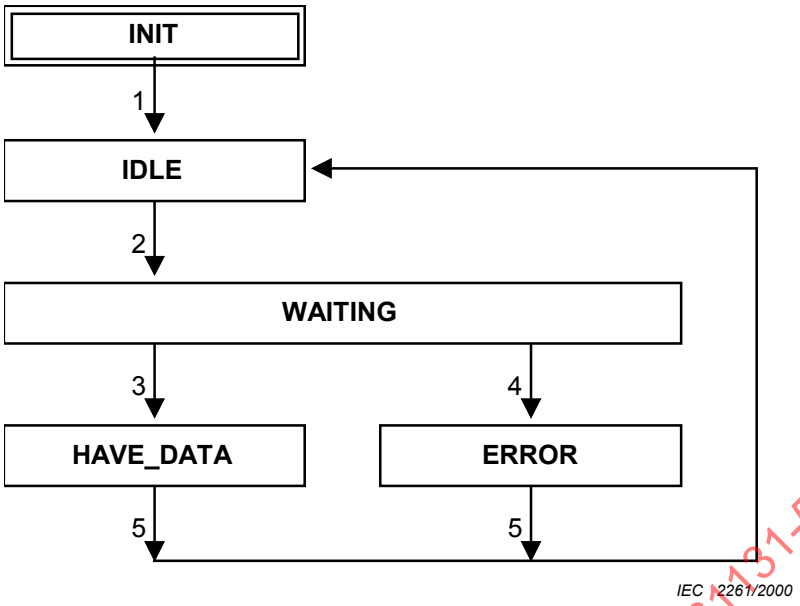


Figure 15 – State diagram of STATUS function block

Table 25 – Transitions of the STATUS state diagram

Transition	Condition
1	Initialization done
2	At raising edge of REQ input
3	Positive response from remote communication partner
4	Negative response from remote communication partner or communication problems detected
5	After next invocation of this instance

Table 26 – Action table for STATUS state diagram

		FB outputs			
State	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**PHYS, LOG, LOCAL**
INIT [a]	Initialize outputs	0	0	0	0
IDLE	No actions	0	0	---	---
WAITING	Request status information from remote communication partner	---	---	–1	---
HAVE_DATA	Deposit status information in instance	1	0	0	New status information
ERROR	Indicate error	0	1	[b]	---

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

The state diagram of figure 16 describes the algorithm of the USTATUS function block. Tables 27 and 28 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the USTATUS function block outputs.

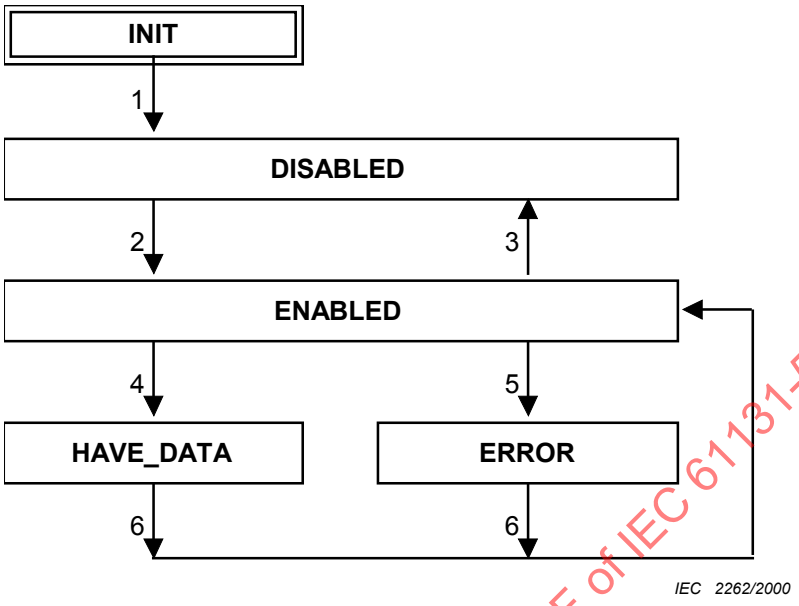


Figure 16 – State diagram of USTATUS function block

Table 27 – Transitions of USTATUS state diagrams

Transition	Condition
1	Initialization done
2	EN_R = 1
3	EN_R = 0
4	Status information received from remote communication partner
5	Communication problems detected
6	After next invocation of this instance

Table 28 – Action table of USTATUS state diagram

		FB outputs			
State	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**PHYS, LOG, LOCAL**
INIT [a]	Initialize outputs	0	0	0	0
DISABLED	No actions	0	0	---	---
ENABLED	No actions	0	0	---	---
HAVE_DATA	Deposit status information	1	0	0	New status information
ERROR	Indicate error	0	1	[b]	---

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

7.4 Polled data acquisition

The PC communication function polled data acquisition uses the READ function block.

One instance of a READ function block provides one instance of the PC function polled data acquisition.

The ID parameter identifies the communication channel to the remote communication partner.

The VAR_i inputs of the READ function block contain a string which can be interpreted by the remote communication partner as variable identifier. The remote communication partner sends the values of these variables back to the requesting READ instance. The READ passes the received variable values to its application program via its RD_i outputs. Each requested variable of the remote communication partner shall have a compatible data type to the variable programmed at the RD_i outputs of the READ instance. The VAR_i and RD_i parameters are extensible. At least VAR_1 and RD_1 shall be present.

If the remote communication partner is a PC, variables with an access path and variables with direct representation may be accessed with a READ function block. The variables with an access path are referenced in the VAR_ACCESS construction of the PC programming languages (see 2.7.1 of IEC 61131-3). The access name specified in this construction shall be used as the identifier of the variable in the VAR_i input. If a variable with direct representation shall be accessed with a READ function block, the VAR_i input shall contain the direct representation, for example %IW17, as a string. It shall be possible to mix the access to variables with an access path and with direct representation in one invocation of a READ function block instance.

If a variable shall be read via an access path which is declared inside a program (see 2.5.3 of IEC 61131-3) the REMOTE_VAR function shall be used. The name of the program instance shall be used at the SC_ID input, the name of the variable at the NAME input, for example to read the variable AB12 in the program DO7 the REMOTE_VAR function shall be invoked with REMOTE_VAR (2, "DO7", "AB12", "").

If a sub-element of a structured variable or an element of an array shall be read, the SUB input of the REMOTE_VAR function shall be used to identify this sub-element or element in the VAR_i inputs.

If an error occurred, the ERROR output pulses one cycle to indicate an error and the STATUS output contains the error code.

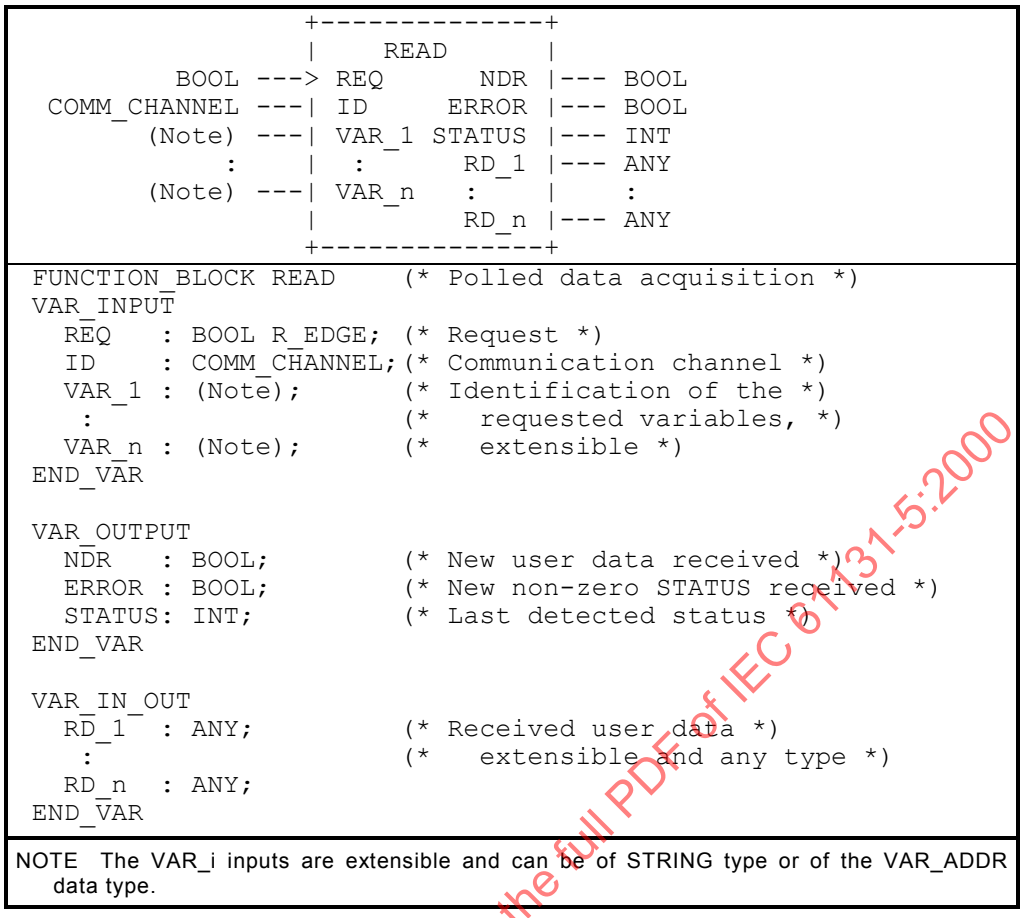


Figure 17 – READ function block

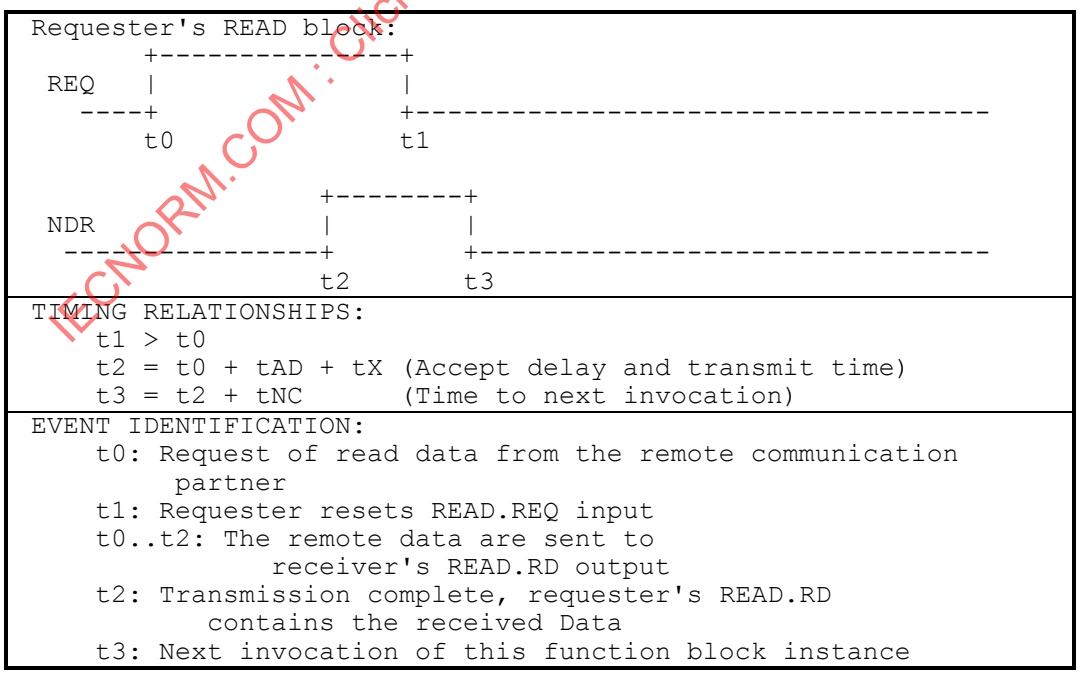


Figure 18 – Timing diagram of READ function block

The state diagram of figure 19 describes the algorithm of the READ function block. Tables 29 and 30 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the READ function block outputs.

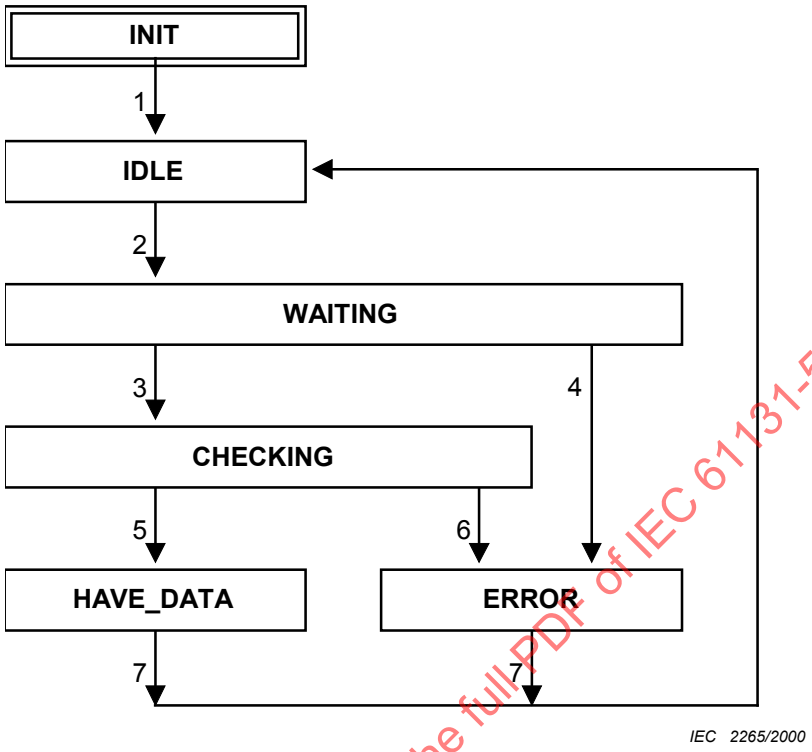


Figure 19 – State diagram of READ function block

Table 29 – Transitions of the READ state diagram

Transition	Condition
1	Initialization done
2	At raising edge of REQ input
3	Positive response from remote communication partner
4	Negative response from remote communication partner or other communication problems detected
5	Data types of RD_i and of received data match
6	Data types of RD_i and of received data do not match
7	After next invocation of this instance

Table 30 – Action table for READ state diagram

		FB outputs			
State	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1..RD..n**
INIT [a]	Initialize outputs	0	0	0	System null
IDLE	No actions	0	0	---	---
WAITING	Request variables from remote communication partner	0	0	–1	---
CHECKING	Verify data type match	0	0	---	---
HAVE_DATA	Deposit data	1	0	0	New data
ERROR	Indicate error	0	1	[b]	---

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

7.5 Programmed data acquisition

7.5.1 USEND/URCV function blocks

The PC communication function programmed data acquisition uses the USEND and the URCV function blocks.

Corresponding instances of one USEND and one URCV function block provide one instance of the PC function programmed data acquisition.

The USEND instance sends data to the URCV instance which may process this data with its application program. When requested the USEND instance takes the data from its SD_i inputs and transmits it to the corresponding URCV instance. Previously received data is overwritten. The URCV instance passes the received data to the application program via its RD_i outputs. This occurs whenever the application program requests its USEND instance to send data. The URCV instance passes newly received data whenever it gets one. It informs the application program when new data have arrived. The following data flow diagram illustrates this.

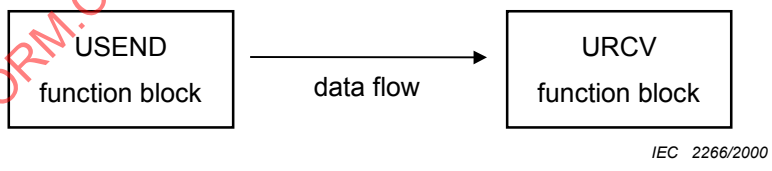


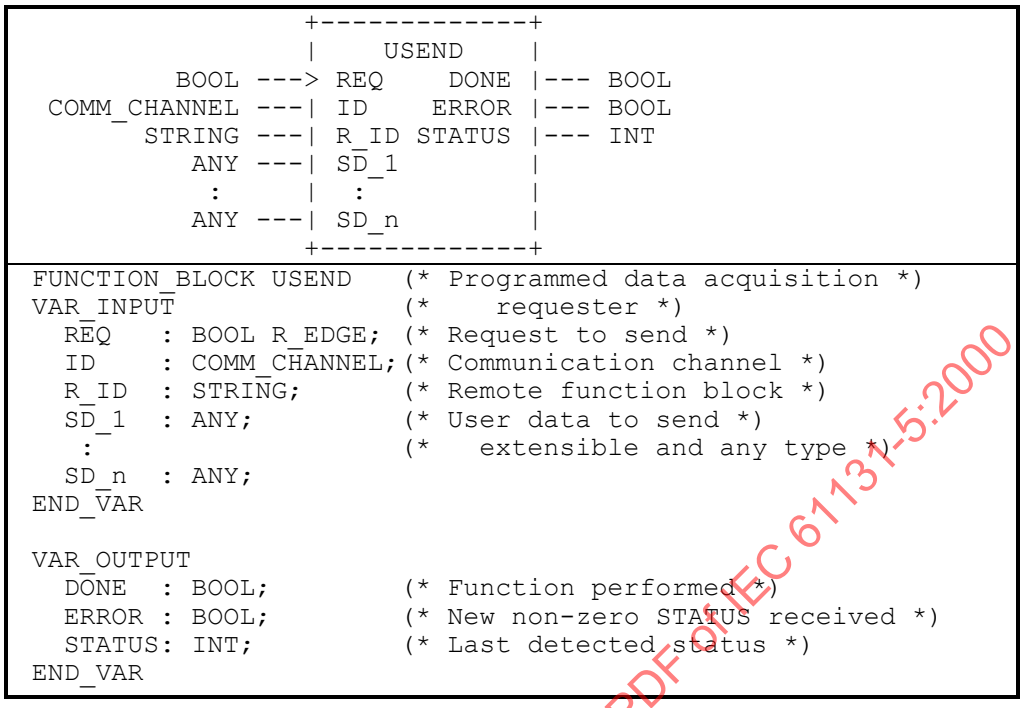
Figure 20 – Programmed data acquisition data flow

The SD_i and RD_i parameters are extensible. At least the SD_1 input at the FB USEND and the RD_1 output at the FB URCV shall be present. The number and each of the data types of the SD_i inputs of the USEND instance and the RD_i outputs of the corresponding URCV instance shall be compatible.

One USEND instance sends data to one URCV instance; that means, they are corresponding instances, if the value of the ID parameter references the same communication channel and if the value of the R_ID parameters are equal within the scope of this communication channel.

NOTE If the communication system provides communication channels which support one-to-many or one-to-all connections, these function blocks may be used to program a data acquisition function from one communication partner to many.

If an error occurred, the ERROR output pulses one cycle to indicate an error and the STATUS output contains the error code.

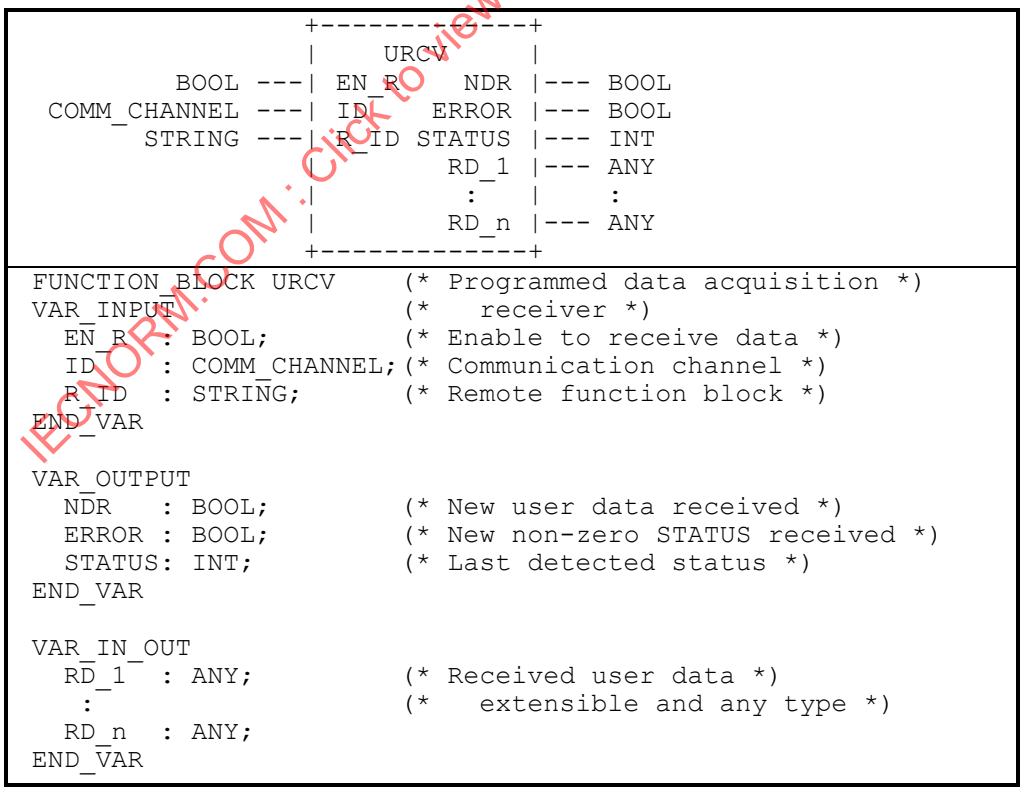



```
FUNCTION_BLOCK USEND    (* Programmed data acquisition *)
VAR_INPUT               (*    requester *)
  REQ    : BOOL R_EDGE; (* Request to send *)
  ID     : COMM_CHANNEL;(* Communication channel *)
  R_ID   : STRING;      (* Remote function block *)
  SD_1   : ANY;         (* User data to send *)
   :                    (*   extensible and any type *)
  SD_n   : ANY;
END_VAR

VAR_OUTPUT
  DONE   : BOOL;        (* Function performed *)
  ERROR  : BOOL;        (* New non-zero STATUS received *)
  STATUS: INT;          (* Last detected status *)
END_VAR
```

IEC 2267/2000

Figure 21 – USEND function block



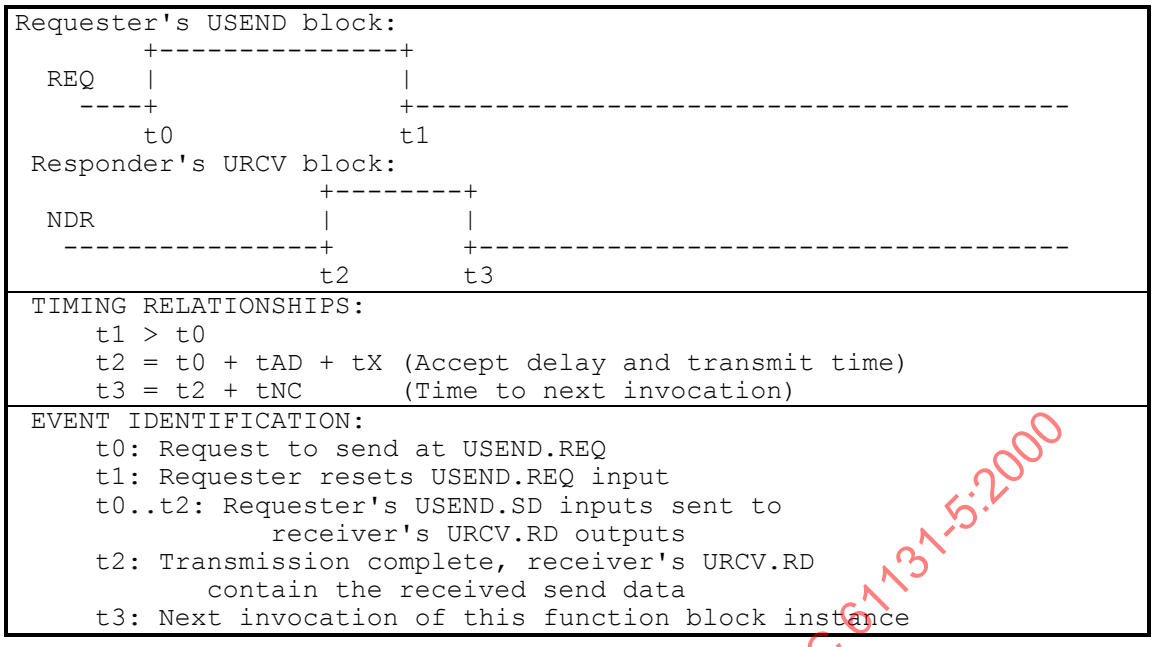

```
FUNCTION_BLOCK URCV     (* Programmed data acquisition *)
VAR_INPUT               (*   receiver *)
  EN_R   : BOOL;        (* Enable to receive data *)
  ID     : COMM_CHANNEL;(* Communication channel *)
  R_ID   : STRING;      (* Remote function block *)
END_VAR

VAR_OUTPUT
  NDR    : BOOL;        (* New user data received *)
  ERROR  : BOOL;        (* New non-zero STATUS received *)
  STATUS: INT;          (* Last detected status *)
END_VAR

VAR_IN_OUT
  RD_1   : ANY;         (* Received user data *)
   :                    (*   extensible and any type *)
  RD_n   : ANY;
END_VAR
```

IEC 2268/2000

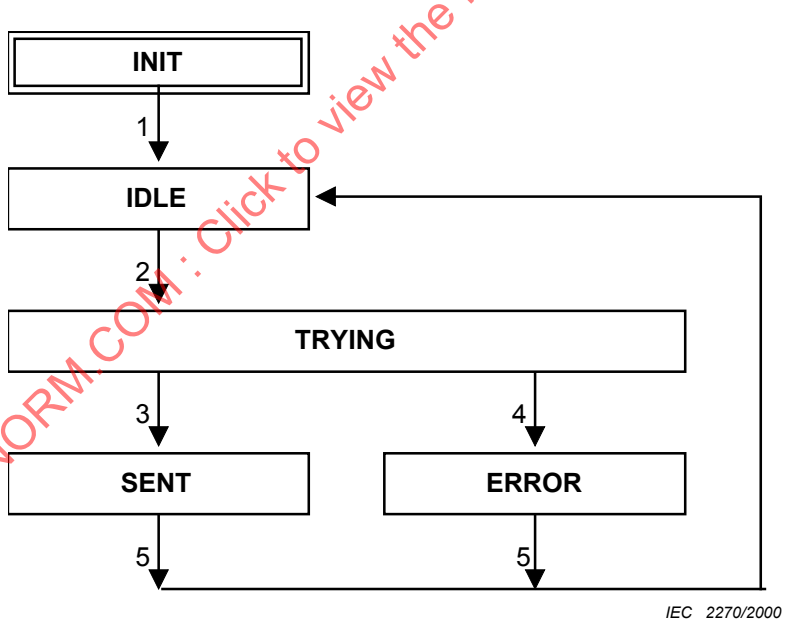
Figure 22 – URCV function block



IEC 2269/2000

Figure 23 – Timing diagram of USEND and URCV function blocks

The state diagram shown in figure 24 describes the algorithm of the USEND function block. Tables 31 and 32 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the USEND function block outputs.



IEC 2270/2000

Figure 24 – State diagram of USEND function block

Table 31 – Transitions of the USEND state diagram

Transition	Condition
1	Initialization done
2	At raising edge of REQ input
3	Communication system indicates "Sent to Remote Communication Partner"
4	Communication system indicates "Cannot Send to Remote Communication Partner" or other communication problems detected
5	After next invocation of this instance

Table 32 – Action table for USEND state diagram

		FB outputs		
State	**Actions**	**DONE** [c]	**ERROR** [c]	**STATUS**
INIT [a]	Initialize outputs	0	0	0
IDLE	No actions	0	0	---
TRYING	Send data given at the SD_i inputs to remote communication partner	---	---	---
SENT	Clear error indication	1	0	0
ERROR	Indicate error	0	1	[b]

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

The state diagram of figure 25 describes the algorithm of the URCV function block. Tables 33 and 34 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the URCV function block outputs.

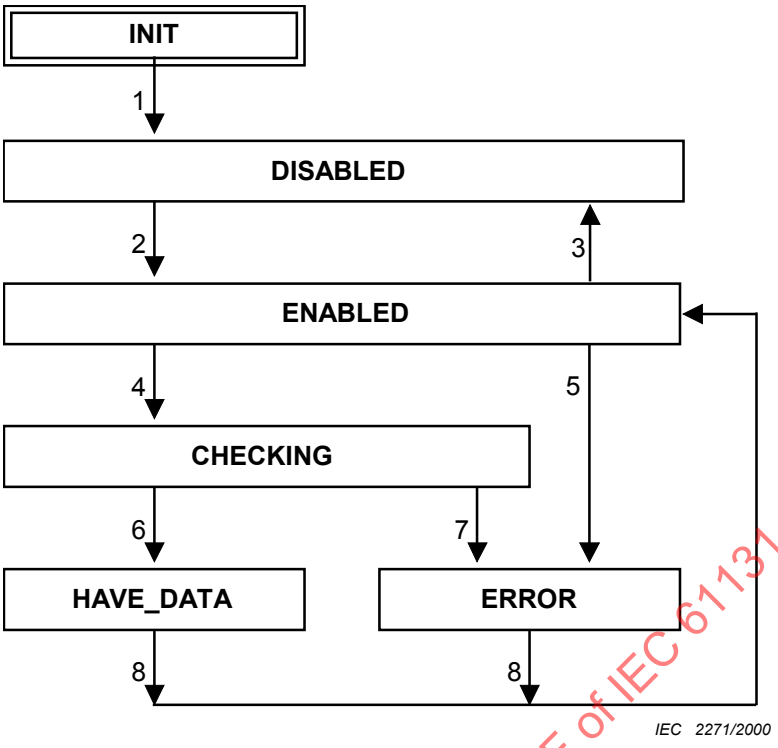


Figure 25 – State diagram of URCV function block

Table 33 – Transitions of URCV state diagrams

Transition	Condition
1	Initialization done
2	EN_R = 1
3	EN_R = 0
4	Data received from remote communication partner
5	Communication problems detected
6	Data types of SD_i of USEND and RD_i of URCV match
7	Data types of SD_i of USEND and RD_i of URCV do not match
8	After next invocation of this instance

Table 34 – Action table of URCV state diagram

		FB outputs			
State	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1..RD..n**
INIT [a]	Initialize outputs	0	0	0	System null
DISABLED	No actions	0	0	---	---
ENABLED	No actions	---	---	---	---
CHECKING	Verify data type match	---	---	---	---
HAVE_DATA	Deposit data	1	0	0, 9	New data
ERROR	Indicate error	0	1	[b]	---

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

7.5.2 BSEND / BRCV Function Blocks

Corresponding instances of one BSEND and one BRCV function block provide one instance of the PC function programmed data acquisition.

The BSEND instance sends data to the BRCV instance, which may process this data with its application program. When requested the BSEND instance takes the data from its SD_1 input and transmits it to the corresponding BRCV instance. Previously received data is overwritten. The BRCV instance passes the received data to the application program via its RD_1 output. This occurs when the application program requests its BSEND instance to send data. The BRCV instance passes newly received data when it has finished its previous request and gets new data. It informs the application program when new data have arrived.

The SD_1 input of the BSEND instance and the RD_1 output of the BRCV instance shall both be of data type ANY and are interpreted as a sequence of bytes.

NOTE The representation of the data in the controller is typically dependent on the implementation. The communication partners shall agree in the interpretation of any data if data of data type other than array of byte are transferred. This restricts the interoperability of programs using these communication function blocks.

The BSEND instance shall send as many bytes out of the data given at the SD_1 input as given at the LEN input of the BSEND instance. If the LEN input has the value 0 the complete variable given at the SD_1 input shall be transferred. The RD_1 output shall be able to store the data sent. The LEN output of the BRCV instance shall contain the count of bytes received in the RD_1 output. An error shall be signalled, if

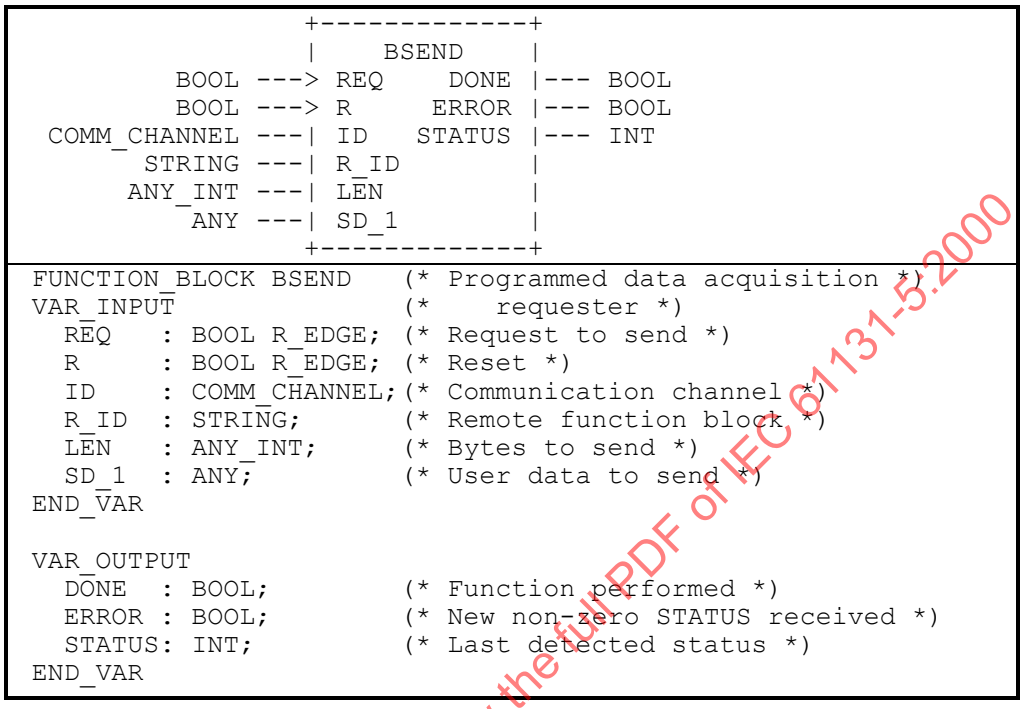
– either the value at the LEN input of the BSEND instance is greater than the total length in bytes of the variable connected to the SD_1 input,
– or the total length in bytes of the received data is greater than the total length in bytes of the variable connected to the RD_1 output of the BRCV instance.

The data transfer phase of the BSEND instance starts with the raising edge at the REQ input. It ends when the DONE or the ERROR output is set to 1. The DONE output shall be set to 1 for one cycle after the complete block of data to be sent is completely transmitted. During the data transfer phase, the implementer may restrict the access to the variable containing the data to be sent. The RD_1 output of the BRCV instance is valid when the NDR output has the value 1.

One BSEND instance sends data to one BRCV instance; that means, they are corresponding instances, if the value of the ID parameter reference the same communication channel and if the value of the R_ID parameters are equal within the scope of this communication channel.

The data transfer shall be cancelled if a raising edge is detected at the R input of the BSEND instance. If the data transfer is cancelled the ERROR and the STATUS outputs of the BRCV instance are set. In case the values of the RD_1 and LEN outputs are undefined.

If an error occurred, the ERROR output pulses one cycle to indicate an error and the STATUS output contains the error code.

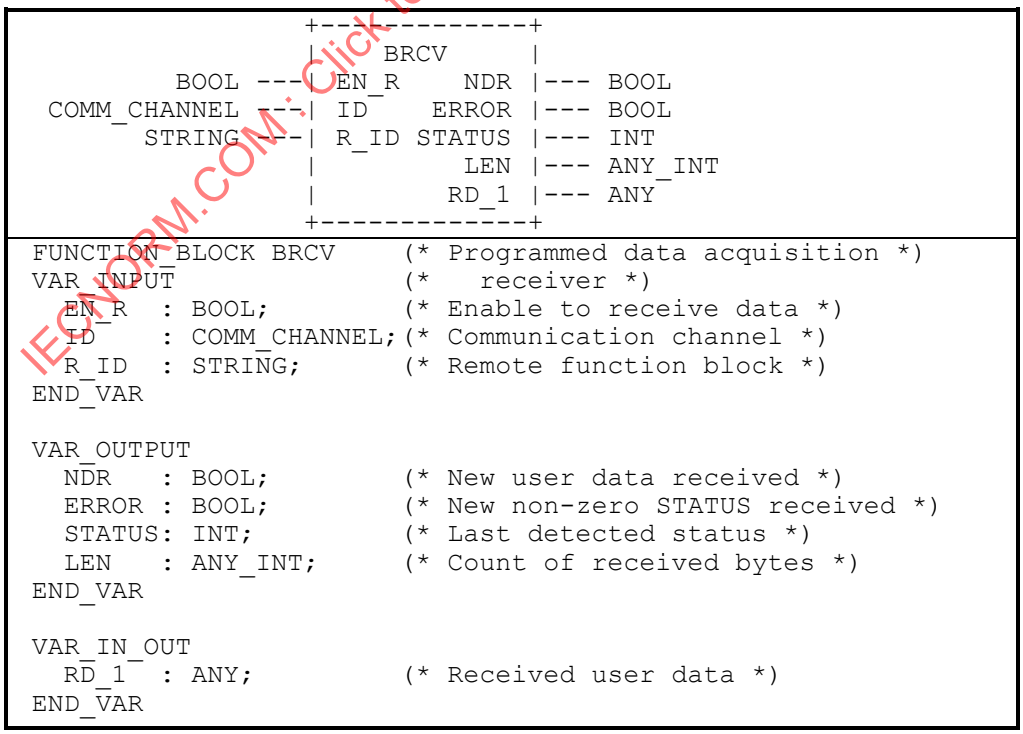



```
FUNCTION_BLOCK BSEND    (* Programmed data acquisition *)
VAR_INPUT               (*    requester *)
  REQ    : BOOL R_EDGE; (* Request to send *)
  R      : BOOL R_EDGE; (* Reset *)
  ID     : COMM_CHANNEL;(* Communication channel *)
  R_ID   : STRING;      (* Remote function block *)
  LEN    : ANY_INT;     (* Bytes to send *)
  SD_1   : ANY;         (* User data to send *)
END_VAR

VAR_OUTPUT
  DONE   : BOOL;        (* Function performed *)
  ERROR  : BOOL;        (* New non-zero STATUS received *)
  STATUS: INT;          (* Last detected status *)
END_VAR
```

IEC 2272/2000

Figure 26 – BSEND function block




```
FUNCTION_BLOCK BRCV     (* Programmed data acquisition *)
VAR_INPUT               (*   receiver *)
  EN_R   : BOOL;        (* Enable to receive data *)
  ID     : COMM_CHANNEL;(* Communication channel *)
  R_ID   : STRING;      (* Remote function block *)
END_VAR

VAR_OUTPUT
  NDR    : BOOL;        (* New user data received *)
  ERROR  : BOOL;        (* New non-zero STATUS received *)
  STATUS: INT;          (* Last detected status *)
  LEN    : ANY_INT;     (* Count of received bytes *)
END_VAR

VAR_IN_OUT
  RD_1   : ANY;         (* Received user data *)
END_VAR
```

IEC 2273/2000

Figure 27 – BRCV function block

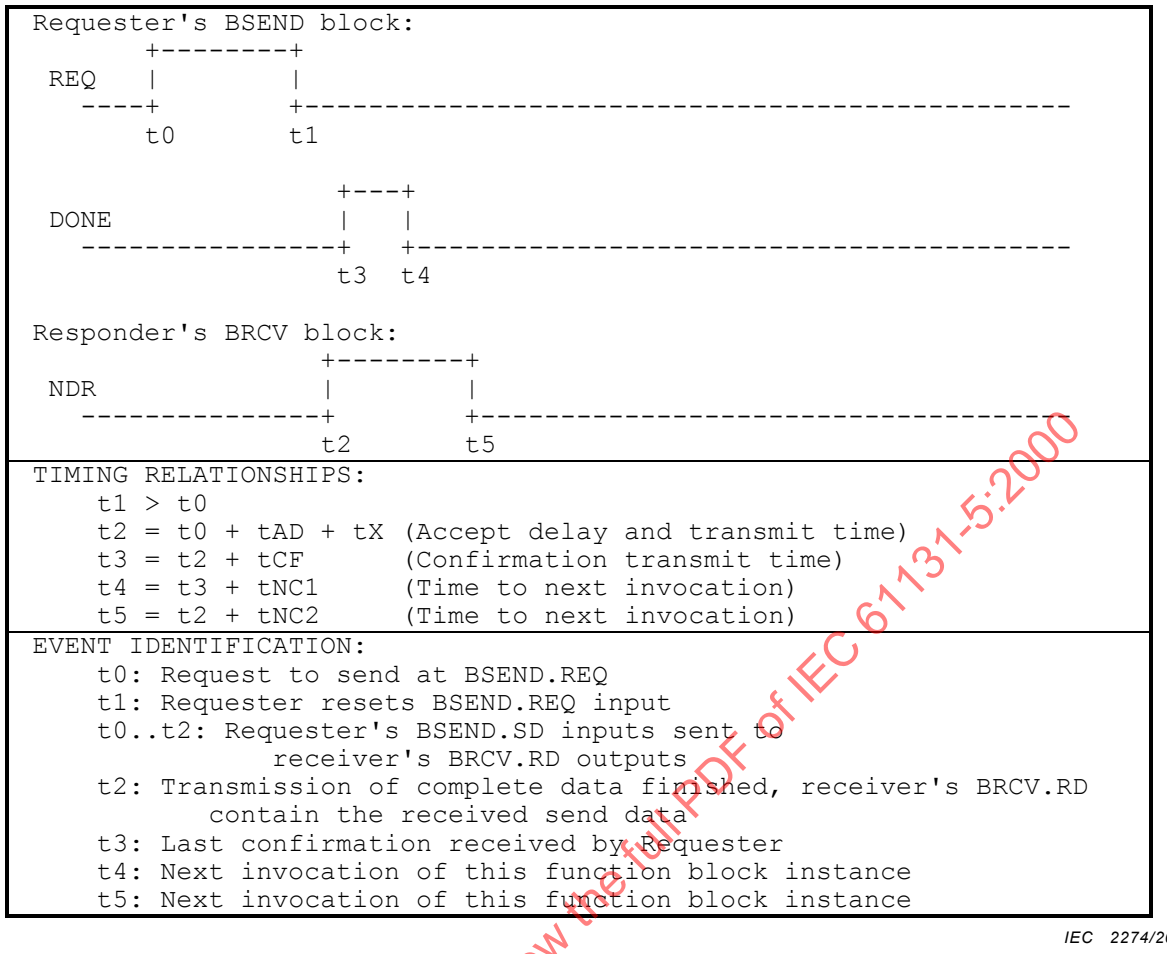


Figure 28 – Timing diagram of BSEND and BRCV function blocks

The state diagram shown in figure 29 describes the algorithm of the BSEND function block. Tables 35 and 36 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the BSEND function block outputs.

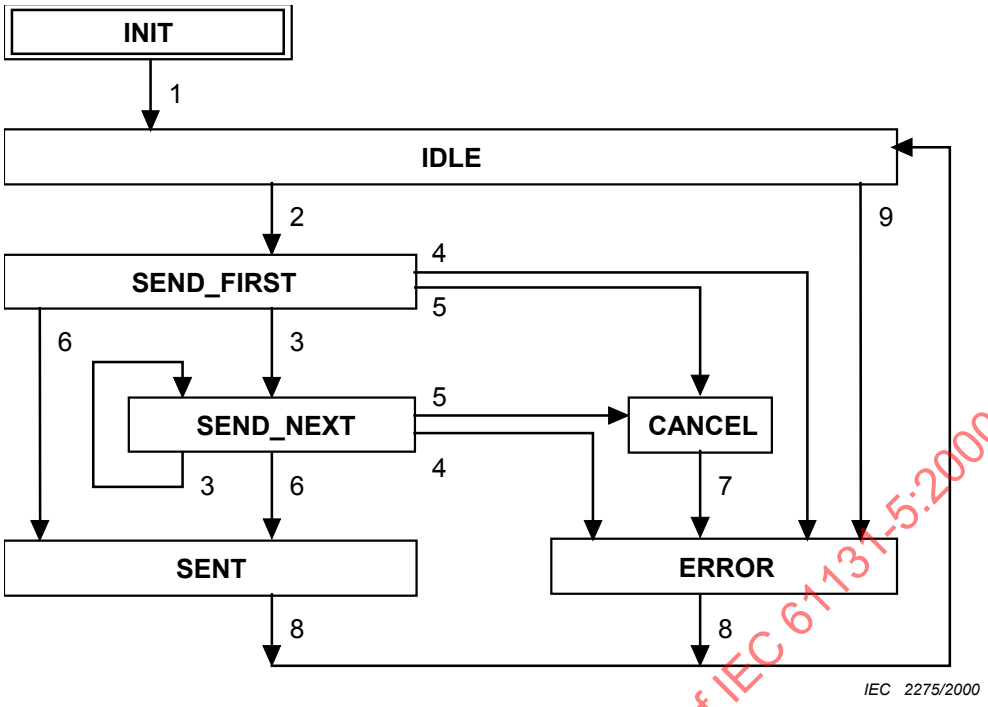


Figure 29 – State diagram of BSEND function block

Table 35 – Transitions of the BSEND state diagram

Transition	Condition
1	Initialization done
2	At raising edge of REQ input
3	Positive confirmation received and more data to send
4	Negative confirmation received or communication problems detected
5	At raising edge of R input
6	Positive confirmation received and no more data to send
7	Immediate
8	After next invocation of this instance
9	Communication problems detected

Table 36 – Action table for BSEND state diagram

		FB outputs		
State	**Actions**	**DONE** [c]	**ERROR** [c]	**STATUS**
INIT [a]	Initialize outputs	0	0	0
IDLE	No actions	0	0	---
SEND_FIRST	Send first block of data given at the SD_1 input to remote communication partner, send a maximum of LEN bytes in total	---	–1	---
SEND_NEXT	Send next block of data given at the SD_1 input to remote communication partner, send a maximum of LEN bytes in total	---	---	---
SENT	Clear error indication	1	0	0
CANCEL	Stop the data transfer	---	---	---
ERROR	Indicate error	0	1	[b]

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

The state diagram shown in figure 30 describes the algorithm of the BRCV function block. Tables 37 and 38 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the BRCV function block outputs.

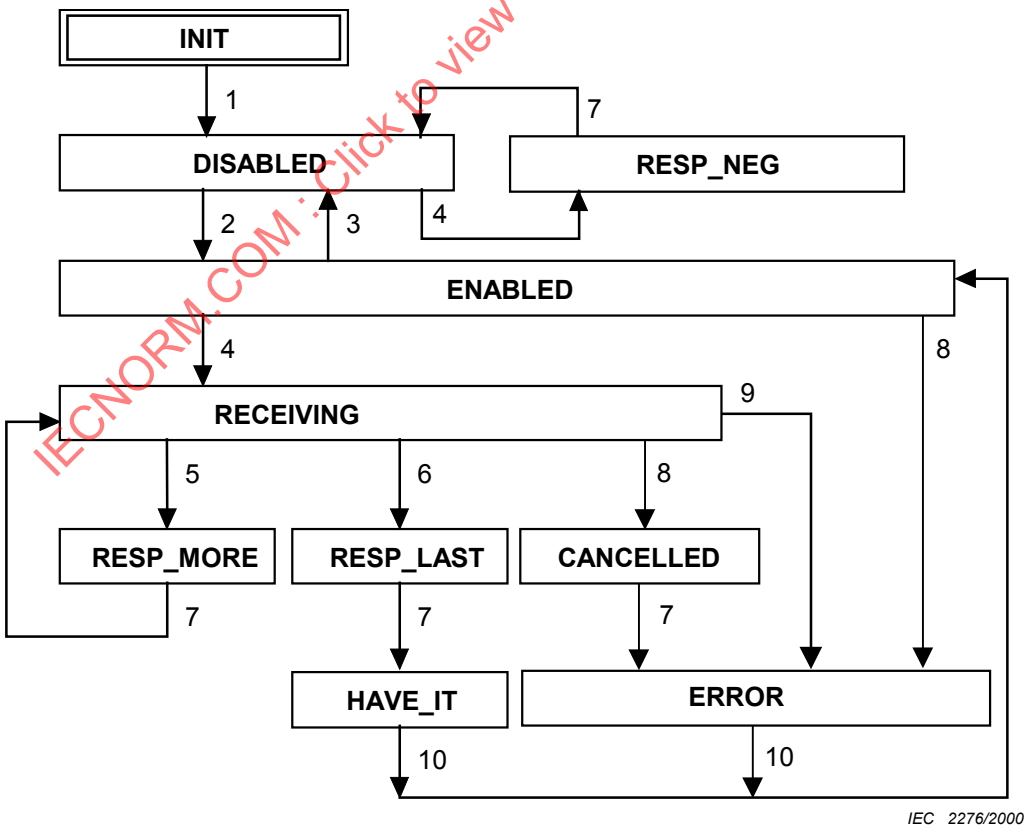


Figure 30 – State diagram of BRCV function block

Table 37 – Transitions of BRCV state diagrams

Transition	Condition
1	Initialization done
2	EN_R = 1
3	EN_R = 0
4	Data received from remote communication partner
5	More data follows is true
6	More data follows is false
7	Immediate
8	Communication problems detected
9	Indication received to cancel data transfer
10	After next invocation of this instance

Table 38 – Action table of BRCV state diagram

		FB outputs			
State	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1, LEN**
INIT [a]	Initialize outputs	0	0	0	System null
DISABLED	No actions	0	0	---	---
RESP_NEG	Send negative response	---	---	---	---
ENABLED	No actions	---	---	---	---
RECEIVING	Verify data can be stored and store at the given index	---	---	[d]	New data [d]
RESP_MORE	Send positive response	---	---	---	---
RESP_LAST	Send positive response	---	---	---	---
CANCELLED	Send positive response	---	---	5	---
HAVE_IT	Deposit data	1	0	0	New data
ERROR	Indicate error	0	1	[b]	---

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

[d] New data may be placed in the RD_1 output, in case the STATUS output shall be set to –1.

7.6 Parametric control

The PC communication function parametric control uses the WRITE function block.

One instance of a WRITE function block provides one instance of the PC function parametric control.

The ID parameter identifies the communication channel to the remote communication partner.

The VAR_i inputs of the WRITE function block contain a string which can be interpreted by the remote communication partner as variable identifier (access path) of it. The SD_i inputs reference the values to be written to the variables identified by the VAR_i inputs. The remote communication partner writes the values to these variables. The VAR_i and SD_i parameters are extensible. At least VAR_1 and SD_1 shall be present.

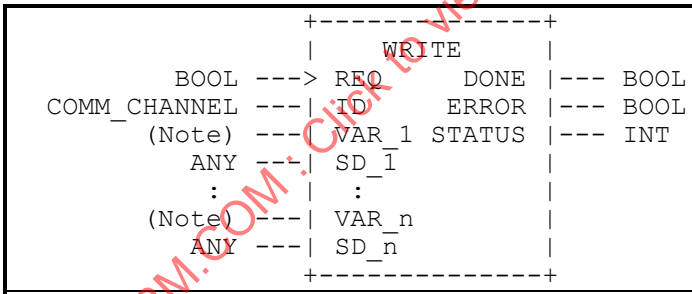
Each variable of the remote communication partner shall have the same data type as programmed at the SD_i inputs of the WRITE instance.

If the remote communication partner is a PC, variables with an access path and variables with direct representation may be accessed with a WRITE function block. The variables with an access path are referenced in the VAR_ACCESS construction of the PC programming languages (see 2.7.1 of IEC 61131-3). The access name specified in this construction shall be used as the identifier of the variable in the VAR_i input. If a variable with direct representation shall be accessed with a WRITE function block, the VAR_i input shall contain the direct representation, for example %IW17, as a string. It is possible to mix the access to variables with an access path and with direct representations in one invocation of a WRITE function block instance.

If a variable shall be written via an access path, which is declared inside a program (see 2.5.3 of IEC 61131-3), the REMOTE_VAR function shall be used. The name of the program instance shall be used at the SC_ID input, the name of the variable at the NAME input, for example to write the variable AB12 in the program DO7 the REMOTE_VAR function shall be invoked with REMOTE_VAR (2, "DO7", "AB12", "").

If a sub-element of a structured variable or an element of an array shall be written, the SUB input of the REMOTE_VAR function shall be used to identify this sub-element or element in the VAR_i inputs.

If an error occurred, the ERROR output pulses one cycle to indicate an error and the STATUS output contains the error code.




```
FUNCTION_BLOCK WRITE    (* Parametric control *)
VAR_INPUT
  REQ    : BOOL R_EDGE; (* Request *)
  ID     : COMM_CHANNEL;(* Communication channel *)
  VAR_1  : (Note);      (* Identifier and *)
  SD_1   : ANY;         (*   value of variable to be set *)
   :                    (*   extensible and any type *)
  VAR_n  : (Note);
  SD_n   : ANY;
END_VAR

VAR_OUTPUT
  DONE   : BOOL;        (* Function performed *)
  ERROR  : BOOL;        (* New non-zero STATUS received *)
  STATUS : INT;         (* Last detected status *)
END_VAR
```

NOTE The VAR_i inputs are extensible and can be of STRING type or of the VAR_ADDR data type.

IEC 2277/2000

Figure 31 – WRITE function block

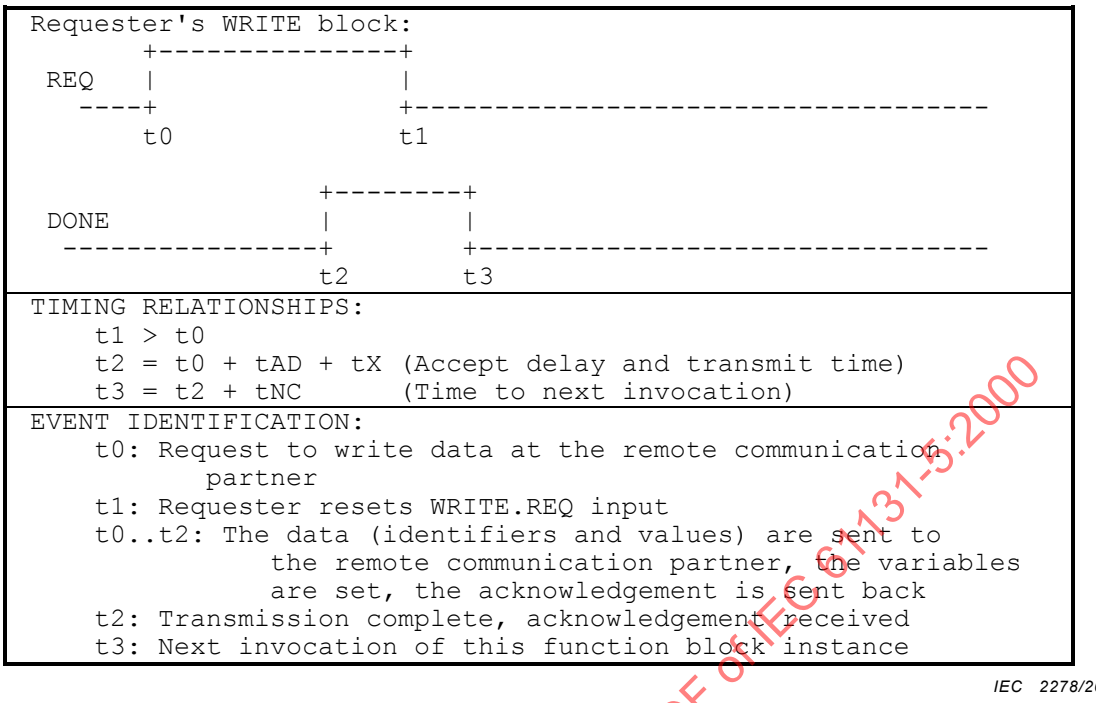


Figure 32 – Timing diagram of WRITE function block

The state diagram shown in figure 33 describes the algorithm of the WRITE function block. Tables 39 and 40 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the WRITE function block outputs.

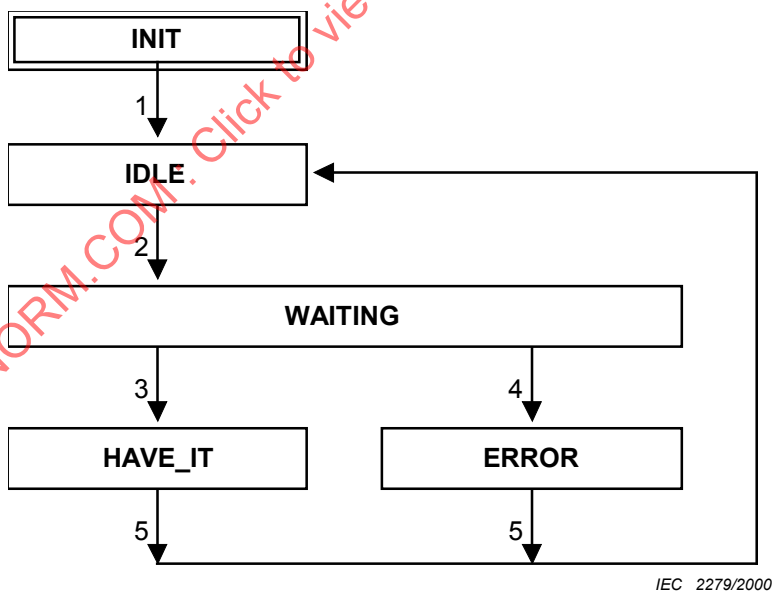


Figure 33 – State diagram of WRITE function block

Table 39 – Transitions of the WRITE state diagram

Transition	Condition
1	Initialization done
2	At raising edge of REQ input
3	Positive response of remote communication partner
4	Negative response from remote communication partner or other communication problems detected
5	After next invocation of this instance

Table 40 – Action table for WRITE state diagram

		FB outputs		
State	**Actions**	**DONE** [c]	**ERROR** [c]	**STATUS**
INIT [a]	Initialize outputs	0	0	0
IDLE	No actions	0	0	--
WAITING	Request to write variables into remote communication partner	---	---	–1
HAVE_IT	Indicate success	1	0	0
ERROR	Indicate error	0	1	[b]

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

7.7 Interlocked control

The PC communication function interlocked control uses the SEND and the RCV function blocks.

Corresponding instances of one SEND and of one RCV function block type provide one PC function interlocked control. Two instances of the SEND and RCV function block are corresponding, if the value of the ID parameters reference the same communication channel and if the value of the R_ID parameters are equal within the scope of this communication channel.

The SEND instance requests the RCV instance to execute an application operation and to inform the SEND instance of the result of the operation. This has two aspects, the synchronization of the application program of the SEND and RCV instances and the exchange of information between them. This function can be used to have the effect of a remote procedure call from one application program to another.

The interlocked control function is requested by a raising edge of the REQ input of the SEND function block instance. When requested the SEND instance takes the data from its SD_i inputs and transmits it to the corresponding RCV instance.

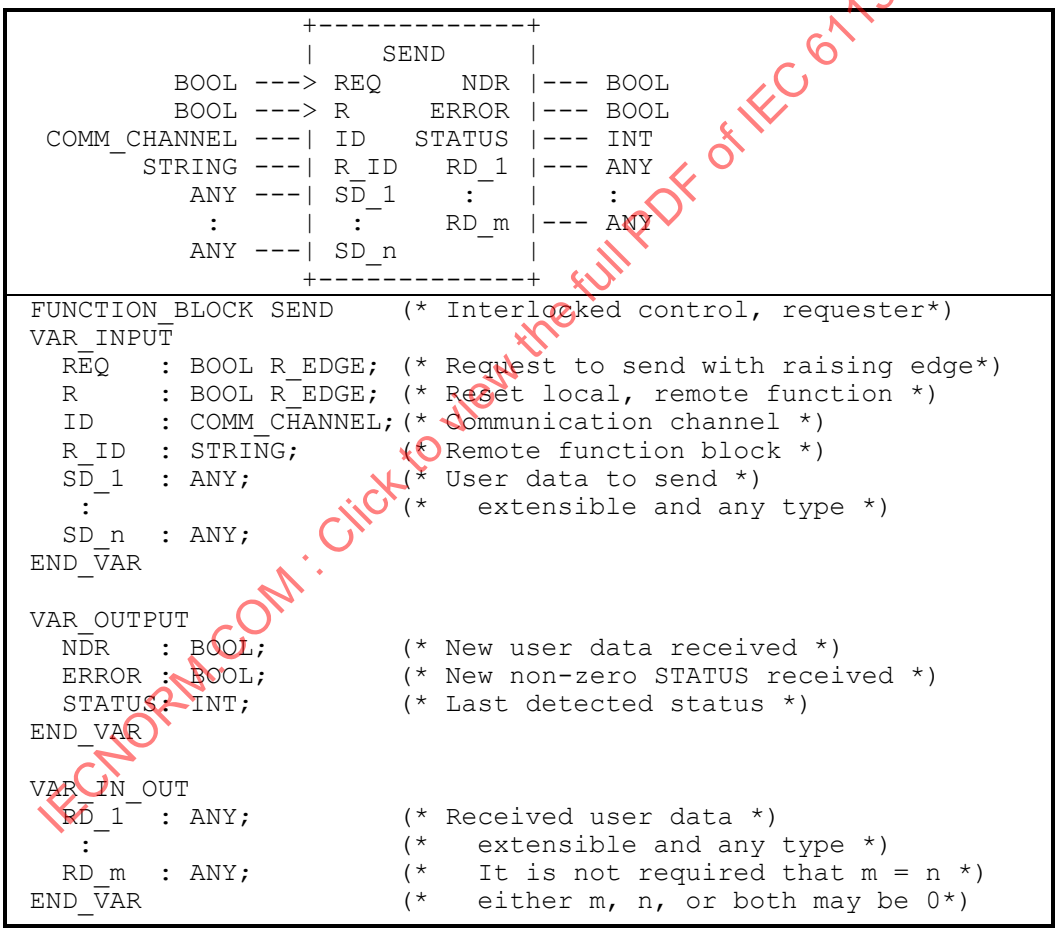
When the EN_R input of the RCV instance has the value of 1, it is enabled to receive data from the corresponding SEND instance and to perform the intended operation of the application program. When the RCV instance receives the data from the SEND instance, it passes the received data to the application program via its RD_i outputs. The NDR output of the RCV instance pulses one cycle to indicate that new data were received. After performing the intended operation of the application program, the result data are taken via the SD_i inputs and the response is initiated on a raising edge of the RESP input of the RCV instance.

When the SEND instance receives this response, it passes the received data to its RD_i outputs. The NDR output of the SEND instance pulses one cycle to indicate that new data are ready.

A raising edge on the R input of the SEND instance resets both the SEND and the corresponding RCV instance.

The number and each of the data types of the SD_i inputs of the SEND instance and the RD_i outputs of the corresponding RCV instance shall be compatible. The same is required for the SD_i inputs of the RCV instance and the RD_i outputs of the SEND instance. The SD_i inputs and the RD_i outputs of the SEND and RCV function blocks are extensible. Either the send data or the response data or both may be empty, that is, the user did not program any SD_i inputs and corresponding RD_i outputs.

If the received data did not match the RD_i outputs of the RCV function block or an error occurred, the ERROR output pulses one cycle to indicate an error and the STATUS output contains the error code.



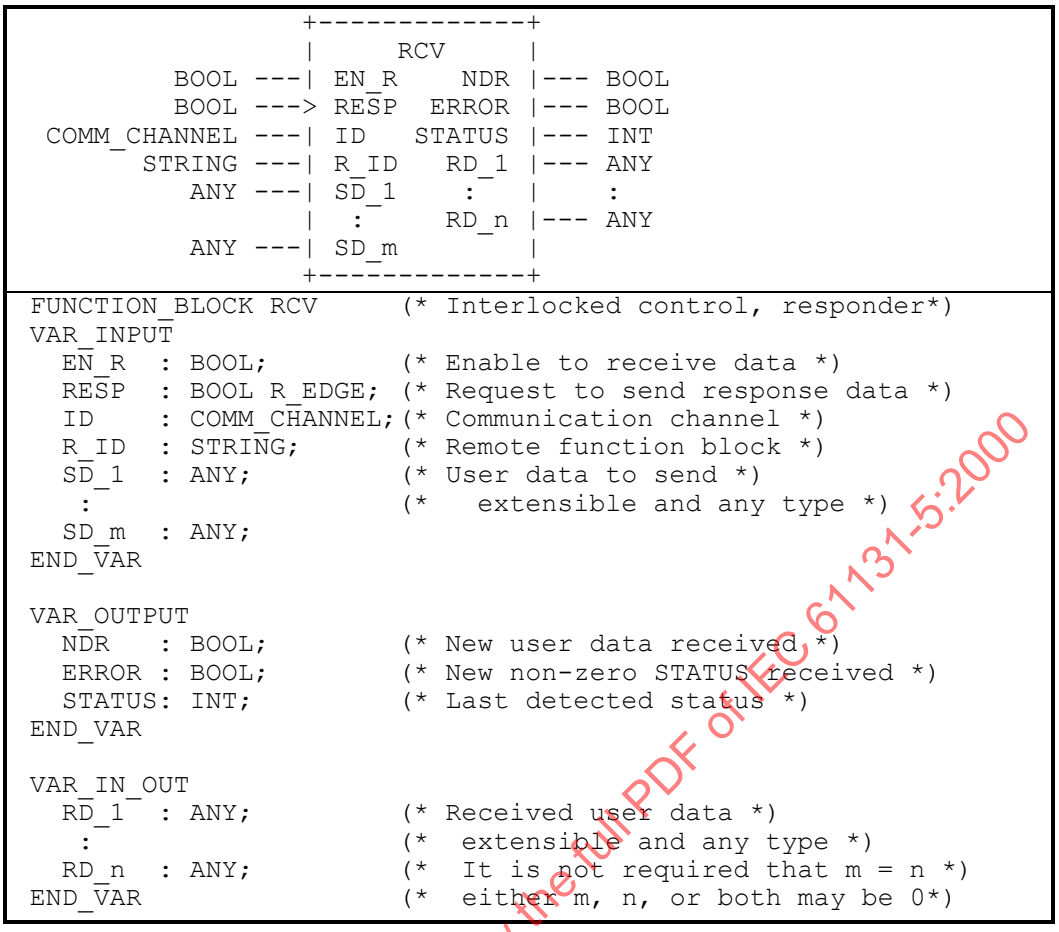

```
FUNCTION_BLOCK SEND     (* Interlocked control, requester*)
VAR_INPUT
  REQ    : BOOL R_EDGE; (* Request to send with raising edge*)
  R      : BOOL R_EDGE; (* Reset local, remote function *)
  ID     : COMM_CHANNEL;(* Communication channel *)
  R_ID   : STRING;      (* Remote function block *)
  SD_1   : ANY;         (* User data to send *)
   :                    (*   extensible and any type *)
  SD_n   : ANY;
END_VAR

VAR_OUTPUT
  NDR    : BOOL;        (* New user data received *)
  ERROR  : BOOL;        (* New non-zero STATUS received *)
  STATUS : INT;         (* Last detected status *)
END_VAR

VAR_IN_OUT
  RD_1   : ANY;         (* Received user data *)
   :                    (*   extensible and any type *)
  RD_m   : ANY;         (*   It is not required that m = n *)
END_VAR                 (*   either m, n, or both may be 0*)
```

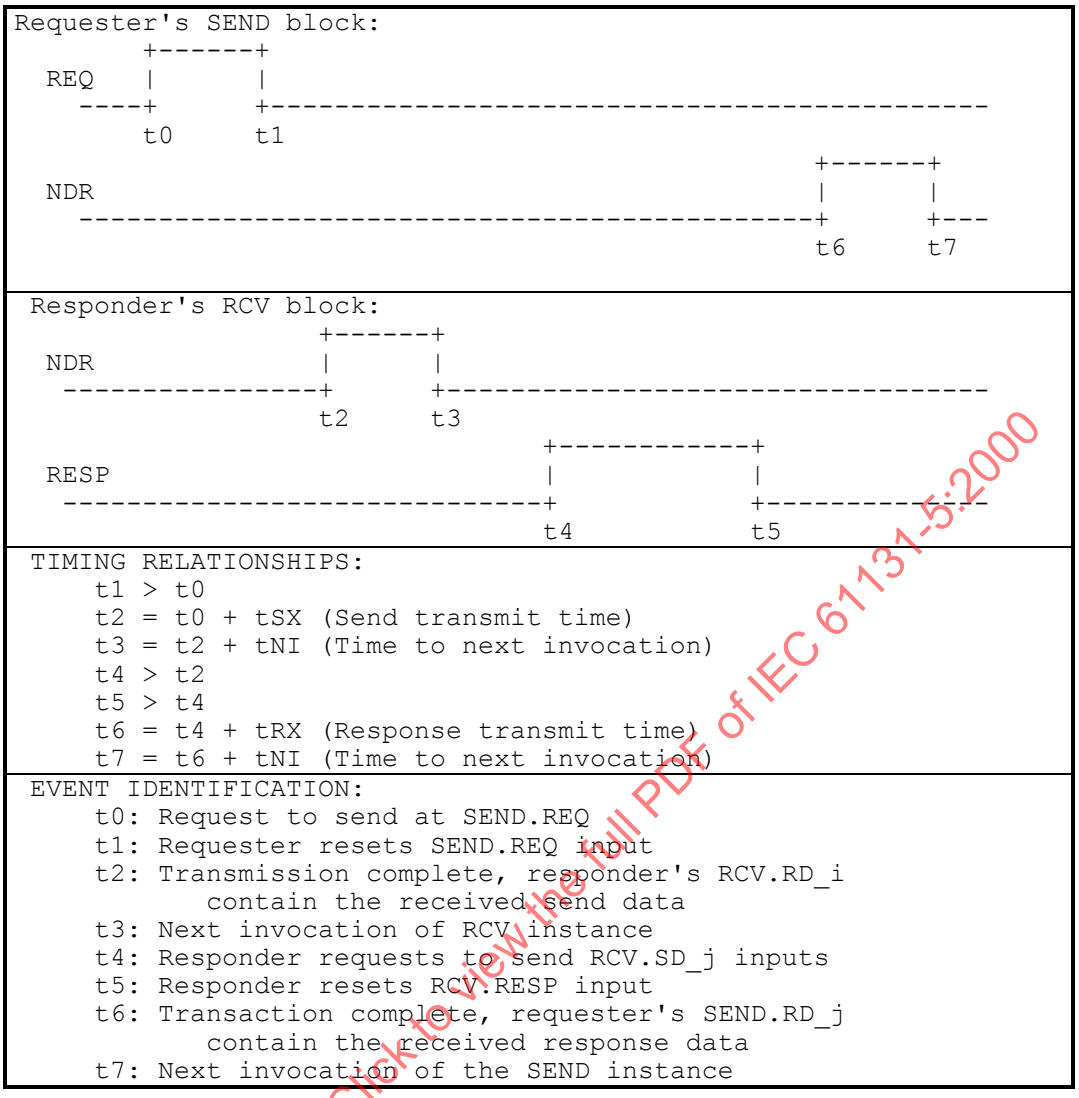
IEC 2280/2000

Figure 34 – SEND function block



IEC 2281/2000

Figure 35 – RCV function block



IEC 2282/2000

Figure 36 – Timing diagram of SEND and RCV function blocks

The state diagram shown in figure 37 describes the algorithm of the SEND function block. Tables 41 and 42 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the SEND function block outputs.

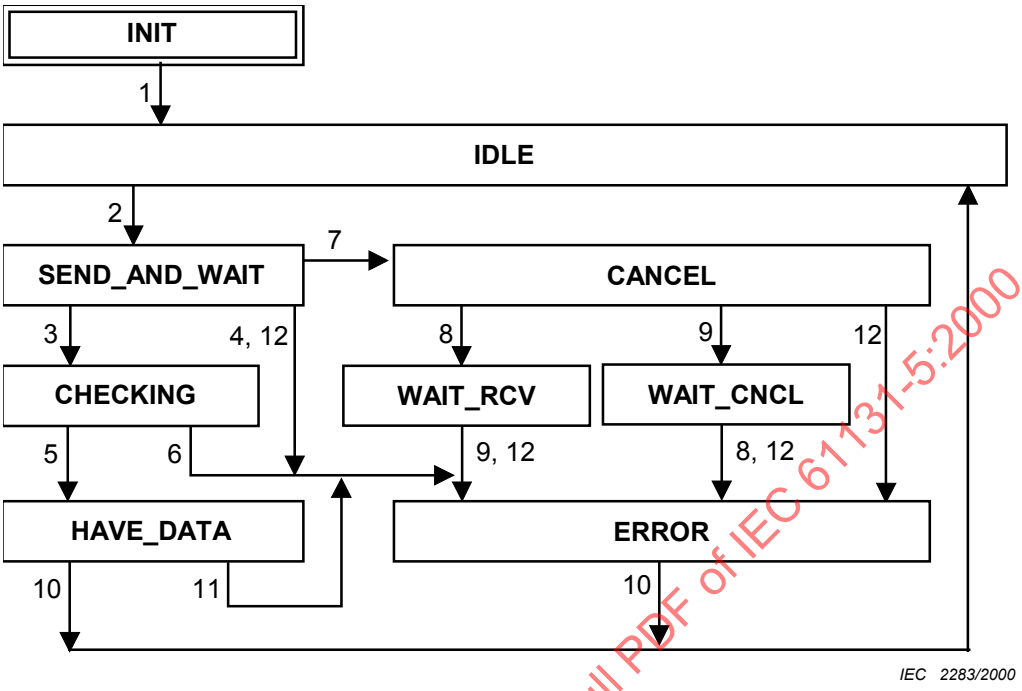


Figure 37 – State diagram of SEND function block

Table 41 – Transitions of the SEND state diagram

Transition	Condition
1	Initialization done
2	At raising edge of REQ input
3	Positive response from RCV
4	Negative response from RCV
5	Data types of received data and RD_1 to RD_m match
6	Data type of received data and RD_1 to RD_m mismatch
7	At raising edge of R input
8	Positive or negative response to the reset request
9	Positive or negative response from RCV
10	After next invocation of the instance
11	Local resource problems detected
12	Communication problems detected

Table 42 – Action table for SEND state diagram

		FB outputs			
State	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1 ... RD_m**
INIT [a]	Initialize outputs	0	0	0	System null
IDLE	No actions	0	0	---	---
SEND_AND_WAIT	Send data to RCV, evaluate transition 7 first	---	---	–1	---
CHECKING	Verify data type match	---	---	---	---
HAVE_DATA	Deposit data in instance	1	0	0	New data from RCV
ERROR	Indicate error	0	1	[b]	---
CANCEL	Request to reset RCV	---	---	5	---
WAIT_RCV	No actions	---	---	---	---
WAIT_CNCL	No actions	---	---	---	---

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

The state diagram shown in figure 38 describes the algorithm of the RCV function block. Tables 43 and 44 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the RCV function block outputs.

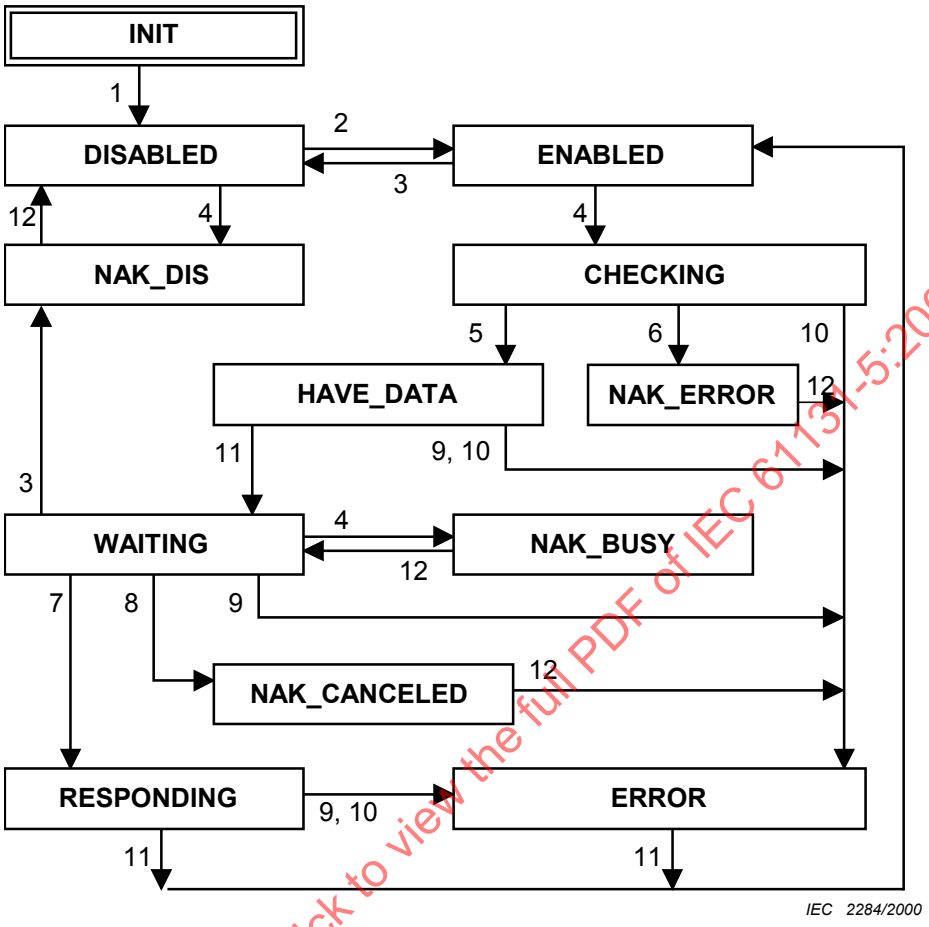


Figure 38 – State diagram of RCV function block

Table 43 – Transitions of RCV state diagrams

Transition	Condition
1	Initialization done
2	EN_R = 1
3	EN_R = 0
4	When data received from SEND
5	Data types of received data and RD_1 to RD_n match
6	Data types of received data and RD_1 to RD_n mismatch
7	Raising edge of RESP input
8	When reset is requested by SEND
9	Communication problems detected
10	Local resource problems detected
11	After next invocation of the instance
12	True, i.e. condition is empty

Table 44 – Action table of RCV state diagram

		FB outputs			
State	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1 ... RD_n**
INIT [a]	Initialize outputs	0	0	0	System null
DISABLED	No actions	0	0	---	---
ENABLED	No actions	0	0	---	---
CHECKING	Verify data type match	---	---	---	---
HAVE_DATA	Deposit data	1	0	0	New data from SEND
NAK_ERROR	Send negative response to SEND	---	---	---	---
WAITING	No actions	0	0	---	---
NAK_BUSY	Send negative response to SEND	---	---	---	---
NAK_CANCELED	Send positive response to the reset request and then send negative response to SEND	---	---	---	---
RESPONDING	Send positive response with user data to SEND	---	---	---	---
ERROR	Indicate error	0	1	[b]	---
NAK_DIS	Send negative response to SEND	---	---	---	---

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

7.8 Programmed alarm report

The PC communication function programmed alarm report uses the NOTIFY and the ALARM function blocks.

One instance of one NOTIFY function block or one instance of one ALARM function block provides one instance of the PC function programmed alarm report.

A PC can be programmed using the ALARM function block to report an alarm message with an acknowledgement capability. Or, it can be programmed using the NOTIFY function block to report an alarm message without an acknowledgement capability.

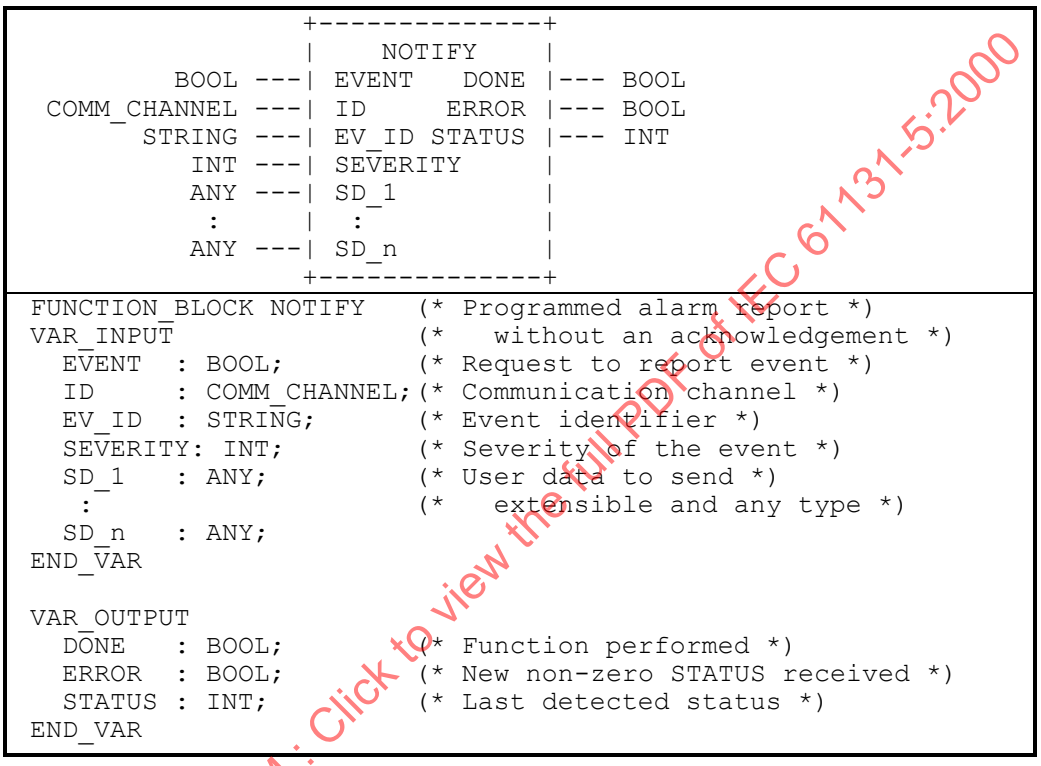
The raising edge at the EVENT input reports an alarm message of a coming event, the falling edge at the EVENT input reports the going of this event. The reason for the event can be given with the EV_ID input, its severity given at the SEVERITY input of the function blocks. The severity shall have a range from 0 to 127, inclusive. 0 shall represent the highest severity, 64 a normal severity, and 127 the lowest severity. Additional data at the SD_i inputs may be used to specify more details of the occurred event.

The ACK_UP output of the ALARM function block shall contain an indication, whether or not the coming of the event was acknowledged by the remote communication partner. The ACK_DN output shall contain this indication for the going of the event. Only the acknowledgement of the most recent event shall set the acknowledgement outputs. The same behavior shall be true for the falling edge of the EVENT input and the ACK_DN output.

The ID parameter identifies the communication channel to the remote communication partner. If the communication system provides communication channels which support one-to-many or one-to-all connections, these function blocks may be used to program an alarm report function from one PC to many. In this case the first valid acknowledgement sets the appropriate ALARM acknowledgement output.

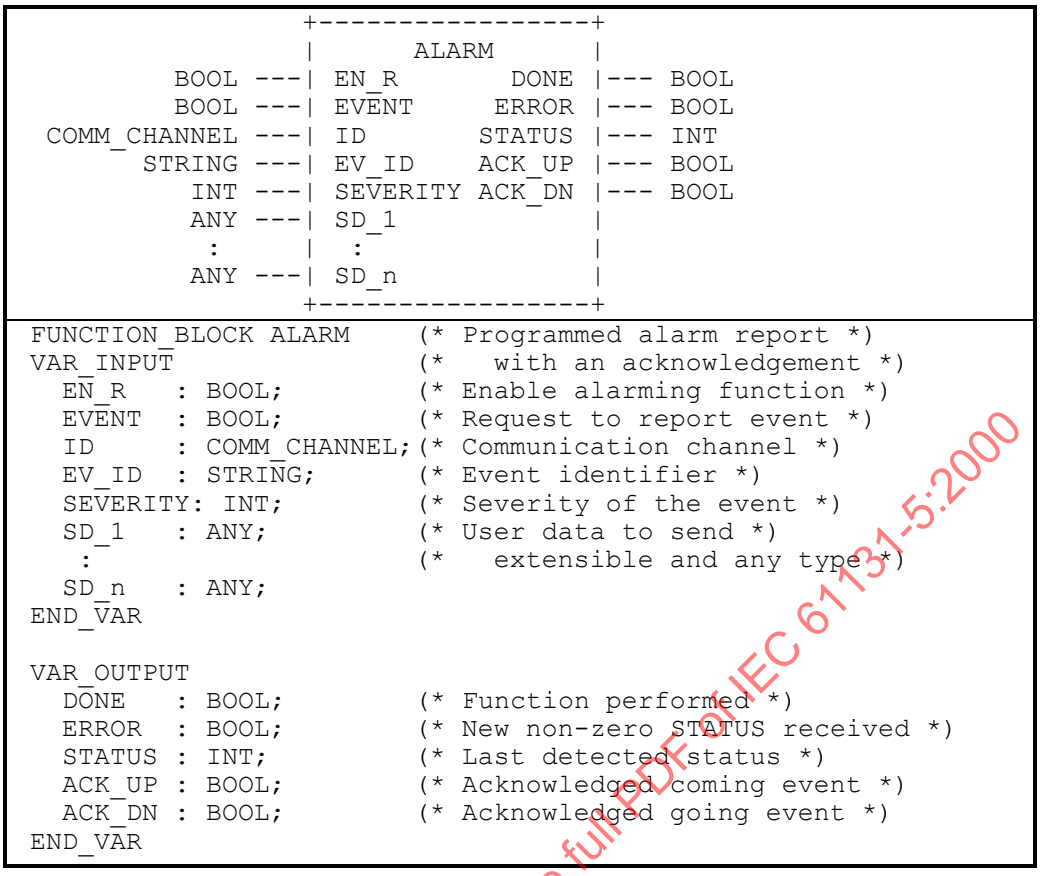
The send data may provide additional information in the alarm message. The send data may be empty, i.e. no SD_i inputs are programmed.

If an error occurred, the ERROR output pulses one cycle to indicate an error and the STATUS output contains the error code.



IEC 2285/2000

Figure 39 – NOTIFY function block



IEC 2286/2000

Figure 40 – ALARM function block

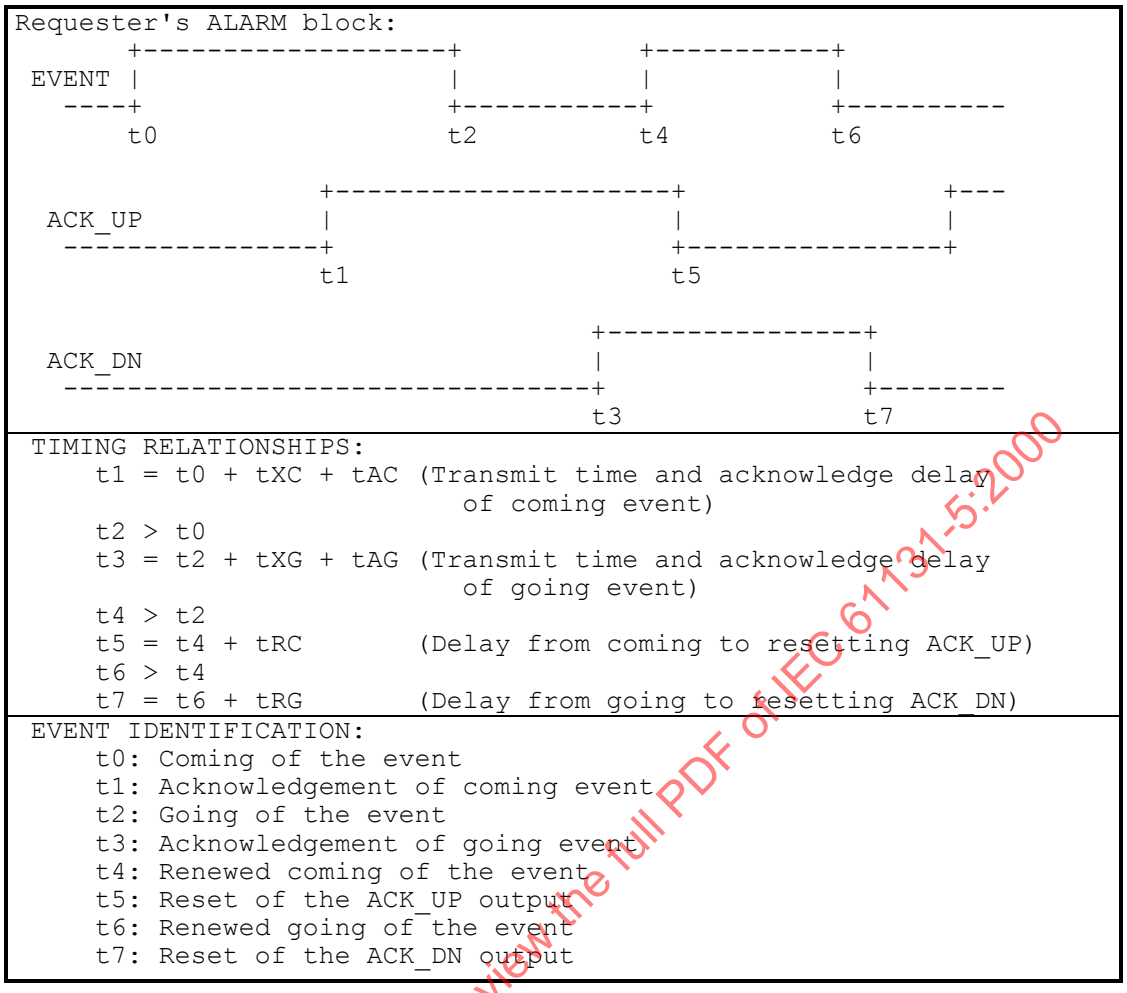


Figure 41 – Timing diagram of ALARM function block

The state diagram shown in figure 42 describes the algorithm of the NOTIFY function block. Tables 45 and 46 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the NOTIFY function block outputs.

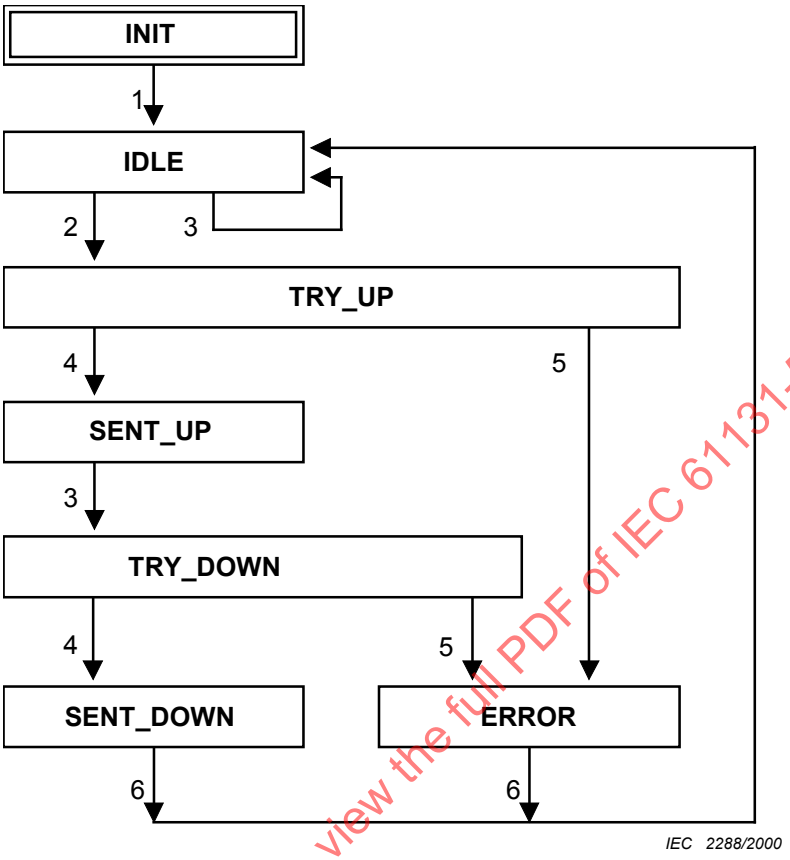


Figure 42 – State diagram of NOTIFY function block

Table 45 – Transitions of the NOTIFY state diagram

Transition	Condition
1	Initialization done
2	Raising edge of EVENT input
3	Falling edge of EVENT input
4	Communication system indicates "Sent to remote communication partner"
5	Communication system indicates "Cannot send to remote communication partner"
6	After next invocation of this instance

Table 46 – Action table for NOTIFY state diagram

		FB outputs		
State	**Actions**	**DONE** [c]	**ERROR** [c]	**STATUS**
INIT [a]	Initialize outputs	0	0	0
IDLE	No actions	0	0	0
TRY_UP	Request to report the coming alarm to the remote communication partner	0	0	---
SENT_UP	No actions	1	0	---
TRY_DOWN	Request to report the going alarm to the remote communication partner	0	0	---
SENT_DOWN	No actions	1	0	---
ERROR	Indicate error	0	1	[b]

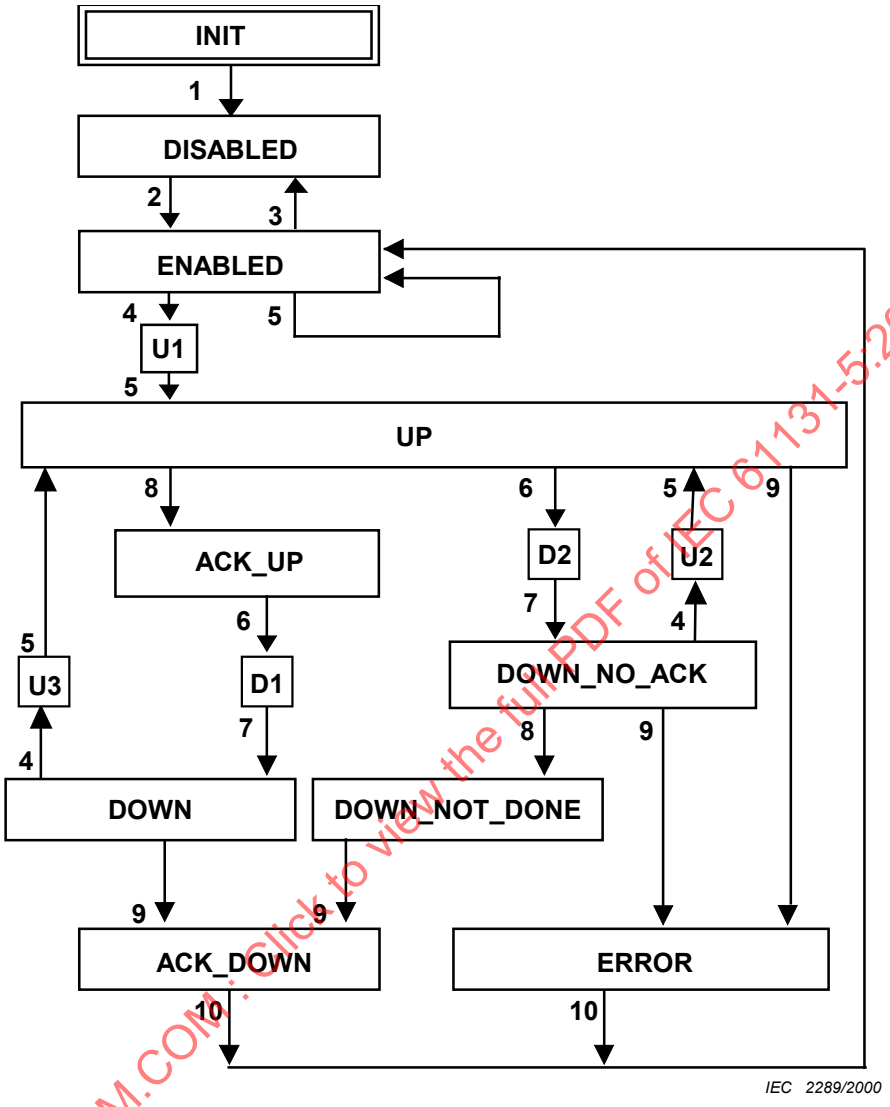
--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

The state diagram shown in figure 43 describes the algorithm of the ALARM function block. Tables 47 and 48 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the ALARM function block outputs.



NOTE The states labelled U1, U2, U3, D1, and D2 may also transition to the ERROR state if the lower layers detect an error while trying to send the event notification.

Figure 43 – State diagram of ALARM function block

Table 47 – Transitions of the ALARM state diagram

Transition	Condition
1	Initialization done
2	EN_R=1
3	EN_R=0
4	Raising edge of EVENT input
5	Coming event notification sent by communication system
6	Falling edge of EVENT input
7	Going event notification sent by communication system
8	Receive acknowledge of the report of the coming alarm
9	Receive acknowledge of the report of the going alarm
10	After next invocation of this instance

Table 48 – Action table for ALARM state diagram

		FB outputs				
State	**Actions**	**DONE** [c]	**ERROR** [c]	**STATUS**	**ACK_UP**	**ACK_DN**
INIT [a]	Initialize outputs	0	0	0	0	0
DISABLED	No actions	0	0	---	0	0
ENABLED	No actions	0	0	0	---	---
U1, U2, U3	Request to report the coming alarm to the remote communication partner	0	0	---	0	---
UP	No actions	1	---	---	---	---
ACK_UP	Confirm received acknowledgement positively	---	0	0	1	---
D1, D2	Request to report the going alarm to the remote communication partner	0	0	---	---	0
DOWN_NO_ACK	No actions	1	---	---	---	---
DOWN	No actions	1	---	---	---	---
DOWN_NOT_DONE	No actions	0	---	---	---	---
ACK_DOWN	Confirm received acknowledgement positively	---	0	0	---	1
ERROR	Indicate error and confirm received acknowledgement negatively	0	1	[b]	0	0

--- indicates "unchanged" FB outputs.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] See figure 10.

7.9 Connection management

The PC communication function connection management uses the CONNECT function block.

One instance of one CONNECT function block provides one instance of the PC function connection management. This communication function allows to establish a connection between the calling communication partner and the remote communication partner.

A PC can request a remote communication partner to establish a connection between this PC and the remote communication partner. The remote communication partner is identified using its name. (The name shall be specified by means of the implementer of the remote communication partner.) A communication channel to the remote communication partner is defined.

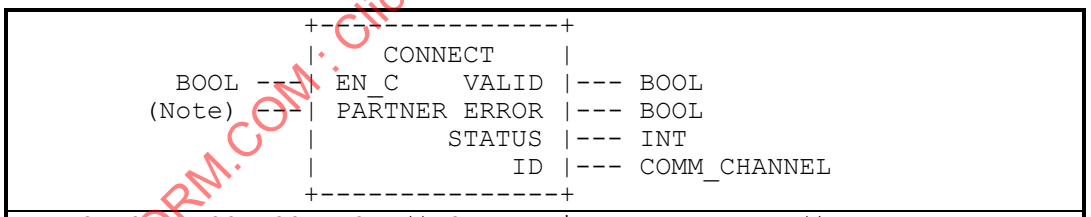
The remote communication partner shall decide whether or not to establish the connection.

If the invocation of a CONNECT function block establishes a communication channel to a remote communication partner, the ID output shall provide the communication channel descriptor which can be used as an input for the ID input to other communication function blocks which communicates with this remote communication partner.

The remote communication partner itself may also use this connection for its communication as a client. The remote communication partner can get the necessary value of the ID parameter by calling a CONNECT function block referencing the communication partner which has already established the connection.

If an error occurred, the ERROR output pulses one cycle to indicate an error and the STATUS output contains the error code.

A communication channel may also be established by local means of the PC on one or both sides of the communication channel. In this case the PC shall show the same behavior as if the CONNECT function block for this communication channel is invoked at the beginning of the application program cycle with the EN_C parameter fixed to 1. The implementer shall define how the application program can get the appropriate value for the ID parameter to use this communication channel.




```
FUNCTION_BLOCK CONNECT (* Connection management *)
VAR_INPUT
  EN_C   : BOOL;         (* Enable connection *)
  PARTNER: (Note);       (* Name of the remote communication *)
                         (* partner *)
END_VAR

VAR_OUTPUT
  VALID : BOOL;          (* Connection ID valid *)
  ERROR : BOOL;          (* New non-zero STATUS received *)
  STATUS: INT;           (* Last detected status *)
  ID    : COMM_CHANNEL;(* Communication channel *)
END_VAR
```

NOTE The data type of the input PARTNER is implementer specific. Additional information may be provided with this parameter which may be used to establish the communication channel, for example, a password or some communication subsystem specific parameters.

IEC 2290/2000

Figure 44 – CONNECT function block

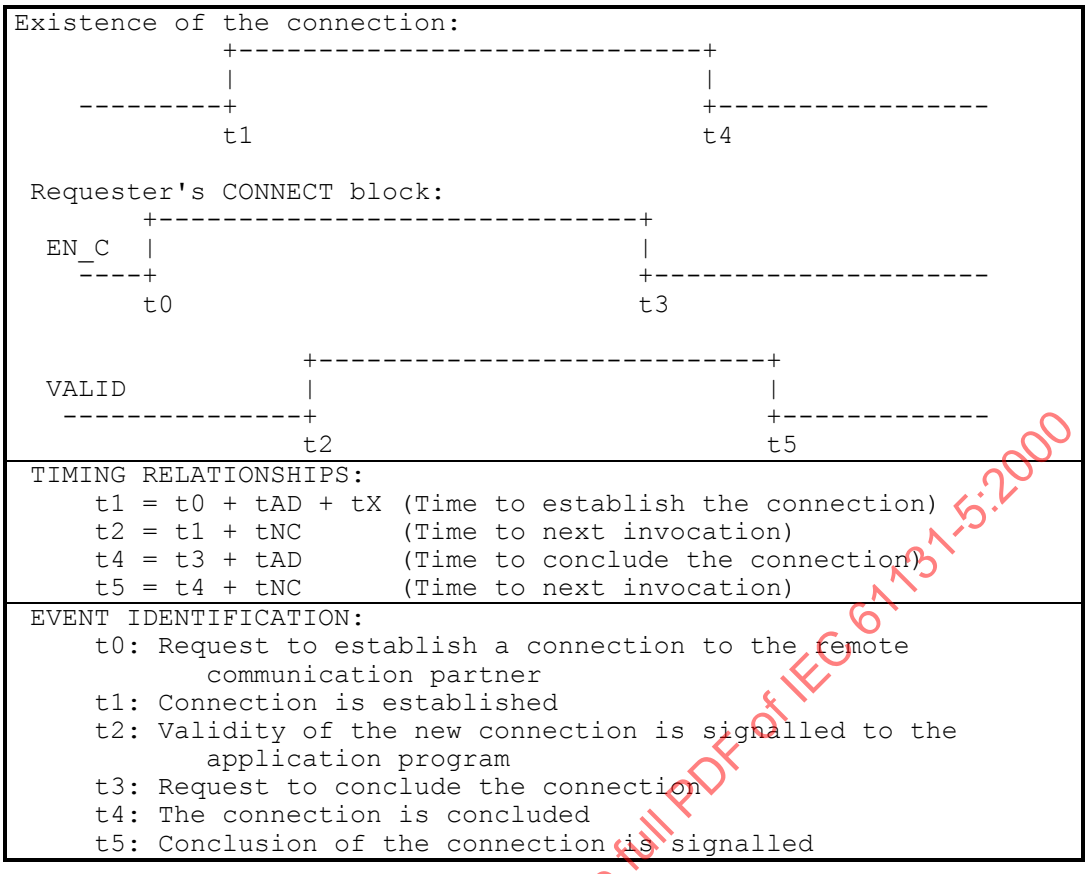


Figure 45 – Timing diagram of CONNECT function block

The state diagram shown in figure 46 describes the algorithm of the CONNECT function block. Tables 49 and 50 describe the transitions of this state diagram and the actions to be performed within the states and the settings of the CONNECT function block outputs.

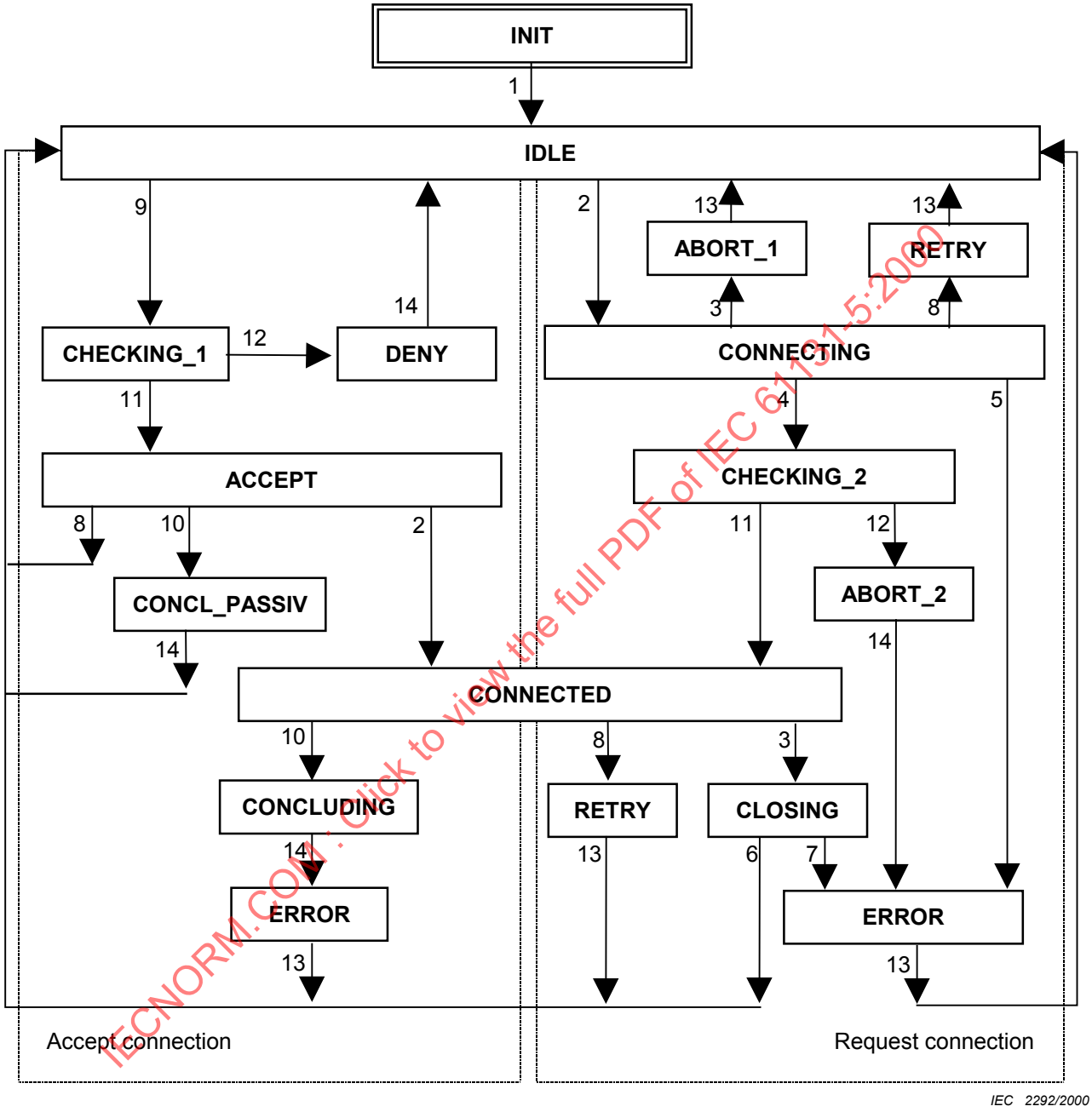


Figure 46 – State diagram of CONNECT function block

Table 49 – Transitions of the CONNECT state diagram

Transition	Condition
1	Initialization done
2	EN_C=1
3	EN_C=0
4	Receive a positive confirmation to the request to establish a new connection from the remote communication partner
5	Receive a negative confirmation to the request to establish a new connection from the remote communication partner
6	Receive a positive confirmation to the request to conclude the connection from the remote communication partner
7	Receive a negative confirmation to the request to conclude the connection from the remote communication partner
8	Loss of connection
9	Receive a request to establish a new connection to the remote communication partner
10	Receive a request to conclude the connection to the remote communication partner
11	Accept to establish a connection
12	Deny to establish a connection
13	Next invocation
14	Immediately

Table 50 – Action table for CONNECT state diagram

		FB outputs			
State	**Actions**	**VALID**	**ERROR [d]**	**STATUS**	**ID [c]**
INIT [a]	Initialize outputs	0	0	0	NG
IDLE	No actions	0	---	---	NG
CHECKING_1	Check if the connection can be established	0	---	---	NG
DENY	Send negative response to the request of the connection	0	---	---	NG
ACCEPT	Accept to establish a connection and a send a positive response	0	0	---	NG
CONCL_PASSIV	Send a positive response to conclude the connection	0	---	---	NG
CONNECTING	Request a connection to the remote communication partner	0	0	–1	NG
ABORT_1	Request to abort communication	0	---	---	NG
CHECKING_2	Check if the connection can be established	0	0	---	NG
ABORT_2	Request to abort communication	0	---	---	NG
CONNECTED	No action	1	0	0	OK
RETRY	No action	0	1	1	NG
CLOSING	Request to conclude the connection	0	0	–1	NG
CONCLUDING	Send positive response to conclude the connection	0	0	---	NG
ERROR	Indicate error	0	1	[b]	NG

--- indicates "unchanged" FB output.

[a] INIT is the cold start state.

[b] The error code is placed in the status output.

[c] NG indicates "invalid connection ID"
OK indicates "valid connection ID".

[d] See figure 10.

7.10 Example for the use of communication function blocks

7.10.1 Establishing a communication channel

Two PCs want to establish a communication channel. Both of the PCs want to use the channel for client and server function, i.e. both need to get an appropriate value for their ID parameter of the Communication function blocks. In this example, it is assumed that the data type COMM_CHANNEL stands for a handle or index of the communication channel. The example is given using the Structured Text programming language as defined in IEC 61131-3.

Extract of the application program of PC1:

```
(* Declaration of the data and the CONNECT instance, e.g. at the beginning
     of an application program *)
VAR
TO_PC2:   COMM_CHANNEL; (* Variable which is set by CONNECT *):
CO1:      CONNECT;      (* Declaration of the CONNECT instance *)
END_VAR;

(* somewhere inside the body of the program *)

CO1(EN_C:=1, PARTNER:='PC2');     (* Invokes the CONNECT instance CO1 and
                                    establishes a communication channel to PC2 *)
IF CO1.ERROR THEN  (* It is recommended to define some error handling *):
IF CO1.VALID THEN  (* Store reference of the communication channel  *)
TO_PC2:= CO1.ID;
```

Extract of the application program of PC2:

```
(* Declaration of the data and the CONNECT instance, e.g. at the beginning
     of an application program *)
VAR
TO_PC1:   COMM_CHANNEL; (* Variable which is set by CONNECT *):
CO2:      CONNECT;      (* Declaration of the CONNECT instance *)
END_VAR;

(* somewhere inside the body of the program *)

CO2(EN_C:=1, PARTNER:='PC1');     (* Invokes the CONNECT instance CO2 and
                                    establishes a communication channel to PC1 *)
IF CO2.ERROR THEN  (* It is recommended to define some error handling *):
IF CO2.VALID THEN  (* Store reference of the communication channel  *)
TO_PC1:= CO2.ID;
```

These parts of the application program of PC1 and PC2 may be nearly identical. The communication channel is established by the PC which invokes its CONNECT instance earlier. The later invoking PC gets the communication channel already established.

7.10.2 Transferring data

Two PCs want to communicate using an already established communication channel. Some data shall be transferred from PC1 to PC2 using the USEND and URCV function blocks. The following parts of the applications programs show how this can be achieved. How the application programs provide and process the data to be transferred is not shown in this example. The example is given using the Structured Text programming language as defined in IEC 61131-3.

Extract of the application program of PC1 which uses an instance of the USEND function block to send the data:

```
(* Declaration of the data and the USEND instance, e.g. in the definition
     of a function block *)
VAR
SENDREQ:  BOOL;           (* Flag to request the send *)
TO_PC2:   COMM_CHANNEL;   (* Variable which allows to use the communication
                             channel to PC2 *)
SDAT1:    ARRAY[0..20] OF BYTE;   (* Declaration of the data to send *)
SDAT2:    REAL;
US1: USEND;               (* Declaration of the USEND instance *)
END_VAR;

(* somewhere inside the body of the function block *)

US1(REQ:=SENDREQ, ID:=TO_PC2, R_ID='PACK1', SD_1:=SDAT1, SD_2:=SDAT2);
     (* Invokes the USEND instance US1 and will send the data on raising
          edge of SENDREQ Boolean *)
IF US1.ERROR THEN  (* It is recommended to define some error handling *):
```

Application program of PC2 which uses an instance of the URCV function block to receive the data sent by PC1:

```
(* Declaration of the data and the URCV instance, e.g. in the definition of a function block *)
VAR
TO_PC1: COMM_CHANNEL;   (* Variable which allows to use the communication
                           channel to PC1 *)
RDAT1: ARRAY[0..20] OF BYTE; (* Declaration of the variable where the data
                        shall be stored, the count of variables and their
                        data type must correspond with the data sent *)
RDAT2: REAL;
UR1: URCV;      (* Declaration of the URCV instance *)
S: REAL;        (* Declaration of an arbitrary floating point variable *)
END_VAR;

(* somewhere inside the body of the function block *)

UR1(EN_C:=1, ID:=TO_PC1, R_ID='PACK1', RD_1:=RDAT1, RD_2:=RDAT2);
               (* Invokes the URCV instance UR1 to wait for data from PC1 *)
IF UR1.NDR THEN    (* process the received data *)
BEGIN
S:= S + RDAT2;     (* e.g. add the received floating point number to a
                   variable *)
IF UR1.ERROR THEN  (* It is recommended to define some error handling *);
```

7.10.3 Using a timer to supervise communication

A PC (PC1) wants to request a process function from PC2. It uses an instance of a SEND function block to transfer the request. It waits for the response from PC2. But if PC2 does not respond within 5 s, it resets the request. The example is given using the Structured Text programming language as defined in IEC 61131-3.

Extract of the application program of PC1 which uses an instance of the SEND function block to request the process function:

```
(* Declaration of the data and the SEND instance, e.g. in the definition of
a function block *)
VAR
SREQ:     BOOL;          (* Flag to send the request *)
FCT1:     INT;           (* Code of the function to request *)
DAT1:     REAL;          (* 1st parameter of this function *)
RDAT1:    INT;           (* Response parameter *)

SR1:      SEND;          (* Instance of the FB SEND *)
M1:       RS;            (* to hold the IN parameter of the timer *)
T1:       TON;           (* Timer for timeout control of the SEND *)
END_VAR;

(* somewhere inside the body of the function block *)

SR1 (REQ:= SREQ,         (* Request on raising edge of SREQ bool *)
     R:= T1.Q,           (* Reset on timeout, i.e. the timer fired *)
     ID:= TO_PC2, R_ID:= 'ORD1',     (* Identifies remote partner and
RCV instance *)
     SD_1:= FCT1, SD_2:= DAT1,  (* Data for the process function *)
     RD_1:= RDAT1);             (* Variables for the results *)

M1  (S:= SREQ,           (* Set when the request is sent *)
     R1:= T1.Q OR SR1.NDR OR SR1.ERROR);
                         (* Reset when the timer fired or the SEND gets a
                         good (NDR=1) or a bad response (ERROR=1) *)

T1  (IN:=M1.Q1,          (* Hold the timer during the request *)
     PT:= T#5s);         (* is active for 5 seconds *)
```

The same program part using the function block diagram as defined in IEC 61131-3 is shown in figure 47:

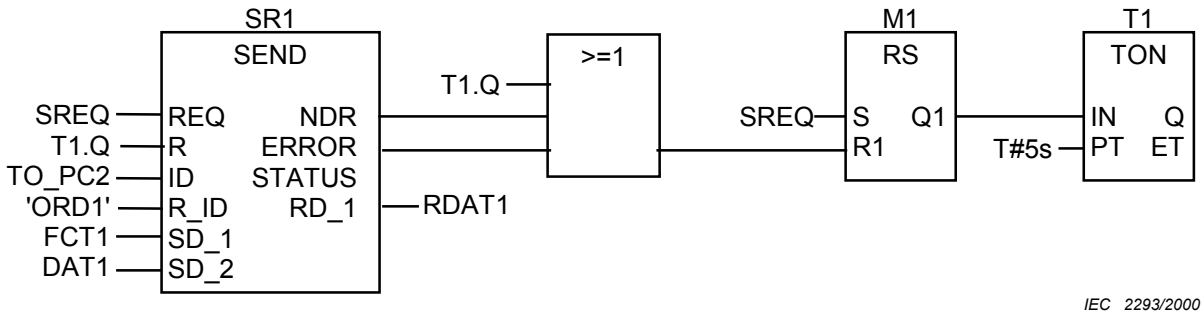


Figure 47 – Example in function block diagram language

8 Compliance and implementer specific features and parameters

8.1 Compliance

A PC system, as defined in IEC 61131-1, which claims to comply, wholly or partially, with the requirements of this part of IEC 61131 shall do so only as described below.

A compliance statement shall be included in the documentation accompanying the system, or shall be produced by the system itself. The form of the compliance statement shall be:

"This system complies with the requirements of this part for the following features:" followed by a set of compliance tables in the following format:

Table title

Table number	Feature number	Feature description

Table and feature numbers and descriptions are to be taken from the tables given in the relevant subclauses of this part of IEC 61131. The table titles and the relevant tables are to be taken from the following table 51.

Table 51 – Table titles and relevant tables for compliance

Table title	For features in table
PC status	Tables 1 to 9
Application specific functions	Tables 10 to 20, except 15
PC communication function blocks	Table 21

A PC system complying with the requirements of this part of IEC 61131 shall

a) not require the inclusion of substitute or additional features in order to accomplish any of the features specified in this part;

b) be accompanied by a document that specifies the values of all implementer specific features or parameters as listed in the following table and in the table 'Implementation specific features and parameters' of the appropriate annex;

c) be accompanied by a document that separately describes any communication relevant feature that is prohibited or not specified in this part. Such features shall be described as being "extensions to the PC communication as defined in IEC 61131-5";

d) not use any of the function or function block names defined in clause 7 for implementer defined features whose functionality differs from that described there.

8.2 Implementation specific features and parameters

Implementation specific features and parameters defined in this part of IEC 61131 and the most relevant subclause for each are listed in the following table 52.

Table 52 – Implementation specific features and parameters

Subclause	Implementation specific features and parameters
6.1	The definition of major and minor faults of the subsystems of the PC
6.1.8	The number of the subsystems of the PC providing status information
6.1.8	The types of the subsystems of the PC providing status information
6.1.8	The maximum size of the string containing names of the PC and of its subsystems
6.1.8	The semantic of the values in the sub-element STATE for the implementer specific subsystems
6.1.8	The size of the sub-element SPECIFIC of the status information
6.1.8	The semantic of the sub-element SPECIFIC of the status information
6.2.2	Restrictions of the access of variables with direct representation
6.2.2	The algorithm to access a variable with direct representation
6.2.2	The conditions (size, location, etc.) under which each data type supported by the PC can be uninterruptedly accessed
6.2.6	The supported values of I/O state
6.2.6	The definition of the outputs if the implementer specified value is requested
6.2.6	The definition of the mechanisms to specify the outputs if the user specified value is requested
6.2.7	Other language elements, for example, function types or function block types which are loadable and the conditions and restrictions for downloading and uploading these
6.2.7	What other clients can do with a PC when one client is downloading it
7.2	The COMM_CHANNEL type used at the ID input of the communication function blocks identifying the remote communication partner
7.2	Data type of the VAR_ADDR output of the REMOTE_VAR function and the restrictions using the output of the REMOTE_VAR function
7.2	Name scopes the implemeter supports for use at SCOPE parameter of the REMOTE_VAR function
7.2	Mapping of the implementer specific name scopes onto the communication system
7.2	The number of SD_i, RD_i, and VAR_i parameters which are supported with one invocation of a communication function block
7.2	The meaning of codes less than –1 or greater than 20 of the STATUS output of the communication function blocks
7.2	Additional means (if there are some) to control the time by which the communication channel is established, if not explicitly controlled using the CONNECT function block
7.3	The used length and the semantics of the additional status information of the LOCAL output of the FB STATUS and FB USTATUS
7.9	How the user can get the information of the communication channel for the use at the ID input of the communication function blocks

Annex A
(normative)

Mapping to ISO/IEC 9506-5

A.1 General

This annex specifies the mapping of the PC communication functions to the MMS objects and services defined in ISO/IEC 9506-1 and extensions defined in ISO/IEC 9506-5. This mapping shall be used when a PC is communicating in the abstract syntax defined in ISO/IEC 9506-5.

Normative references

ISO 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*

ISO/IEC 9506-1:1990, *Industrial automation systems – Manufacturing Message Specification – Part 1: Service definition*
Amendment 1:1993, *Data exchange*

ISO/IEC 9506-2:1990, *Industrial automation systems – Manufacturing Message Specification – Part 2: Protocol specification*
Amendment 1:1993, *Data exchange*

ISO/IEC 9506-5:1999, *Industrial automation systems – Manufacturing Message Specification – Part 5: Companion standard for programmable controllers*

Definitions from other publications

ISO 7498-1

application entity (AE)

ISO/IEC 9506-1

domain

program invocation (PI)

server

Virtual Manufacturing Device (VMD) (clause 7)

Abbreviations

Cnf This is the confirmation primitive of a communication service

Ind This is the indication primitive of a communication service

Req This is the request primitive of a communication service

Rsp This is the response primitive of a communication service

VMD Virtual Manufacturing Device

A.2 Application specific functions

A.2.1 Device verification

A PC can provide some very general status to a requesting device via the MMS Status service. The contents of this status is defined by MMS. The implementer can provide Local Detail (bit string with a maximum length of 128 bits) that contains additional information. The PC can initiate an unsolicited status report of this same information using the MMS UnsolicitedStatus service.

NOTE The 16 lowest numbered bits of Local Detail may provide, for example the same information as in the P_PCSTATE variable (see table 2 and 6.1.8).

A PC can provide detailed status information about the PC and its subsystems via MMS Named Variables or MMS Unnamed Variables or both (see 6.1.8). These variables can be read using the MMS Read service. For the specification of these variables (see 6.1.8).

A.2.2 Data acquisition

Data contained in a PC is presented as PC variables. Selected PC variables are mapped onto MMS Named Variables, using the access path language construct (see A.3.2). Additionally, variables with direct representation are mapped onto MMS Unnamed Variables, with implementer specified restrictions. The content of these variables can be provided to a client using the MMS Read and InformationReport services.

A.2.3 Parametric control

Parametric control is when the operation of the PC is directed by writing values to PC variables. This change in operation is determined by either the application program or other local mechanisms. The MMS Write service is used to write a new value to a PC variable.

A.2.4 Interlocked control

Interlocked control is when the client requests the server to execute an application operation and to inform the client of the result of the operation. The SEND and RCV function blocks are used to provide this function. See A.4 for the mapping of these function blocks onto MMS objects and services.

A.2.5 Synchronization between user applications

User applications may need a synchronization service. This PC function is provided by the communication function interlocked control (see 6.2.3 and A.2.4).

A.2.6 Alarm reporting

The PC can have the ability to signal alarm messages to a client when a predetermined condition occurs. The ALARM and NOTIFY function blocks are used to provide this function. See A.4 for the mapping of these function blocks onto MMS objects and services.

A.2.7 Application program execution and I/O control

Program execution and I/O control uses the language elements configuration and resource of IEC 61131-3 (see 6.2).

The configuration and resource are mapped onto MMS Program Invocation objects.

The state of the I/O (inputs and outputs) associated with a resource is specified using the various MMS services which operate on Program Invocations using the I/O State parameter, as defined in ISO/IEC 9506-5.

A.2.8 Application program transfer

Application program transfer allows the client to upload the complete contents of the programmable memory or portions thereof for archiving or verification or to download it for restoring the PC to a known state. The portions of the programmable memory which can be uploaded or downloaded were specified in 6.2.7. These loadable units are mapped onto the MMS Domain object. They are transferred using the MMS Domain services.

A.2.9 Connection management

The MMS Environment and General Management services are used to manage the connections.

A.3 PC object mapping

ISO/IEC 9506 supports an object-oriented view to the communication. A PC communicating in the abstract syntax of ISO/IEC 9506-5 is mapped onto a virtual device. This virtual device contains all objects visible to a remote communication partner and services all requests coming from a remote communication partner.

A.3.1 VMD

One PC, a part of a PC, or a set of several PCs may be mapped onto one VMD. The communication to this VMD shall conform to ISO/IEC 9506-5.

The implementer shall specify this mapping. This part of IEC 61131 does not define any restrictions to the attributes of the VMD.

A.3.2 Named Variables

Each access path is mapped onto one MMS Named Variable. The attributes of the MMS Named Variable are as follows:

Object: Named Variable

Attribute: Variable Name – This shall be the access name from the access path declaration. If the access path definition references variables with direct representation, program inputs or outputs, or global variables of configurations or resources (see 2.7.1 of IEC 61131-3), the name shall have VMD-specific scope. If the access path definition is at program level (see 2.5.3 of IEC 61131-3), the name shall be domain specific of the domain which is associated with the program which contains the referenced variable (see A.2.9).

Attribute: Reference to Access Control List – If the access path contains the READ_ONLY language element this object shall reference the Access Control List object M_ReadOnly otherwise M_NonDeletable.

Attribute: Type Description – This shall be the data type as specified in the access path declaration mapped to MMS according to table A.1.

Table A.1 – Type description mapping

No.	Access path data type keyword	MMS type description class and size	Notes
1	BOOL	boolean	
2	SINT	integer 8	4
3	INT	integer 16	4
4	DINT	integer 32	4
5	LINT	integer 64	4
6	USINT	unsigned 8	4
7	UINT	unsigned 16	4
8	UDINT	unsigned 32	4
9	ULINT	unsigned 64	4
10	REAL	floating-point 32,8	4
11	LREAL	floating-point 64,11	4
12	TIME	unsigned 32	4
13	DATE	binary-time true	1
14	TIME_OF_DAY, TOD	binary-time false	1
15	DATE_AND_TIME, DT	binary-time true	1
16	STRING[N]	octet-string N	2
17	BYTE	bit-string 8	4
18	WORD	bit-string 16	4
19	DWORD	bit-string 32	4
20	LWORD	bit-string 64	4
21	enumerated_specification	integer	3
22	subrange_specification	basic data type of the subrange	
23	ARRAY	array	
24	STRUC	structure	

NOTE 1 The values true and false indicate that data is included or not included, respectively.

NOTE 2 The implementer shall specify the maximum length permitted, the given size N is the maximum size of the string.

NOTE 3 The size of the integer shall be selected to hold all possible enumerated values.

NOTE 4 The given size is fixed.

Variables may be addressed using the REMOTE_VAR function. The mapping of the parameters SCOPE and SC_ID are as follows:

Table A.2 – Mapping of the SCOPE and SC_ID parameter

Value of SCOPE parameter	MMS name scope	Usage of the SC_ID parameter
0	VMD	Not used
1	VMD	Not used
2	Domain	Domain name
3	Domain	Domain name
10	VMD	Not used
11	Domain	Domain name
12	Application Association	AA name

A.3.3 Unnamed Variables

Object: Unnamed Variable

Attribute: Address – The Kind of Address parameter has the value 'Symbolic Address'. The value of the Symbolic Address parameter shall be the direct representation of the PC variable.

Attribute: Type Description – This shall be the data type derived from the size prefix of the direct representation mapped according to the following table.

Table A.3 – Size prefix of direct representation

Number	Size prefix of direct representation	MMS type description class and size
1	(none)	boolean
2	X	boolean
3	B	bit-string 8
4	W	bit-string 16
5	D	bit-string 32
6	L	bit-string 64
NOTE The programming languages of IEC 61131-3 define the means of representing variables symbolically, or alternatively in a manner which directly represents the association of the data element with physical or logical locations in the programmable controller's input, output, or memory structure. The Unnamed Variable object is used to access the variables with direct representation.		

A.3.4 Program Invocations

Object: Program Invocation

Attribute: Program Invocation Name – This shall be the name of the associated configuration or resource.

Attribute: Reusable – This shall have the value TRUE.

Attribute: Additional Detail – This consists of the following attributes defined in ISO/IEC 9506-5:

1. Independent – This attribute has the value TRUE if the associated PC language element is a configuration, otherwise it has the value FALSE.
2. Constraint: Independent = FALSE

   Program Invocation Reference – This attribute has the value of the identifier of the configuration.
3. I/O state

The implementer may define additional means to create additional Program Invocation objects.

A.3.5 Domains

Object: Domain

Attribute: Domain Name – This shall be the name of the associated loadable unit.

Attribute: List of Subordinate Objects – If the Named Variable P_DDATE is supported by the implementation, it shall appear in this attribute.

The implementer may define additional means to create additional Domain objects.

A.4 Communication function block mapping to MMS objects and services

The communication function blocks are described using state diagrams with transitions and actions. In this part of the annex, all transitions and actions which refer to the PC communication system are mapped onto ISO/IEC 9506-5.

A.4.1 Using communication channels

Only one-to-one communication channels are provided with ISO/IEC 9506-5.

All requests and all responses mentioned in the following clauses are transmitted using the application association which is referenced by the ID parameter of the communication function blocks. Indications or confirmations have effect to a certain communication function block instance only if they were received using the application association referenced by its ID parameter.

A.4.2 Rules for data type compatibility

The following rules shall apply, when a data protocol unit is received and the action requests to verify if the data type of the received variable object matches with the data type of an application program variable programmed at an RD_i output parameter of the receiving function block.

a) If the application program variable programmed at the RD_i parameter is of a signed or unsigned integer data type, any received variable of signed or unsigned integer type, the value of which can be stored without loss of information shall pass the type check. The application program variable shall get the value of the received variable object.

b) If the application program variable programmed at the RD_i parameter is of BYTE, WORD, DWORD, or LWORD data type, any received variable of bit-string type with a length of which is not bigger than the bit length of the application program variable shall pass the type check. The application program variable shall get the value of the received variable object, storing bit 0 of the received bit-string into bit 0 of the application program variable and so on. The application program variable is filled up with 0 if its length is bigger.

c) If the application program variable programmed at the RD_i parameter is of STRING data type, any received variable with a byte length of which is not bigger than the byte length of the application program variable shall pass the type check. The application program STRING variable shall get the value of the received variable object, storing byte 1 of the received variable into byte 1 of the application program STRING variable and so on. The length of the string shall be set to the count of the received and stored bytes.

d) If the received variable object has the data type octet string the type check shall be passed, if the application program variable has a byte length which is not smaller than the length of the received octet string. The application program variable shall get the value of the received octet string, storing byte 1 of the received variable into byte 1 of the application program variable and so on. The application program variable is filled up with 0 if its length is bigger.

e) If the application program variable programmed at the RD_i parameter is of a data type not mentioned in rule a) to c) and rule d) does not apply, the type check shall be passed if the data type of the received MMS variable object corresponds with the data type of the application program variable using tables A.1 and A.3. The application program variable shall get the value of the received variable object.

A.4.3 Device verification

The STATUS function block uses the Status service as a client.

Table A.4 – Transition mapping of the STATUS state diagram

Transition	Condition
3	Positive response from remote communication partner: Status.Cnf(+)
4	Negative response from remote communication partner received or communication problems detected: Status.Cnf(–) or lower layers indicate error

Table A.5 – Action mapping for STATUS state diagram

State	Actions
WAITING	Request status information from remote device: Status.Req: Extended Derivation: FALSE
HAVE_DATA	Deposit status information in instance: Status.Cnf(+): VMD Logical Status to LOG output VMD Physical Status to PHYS output Local Detail to LOCAL output

The USTATUS function block uses the Unsolicited Status service as a receiver.

Table A.6 – Transition mapping of USTATUS state diagram

Transition	Condition
4	Status information received from remote communication partner: UnsolicitedStatus.Ind
5	Communication problems detected: Lower layers indicate error

Table A.7 – Action mapping of USTATUS state diagram

State	Actions
HAVE_DATA	Deposit status information in instance: UnsolicitedStatus.Ind: VMD Logical Status to LOG output VMD Physical Status to PHYS output Local Detail to LOCAL output

A.4.4 Polled data acquisition

The READ function block uses the Read service as a client.

The strings and the outputs of the REMOTE_VAR function given in the VAR_i input as identifiers of the variables to be read are mapped onto the Variable Access Specification parameter of the Read request. Each identifier of a VAR_i input forms one element of the List of Variable selection.

If the string of a VAR_i input starts with a "%" character, an Unnamed Variable object with a Symbolic Address shall be read. The string of the VAR_i input is used in the symbolicAddress parameter. Otherwise a VMD-specific variable shall be read. The Name Scope parameter of the Object Name parameter shall be set to VMD-specific. The string is used as the identifier.

If a REMOTE_VAR function output is used to identify the remote variable to be read, the Name Scope parameter of the Object Name parameter shall be set as given in the SCOPE input of the REMOTE_VAR function and mapped using table A.2. If the scope is VMD-specific or AA-specific, the value of the NAME input is used as the identifier. If the scope is domain specific the value of the SC_ID input shall be used as identifier of the Domain, and the value of the NAME input as item identifier.

If a derived variable is to be read, the Alternate Access option of the Variable Access Specification is used to describe which sub-element of a structure or element of an array is to be read. The List of Alternate Access Selections shall contain exactly one element. If the SUB input of the REMOTE_VAR function is an integer (signed or unsigned), it shall be used as the Index of the array element to be read. If the SUB input is a not empty string, it shall be used as the component name of the sub-element to be read.

Table A.8 – Transition mapping of the READ state diagram

Transition	Condition
3	Positive response from remote communication partner: Read.Cnf(+)
4	Negative response from remote communication partner or other communication problems detected: Read.Cnf(–) or lower layers indicate error

Table A.9 – Action mapping for READ state diagram

State	Actions
WAITING	Request variables from remote device: Read.Req Specification With Result= FALSE Variable Access Specification
CHECKING	Verify data type match: Check all elements of the List of Access Results against the RD_i output declaration: If Success is false, the check fails If the Kind of Data parameter does not match (see A.4.2) with the data type of the appropriate RD_i output, the check fails. If the check of one data element fails, the complete check fails.
HAVE_DATA	Deposit data: Deposit all data elements of the List of Access Results in the RD_i outputs

A.4.5 Programmed data acquisition

The USEND function block uses the Information Report service as a requester.

One instance of an USEND function block defines a Named Variable object with AA-specific scope. The attributes of this object are as follows:

Object: Named Variable

Attribute: Variable Name – The string given at the R_ID input parameter is the name of the variable with AA-specific scope.

Attribute: Reference to Access Control List – This object shall reference the Access Control List object M_ReadOnly.

Attribute: Type Description – If only one send data parameter is given, the type of the Named Variable object is exactly the type of the send data using tables A.1 and A.3. If more than one send data parameter is programmed, the type of the Named Variable object is a structure containing the send data as components in the same order and with the same type as given as send data parameters.

Table A.10 – Transition mapping of the USEND state diagram

Transition	Condition
3	Communication system indicates "Sent to Remote Communication partner" Lower Layers indicate no errors
4	Communication system indicates "Cannot Send to Remote Communication partner" or other communication problems detected: Lower layers indicate error

Table A.11 – Action mapping for USEND state diagram

State	Actions
TRYING	Send data given at the SD_i inputs to remote communication partner: InformationReport.Req Variable Access Specification Kind Of Variable= NAMED Name Name Scope= AA-SPECIFIC Item Identifier= Content of R_ID input List of Access Results

The URCV function block uses the Information Report service as a receiver.

The string given in the R_ID input is used as the name of the Named Variable object with AA-specific scope which shall be received by the instance of the URCV function block. If only one data output RD_1 is given the type of this parameter shall be checked with the type of the received variable, see A.4.2. If more than one data output RD_1 ... RD_n is given, the type of the received variable object shall be a structure containing the data for the different parameters as components in the same order and with the same type as given in the function block instance.

Table A.12 – Transition mapping of URCV state diagram

Transition	Condition
4	Data from remote communication partner received: InformationReport.Ind Variable Access Specification List of Variable Variable Specification Name Name Scope= AA-SPECIFIC Item Identifier= Value of R_ID parameter List of Access Results
5	Communication problems detected: Lower layers indicate error

6	Data types of SD_i of USEND and RD_i of URCV match: Data parameter of the Access Result parameter match with the count and data type of the RD_i outputs of the URCV instance (see A.4.2).
7	Data types of SD_i of USEND and RD_i of URCV do not match: Data parameter of the Access Result parameter do not match with the count and data type of the RD_i outputs of the URCV instance (see A.4.2).

Table A.13 – Action mapping for URCV state diagram

State	Actions
CHECKING	Verify data type match: Check all elements of the List of Access Results against the RD_i output declaration: If Success is false, the check fails If the Kind of Data parameter does not match with the data type of the appropriate RD_i output (see A.4.2), the check fails If the check of one data element fails, the complete check fails
HAVE_DATA	Deposit data: Deposit all data elements of the List of Access Results in the RD_i outputs

The BSEND function block uses the Write service as a client.

The Named Variable object with AA-specific scope which is written is defined by the corresponding instance of the BRCV function block. The attributes of this object are as follows:

Object: Named Variable

Attribute: Variable Name – The string given at the R_ID input parameter is the name of the variable with AA-specific scope.

Attribute: Reference to Access Control List – This object shall reference the Access Control List object M_ReadOnly.

Attribute: Type Description – The data type is a structure containing an index of data type unsigned integer 32, a more_follows mark of data type boolean and the data to send as an octet string.

```
structure {
    components {
        {component Name = "Index",
         component Type = unsigned -32},
        {component Name = "More_Follows",
         component Type = boolean},
        {component Name = "Data",
         component Type = octet-string} } }
```

Table A.14 – Transition mapping of the BSEND state diagram

Transition	Condition
3	Positive confirmation received and more data to send: Write.Cnf(+)
4	Negative confirmation received or communication problems detected: Write.Cnf(–) or lower layers indicate error
6	Positive confirmation received and no more data to send: Write.Cnf(+)
9	Communication problems detected: Lower layers indicate error

Table A.15 – Action mapping for BSEND state diagram

State	Actions
SEND_FIRST	Send first block of data given at the SD_1 input to remote communication partner, send a maximum of LEN bytes in total: Write.Req Variable Access Specification List of Data with component Index= 1 (= element number of the first byte in the byte array to send) component More_Follows= true if not all data to send are contained in component Data component Data= the first bytes out of the data to send
SEND_NEXT	Send next block of data given at the SD_1 input to remote communication partner, send a maximum of LEN bytes in total: Write.Req Variable Access Specification List of Data with component Index= Element number of the first byte of the octet string in the byte array to send component More_Follows= true if not all data still to send are contained in component Data component Data= bytes out of the data to send starting with the byte the number of which is contained in component Index
CANCEL	Stop the data transfer: Write.Req Variable Access Specification List of Data with component Index=0 with component More_Follows= false with component Data= null

The BRCV function block uses the Write service as a server.

The string given in the R_ID input is the name of the Named Variable object with AA-specific scope, which is defined by the instance of the BRCV function block.

Table A.16 – Transition mapping of BRCV state diagram

Transition	Condition
4	Data received from remote communication partner: Write.Ind
5	More data follows is true: Component More_Follows= true
6	More data follows is false: Component More_Follows= false
7	Indication to cancel data transfer received: Write.Ind with component Index= 0
9	Communication problems detected: Lower layers indicate error

Table A.17 – Action mapping for BRCV state diagram

State	Actions
RECEIVING	Verify data type match: The check fails if the length of octet string of the just received component Data plus the value contained in the just received component Index minus 1 is greater than the element count of the RD_1 array of byte
RESP_MORE	Send positive response: Write.Rsp(+)
RESP_LAST	Send positive response: Write.Rsp(+)
HAVE_IT	Deposit data: Store the bytes of the component Data successively starting with the element whose number is given in the component Index
CANCELLED	Send positive response: Write.Rsp (+)

A.4.6 Parametric control

The WRITE function block uses the Write service as a client.

The strings and the outputs of the REMOTE_VAR function given in the VAR_i input as identifiers of the variables to be written are mapped onto the Variable Access Specification parameter of the Write request. Each identifier of a VAR_i input forms one element of the List of Variable selection.

If the string of a VAR_i input starts with a "%" character, an Unnamed Variable object with a Symbolic Address shall be written. The string of the VAR_i input is used in the symbolicAddress parameter. Otherwise a VMD-specific variable shall be written. The Name Scope parameter of the Object Name parameter shall be set to VMD-specific. The string is used as the identifier.

If a REMOTE_VAR function output is used to identify the remote variable to be written, the Name Scope parameter of the Object Name parameter shall be set as given in the SCOPE input of the REMOTE_VAR function and mapped using table A.2. If the scope is VMD-specific or AA-specific, the value of the NAME input is used as the identifier. If the scope is domain specific the value of the SC_ID input shall be used as identifier of the Domain, and the value of the NAME input as item identifier.

If a derived variable is to be written, the Alternate Access option of the Variable Access Specification is used to describe which sub-element of a structure or element of an array is to be written. The List of Alternate Access Selections shall contain exactly one element. If the SUB input of the REMOTE_VAR function is an integer (signed or unsigned), it shall be used as Index of the array element to be written. If the SUB input is a not empty string, it shall be used as component name of the sub-element to be written.

The data given at the SD_i inputs give the List of Data parameter of the Write request.

Table A.18 – Transition mapping of the WRITE state diagram

Transition	Condition
3	Positive response of remote communication partner: Write.Cnf(+)
4	Negative response from remote communication partner or other communication problems detected: Write.Cnf(–) or lower layers indicate error

Table A.19 – Action mapping for WRITE state diagram

State	Actions
WAITING	Request to write variables into remote communication partner: Write.Req Variable Access Specification List of Data

A.4.7 Interlocked control

The SEND function block uses the Exchange Data service as a client.

The string given in the R_ID input is used as Item Identifier of the AA-specific data exchange object. The data inputs SD_1 ... SD_n form the List of Request Data parameter of the Data Exchange request in the same order and with the same type as given in the SEND function block invocation. The data of the List of Response Data parameter of the Data Exchange confirmation is stored in the data outputs RD_1 ... RD_m in the same order as given in the SEND function block invocation.

Table A.20 – Transition mapping of the SEND state diagram

Transition	Condition
3	Positive response from RCV: ExchangeData.Cnf(+)
4	Negative response from RCV: ExchangeData.Cnf(–)
5	Data types of the received data and RD_1 to RD_m match: Data of the List of Response Data parameter and RD_1 to RD_m match
6	Data type of the received data and RD_1 to RD_m mismatch Data of the List of Response Data parameter and RD_1 to RD_m mismatch
8	Positive or negative response to the reset request: Cancel.Cnf
9	Positive or negative response from RCV: ExchangeData.Cnf
12	Communication problems detected: Lower layers indicate error

Table A.21 – Action mapping for SEND state diagram

State	Actions
SEND_AND_WAIT	Send Data to RCV: ExchangeData.Req Data Exchange Name Name Scope= AA-SPECIFIC Item Identifier= Value of R_ID parameter List of Request Data
CHECKING	Verify data type match: Check data type of List of Response Data parameter of the ExchangeData.Cnf(+), see A.4.2
CANCEL	Request to reset RCV: Cancel.Req Original Invoke ID

The RCV function block uses the Exchange Data service as a server.

One instance of an RCV function block defines a Data Exchange object with AA-specific scope. The attributes of this object are as follows:

Object: Data Exchange

Attribute: Data Exchange Name – The string given at the R_ID input parameter is the name of the Data Exchange object with AA-specific scope.

Attribute: Reference to Access Control List – This object shall reference an Access Control List object M_NonDeletable.

Attribute: List of Request Type Specifications – The List of Request Type Specifications contains exactly the types of the receive data given at the RD_i outputs using tables A.1 and A.3.

Attribute: List of Response Type Specifications – The List of Response Type Specifications contains exactly the types of the send data given at the SD_i inputs using tables A.1 and A.3.

Attribute: Linked – This attribute shall have the value FALSE.

Table A.22 – Transition mapping of RCV state diagram

Transition	Condition
4	When data received from SEND: ExchangeData.Ind Data Exchange Name Name Scope= AA-SPECIFIC Item Identifier= Value of R_ID parameter List of Request Data
5	Data types of received data and RD_1 to RD_n match: Data of the List of Request Data parameter and RD_1 to RD_n match
6	Data types of the received data and RD_1 to RD_n mismatch: Data of the List of Request Data parameter and RD_1 to RD_n mismatch
8	When reset is requested by SEND: Cancel.Ind Original Invoke ID
9	Communication problems detected: Lower layers indicate error

Table A.23 – Action mapping of RCV state diagram

State	Actions
CHECKING	Verify data type match: Check result of the verification of the conformance of the List of Request Type Specification attribute and the List of Request Data parameter (see A.4.2).
NAK_ERROR	Send negative response to SEND: ExchangeData.Rsp(–) Error Class= DEFINITION Error Code= TYPE-INCONSISTENT
NAK_BUSY	Send negative response to SEND: ExchangeData.Rsp(–) Error Class= SERVICE Error Code= OBJECT-STATE-CONFLICT
NAK_CANCELLED	Send positive response to the reset request and then send negative response to SEND: Cancel.Rsp(+) ExchangeData.Rsp(–) Error Class= SERVICE-PREEMPT Error Code= CANCEL
RESPONDING	Send positive response with user data to SEND: ExchangeData.Rsp(+) List of Response Data= Data given at the SD_i inputs
NAK_DIS	Send negative response to SEND: ExchangeData.Rsp(–) Error Class= ACCESS Error Code= OBJECT-NON-EXISTENT

A.4.8 Programmed alarm report

The NOTIFY and the ALARM function block use the Event Notification service as a requester. Additionally, the ALARM function block uses the Acknowledge Event Notification service as a server.

One instance of a NOTIFY or an ALARM function block defines a Named Variable object with AA-specific scope. This variable object shall be transmitted in a Read response of the Confirmed Service Response selection of the Action Result parameter. The attributes of this object are as follows:

Object: Named Variable

Attribute: Variable Name – The string given at the EV_ID input parameter is the name of the variable with AA-specific scope.

Attribute: Reference to Access Control List – This object shall reference an Access Control List object M_ReadOnly.

Attribute: Type Description – If only one send data parameter is given, the type of the Named Variable object is exactly the type of the send data using tables A.1 and A.3. If more than one send data parameter is programmed, the type of the Named Variable object is a structure containing the send data as components in the same order and with the same type as given as send data parameters.

One instance of a NOTIFY or an ALARM function block defines a Event Condition object with AA-specific scope. The attributes of this object are as follows:

Object: Event Condition

Attribute: Event Condition Name – This attribute shall have the value of the EV_ID input parameter.

Attribute: Event Condition Class – This attribute shall have the value MONITORED.

Attribute: Reference to Access Control List – This object shall reference an Access Control List object M_NonDeletable.

Attribute: Priority – The value of this attribute shall be specified by the implementer.

Attribute: Severity – This attribute is taken from the SEVERITY input parameter of the function block invocation.

Attribute: List of Event Enrolment Reference – This attribute contains the reference to the Event Enrolment object predefined by this function block instance.

Attribute: Enabled – This attribute is taken from the EN_R input parameter of the function block invocation.

Attribute: Alarm Summary Reports – This attribute shall have the value FALSE.

Attribute: Monitored Variable Reference – This attribute shall have the value UNSPECIFIED.

Attribute: Evaluation Interval – This attribute is not supported, because the Parameter CBB and CEI is not requested in any conformance class of ISO/IEC 9506-5.

One instance of a NOTIFY or an ALARM function block defines a Event Enrolment object with AA-specific scope. The attributes of this object are as follows:

Object: Event Enrolment

Attribute: Event Enrolment Name – This attribute shall have the value of the EV_ID input parameter.

Attribute: Reference to Access Control List – This object shall reference an Access Control List object M_NonDeletable.

Attribute: Enrolment Class – This attribute shall have the value NOTIFICATION.

Attribute: Event Condition Reference – The Event Condition predefined by this function block instance is referenced.

Attribute: Notification Lost – The value of this attribute is specified by the implementer.

Attribute: Event Action Reference – This attribute contains the reference to the Event Action object predefined by this function block instance.

Attribute: Duration – This attribute has the value PERMANENT.

Attribute: Alarm Acknowledgement Rule – This attribute shall have the value NONE, if a NOTIFY function block is programmed. It shall have the value ACK-ALL, if an ALARM function block is programmed.

If one or more event data inputs SD_i are programmed at the instance of a NOTIFY or an ALARM function block, this instance defines an Event Action object with AA-specific scope. The attributes of this object are as follows:

Object: Event Action

Attribute: Event Action Name – This attribute shall have the value of the EV_ID input parameter.

Attribute: Reference to Access Control List – This object shall reference an Access Control List object M_NonDeletable.

Attribute: Confirmed Service Request – The Read service is used as Event Action. The service arguments shall be:

Attribute: Specification with Result – FALSE

Attribute: Variable Access Specification – This argument shall specify the associated variable of the function block instance.

Attribute: List of Modifier – This attribute shall be empty.

Attribute: List of Event Enrolment Reference – This attribute contains the reference to the Event Enrolment object predefined by the function block instance.

Table A.24 – Transition mapping of the NOTIFY state diagram

Transition	Condition
4	Communication system indicates "Sent to remote communication partner": Lower layers indicated no error
5	Communication system indicates "Cannot send to remote communication partner": Lower layers indicate error

Table A.25 – Action mapping for NOTIFY state diagram

State	Actions
TRY_UP	If the Event Condition is in DISABLED state, an error with status code 10 shall be reported, otherwise request to report the coming alarm to the remote communication partner: EventNotification.Req Event Enrolment Name Name Scope= AA-SPECIFIC Item Identifier= Value of EV_ID parameter Event Condition Name Name Scope= AA-SPECIFIC Item Identifier= Value of EV_ID parameter Severity= Value of SEVERITY parameter Current State= ACTIVE Alarm Acknowledge Rule= NONE Action Result Event Action Name Name Scope= AA-SPECIFIC Item Identifier= Value of EV_ID parameter Success Confirmed Service Response
TRY_DOWN	If the Event Condition is in DISABLED state, an error with status code 10 shall be reported, otherwise request to report the going alarm to the remote communication partner: EventNotification.Req Event Enrolment Name Name Scope= AA-SPECIFIC Item Identifier=Value of EV_ID parameter Event Condition Name Name Scope= AA-SPECIFIC Item Identifier=Value of EV_ID parameter Severity= Value of SEVERITY parameter Current State= IDLE Transition Time= Time the raising edge of the EVENT parameter was detected Alarm Acknowledge Rule= NONE Action Result Event Action Name Name Scope= AA-SPECIFIC Item Identifier= Value of EV_ID parameter Success Confirmed Service Response

Table A.26 – Transition mapping of the ALARM state diagram

Transition	Condition
6	Receive acknowledge of the report of the coming alarm: AcknowledgeEventNotification.Ind Event Enrolment Name Name Scope= AA-SPECIFIC Item Identifier= Value of EV_ID parameter Acknowledged State=ACTIVE
7	Receive acknowledge of the report of the going alarm AcknowledgeEventNotification.Ind Event Enrolment Name Name Scope= AA-SPECIFIC Item Identifier= Value of EV_ID parameter Acknowledged State=IDLE

Table A.27 – Action mapping for ALARM state diagram

State	Actions
U1, U2, U3	If the Event Condition is in DISABLED state, an error with status code 10 shall be reported, otherwise request to report the coming alarm to the remote communication partner: EventNotification.Req Event Enrolment Name Name Scope= AA-SPECIFIC Item Identifier= Value of EV_ID parameter Event Condition Name Name Scope= AA-SPECIFIC Item Identifier= Value of EV_ID parameter Severity= Value of SEVERITY parameter Current State= ACTIVE_NO-ACK-A Transition Time= Time the raising edge of the EVENT parameter was detected Alarm Acknowledge Rule= ACK-ALL Action Result Event Action Name Name Scope= AA-SPECIFIC Item Identifier=Value of EV_ID parameter Success Confirmed Service Response
ACK_UP	Confirm received acknowledgement positively: AcknowledgeEventNotification.Rsp(+) Set the states of all event enrolments identified by the EV_ID parameter to ACTIVE ACKED
D2	If the Event Condition is in DISABLED state, an error with status code 10 shall be reported, otherwise request to report the going alarm to the remote communication partner, see state U1 but: EventNotification.Req Current State= IDLE NO_ACK_A
D1	If the Event Condition is in DISABLED state, an error with status code 10 shall be reported, otherwise request to report the going alarm to the remote communication partner, see state U1 but: EventNotification.Req Current State= IDLE ACKED
ACK_DOWN	Confirm received acknowledgement positively: AcknowledgeEventNotification.Rsp(+) Set the states of all event enrolments identified by the EV_ID parameter to IDLE ACKED
ERROR	Confirm received acknowledgement negatively: AcknowledgeEventNotification.Rsp(–) Error Class= SERVICE Error Code= OBJECT-STATE-CONFLICT

A.4.9 Connection management

The CONNECT function block uses the Initiate service and the Conclude service as a client and a server.

The PARTNER input is used to identify the application entity to which a connection is requested.

Table A.28 – Transitions of the CONNECT state diagram

Transition	Condition
4	Receive a positive confirmation to the request to establish a new connection from the remote communication partner: Initiate.Cnf(+)
5	Receive a negative confirmation to the request to establish a new connection from the remote communication partner: Initiate.Cnf(–)
6	Receive a positive confirmation to the request to conclude the connection from the remote communication partner: Conclude.Cnf(+)
7	Receive a negative confirmation to the request to conclude the connection from the remote communication partner: Conclude.Cnf(–)
8	Loss of connection, i.e. loss of the application association
9	Receive a request to establish a new connection to the remote communication partner: Initiate.Ind
10	Receive a request to conclude the connection to the remote communication partner: Conclude.Ind

Table A.29 – Action mapping for CONNECT state diagram

State	Actions
CHECKING_1	Check if the connection can be established: Check if the constraints identified in the Initiate.Ind primitive can be accepted, or if alternate values can be proposed
DENY	Send a negative response to the request of the connection: Initiate.Rsp(–) Error Type Error Class= INITIATE Error Code= OTHER
ACCEPT	Accept to establish a connection and send a positive response: Initiate.Rsp(+)
CONCL_PASSIV	Send positive response to conclude the connection: Conclude.Rsp(+)
CONNECTING	Request a connection to the remote communication partner: Initiate.Req
ABORT_1	Request to abort communication: Abort.Req
CHECKING_2	Check if the connection can be established: Check if the application association can be established
ABORT_2	Request to abort communication: Abort.Req

State	Actions
CLOSING	Request to conclude the connection: Conclude.Req
CONCLUDING	Send positive response to conclude the connection: Conclude.Rsp(+)

A.5 Implementation specific features and parameters

Implementation specific features and parameters defined in this annex and the most relevant subclause for each are listed in the following table.

Table A.30 – Implementation specific features and parameters

Subclause	Implementation specific features and parameters
A.6.1	Mapping of the PC system onto the VMD of ISO/IEC 9506-5
A.6.2	The maximum length permitted for data of data type STRING when communicating in the abstract syntax of ISO/IEC 9506-5
A.6.4	Means to create additional Program Invocation objects
A.6.5	Means to create additional Domain objects
A.7	If communication channels established using the FB CONNECT are reusable or not
A.7.6	The value of the Priority attribute of the MMS Event Condition object
A.7.6	The value of the Notification Lost attribute of the MMS Event Enrolment object

Annex B
(normative)

PC behavior using ISO/IEC 9506-2

B.1 PC communications mapping to MMS

This annex specifies the behavior of a PC when it is communicating in the abstract syntax defined by ISO/IEC 9506-2.

ISO/IEC 9506-1/2 specifies the model of a VMD and specifies how services are provided by a VMD. ISO/IEC 9505-5 extends the VMD for a PC e.g. by adding the concept of I/O state. It also extends the services provided by a PC to specify the behavior of a PC.

The general approach is that the PC behaves as if it is communicating in the abstract syntax defined in ISO/IEC 9506-5, with defaults applied where the services has been extended in ISO/IEC 9506-5.

For the CreateProgramInvocation Service, the following defaults shall be used:

Table B.1 – CreateProgramInvocation service defaults

PC language element of IEC 61131-3	Independent attribute	Program Invocation Reference attribute
Configuration	TRUE	
Resource	FALSE	Identifier of the Program Invocation of the configuration
Others	TRUE	

For the following services, the PC shall behave according to the service extensions defined in ISO/IEC 9506-5 with the value of the I/O State used in the following table, including the service procedure extensions related to the Independent and Program Invocation Reference attributes.

Table B.2 – Program Invocation service defaults for I/O State parameter

MMS service	I/O State value
Start	0 – Controlled
Resume	0 – Controlled
Stop	3 – Implementer State
Kill	3 – Implementer State

The service procedure extensions defined for the GetProgramInvocationAttributes service shall not be used.

B.2 Implementation specific features and parameters

Implementation specific features and parameters used in this annex of IEC 61131-5 and the most relevant subclause for each are listed in the following table.

Table B.3 – Implementation specific features and parameters

Subclause	Implementation specific features and parameters
A.6.1	Mapping of the PC system onto the VMD of ISO/IEC 9506-1
A.6.2	The maximum length permitted for data of data type STRING when communicating in the abstract syntax of ISO/IEC 9506-1
A.6.4	Means to create additional Program Invocation objects
A.6.5	Means to create additional Domain objects
A.7	If communication channels established using the FB CONNECT are reusable or not
A.7.6	The value of the Priority attribute of the MMS Event Condition object
A.7.6	The value of the Notification Lost attribute of the MMS Event Enrolment object

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

AUTOMATES PROGRAMMABLES –

Partie 5: Communications

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.

La Norme internationale CEI 61131-5 a été établie par le sous-comité 65B: Dispositifs, du comité d'études 65 de la CEI: Mesure et commande dans les processus industriels.

La présente version bilingue (2012-08) correspond à la version anglaise monolingue publiée en 2000-11.

Le texte anglais de cette norme est issu des documents 65B/411/FDIS et 65B/420/RVD.

Le rapport de vote 65B/420/RVD donne toute information sur le vote ayant abouti à l'approbation de cette norme.

La version française n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 3.

Il convient de lire la présente partie conjointement avec les autres parties de la CEI 61131. La CEI 61131 est constituée des parties suivantes, sous le titre général: *Automates programmables*.

Partie 1:1992, Informations générales

Partie 2:1992, Spécifications et essais des équipements

Partie 3:1993, Langages de programmation

Partie 4:1994: Guide pour l'utilisateur (publié comme rapport technique CEI/TR 61131-4)

Partie 5:2000, Communications

Partie 8:2000, Lignes directrices pour l'application et la mise en œuvre des langages de programmation (publié comme rapport technique CEI/TR 61131-8)

Les annexes A et B font partie intégrante de la présente norme.

L'annexe C est donnée uniquement à titre d'information.

En cas de conflit entre cette norme et d'autres normes CEI (à l'exception des normes de base sur la sécurité), il convient de considérer les dispositions de cette norme comme régissant le domaine des automates programmables et de leurs périphériques associés.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant 2006. A cette date, la publication sera

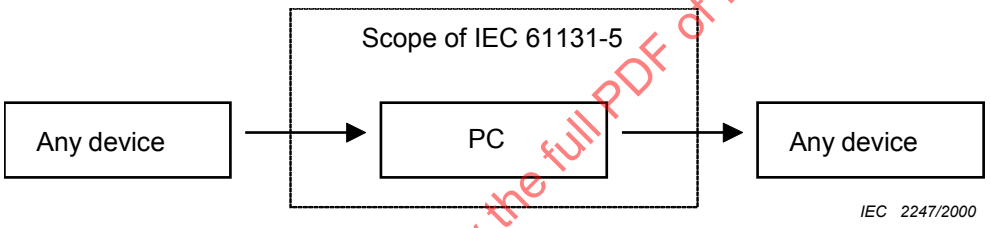
- reconduite;
- supprimée;
- remplacée par une édition révisée, ou
- amendée.

AUTOMATES PROGRAMMABLES –

Partie 5: Communications

1 Domaine d'application

La présente partie de la CEI 61131 spécifie les aspects des automates programmables (AP) relatifs à la communication. Elle spécifie, du point de vue d'un AP, la manière dont chaque périphérique peut communiquer avec un AP en tant que serveur et la manière dont l'AP peut communiquer avec chaque périphérique. Elle spécifie, en particulier, le comportement de l'AP lorsqu'il fournit des services pour le compte d'autres périphériques et les services que le programme application de l'AP peut demander aux autres périphériques. Elle n'est pas destinée à spécifier la manière dont chaque périphérique peut communiquer avec un autre périphérique en utilisant un AP comme routeur ou comme passerelle. Le comportement de l'AP comme client et serveur de communication est spécifié indépendamment du sous-système de communication particulier mais la fonctionnalité de communication peut dépendre des capacités du sous-système de communication utilisé.



Légende

Anglais	Français
Scope of IEC 61131-5	Domaine d'application de la CEI 61131-5
Any device	Tout périphérique
PC	AP

Figure 1 – Domaine d'application de la présente partie de la CEI 61131

Le domaine d'application de la présente partie est un sous-ensemble du "modèle de communication" représenté sur la Figure 2 de la CEI 61131-3; c'est-à-dire que les Figures 2c et 2 d sont incluses dans le domaine d'application de la présente partie. De plus, les moyens définis dans cette partie de la CEI 61131 peuvent être utilisés pour les communications dans un programme ou entre des programmes.

Le mapping du comportement de l'AP pour certains sous-systèmes de communication particuliers est présenté dans les annexes.

2 Références normatives

Les documents normatifs suivants contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions valables pour la présente partie de la CEI 61131. Pour les références datées, les amendements ultérieurs ou les révisions de ces publications ne s'appliquent pas. Toutefois, les parties qui veulent établir des accords fondés sur la présente partie de la CEI 61131 sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des documents normatifs indiqués ci-après. Pour les références non datées, la dernière édition du document normatif en référence s'applique. Les membres de la CEI et de l'ISO possèdent le registre des Normes internationales en vigueur.

CEI 60050-351:1998, *Vocabulaire électrotechnique international – Partie 351: Commande et régulation automatiques*

CEI 61131-1:1992, *Automates programmables – Partie 1: Informations générales*

CEI 61131-2:1992, *Automates programmables – Partie 2: Spécifications et essais des équipements*

CEI 61131-3:1993, *Automates programmables – Partie 3: Langages de programmation*

ISO/CEI 2382-1:1993, *Traitement de l'information – Vocabulaire – Partie 1: Termes fondamentaux*

ISO/CEI 9506-1:1990, *Systèmes d'automatisation industrielle – Spécification de messagerie industrielle – Partie 1: Définition des services*

ISO/CEI 9506-2:1990, *Systèmes d'automatisation industrielle – Spécification de messagerie industrielle – Partie 2: Spécification de protocole*

3 Définitions

Pour les besoins de la présente partie de la CEI 61131, les définitions suivantes s'appliquent.

La présente partie de la CEI 61131 est basée sur les concepts des parties 1 à 3 de la CEI 61131 et utilise les termes suivants, définis dans d'autres normes internationales.

Définitions d'autres publications

CEI 60050-351

régulation

surveillance

CEI 61131-1

programme application (2.1)

archivage de programme application (4.6.4)

redémarrage à froid (2.56)

entrée (2.25)

processeur principal (2.32)

modification du programme application (4.6.2.6)

sortie (2.40)

automate programmable (2.50)

système d'automate programmable (2.51)

essai du programme application (4.6.2.5)

redémarrage à chaud (2.56)

CEI 61131-3

chemin d'accès (1.3.2)

représentation directe (1.3.23)

invocation (1.3.43)

programmer (verbe, 1.3.60)

sous-élément (2.3.3.1)

ISO/CEI 2382-1

données

ISO/CEI 9506-1

client

télécharger

événement (article 15)

serveur

accès variable sans interruption (12.1.1.1)

charger

variable

Définitions de la présente partie

3.1
alarme
événement qui signale une condition spécifique

3.2
acquisition de données
collecte de données dans un but de surveillance du processus et de génération de rapports

3.3
interface opérateur directe
lorsque le client peut communiquer avec l'interface opérateur via le système de communication sans interaction d'un programme application

3.4
vérification de périphérique
permet aux autres périphériques de déterminer si l'AP est capable d'exécuter ses fonctions dans le système de commande

3.5
santé
la santé d'un AP ou de ses sous-systèmes est spécifiée par le retour de une et seulement une parmi trois valeurs possibles. Il s'agit, dans l'ordre de santé décroissante, de: GOOD, WARNING et BAD (bonne, avertissement, mauvaise)

3.6
commande verrouillée
commande effectuée par l'intermédiaire d'échanges de données entre deux parties. À plusieurs moments, l'une des parties attend que l'autre partie délivre certaines données attendues

3.7
local
interne à l'AP; inverse de distant

3.8
commande paramétrique
commande effectuée par le client qui écrit des variables de commande résidant dans l'AP

3.9
unité de traitement
partie du processeur principal. Il s'agit de la partie du système d'un AP responsable du stockage du programme et des données de l'application et de l'exécution du programme application. Un système d'AP possède une ou plusieurs unités de traitement.

3.10
vérification de programme
essai du programme application d'un AP destiné à vérifier qu'il exécute la ou les fonction(s) pour laquelle ou lesquelles il a été conçu dans l'environnement du processus

3.11
recette
description des procédures ou des données pour ces procédures, ou des deux, permettant de fabriquer un produit qui utilise le processus ou les machines auxquels est associé l'automate et qui est différent du produit précédent

3.12
distant
externe à l'AP; inverse de local

3.13
état
l'état du système d'un AP est indiqué par une liste d'attributs dont chacun peut être TRUE (vrai) ou FALSE (faux). Aucun, un ou plusieurs de ces attributs peuvent être vrais en même temps

3.14
non sollicité
exécuté sans requête explicite

4 Symboles et abréviations

Il s'agit de certaines abréviations fréquemment utilisées dans la présente partie de la CEI 61131. Ces termes sont définis ou référencés à l'Article 3 de cette partie de la CEI 61131.

CFB	Bloc fonctionnel de communication (*Communication function block*)
FB	Bloc fonctionnel (*Function block*)
E/S	Entrée et sortie
CEI	Commission Electrotechnique Internationale
ISO	International Organization for Standardization
MMS	Spécification de messagerie industrielle (*Manufacturing Message Specification*), ISO/CEI 9506-1 et ISO/CEI 9506-2
OSI	Interconnexion de systèmes ouverts (*Open Systems Interconnection*)
PADT	Outil de programmation et de mise au point (*Programming and debugging tool*)
AP	Automate programmable
PU	Unité de traitement (*Processing unit*)

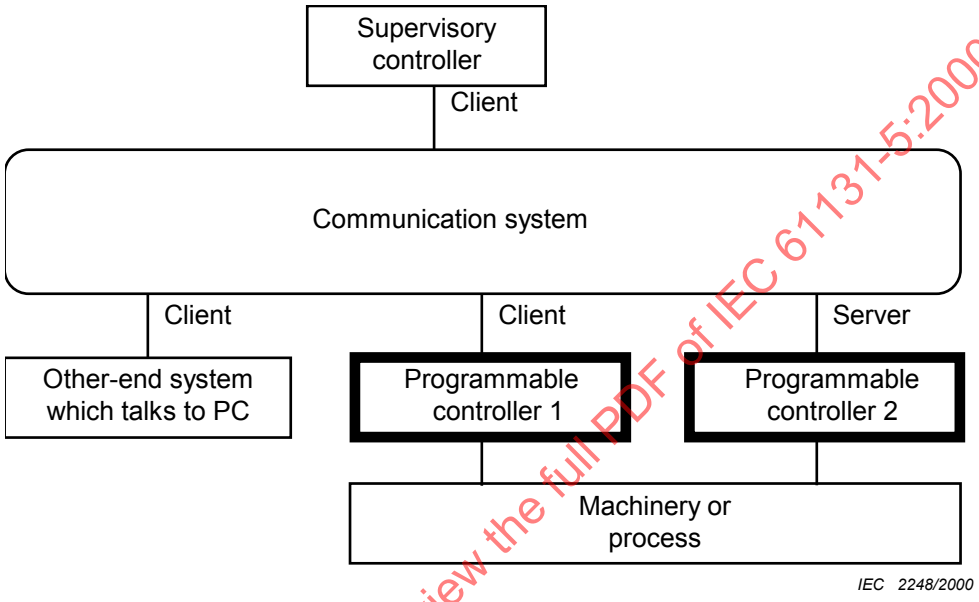
5 Modèles

Le présent article spécifie les modèles utilisés dans le reste de cette partie de la CEI 61131.

5.1 Modèle du réseau de communication de l'AP

Un automate programmable fournit des fonctions d'application spécifiques au reste du système de commande. Il peut également demander des fonctions à d'autres automates programmables. Les fonctions de communication définies dans la présente partie de la norme CEI 61131 sont fondées sur un sous-système de communication qui peut signaler des erreurs de communication à la fonction de traitement du signal de l'AP (voir 5.2).

Le schéma qui suit illustre les dispositifs d'un réseau de communication, représentant trois dispositifs pouvant solliciter les fonctions de l'AP (client) depuis l'AP 2. Les deux AP surlignés font partie du domaine d'application de la présente partie de la CEI 61131.



Légende

Anglais	Français
Supervisory controller	Contrôleur de supervision
Client	Client
Communication system	Système de communication
Other-end system which talks to PC	Autres systèmes périphériques qui échangent avec l'AP
Programmable controller	Automate programmable
Machinery or process	Machines ou processus

NOTE Du point de vue de la communication, le "contrôleur de supervision" et le "système périphérique qui communique avec l'AP" mentionnés sur cette figure affichent le même comportement qu'un serveur de communication de l'AP, c'est-à-dire qu'ils soumettent leurs requêtes à l'AP2.

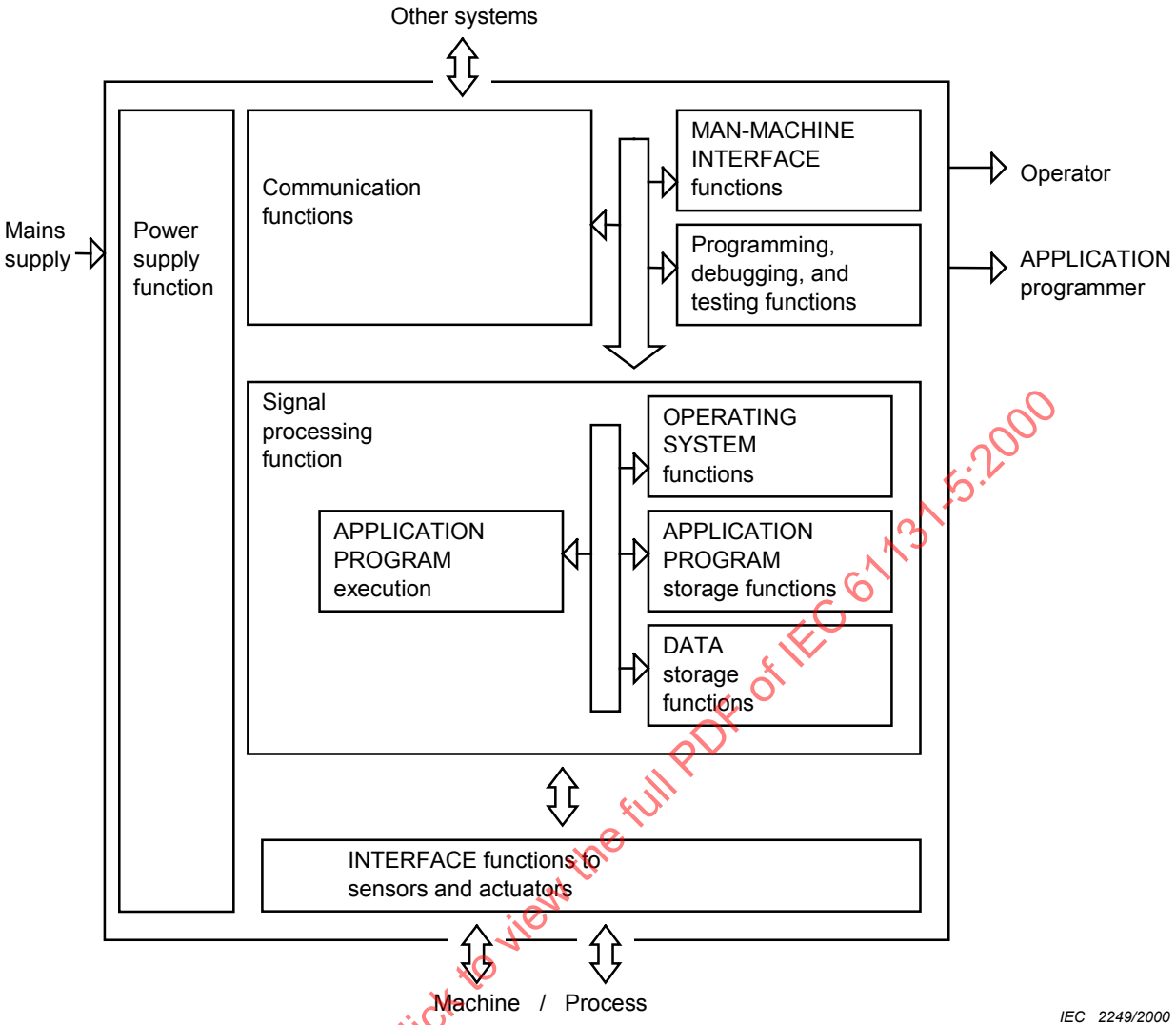
Figure 2 – Modèle de communication de l'AP

Un AP peut utiliser sa fonction client pour communiquer avec tout dispositif se comportant comme un AP.

5.2 Modèle fonctionnel de l'AP

Un AP est constitué de plusieurs fonctions (voir Figure 3). Pour un AP relevant du domaine d'application de cette partie de la CEI 61131, au moins une fonction de communication est présente.

Le schéma qui suit provient de la CEI 61131-1, Figure 1. Il est conçu pour illustrer certains des sous-systèmes d'un AP classique.



Légende

Anglais	Français
Other systems	Autres systèmes
Mains supply	Réseau d'alimentation
Operator	Opérateur
Application program	Programme application
Power supply function	Fonction d'alimentation
Communication functions	Fonctions de communication
Signal processing function	Fonction de traitement de signal
Man-machine interface functions	Fonctions d'interface homme-machine
Programming, debugging and testing functions	Fonctions de programmation, mise au point et essais
Application program execution	Exécution du programme application
Operating system functions	Fonctions du système d'exploitation
Application program storage functions	Fonctions de stockage du programme application
Data storage functions	Fonctions de stockage des données
Interface functions to sensors and actuators	Fonctions d'interface avec les capteurs et les actionneurs
Machine / process	Machine / processus

Figure 3 – Modèle fonctionnel de l'automate programmable

Une fonction, intégrée au système de l'AP mais généralement externe à l'AP lui-même, est connue sous le nom d'outil de programmation et de mise au point (PADT). Le PADT est modélisé pour interagir avec l'AP via la fonction de communication.

La fonction d'interface avec les capteurs et les actionneurs peut disposer d'E/S locales ou distantes au processeur principal (voir le modèle matériel en 5.3). La fonction d'interface avec les capteurs et les actionneurs possède deux attributs pour chaque programme application, qui définissent la manière dont l'AP surveille et commande la machine/le processus. L'attribut d'entrée peut présenter les états suivants:

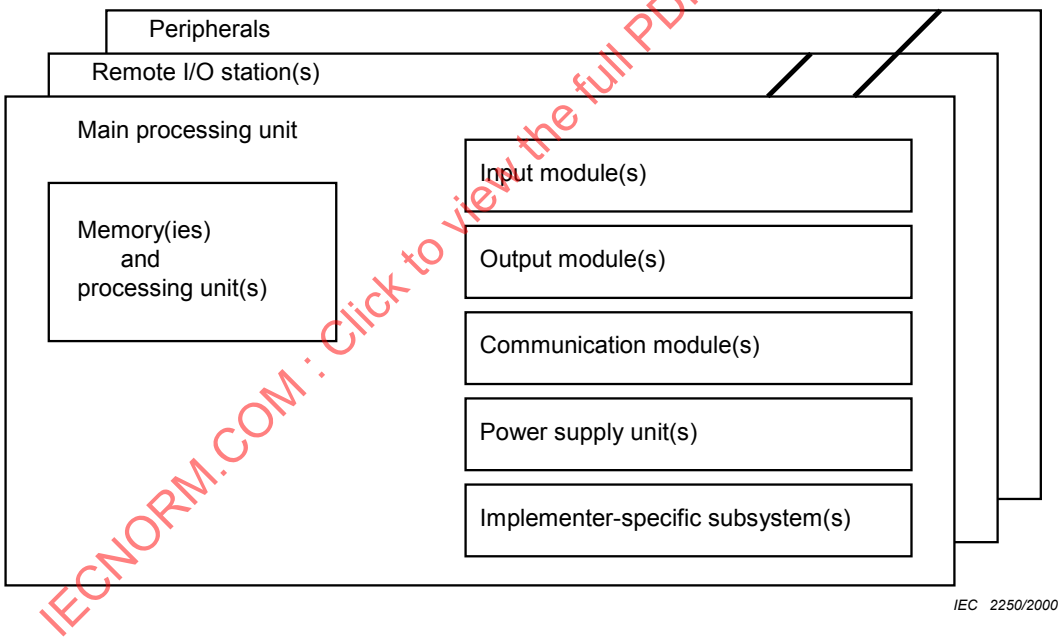
- les entrées appliquées au programme application sont fournies par les capteurs,
- les entrées appliquées au programme application sont maintenues dans l'état courant.

L'attribut de sortie peut présenter les états suivants:

- les actionneurs sont contrôlés par le programme application,
- les actionneurs sont maintenus dans l'état courant.

5.3 Modèle matériel de l'AP

La figure qui suit représente le modèle matériel de l'AP. Elle représente les modules qui constituent un AP. Un sous-système d'AP est constitué d'un ou plusieurs modules. La figure qui suit correspond à la Figure B.1 de la CEI 61131-1 et à la Figure 1 de la CEI 61131-2.



Légende

Anglais	Français
Peripherals	Périphériques
Remote I/O station(s)	Station(s) d'E/S distante
Main processing unit	Processeur principal
Memory(ies) and processing unit(s)	Mémoire(s) et unité(s) de traitement
Input module(s)	Module(s) d'entrée
Output module(s)	Module(s) de sortie
Communication module(s)	Module(s) de communication
Power supply unit(s)	Unité(s) d'alimentation
Implementer specific subsystem(s)	Sous-système(s) spécifique(s) à l'intégrateur

Figure 4 – Modèle matériel de l'automate programmable

5.4 Modèle logiciel

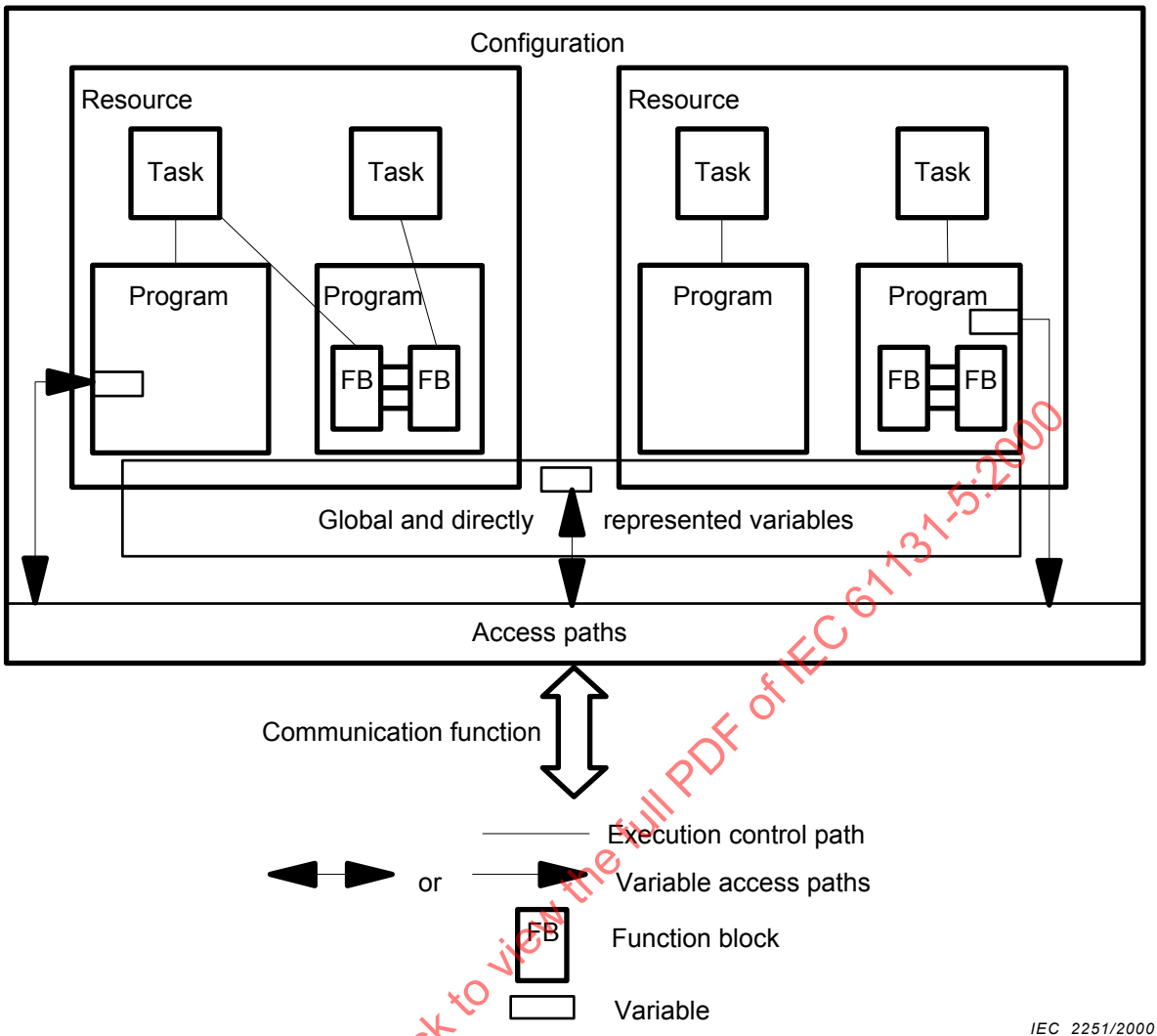
La Figure 5 représente le modèle logiciel de l'AP défini dans la CEI 61131-3, Figure 1. Elle illustre les éléments de langage de base de haut niveau des langages de programmation de l'AP et leurs relations. Ceux-ci sont constitués d'éléments programmés au moyen des langages définis dans la CEI 61131-3, c'est-à-dire des programmes et des blocs fonctionnels; et des éléments de configuration, à savoir des configurations, des ressources, des tâches, des variables globales et des chemins d'accès, qui prennent en charge l'installation de programmes d'automate programmable dans les systèmes d'automates programmables.

Une configuration est l'élément de langage qui correspond à un système d'automate programmable tel que défini dans la CEI 61131-1. Une ressource correspond à une "fonction de traitement de signal" et à ses fonctions "d'interface homme-machine" et "d'interface capteur et actionneur" (lorsqu'elles existent) telles que définies dans la CEI 61131-1. Une configuration contient une ou plusieurs ressources, chacune contenant un ou plusieurs programmes exécutés sous le contrôle de zéro ou plusieurs tâches. Un programme peut contenir zéro ou plusieurs blocs fonctionnels ou d'autres éléments de langage tels que définis dans la CEI 61131-3.

Les configurations et les ressources peuvent être démarrées et arrêtées via les fonctions "interface opérateur", "programmation, essais et surveillance" ou "système d'exploitation" définies dans la CEI 61131-1. Les mécanismes de démarrage et d'arrêt des configurations et des ressources via les services de communication sont définis dans la présente partie de la CEI 61131.

Les programmes, les ressources, les variables globales, les chemins d'accès (et leurs privilèges d'accès correspondants) et les configurations peuvent être chargées ou supprimées par la "fonction de communication" définie dans la CEI 61131-1. Le chargement et la suppression d'une configuration ou d'une ressource doivent être équivalents au chargement et à la suppression de tous les éléments qu'elle contient.

Les chemins d'accès et leurs privilèges d'accès correspondants permettent d'accéder aux variables d'un AP via les services de communication.



Légende

Anglais	Français
Configuration	Configuration
Resource	Ressource
Task	Tâche
Program	Programme
FB Function block	Bloc fonctionnel
Global and directly represented variables	Variables globales et à représentation directe
Access paths	Chemins d'accès
Communication function	Fonction de communication
Execution control path	Chemin de contrôle d'exécution
Variable access paths	Chemins d'accès aux variables
Or Variable	Variable ou

NOTE 1 Cette figure est représentée à titre illustratif uniquement. La représentation graphique n'est pas normative.

NOTE 2 Dans une configuration avec ressource unique, il n'est pas nécessaire que la ressource soit explicitement représentée.

Figure 5 – Modèle logiciel de l'AP

6 Services de communication de l'AP

Le présent article décrit le concept d'informations de statut d'un AP et donne une spécification des services que l'AP fournit au système de commande via le sous-système de communication. (L'article suivant spécifie la manière dont le programme application de l'AP peut utiliser le sous-système de communication pour interagir avec d'autres dispositifs.)

6.1 Sous-systèmes de l'AP et leur statut

Un AP peut produire des statuts qui incluent des informations d'état et des indications de défaut.

Le statut peut être indiqué sur certains des sous-systèmes identifiés dans la figure qui suit. De plus, il existe un résumé du statut qui donne des informations générales sur l'AP.

Tableau 1 – Entités de présentation du statut

N°	Entités de présentation du statut
1	AP (dans son ensemble)
2	Sous-système d'E/S (inclut les modules d'entrée et de sortie et d'autres dispositifs d'E/S intelligents)
3	Unité de traitement
4	Sous-système d'alimentation
5	Sous-système de mémoire
6	Sous-système de communication
7	Sous-systèmes spécifiés par l'intégrateur
NOTE Le statut est destiné à donner des informations sur l'automate et ses sous-systèmes matériels et logiciels, sans tenir compte des informations de configuration. Il n'est pas destiné à donner des informations sur le processus sous contrôle ni sur le programme application de l'AP. Les données de statut contiennent des informations concernant l'état et la santé de l'AP et de ses sous-systèmes.	

La présente partie de la CEI 61131 utilise deux concepts relatifs au statut: santé et état. La "santé" d'un AP ou de ses sous-systèmes est spécifiée par le retour d'une et une seule parmi trois valeurs possibles. La sémantique associée à chaque valeur est précisée ci-dessous. Il s'agit, dans l'ordre de santé décroissante, de:

a) GOOD (bonne) – Si cette valeur est VRAI, l'AP, ou le sous-système spécifié, n'a détecté aucun problème qui l'empêche d'exécuter la fonction prévue;

b) WARNING (avertissement) – Si cette valeur est VRAI, l'AP, ou le sous-système spécifié, n'a détecté aucun problème qui l'empêche d'exécuter la fonction prévue mais a détecté au moins un problème qui pourrait limiter ses capacités. Ces limites peuvent affecter les temps d'exécution ou les performances, etc. (voir les énoncés ci-dessous pour une définition plus précise de ces limites);

c) BAD (mauvaise) – Si cette valeur est VRAI, l'AP, ou le sous-système spécifié, a détecté au moins un problème qui peut l'empêcher d'exécuter la fonction prévue.

L'état système d'un AP est indiqué par une liste d'attributs dont chacun peut être TRUE (vrai) ou FALSE (faux). Zéro, un ou plusieurs de ces attributs peuvent être TRUE en même temps. La sémantique associée à chaque attribut est spécifiée dans la suite du présent article.

Chaque information de statut peut également présenter des attributs spécifiés par l'intégrateur. Parmi ces attributs spécifiés par l'intégrateur, on trouve les suivants:

a) des diagnostics d'erreurs supplémentaires (par exemple, dépassement de cycle d'écriture d'EEPROM);

b) des états opérationnels supplémentaires (par exemple, activation de l'auto-étalonnage);

c) des statuts locaux importants (par exemple, redémarrage automatique requis).

Il n'est pas nécessaire que les mises en œuvre donnent le statut des sous-systèmes. Toutes les instances de types similaires des sous-systèmes présents dans un système sont représentées séparément. Le nom du sous-système peut être prévu afin de pouvoir différencier les sous-systèmes du même type.

6.1.1 Résumé du statut de l'AP

L'AP donne les informations suivantes comme résumé du statut.

Tableau 2 – Résumé de statut de l'AP

N°	Élément	Description	
1	Santé	GOOD	Tous les sous-systèmes de l'AP indiquent une bonne condition de santé
2		WARNING	Au moins un sous-système indique une condition de santé avec un avertissement et aucun sous-système n'indique une mauvaise condition de santé
3		BAD	Au moins un sous-système indique une mauvaise condition de santé
4	En fonctionnement	Si la valeur est VRAI, cet attribut indique si au moins une partie de l'application utilisateur est chargée et sous contrôle de l'AP	
5	Commande locale	Si la valeur est VRAI, cet attribut indique si la commande de neutralisation locale est active. Si elle est active, la capacité à commander un AP et ses sous-systèmes depuis le réseau peut être limitée. Par exemple, elle pourrait être étroitement associée à l'utilisation d'un interrupteur local	
6	Aucune sortie désactivée	Si la valeur est VRAI, cet attribut indique que l'AP peut modifier l'état physique de toutes les sorties à la suite de l'exécution du programme application ou par d'autres moyens. Si elle n'est pas VRAI, l'état physique de certaines des sorties n'est pas affecté (l'état logique peut être affecté). Ceci est généralement utilisé lors des essais et de la modification des programmes application de l'AP	
7	Aucune entrée désactivée	Si la valeur est VRAI, cet attribut indique que l'AP peut accéder à l'état physique de toutes les entrées à la suite de l'exécution d'un programme application ou par d'autres moyens. Si elle n'est pas VRAI, l'état physique de certaines entrées ne peut être accessible. Ceci est généralement utilisé lors des essais et de la modification des programmes application dans lesquels les entrées peuvent être simulées	
8	Forcé	Si la valeur est VRAI, cet attribut indique qu'au moins un point d'E/S associé à l'AP a été forcé. Lorsqu'une entrée est forcée, le programme application reçoit la valeur spécifiée par le PADT à la place de la valeur réelle de la machine ou du processus. Lorsqu'une sortie est forcée, le programme application reçoit la valeur spécifiée par le PADT à la place de la valeur générée par l'exécution du programme application. Lorsqu'une variable est forcée, le programme application utilise la valeur spécifiée par le PADT à la place de la valeur générée par l'exécution du programme normal	
9	Application utilisateur présente	Si la valeur est VRAI, cet attribut indique que l'unité de traitement possède au moins une application utilisateur présente	
10	Sous-système d'E/S	Si la valeur est VRAI, cet attribut indique un "WARNING" ou "BAD" causé par un sous-système d'E/S	
11	Sous-système d'unité de traitement	Si la valeur est VRAI, cet attribut indique un "WARNING" ou "BAD" causé par un sous-système de l'unité de traitement	
12	Sous-système d'alimentation	Si la valeur est VRAI, cet attribut indique un "WARNING" ou "BAD" causé par un sous-système d'alimentation	
13	Sous-système de mémoire	Si la valeur est VRAI, cet attribut indique un "WARNING" ou "BAD" causé par un sous-système mémoire	
14	Sous-système de communication	Si la valeur est VRAI, cet attribut indique un "WARNING" ou "BAD" causé par un sous-système de communication	
15	Sous-système spécifié par l'intégrateur	Si la valeur est VRAI, cet attribut indique un "WARNING" ou "BAD" causé par un sous-système spécifié par l'intégrateur	

6.1.2 Sous-système d'E/S

L'AP donne les informations de statut suivantes sur son sous-système d'E/S.

Tableau 3 – Statut du sous-système d'E/S

N°	Élément	Description	
1	Santé	GOOD	indique qu'aucune erreur n'a été détectée dans ce sous-système d'E/S
2		WARNING	indique qu'un défaut mineur a été détecté dans le sous-système d'E/S. La survenue d'erreurs récupérables dans la communication avec un poste d'E/S à distance est un exemple de défaut mineur
3		BAD	indique qu'un défaut majeur a été détecté dans le sous-système d'E/S. La perte de communication avec un poste d'E/S déporté est un exemple de défaut majeur
4	Aucune sortie désactivée	Si la valeur est VRAI, cet attribut indique que l'AP peut modifier l'état physique de toutes les sorties associées au sous-système d'E/S spécifié à la suite de l'exécution du programme application ou par d'autres moyens. Si elle n'est pas VRAI, l'état physique de certaines des sorties n'est pas affecté (l'état logique peut être affecté). Ceci est généralement utilisé lors des essais et de la modification des programmes application de l'AP	
5	Aucune entrée désactivée	Si la valeur est VRAI, cet attribut indique que l'AP peut accéder à l'état physique de toutes les entrées associées au sous-système d'E/S spécifié à la suite de l'exécution du programme application ou par d'autres moyens. Si elle n'est pas VRAI, l'état physique de certaines entrées ne peut être accessible. Ceci est généralement utilisé lors des essais et de la modification des programmes application dans lesquels les entrées peuvent être simulées	
6	E/S forcée	Si la valeur est VRAI, cet attribut indique qu'au moins un point d'E/S associé à ce sous-système a été forcé. Lorsqu'une entrée est forcée, le programme application reçoit la valeur spécifiée par le PADT à la place de la valeur réelle de la machine ou du processus. Lorsqu'une sortie est forcée, la machine ou le processus reçoit la valeur spécifiée par le PADT à la place de la valeur générée par l'exécution du programme application	
NOTE La définition de "défaut majeur" et "défaut mineur" doit être fournie par l'intégrateur.			

6.1.3 Unité de traitement

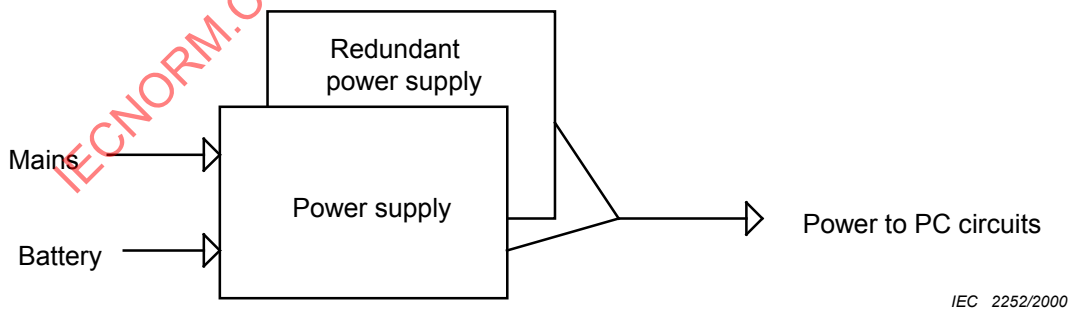
L'AP donne les informations de statut suivantes sur son unité de traitement.

Tableau 4 – Statut de l'unité de traitement

N°	Élément	Description
1 2 3	Santé	Cet attribut identifie la santé de l'unité de traitement. L'intégrateur doit spécifier les conditions dans lesquelles les attributs GOOD, WARNING et BAD sont valides
4	En fonctionne-ment	Si la valeur est VRAI, cet attribut indique si au moins une partie de l'application utilisateur est chargée et sous contrôle de l'unité de traitement
5	Commande locale	Si la valeur est VRAI, cet attribut indique si la commande de neutralisation locale est active. Si elle est active, la capacité à commander l'unité de traitement depuis le réseau peut être limitée. Par exemple, elle pourrait être étroitement associée à l'utilisation d'un interrupteur local
6	Aucune sortie désactivée	Si la valeur est VRAI, cet attribut indique que l'unité de traitement peut modifier l'état physique de toutes les sorties commandées par cette unité de traitement à la suite de l'exécution du programme application ou par d'autres moyens. Si elle n'est pas VRAI, l'état physique de certaines des sorties n'est pas affecté (l'état logique peut être affecté). Ceci est généralement utilisé lors des essais et de la modification des programmes application de l'unité de traitement
7	Aucune entrée désactivée	Si la valeur est VRAI, cet attribut indique que l'unité de traitement peut accéder à l'état physique de toutes les entrées accessibles depuis cette unité de traitement à la suite de l'exécution du programme application ou par d'autres moyens. Si elle n'est pas VRAI, l'état physique de certaines entrées ne peut être accessible. Ceci est généralement utilisé lors des essais et de la modification des programmes application dans lesquels les entrées peuvent être simulées
8	Application utilisateur présente	Si la valeur est VRAI, cet attribut indique que l'unité de traitement possède au moins une application utilisateur présente
9	Forcé	Si la valeur est VRAI, cet attribut indique qu'au moins une variable associée à cette unité de traitement a été forcée. Lorsqu'une variable est forcée, le programme application utilise la valeur spécifiée par le PADT à la place de la valeur générée par l'exécution du programme normal.

6.1.4 Sous-système d'alimentation

L'AP peut donner des informations de statut sur tous les sous-systèmes d'alimentation; voir la Figure 6 pour la configuration présumée de l'alimentation d'un AP. Les exigences applicables aux alimentations des systèmes d'AP et à leur comportement sont décrites dans la CEI 61131-1 et la CEI 61131-2.



Légende

Anglais	Français
(Redundant) power supply	Alimentation (redondante)
Mains	Réseau
Battery	Batterie
Power to PC circuits	Alimentation vers les circuits de l'AP

Figure 6 – Alimentation de l'automate programmable

Tableau 5 – Statut de l'alimentation

N°	Élément	Description	
1	Santé	GOOD	indique qu'aucun problème n'a été détecté dans l'alimentation pouvant l'empêcher de fonctionner pendant un temps indéterminé
2		WARNING	indique qu'un problème a été détecté dans l'alimentation qui peut la rendre inopérante pendant un temps limité
3		BAD	indique que l'alimentation ne fonctionne pas
4	En cours d'utilisation	Si la valeur est VRAI, cet attribut indique que le sous-système d'alimentation est en cours d'utilisation, c'est-à-dire qu'il alimente l'AP	
5	Réseau en fonctionnement	Si la valeur est VRAI, cet attribut indique que le réseau fournit une puissance dans la plage d'alimentation spécifiée	
6	Réseau faible	Si la valeur est VRAI, cet attribut indique que le réseau ne fournit pas d'alimentation dans la plage d'alimentation spécifiée	
7	Batterie en fonctionnement	Si la valeur est VRAI, cet attribut indique que la batterie fournit une puissance dans la plage d'alimentation spécifiée	
8	Batterie faible	Si la valeur est VRAI, cet attribut indique que la batterie n'est pas capable de fournir une alimentation dans la plage d'alimentation spécifiée	
9	Protection déclenchée	Si la valeur est VRAI, cet attribut indique qu'un dispositif de protection de l'alimentation a supprimé une partie de la puissance à destination de l'AP	

6.1.5 Sous-système de mémoire

L'AP donne les informations de statut suivantes sur son sous-système de mémoire.

Tableau 6 – Statut de la mémoire

N°	Élément	Description	
1	Santé	GOOD	Aucune erreur n'a été détectée dans la mémoire associée à ce sous-système
2		WARNING	Au moins une erreur pouvant être corrigée a été détectée et aucune erreur irrémédiable n'a été détectée
3		BAD	Au moins une erreur irrémédiable a été détectée
4	Protégé[1)]	Si la valeur est VRAI, cet attribut indique que la mémoire de ce sous-système de mémoire a été protégée en ce qu'elle ne peut pas être modifiée. Ceci indique généralement que le programme application situé dans ce sous-système de mémoire ne peut pas être modifié. [1)] Cet attribut modélise un état logique et non des caractéristiques physiques du sous-système. Lorsque certaines parties de la mémoire sont protégées et d'autres non, celles-ci doivent être indiquées comme sous-systèmes multiples.	

6.1.6 Sous-système de communication

L'AP donne les informations de statut suivantes sur son sous-système de communication.

Tableau 7 – Statut du sous-système de communication

N°	Élément	Description	
1	Santé	GOOD	indique qu'aucune erreur ne s'est produite ou il y a un nombre acceptable d'erreurs récupérables
2		WARNING	indique que le nombre acceptable d'erreurs récupérables a été dépassé
3		BAD	indique que le sous-système de communication n'est pas capable de communiquer avec tous les dispositifs comme prévu
4	En cours d'utilisation	Si la valeur est VRAI, cet attribut indique que le sous-système de communication est en cours de fonctionnement. Par exemple, dans le cas d'une interface de communication MMS, cela signifie qu'au moins une association avec une application est établie. Sinon l'intégrateur doit définir la sémantique de cet attribut	
5	Erreur locale	Si la valeur est VRAI, cet attribut indique qu'il existe quelques erreurs internes au sous-système de communication qui entravent le fonctionnement	
6	Erreur distante	Si la valeur est VRAI, cet attribut indique qu'il existe quelques erreurs sur des dispositifs avec lesquels la communication est établie qui entravent le fonctionnement	
NOTE 1 Le sous-système de communication qui indique son état peut ne pas être capable d'indiquer son propre état "BAD" de la manière définie dans le présent article. Toutefois, dans un système d'AP, plusieurs sous-systèmes de communication indépendants peuvent fonctionner, et tous peuvent donner des informations de statut. NOTE 2 Il est prévu que les informations spécifiées par l'intégrateur donnent des informations supplémentaires sur chaque interface particulière. Les interfaces du réseau ISO donnent également des informations supplémentaires via des fonctions de gestion du réseau.			

6.1.7 Sous-systèmes spécifiques à l'intégrateur

D'autres sous-systèmes d'un système d'AP doivent être modélisés sous forme de sous-systèmes spécifiques à l'intégrateur. Voici quelques exemples de ces sous-systèmes:

a) automate PID;

b) contrôleur d'exécution;

c) autres processeurs auxiliaires.

Tableau 8 – Statut d'un sous-système spécifique à l'intégrateur

N°	Élément	Description	
1	Santé	GOOD	indique qu'aucune erreur n'a été détectée dans ce sous-système
2		WARNING	indique qu'un défaut mineur a été détecté dans ce sous-système
3		BAD	indique qu'un défaut majeur a été détecté dans ce sous-système
NOTE La définition de "défaut majeur" et "défaut mineur" doit être fournie par l'intégrateur.			

6.1.8 Présentation des informations de statut

Les informations de statut doivent être présentées à l'aide de variables, avec un chemin d'accès prédéfini dans la déclaration de configuration du programme application de l'AP, ou doivent être présentées sous forme de variables avec une représentation directe pour un partenaire de communication distant.

Tableau 9 – Présentation des informations de statut

N°	Présentation des informations de statut
1	Résumé du statut de l'AP sous forme de variable avec chemin d'accès prédéfini P_PCSTATE
2	Résumé du statut de l'AP sous forme de variable avec représentation directe %S
3	Résumé du statut de l'AP et statut de tous les sous-systèmes sous forme de variable avec chemin d'accès prédéfini P_PCSTATUS
4	Informations de statut de chaque sous-système sous forme d'ensemble de variables avec représentation directe %SC<n>
5	Type de chaque sous-système sous forme d'ensemble de variables avec représentation directe %SU<n>
6	Nom de chaque sous-système sous forme d'ensemble de variables avec représentation directe %SN<n>
7	État de chaque sous-système sous forme d'ensemble de variables avec représentation directe %SS<n>
8	Statut spécifique à l'intégrateur de chaque sous-système sous forme d'ensemble de variables avec représentation directe %SI<n>

Si le résumé du statut de l'AP doit être présenté sous forme de variable, son chemin d'accès doit être P_PCSTATE et doit être prédéfini dans la déclaration de la configuration. La variable doit être du type WORD (mot) et doit contenir le résumé du statut de l'AP en commençant par l'élément numéro 1 du bit le moins significatif vers le plus significatif .

Si le résumé du statut de l'AP doit être présenté sous forme de variable avec représentation directe, la représentation directe doit être %S et doit être du type WORD. Elle doit contenir le résumé du statut de l'AP en commençant par l'élément numéro 1 du bit le moins significatif vers le plus significatif.

Si l'information de statut complète doit être présentée sous forme de variable, son chemin d'accès doit être P_PCSTATUS et être prédéfini dans la déclaration de configuration. Cette variable doit être de type structuré comme suit:

```
ARRAY [0..p_NOS] OF
    STRUCT
        SUBSYSTEM  : (SUMMARY, IO, PU, POWER, MEMORY, COMMUNICATION,
                IMPLEMENTER);
        NAME    : STRING[<Max_Name_Len>];
        STATE   : ARRAY[0..15] OF BOOL;
        SPECIFIC    : ARRAY[0..p_BIT] OF BOOL;
    END_STRUCT;
```

IEC 2253/2000

Figure 7 – Description type des informations de statut

L'élément du tableau portant le numéro 0 doit contenir le résumé du statut de l'AP, chaque élément portant un numéro plus élevé doit contenir le statut d'un sous-système. Le sous-élément SUBSYSTEM doit contenir le type de l'AP ou d'un sous-système. Le sous-élément NAME doit contenir le nom de l'AP ou d'un sous-système. L'intégrateur doit spécifier la longueur maximale prise en charge pour les chaînes de noms, c'est-à-dire la valeur de Max_Name_Len. Le sous-élément STATE doit contenir les informations d'état de l'AP ou d'un sous-système sous forme de tableau de BOOL dans le même ordre que celui spécifié aux Tableaux 2 à 8. L'intégrateur doit spécifier le nombre d'éléments du tableau P_PCSTATUS, c'est-à-dire la valeur de p_NOS, les types de sous-systèmes pris en charge, la sémantique des valeurs du sous-élément STATE pour le sous-système spécifique du concepteur, la taille du sous-élément SPECIFIC, c'est-à-dire la valeur de p_BIT et la sémantique du sous-élément SPECIFIC.

Les informations de statut de chaque sous-système peuvent être présentées sous forme d'une variable avec représentation directe %SC<n>, où <n> représente un nombre entre 0 (représentant le résumé de statut de l'AP) et le nombre de sous-systèmes p_NOS. La variable doit présenter la même représentation interne qu'une variable du type de partie de structure décrit sur la figure ci-dessus.

De plus, il peut y avoir un ensemble de variables avec représentation directe %SU<n>, %SN<n>, %SS<n> et %SI<n>. Le <n> représente un nombre entre 0 (représentant le résumé du statut de l'AP) et le nombre de sous-systèmes p_NOS. Les variables doivent présenter la même représentation interne qu'une variable du type de l'un des sous-éléments de structure décrits sur la figure ci-dessus. Dans le détail, %SU<n> doit correspondre au sous-élément SUBSYSTEM, %SN<n> au sous-élément NAME, %SS<n> au sous-élément STATE et %SI<n> au sous-élément SPECIFIC.

6.2 Fonctions spécifiques à l'application

La suite du présent article décrit les fonctions fournies par un AP à un système de commande à l'aide du sous-système de communication tel qu'illustré à la Figure 2.

Fonction de communication de l'AP	AP demandeur	AP répondant	Bloc fonctionnel disponible
Vérification de périphérique	oui	oui	oui
Acquisition de données	oui	oui	oui
Contrôle	oui	oui	oui
Synchronisation entre les applications utilisateur	oui	oui	oui
Rapport d'alarme	oui	non	oui
Exécution du programme et contrôle d'E/S	non	oui	non
Transfert du programme application	non	oui	non
Gestion de la connexion	oui	oui	oui

Chacune de ces fonctions est traitée séparément dans la suite du présent article. Toutes les fonctions ne sont pas disponibles dans tous les AP. Voir l'Article 7 pour la définition du bloc fonctionnel.

Certaines applications combinent les catégories d'application définies ci-dessous, par exemple la supervision et l'acquisition de données.

Les éléments qui suivent, bien que généralement fournis par les AP, sont en dehors du domaine d'application de cette partie de la CEI 61131:

a) interface opérateur;

b) programmation, essais et modification du programme application et vérification du programme.

Les AP ont la capacité d'utiliser des interfaces opérateur. Ces dispositifs sont utilisés par un opérateur pour surveiller ou modifier le processus sous contrôle, ou les deux. Ils peuvent également servir à un système client pour communiquer avec l'opérateur.

L'interface opérateur directe signifie que le client peut communiquer avec l'interface opérateur via le système de communication sans interaction d'un programme application.

La programmation est le processus de création d'un programme application pour l'AP, la base étant instruction par instruction ou bloc fonctionnel par bloc fonctionnel. Les essais et les modifications forment le processus de recherche et de correction des erreurs (" bugs") dans un programme application existant par l'apport de modifications. La vérification du programme consiste en l'essai du programme application d'un AP afin de vérifier qu'il exécute la ou les fonction(s) pour laquelle ou lesquelles il a été conçu dans l'environnement process.

6.2.1 Vérification de périphérique

Cette fonction est conçue pour permettre aux autres périphériques de déterminer si l'AP est apte à exécuter sa fonction prévue dans le système automatisé. Un AP peut produire son propre statut et celui de ses sous-systèmes. Le statut comprend les informations de santé et d'état. Un périphérique peut explicitement demander son statut à l'AP ou bien l'AP peut initier un rapport de statut non sollicité à l'aide des services fournis par l'interface de communication. Voir 6.1 pour la définition des informations de santé et d'état d'un AP et de ses sous-systèmes.

Tableau 10 – Fonctions de vérification de périphérique

N°	Vérification de périphérique
1	Donne des informations de statut
2	Initie des rapports de statut non sollicités

6.2.2 Acquisition de données

Les données contenues dans un AP sont présentées sous forme de variables. Ces données peuvent provenir de plusieurs sources et peuvent avoir une grande variété de significations. Le client peut les obtenir au moyen d'une ou de plusieurs méthodes.

a) Interrogées – Le client lit la valeur d'une ou plusieurs variables à un instant ou lors d'une condition déterminé(e) par le client. L'accès aux variables peut être contrôlé par l'AP. Seules les variables sélectionnées sont accessibles sur le réseau.

b) Programmées – Les données sont fournies par l'AP au client à un instant ou lors d'une condition déterminé(e) par le programme application de l'AP.

c) Configurées – L'interface de communication avec l'AP peut être configurée par un client pour lancer un transfert de données vers le client.

Les types de variables de l'AP que le système de communication peut voir sont les suivants:

a) variables avec représentation directe;

b) autres variables ayant des chemins d'accès (voir la CEI 61131-3 pour la définition des chemins d'accès).

Si les variables avec représentation directe sont accessibles à la communication, ces variables doivent utiliser la représentation directe comme identifiant. Le serveur d'AP (c'est-à-dire l'AP qui est propriétaire des variables) peut interpréter l'identifiant au moyen d'un algorithme défini par l'intégrateur.

NOTE Les variables avec représentation directe peuvent être utilisées comme des variables "normales" pendant la programmation d'un programme d'application. Un nom symbolique supplémentaire peut être assigné à une variable à représentation directe au moyen de la construction AT dans la déclaration de la variable (voir la CEI 61131-3).

Il existe des milliers de ces variables avec représentation directe, même dans un AP de petite taille. Il n'est pas raisonnable de conserver le nom et l'adresse de toutes ces variables dans le dictionnaire d'objets d'un AP.

Le système de l'AP peut restreindre l'accès aux variables avec représentation directe. Les conditions (taille, emplacement, etc.) dans lesquelles chaque type de données pris en charge par l'AP peut être accessible sans interruption doivent être spécifiées par l'intégrateur.

Tableau 11 – Fonctions d'acquisition de données

N°	Acquisition de données
1	Les variables avec représentation directe sont accessibles
2	Chemins d'accès au niveau de la configuration
3	Chemins d'accès au niveau du programme
4	Moyens de restriction de l'accès aux variables avec représentation directe
5	Conditions d'accès aux variables sans interruption

6.2.3 Contrôle

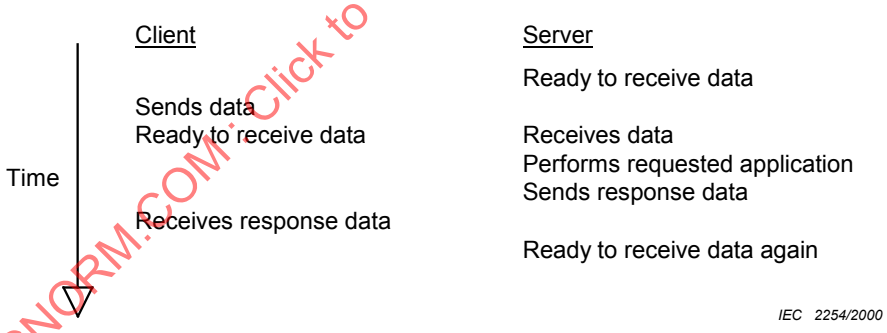
Un AP peut prendre en charge deux méthodes de contrôle: paramétrique et verrouillé.

Le contrôle paramétrique signifie que le fonctionnement de l'AP est commandé par l'écriture de valeurs sur les variables résidant dans l'AP. Cette modification du fonctionnement est déterminée soit par le programme application, soit par d'autres mécanismes locaux.

L'accès aux variables est contrôlé par l'AP qui contient les variables. Seules les variables dont le qualificatif READ_WRITE est sélectionné dans la déclaration du chemin d'accès sont accessibles au contrôle paramétrique sur le réseau.

Le contrôle verrouillé signifie que le client demande au serveur d'exécuter une opération de l'application et d'informer le client du résultat de l'opération. Ce service comporte deux aspects, la synchronisation du client et du serveur et l'échange entre eux de données.

Dans le contrôle verrouillé, cet échange de données a lieu sur des points de synchronisation dans le programme application. Ce service peut être utilisé avec pour effet l'appel à distance de procédure entre un programme application et un autre. La Figure 8 illustre la chronologie de ce service.



Légende

Anglais	Français
Client	Client
Server	Serveur
Time	Temps
Sends data	Envoie les données
Ready to receive data	Prêt à recevoir les données
Receives response data	Reçoit les données de réponse
Receives data	Reçoit les données
Performs requested application	Exécute l'application demandée
Sends response data	Envoie les données de réponse
Ready to receive data again	Prêt à recevoir à nouveau les données

Figure 8 – Chronologie du contrôle verrouillé

L'AP met en œuvre le contrôle verrouillé au moyen des blocs fonctionnels SEND (client) et RCV (serveur). D'autres dispositifs peuvent utiliser d'autres moyens pour émuler le comportement de ces blocs fonctionnels afin d'accéder à cette fonction de communication.

Tableau 12 – Fonctions de contrôle

N°	Contrôle
1	Les variables avec représentation directe sont accessibles
2	Chemins d'accès au niveau de la configuration
3	Chemins d'accès au niveau du programme
4	Moyens de restriction de l'accès aux variables avec représentation directe
5	Conditions d'accès aux variables sans interruption
6	Contrôle verrouillé

6.2.4 Synchronisation entre les applications utilisateur

Les applications utilisateurs peuvent nécessiter un service de synchronisation. Par exemple, une application utilisateur peut démarrer l'exécution d'une autre application après l'exécution d'un algorithme. Le service de synchronisation est fourni par le mécanisme de contrôle verrouillé (voir 6.2.3).

6.2.5 Rapport d'alarme

L'AP peut avoir la capacité de signaler les messages d'alarme à un client lors d'une condition prédéterminée. Le client peut indiquer un acquittement de ces alarmes à l'AP. Ceci est différent de l'acquisition de données normale en ce que l'AP se souvient de l'état d'un point d'alarme jusqu'à son acquittement par le client. L'AP peut générer un résumé des alarmes acquittées à la demande du client.

Tableau 13 – Fonctions de rapport d'alarme

N°	Rapport d'alarme
1	Signalement des messages
2	Réception des acquittements
3	Génération d'un résumé des alarmes acquittées

6.2.6 Exécution du programme application et contrôle des E/S

L'exécution d'un programme application d'AP est gérée par la fonction exécution du programme application (voir 5.2). Les programmes application d'AP peuvent être démarrés et arrêtés. Les programmes application d'AP peuvent être démarrés soit à partir d'un état initial, soit à partir de l'état dans lequel ils se trouvaient au moment de leur arrêt.

Le programme application dans un AP est constitué d'une configuration et zéro, une ou plusieurs ressources (voir la CEI 61131-3). Les configurations et les ressources peuvent être démarrées et arrêtées. Les ressources sont démarrées et arrêtées lorsque la configuration est démarrée et arrêtée et elles peuvent être démarrées et arrêtées indépendamment de la configuration.

La fonction d'interface avec les actionneurs (sorties) associée à un programme application en cours d'exécution peut être contrôlée pour utiliser les valeurs fournies par le programme application ou peut être maintenue dans un état connu. Cet état de l'interface est spécifié au moment de la modification de l'état du programme application. Les sorties peuvent être contrôlées pour être définies selon des états spécifiés par l'intégrateur, conserver les sorties dans l'état courant, positionner toutes les sorties à zéro ou modifier certaines sorties selon des

états spécifiés par l'utilisateur (marche ou arrêt, les sorties non spécifiées conservant l'état précédent) par l'intermédiaire d'un mécanisme spécifié par l'intégrateur (par exemple des tableaux, une procédure, etc.).

La fonction d'interface avec les capteurs (entrées) associée à un programme application en cours d'exécution peut être contrôlée pour fournir les données réelles des capteurs ou continuer à utiliser les valeurs précédentes. L'état de l'entrée est spécifié au moment de la modification de l'état du programme application.

Tableau 14 – Unités pouvant être démarrées et arrêtées

N°	Unité pouvant être démarrée et arrêtée
1	Configuration
2	Ressource

Les E/S (entrées et sorties) associées à une ressource en cours d'exécution doivent soit être contrôlées par le programme, soit être conservées dans un état connu qui est déterminé lorsque la ressource est démarrée. Les ressources doivent pouvoir être démarrées soit à partir d'un état initial (redémarrage à froid à l'aide de START), soit à partir de l'état dans lequel elles se trouvaient au moment de leur arrêt (redémarrage à chaud à l'aide de RESUME). L'état souhaité des sorties doit pouvoir être spécifié dans le cadre du processus d'arrêt.

L'état des E/S peut être défini avec les valeurs suivantes lorsqu'une configuration ou une ressource est démarrée ou arrêtée:

Tableau 15 – Signification de l'état des E/S

Valeur de l'état d'E/S	Signification	Défini par
Contrôlé	Les actionneurs sont contrôlés par le programme application ou la partie du programme application démarrée. Les entrées sont fournies au programme application ou à la partie du programme application démarrée par la fonction d'interface avec les capteurs et les actionneurs.	Démarrage
Conserver les sorties	Les actionneurs ne sont pas contrôlés par le programme application ou la partie du programme application démarrée, ils sont conservés dans l'état courant. Les capteurs sont fournis au programme application ou à la partie du programme application démarrée par la fonction d'interface avec les capteurs et les actionneurs.	Démarrage
Conserver l'état courant	Les actionneurs ne sont pas contrôlés par le programme application ou la partie du programme application démarrée ou arrêtée, ils sont conservés dans l'état courant. Les capteurs ne sont pas fournis au programme application ou à la partie du programme application démarrée, ils sont conservés dans l'état courant.	Démarrage, arrêt
État défini par l'intégrateur	Les actionneurs ne sont pas contrôlés par le programme application ou la partie du programme application arrêtée, ils sont conservés dans un état spécifié par l'intégrateur. Le programme application ne fonctionne pas, par conséquent l'état de l'interface avec les capteurs n'est pas spécifié.	Arrêt
Sorties à zéro	Les actionneurs ne sont pas contrôlés par le programme application ou la partie du programme application arrêtée, ils sont conservés dans l'état zéro. Le programme application ne fonctionne pas, par conséquent l'état de l'interface avec les capteurs n'est pas spécifié.	Arrêt
Spécifié par l'utilisateur	Les actionneurs ne sont pas contrôlés par le programme application ou la partie du programme application arrêtée, ils sont conservés dans un état spécifié par l'utilisateur. Le programme application ne fonctionne pas, par conséquent l'état de l'interface avec les capteurs n'est pas spécifié.	Arrêt

Tableau 16 – État d'E/S

N°	État d'E/S
1	Contrôlé
2	Conserver les sorties
3	Conserver l'état courant
4	Spécifié par l'intégrateur
5	Sorties à zéro
6	Spécifié par l'utilisateur
NOTE Il convient de ne pas être tributaire du sous-système de communication pour le remplacement des interrupteurs d'arrêt d'urgence câblés. Il convient de suivre les pratiques de sécurité normales.	

Tableau 17 – Fonctions d'exécution et de contrôle des E/S

N°	Exécution et contrôle des E/S
1	Réception des demandes de démarrage d'une unité pouvant être démarrée et arrêtée
2	Réception des demandes d'arrêt d'une unité pouvant être démarrée et arrêtée
3	Réception des demandes de reprise d'une unité pouvant être démarrée et arrêtée

6.2.7 Transfert de programme application

Le programme application est transféré à l'aide de la fonction de stockage du programme application et de stockage des données d'un AP (voir 5.2). Le transfert du programme application permet au client de charger le contenu complet de la mémoire programmable ou des portions de celle-ci à des fins d'archivage ou de vérification ou de la télécharger pour restaurer l'AP dans un état connu. Cette fonction donne également la possibilité de placer l'AP dans un état sécurisé avant de modifier le contenu de sa mémoire programmable et de le redémarrer en toute sécurité lorsque le transfert du programme application est terminé.

L'initiation du transfert du programme est généralement réalisée par un dispositif qui n'est pas un AP. Les services comprennent:

a) chargement pour archivage;

b) chargement pour vérification;

c) téléchargement pour restauration dans un état correct précédent connu; et

d) téléchargement sur un système hors ligne.

Les portions de la mémoire programmable qui peuvent être chargées ou téléchargées sont indiquées dans le Tableau 18.

NOTE Une unité de chargement contient la variable P_DDATE, sa valeur est la date de la dernière modification de l'unité de chargement.

Tableau 18 – Unités chargeables

N°	Unités chargeables
1	Configuration
2	Ressource
3	Programmes
4	Variables globales
5	Chemins d'accès au niveau de la configuration
6	Chemins d'accès au niveau du programme
7	Unités de chargement contenant la variable P_DDATE

L'intégrateur doit spécifier si d'autres éléments de langage, par exemple des types de fonction ou des types de bloc fonctionnel, peuvent être chargés, ainsi que les conditions et les restrictions de leur chargement et téléchargement.

La présente partie de la CEI 61131 définit un moyen d'exécuter les chargements et les téléchargements sur l'AP dans son entier, sur un sous-système de l'AP et sur une partie d'un sous-système. L'AP entier ou la partie de l'AP à télécharger doivent être explicitement arrêtés avant le téléchargement. Le système entier ou une partie de celui-ci qui est téléchargé(e) ne doit pas être disponible pour d'autres utilisations jusqu'à la fin du téléchargement. L'intégrateur doit spécifier ce que les autres clients peuvent faire avec un AP lorsqu'un client est en cours de téléchargement.

Tableau 19 – Fonctions de transfert du programme application

N°	Transfert du programme application
1	Réception des requêtes de téléchargement d'une unité chargeable
2	Réception des requêtes de chargement d'une unité chargeable

6.2.8 Gestion de la connexion

La gestion de la connexion fournit les moyens permettant d'installer, d'entretenir et de fermer les connexions de communication avec un partenaire de communication distant. Lorsque plusieurs requêtes sont lancées pour un dispositif, elles peuvent fonctionner toutes sur la même connexion. Tous les sous-systèmes de communication ne nécessitent pas de connexion, par exemple la communication point à point.

Les connexions sont contrôlées explicitement par le programme application au moyen du bloc fonctionnel CONNECT ou sont fournies si nécessaire par le sous-système de communication et au moment requis.

Tableau 20 – Fonctions de gestion de la connexion

N°	Gestion de la connexion
1	Installer les connexions
2	Fermer les connexions
3	Utiliser une connexion pour plusieurs requêtes

7 Blocs fonctionnels de communication de l'AP

7.1 Présentation des blocs fonctionnels de communication

Les fonctions de communication de l'AP et leurs blocs fonctionnels correspondants sont décrits ci-dessous.

Tableau 21 – Présentation des blocs fonctionnels de communication

N°	Paragraphe	Nom du bloc fonctionnel ou de la fonction de communication
1	7.2 Sémantique des paramètres du FB (bloc fonctionnel) de communication (adressage des variables à distance)	REMOTE_VAR
2 3	7.3 Vérification de périphérique	STATUS, USTATUS
4	7.4 Acquisition des données interrogées	READ,
5 6 7 8	7.5 Acquisition des données programmées	USEND, URCV, BSEND, BRCV
9	7.6 Contrôle paramétrique	WRITE,
10 11	7.7 Contrôle verrouillé	SEND, RCV
12 13	7.8 Rapport d'alarme programmé	NOTIFY, ALARM
14	7.9 Gestion de la connexion	CONNECT

Les numéros donnés dans le tableau ci-dessus doivent être utilisés pour déclarer la conformité à ces blocs fonctionnels de communication (CFB).

7.1.1 Vérification de périphérique

Les blocs fonctionnels STATUS et USTATUS sont fournis afin que l'AP puisse collecter le statut des autres périphériques. Ils sont prévus pour que l'AP puisse déterminer si les autres périphériques du système automatisé sont aptes à exécuter leur fonction attendue .

7.1.2 Acquisition de données

Les données contenues dans les autres périphériques peuvent être présentées sous forme de variables. Ces données peuvent provenir de plusieurs sources et peuvent avoir une grande variété de significations. Elles peuvent être obtenues par l'AP au moyen de l'une des deux méthodes en utilisant les blocs fonctionnels de communication.

a) Interrogées – L'AP utilise le bloc fonctionnel READ pour obtenir la valeur d'une ou de plusieurs variables à la fois à un instant ou lors d'une condition déterminé(e) par le programme application de l'AP. L'accès aux variables peut être contrôlé par le périphérique lu.

b) Programmées – Les données sont fournies à l'AP à un instant ou lors d'une condition déterminé(e) par l'autre périphérique. L'AP utilise le bloc fonctionnel URCV pour fournir les données au programme application de l'AP. L'AP utilise le bloc fonctionnel USEND pour fournir des données non sollicitées aux autres périphériques.

Les conditions (taille, emplacement, etc.) dans lesquelles chaque type de données pris en charge par l'autre périphérique peut être accessible sans interruption sont déterminées par l'autre périphérique.

7.1.3 Contrôle

Deux méthodes de contrôle doivent être prises en charge par les AP: paramétrique et verrouillé.

Le contrôle paramétrique signifie que le fonctionnement de l'autre périphérique est commandé par l'écriture de valeurs sur les variables résidant dans ce périphérique. L'accès aux variables peut être contrôlé par l'AP qui contient les variables. L'AP utilise le bloc fonctionnel WRITE pour exécuter cette action depuis le programme application de l'AP.

Le contrôle verrouillé signifie que le client demande au serveur d'exécuter une opération de l'application et d'informer le client du résultat de l'opération. L'AP utilise les blocs fonctionnels SEND et RCV pour mettre en œuvre, respectivement, les rôles de client et de serveur.

7.1.4 Rapport d'alarme

L'AP peut avoir la capacité de signaler les messages d'alarme à un client lors d'une condition prédéterminée. Le client peut indiquer un acquittement de ces alarmes à l'AP. Les blocs fonctionnels ALARM et NOTIFY sont utilisés par le programme application de l'AP pour générer des alarmes, respectivement acquittées et non acquittées.

7.1.5 Gestion de la connexion

Le programme application de l'AP utilise le bloc fonctionnel CONNECT pour gérer les connexions.

7.2 Sémantique des paramètres des FB de communication

Les blocs fonctionnels de communication utilisent une sémantique commune pour leurs entrées et sorties. La signification de ces entrées et sorties est décrite ci-dessous. Certains blocs fonctionnels de communication possèdent des paramètres d'entrée ou de sortie spéciaux, ils sont décrits conjointement à la description des blocs fonctionnels de communication eux-mêmes.

Tableau 22 – Sémantique des paramètres des FB de communication

Nom du paramètre	Type de données du paramètre	Interprétation
EN_R	BOOL	Activé pour recevoir des données
REQ/RESP	BOOL	Exécution de la fonction sur le front montant
ID	COMM_CHANNEL	Identification du canal de communication
R_ID	STRING	Identification du FB distant d'un canal
SD_i	ANY	Données utilisateur à envoyer
VAR_i	STRING ou type de données de la sortie de la fonction REMOTE_VAR	Identification d'une variable du partenaire de communication distant
DONE	BOOL	Fonction demandée exécutée (bonne et valide)
NDR	BOOL	Nouvelles données utilisateur reçues (bonnes et valides)
ERROR	BOOL	Nouveau statut reçu différent de zéro
STATUS	INT	Dernier statut détecté (erreur ou bon)
RD_i	ANY	Dernières données utilisateur reçues

L'entrée ID référence le canal de communication utilisé par l'instance du bloc fonctionnel de communication, c'est-à-dire qu'elle détermine le partenaire de communication distant. L'entrée ID est du type COMM_CHANNEL et doit être définie par l'intégrateur.

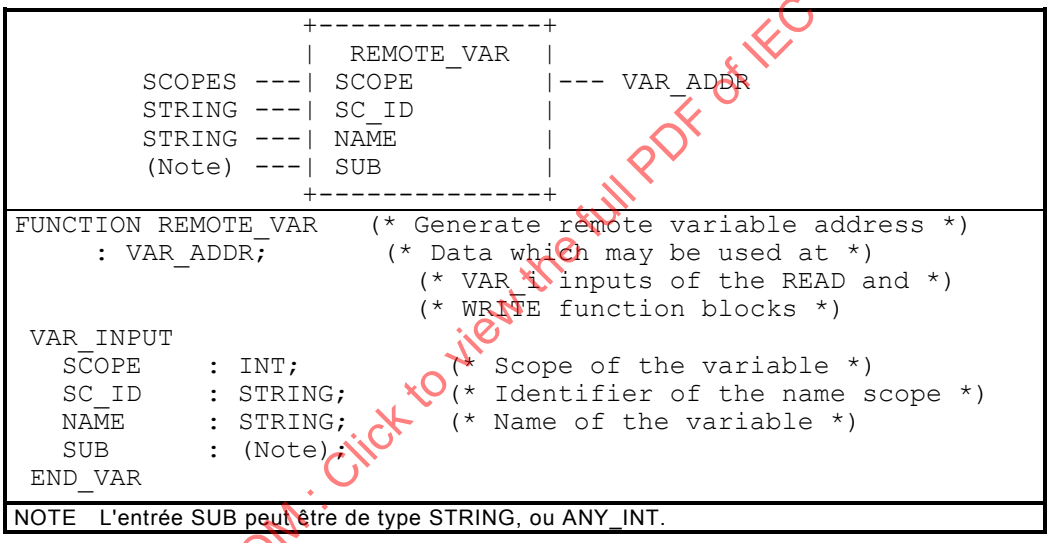
NOTE La valeur donnée à l'entrée ID d'une instance d'un bloc fonctionnel de communication est destinée à contenir ou à référencer les informations nécessaires à la gestion de la communication avec le partenaire de communication distant. Ces informations peuvent dépendre de la mise en œuvre et du sous-système de communication utilisé.

L'entrée R_ID sert à identifier l'instance correspondante du bloc fonctionnel de communication sur le partenaire distant si la fonction de communication de l'AP est fournie par une paire correspondante d'instances d'un bloc fonctionnel.

Une instance d'un bloc fonctionnel de communication doit utiliser le même canal de communication et communiquer avec la même instance de bloc fonctionnel distant tout au long de sa durée de vie.

Les variables à lire ou à écrire sont identifiées au moyen des entrées VAR_i des blocs fonctionnels READ et WRITE. Le paramètre effectif est généralement une chaîne qui contient le nom de la variable distante.

De plus, le paramètre VAR_i peut posséder un type de données défini par l'intégrateur, nommé VAR_ADDR. Une fonction REMOTE_VAR est définie pour générer les informations d'accès aux variables imbriquées.



NOTE L'entrée SUB peut être de type STRING, ou ANY_INT.

IEC 2255/2000

Figure 9 – Fonction REMOTE_VAR

SCOPE est un entier qui identifie les domaines d'application des noms des langages de programmation de la CEI 61131-3, le système de communication ou les prises en charge du concepteur dans le domaine d'application d'une variable distante.

Tableau 23 – Valeurs du paramètre SCOPE

Domaine d'application de nom de la CEI 61131-3	Valeur
Configuration	0
Ressource	1
Programme	2
Instance de bloc fonctionnel	3
Réservé à une normalisation ultérieure	4 .. 9
Réservé aux domaines d'application des noms spécifiques aux sous-systèmes de communication	10 .. 99
Spécifique au concepteur	<0, > 99

Si le SCOPE de la variable nécessite un identifiant, le SC_ID est utilisé pour fournir cet identifiant. NAME contient le nom de la variable distante. Si SUB est du type chaîne, la valeur de la chaîne est interprétée comme le nom d'un sous-élément. Si SUB est un ANY_INT, la valeur est interprétée comme un index. Si SUB est une chaîne vide, la variable entière est adressée.

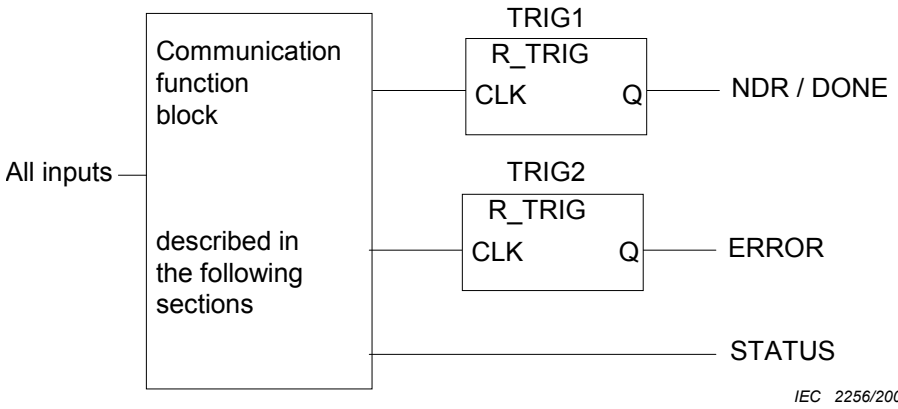
Le type de données de la sortie de la fonction est défini par l'intégrateur. Le résultat de la fonction peut être transféré aux entrées VAR_i des blocs fonctionnels READ et WRITE ou peut être stocké dans une variable du même type de données. L'utilisation de la fonction REMOTE_VAR peut être restreinte de manière à ne produire que des valeurs valides pour les paramètres VAR_i des blocs fonctionnels READ et WRITE.

Il peut exister des fonctions supplémentaires de prise en charge des schémas d'adressage spécifiques au sous-système de communication ou spécifiques au concepteur qui produisent une sortie de type VAR_ADDR. Les fonctions spécifiques au sous-système de communication sont spécifiées dans les annexes relatives au mapping de la présente partie de la CEI 61131.

Tous les paramètres des blocs fonctionnels de communication sont obligatoires, à l'exception des paramètres extensibles SD_i, RD_i et VAR_i, selon le type de bloc fonctionnel. Il n'est pas obligatoire que les paramètres SD_i ou RD_i aient le même type de données lorsque plusieurs paramètres SD_i sont utilisés sur une instance du bloc fonctionnel ou lorsqu'un paramètre SD_i est utilisé avec différentes instances du bloc fonctionnel. L'intégrateur doit spécifier le nombre de paramètres SD_i, RD_i et VAR_i pris en charge avec une invocation d'un bloc fonctionnel de communication. De plus, il doit indiquer s'il existe des restrictions dans l'utilisation de ces paramètres, par exemple le type de données ou la taille des paramètres effectifs.

Lorsqu'un bloc fonctionnel de communication exige que les types de données soient compatibles, le contrôle de la compatibilité doit toujours être vrai si les mêmes types de données CEI 61131-3 sont utilisés sur un AP client et un AP serveur. Des règles de compatibilité supplémentaires spécifiques aux sous-systèmes de communication peuvent être définies.

Les sorties sont initialisées avec le zéro système. L'impulsion de NDR, DONE et ERROR est vraie jusqu'à l'invocation suivante de cette instance. C'est-à-dire que chacun des blocs fonctionnels de communication implique la structure qui suit, qui n'est pas représentée sur les schémas d'état de ces blocs fonctionnels.



Légende

Anglais	Français
All inputs	Toutes les entrées
Communication function block	Bloc fonctionnel de communication
Described in the following sections	Description dans les sections suivantes

Figure 10 – Principe de la signalisation de statut

Lorsqu'une erreur de communication d'une instance d'un bloc fonctionnel de communication est détectée par le système de l'AP ou par l'algorithme du bloc fonctionnel de communication, les sorties ERROR et STATUS de l'instance du bloc fonctionnel sont définies. La sortie ERROR reste vraie pendant le temps écoulé entre deux invocations de cette instance (voir Figure 11).

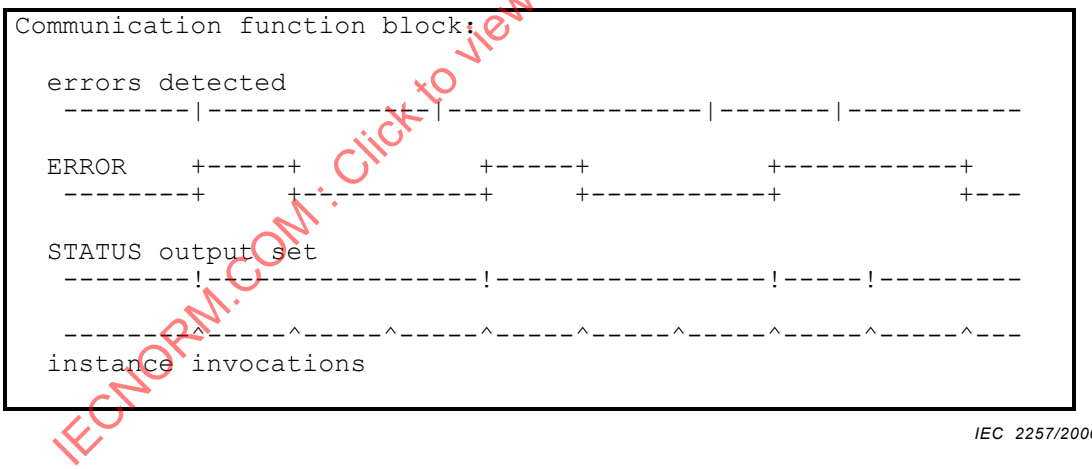


Figure 11 – Chronogramme des temps des sorties ERROR et STATUS

Les NDR, DONE, ERROR et la sortie STATUS doivent prendre de nouvelles valeurs que de manière synchronisée avec les invocations d'instance des blocs fonctionnels de communication, même si une erreur est détectée entre-temps.

Les valeurs suivantes ont été définies pour la sortie STATUS des divers blocs fonctionnels. Tous les blocs fonctionnels n'utilisent pas toutes les valeurs.

Tableau 24 – Valeur et interprétation de la sortie STATUS

Valeur de STATUS	Interprétation
0	Pas d'erreur
1	Erreur des couches inférieures, pas de communication possible
2	Autre réponse négative de la part du partenaire de communication distant
3	R_ID n'existe pas dans le canal de communication
4	Défaut de correspondance du type de données
5	Réinitialisation reçue
6	Récepteur non activé
7	Partenaire de communication distant dans un état défaillant
8	Accès refusé à l'objet distant
9	Récepteur dépassé (les données utilisateur sont nouvelles)
10	Accès à l'objet local rejeté
11	Le service demandé dépasse les ressources locales
12 .. 20	Réservé à une normalisation ultérieure
–1	L'instance de cette fonction est occupée et ne peut pas fournir de services supplémentaires à ce moment
< –1 ou > 20	Les codes inférieurs à –1 ou supérieurs à 20 doivent être spécifiés par l'intégrateur

Lorsque des erreurs sont reçues depuis un ou plusieurs partenaires de communication en cas d'une communication d'une source vers un ou plusieurs destinataires ou d'une source vers tous les destinataires, le paramètre STATUS est défini en fonction de la première erreur reçue.

Les paramètres RD_i doivent contenir les données reçues. Ces paramètres doivent être des paramètres d'entrée/sortie.

Les paragraphes qui suivent contiennent une description des blocs fonctionnels de communication. La représentation des blocs fonctionnels est donnée de manière graphique et textuelle et les types et significations des entrées et des sorties associées sont représentés. La sortie STATUS doit prendre la valeur appropriée définie.

Le fonctionnement normal est illustré par un diagramme d'état.

Le fonctionnement des blocs fonctionnels est décrit sur la base des diagrammes d'état. Les transitions qui dépendent du programme application sont mappées pour les conditions des entrées des blocs fonctionnels. Les transitions qui dépendent du système de communication sont décrites indépendamment du système de communication utilisé. Les mappings explicites pour certains systèmes de communication sont décrits dans les annexes. La valeur des sorties du bloc fonctionnel est donnée pour chaque état du diagramme d'état.

Les erreurs causées par le système de communication ou par des problèmes locaux peuvent se produire de manière asynchrone dans tous les états d'un bloc fonctionnel de communication. Seules les erreurs qui entraînent des transitions d'état ou nécessitent des actions sont explicitement décrites dans les diagrammes d'état. Sinon, les erreurs doivent seulement être signalées au programme application au moyen des sorties ERROR et STATUS telles que représentées sur les Figures 10 et 11.

Il est nécessaire d'initialiser les blocs fonctionnels de communication. L'état d'initialisation INIT contenu dans tous les diagrammes d'état doit être conservé au moins jusqu'au retour de la

première invocation de l'instance du bloc fonctionnel. Dans cet état, toutes les actions doivent être entreprises pour activer la communication. Le canal de communication avec le partenaire de communication distant doit être établi, si la gestion de la connexion du canal n'est pas explicitement programmée.

7.3 Vérification de périphérique

La fonction de communication de vérification d'un périphérique de l'AP utilise les blocs fonctionnels STATUS et USTATUS.

Une instance d'un bloc fonctionnel STATUS ou USTATUS produit une instance d'une fonction de vérification du périphérique de l'AP.

Un AP peut demander à un partenaire de communication distant de lui renvoyer ses informations de statut au moyen du bloc fonctionnel STATUS.

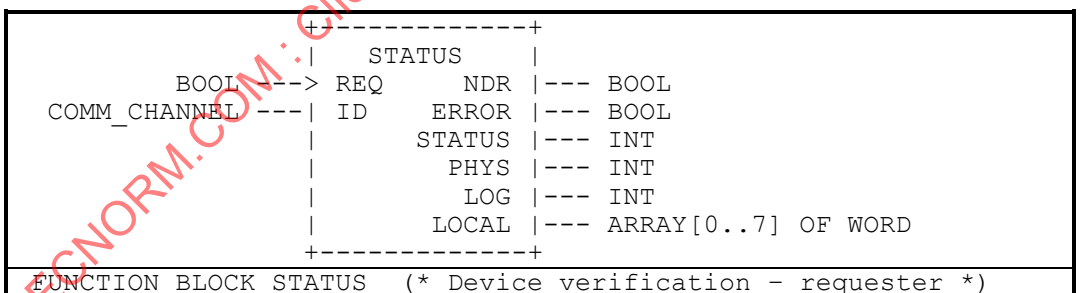
Un AP peut s'activer lui-même pour recevoir des informations de statut d'un partenaire de communication distant au moyen du bloc fonctionnel USTATUS. Le partenaire de communication distant doit au moins informer l'instance USTATUS dès lors que ses informations de statut, présentées dans les sorties PHYS et LOG, changent.

La sortie PHYS du FB contient le statut physique du périphérique distant, la sortie LOG du FB contient son statut de communication logique. La sortie LOCAL du FB peut contenir des informations supplémentaires de statut jusqu'à 128 bits. L'intégrateur doit spécifier la longueur utilisée des informations supplémentaires de statut et doit définir leur sémantique.

NOTE Le bloc fonctionnel READ peut être utilisé pour obtenir des informations supplémentaires de statut.

Le paramètre ID identifie le canal de communication vis-à-vis du partenaire de communication distant.

Lorsqu'une erreur s'est produite, la sortie ERROR démarre un cycle pour indiquer une erreur et la sortie STATUS contient le code d'erreur.

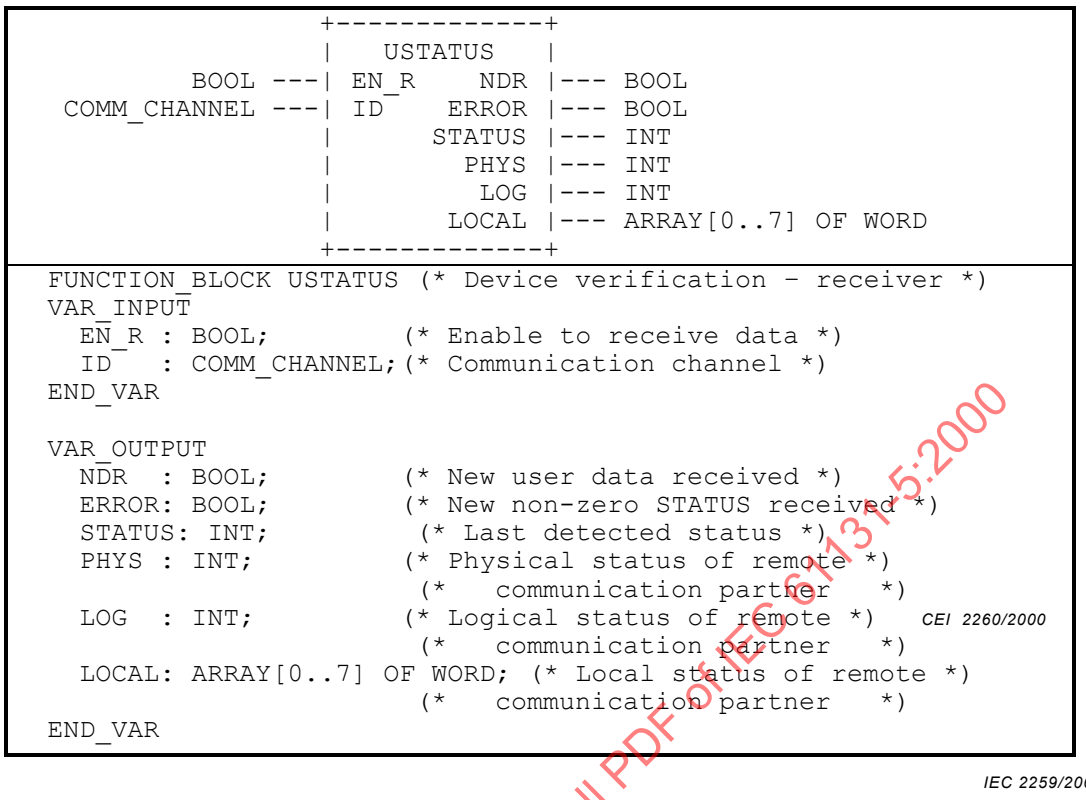



```
FUNCTION_BLOCK STATUS  (* Device verification - requester *)
VAR_INPUT
  REQ   : BOOL R_EDGE; (* Request *)
  ID    : COMM_CHANNEL;(* Communication channel *)
END_VAR

VAR_OUTPUT
  NDR   : BOOL;         (* New user data received *)
  ERROR: BOOL;          (* New non-zero STATUS received *)
  STATUS: INT;           (* Last detected status *)
  PHYS  : INT;          (* Physical status of remote *)
                         (*   communication partner   *)
  LOG   : INT;          (* Logical status of remote *)
                         (*   communication partner   *)
  LOCAL: ARRAY[0..7] OF WORD;  (* Local status of remote *)
END_VAR                  (*   communication partner   *)
```

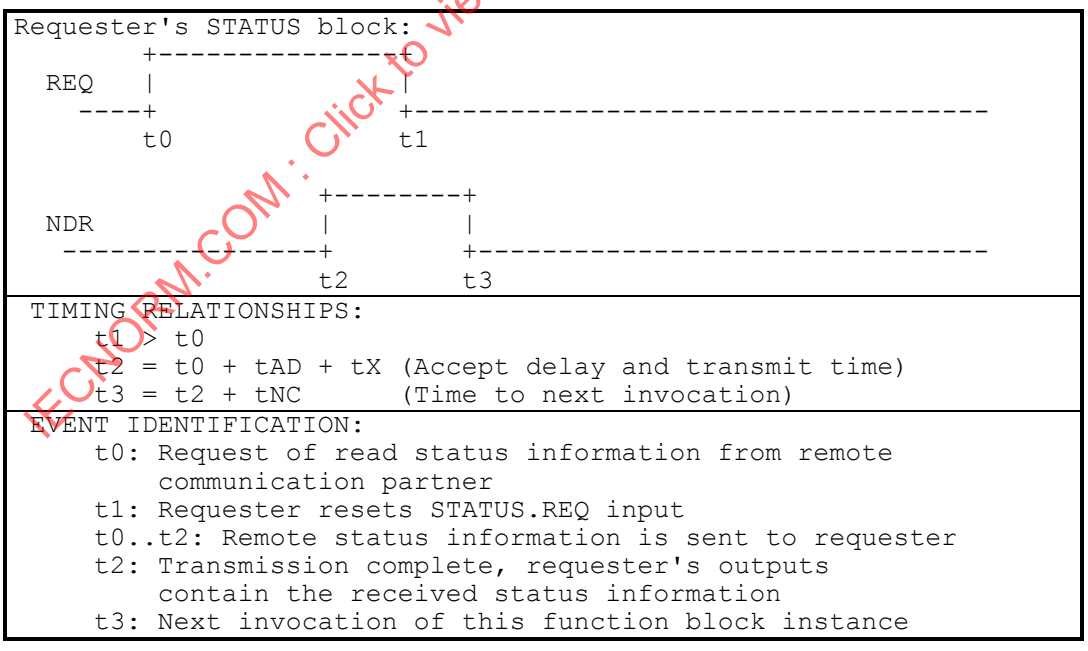
IEC 2258/2000

Figure 12 – Bloc fonctionnel STATUS



IEC 2259/2000

Figure 13 – Bloc fonctionnel USTATUS



IEC 2260/2000

Figure 14 – Chronogramme du bloc fonctionnel STATUS

Le diagramme d'état représenté à la Figure 15 décrit l'algorithme du bloc fonctionnel STATUS. Les Tableaux 25 et 26 décrivent les transitions de ce diagramme d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel STATUS.

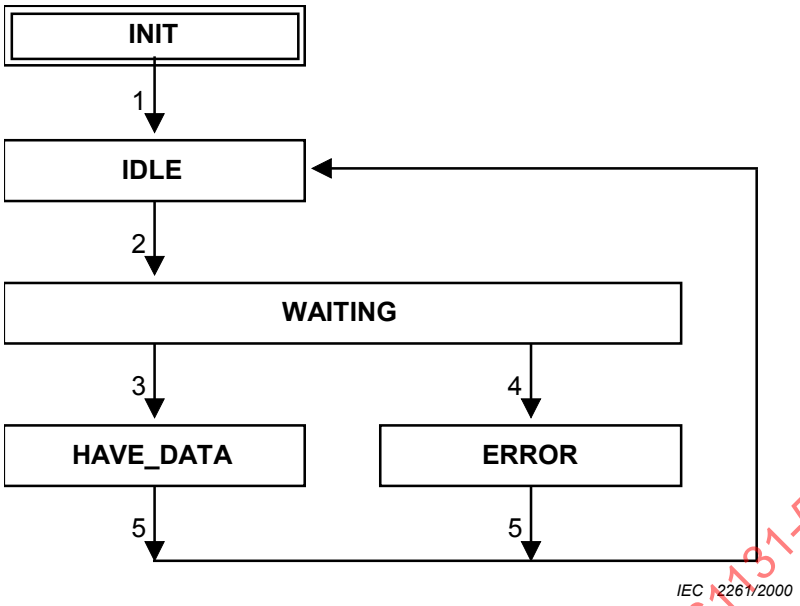


Figure 15 – Diagramme d'état du bloc fonctionnel STATUS

Tableau 25 – Transitions du diagramme d'état STATUS

Transition	Condition
1	Initialisation effectuée
2	Au front montant sur l'entrée REQ
3	Réponse positive du partenaire de communication distant
4	Réponse négative du partenaire de communication distant ou problèmes de communication détectés
5	Après la prochaine invocation de cette instance

Tableau 26 – Table d'actions pour le diagramme d'état STATUS

		Sorties du FB			
Etat	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**PHYS, LOG, LOCAL**
INIT [a]	Initialiser les sorties	0	0	0	0
IDLE	Aucune action	0	0	---	---
WAITING	Demander les informations de statut au partenaire de communication distant	---	---	–1	---
HAVE_DATA	Déposer les informations de statut dans l'instance	1	0	0	Nouvelles informations de statut
ERROR	Indiquer une erreur	0	1	[b]	---

--- indique les sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

Le diagramme d'état de la Figure 16 décrit l'algorithme du bloc fonctionnel USTATUS. Les Tableaux 27 et 28 décrivent les transitions de ce diagramme d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel USTATUS.

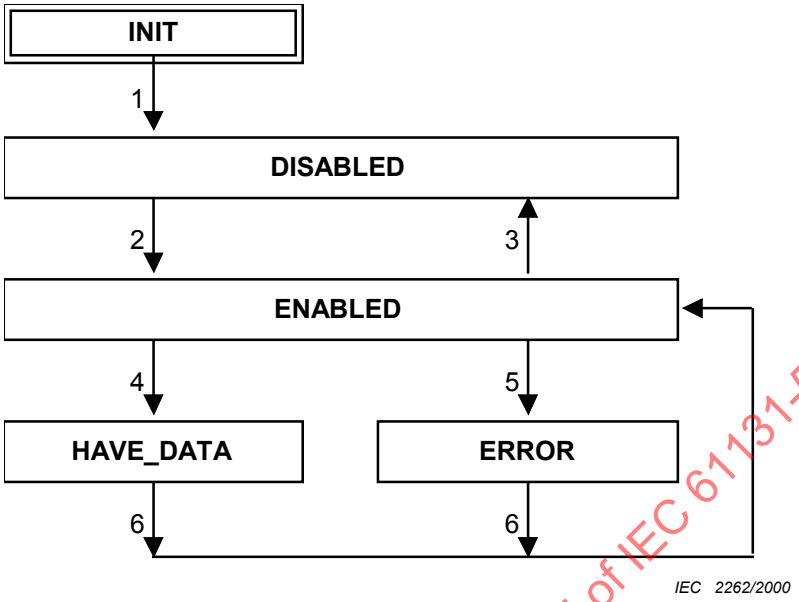


Figure 16 – Diagramme d'état du bloc fonctionnel USTATUS

Tableau 27 – Transitions du diagramme d'état USTATUS

Transition	Condition
1	Initialisation effectuée
2	EN_R = 1
3	EN_R = 0
4	Informations de statut reçues du partenaire de communication distant
5	Problèmes de communication détectés
6	Après la prochaine invocation de cette instance

Tableau 28 – Table d'actions pour le diagramme d'état USTATUS

		Sorties du FB			
Etat	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**PHYS, LOG, LOCAL**
INIT [a]	Initialiser les sorties	0	0	0	0
DISABLED	Aucune action	0	0	---	---
ENABLED	Aucune action	0	0	---	---
HAVE_DATA	Déposer les informations de statut	1	0	0	Nouvelles informations de statut
ERROR	Indiquer une erreur	0	1	[b]	---

--- indique les sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

7.4 Acquisition de données interrogées

La fonction de communication d'acquisition des données interrogées de l'AP utilise le bloc fonctionnel READ.

Une instance d'un bloc fonctionnel READ produit une instance de la fonction d'acquisition des données interrogées de l'AP.

Le paramètre ID identifie le canal de communication vis-à-vis du partenaire de communication distant.

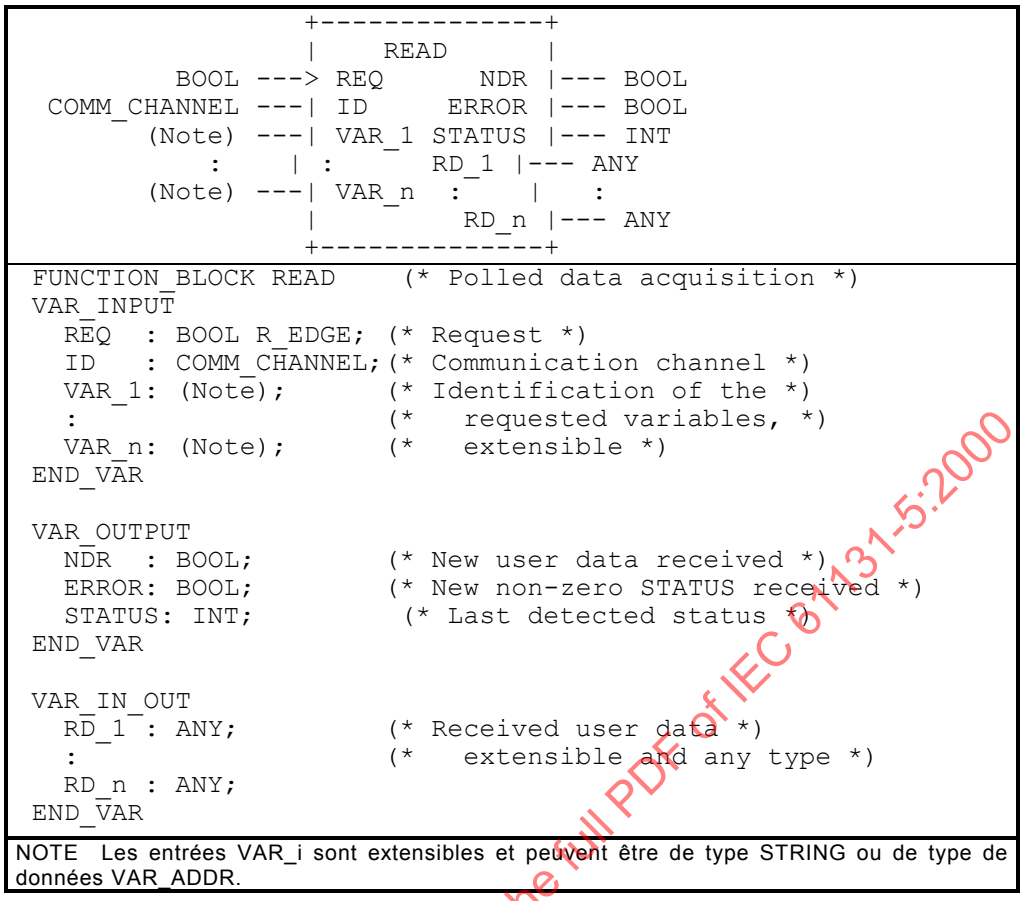
Les entrées VAR_i du bloc fonctionnel READ contiennent une chaîne qui peut être interprétée par le partenaire de communication à distance comme un identifiant de variable. Le partenaire de communication distant envoie les valeurs de ces variables en retour à l'instance READ qui a fait la demande. La fonction READ transfère les valeurs des variables reçues à son programme application via ses sorties RD_i. Chaque variable demandée au partenaire de communication distant doit être d'un type de données compatible avec la variable programmée sur les sorties RD_i de l'instance READ. Les paramètres VAR_i et RD_i sont extensibles. Au moins VAR_1 et RD_1 doivent être présents.

Si le partenaire de communication distant est un AP, les variables avec chemin d'accès et les variables à représentation directe peuvent être accédées à l'aide d'un bloc fonctionnel READ. Les variables avec chemin d'accès sont référencées dans la construction de VAR_ACCESS des langages de programmation de l'AP (voir 2.7.1 de la CEI 61131-3). Le nom d'accès spécifié dans cette construction doit être utilisé comme identifiant de la variable dans l'entrée VAR_i. Lorsqu'une variable à représentation directe doit être accédée à l'aide d'un bloc fonctionnel READ, l'entrée VAR_i doit contenir la représentation directe, par exemple %IW17, sous forme de chaîne. Il doit être possible de mélanger l'accès aux variables avec chemin d'accès et à représentation directe dans une seule invocation d'une instance d'un bloc fonctionnel READ.

Lorsqu'une variable doit être lue via un chemin d'accès, qui est déclaré dans un programme (voir 2.5.3 de la CEI 61131-3), la fonction REMOTE_VAR doit être utilisée. Le nom de l'instance du programme doit être utilisé sur l'entrée SC_ID, le nom de la variable sur l'entrée NAME; par exemple pour lire la variable AB12 du programme DO7, la fonction REMOTE_VAR doit être invoquée à l'aide de REMOTE_VAR (2, "DO7", "AB12", "").

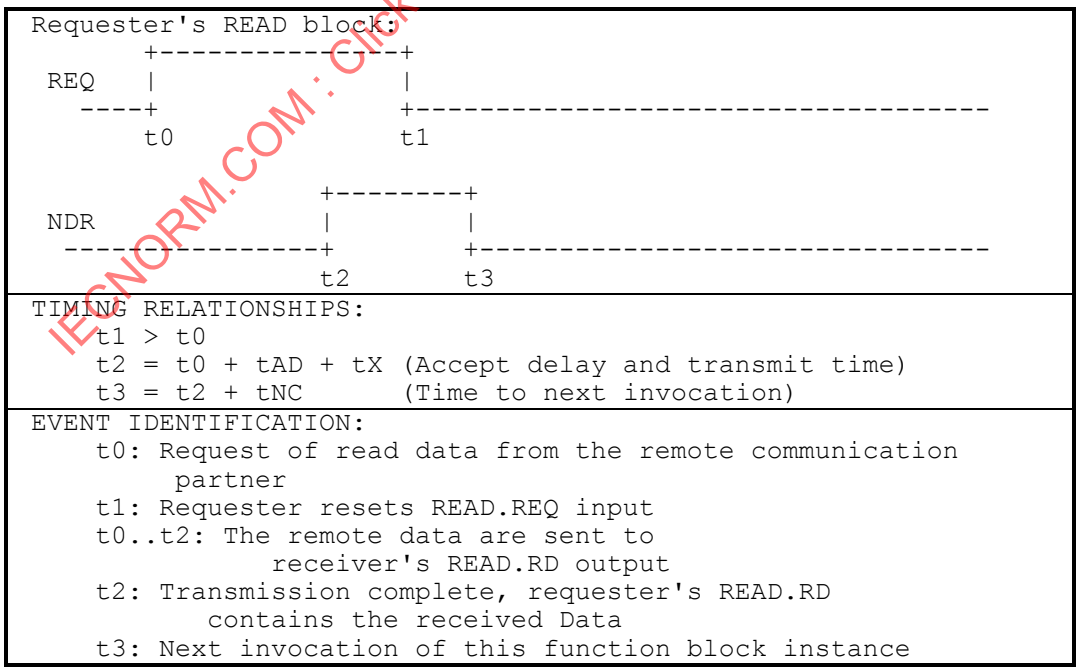
Lorsqu'un sous-élément d'une variable structurée ou d'un élément d'un tableau doit être lu, l'entrée SUB de la fonction REMOTE_VAR doit être utilisée pour identifier ce sous-élément ou cet élément dans les entrées VAR_i.

Lorsqu'une erreur s'est produite, la sortie ERROR démarre un cycle pour indiquer une erreur et la sortie STATUS contient le code d'erreur.



IEC 2263/2000

Figure 17 – Bloc fonctionnel READ



IEC 2264/2000

Figure 18 – Chronogramme du bloc fonctionnel READ

Le diagramme d'état de la Figure 19 décrit l'algorithme du bloc fonctionnel READ. Les Tableaux 29 et 30 décrivent les transitions de ce diagramme d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel READ.

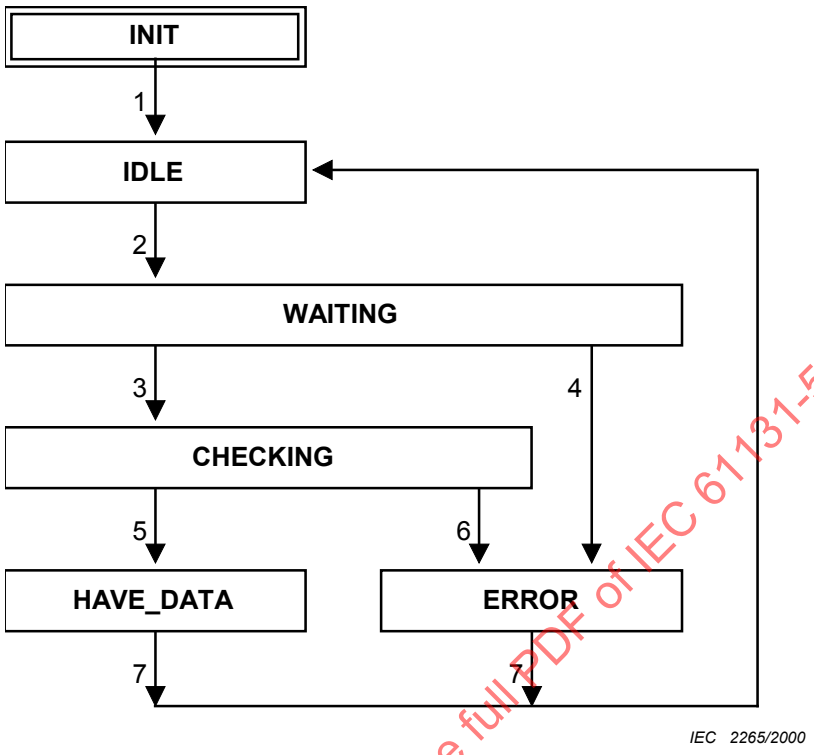


Figure 19 – Diagramme d'état du bloc fonctionnel READ

Tableau 29 – Transitions du diagramme d'état READ

Transition	Condition
1	Initialisation effectuée
2	Au front montant sur l'entrée REQ
3	Réponse positive du partenaire de communication distant
4	Réponse négative du partenaire de communication distant ou autres problèmes de communication détectés
5	Les types de données de RD_i et des données reçues correspondent
6	Les types de données de RD_i et des données reçues ne correspondent pas
7	Après la prochaine invocation de cette instance

Tableau 30 – Table d'actions pour le diagramme d'état READ

		Sorties du FB			
Etat	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1..RD..n**
INIT [a]	Initialiser les sorties	0	0	0	Nul système
IDLE	Aucune action	0	0	---	---
WAITING	Demander les variables au partenaire de communication distant	0	0	–1	---
CHECKING	Vérifier la correspondance des types de données	0	0	---	---
HAVE_DATA	Déposer les données	1	0	0	Nouvelles données
ERROR	Indiquer une erreur	0	1	[b]	---

--- indique les sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

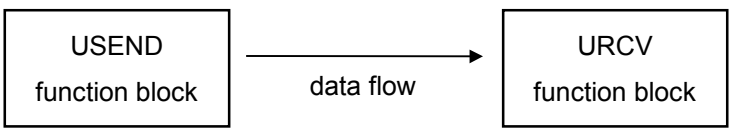
7.5 Acquisition de données programmées

7.5.1 Blocs fonctionnels USEND/URCV

La fonction de communication d'acquisition de données programmées de l'AP utilise les blocs fonctionnels USEND et URCV.

Des instances correspondantes d'un bloc fonctionnel USEND et un bloc fonctionnel URCV produisent une instance de la fonction d'acquisition des données programmées de l'AP.

L'instance USEND envoie les données à l'instance URCV qui peut traiter ces données à l'aide de son programme application. Suite à une demande, l'instance USEND prend les données de ses entrées SD_i et les transmet à l'instance URCV correspondante. Les données reçues auparavant sont écrasées. L'instance URCV transfère les données reçues au programme application via ses sorties RD_i. Ceci se produit dès lors que le programme application demande à son instance USEND d'envoyer des données. L'instance URCV transfère les nouvelles données reçues dès qu'elle les obtient. Elle informe le programme application de l'arrivée de nouvelles données. Le schéma de flux de données qui suit illustre ce processus.



IEC 2266/2000

Légende

Anglais	Français
USEND function block	Bloc fonctionnel USEND
Data flow	Flux de données
URCV function block	Bloc fonctionnel URCV

Figure 20 – Flux de données d'acquisition de données programmées

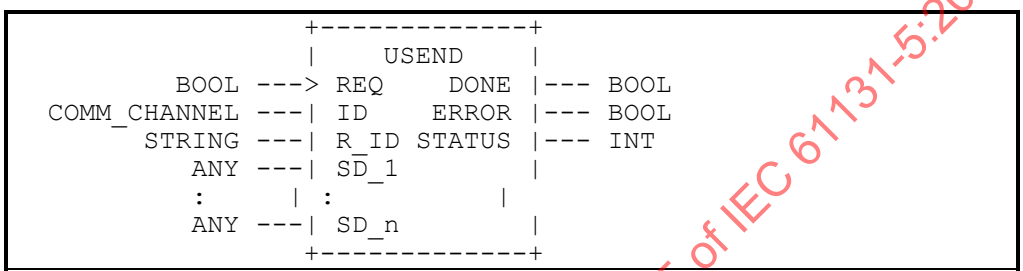
Les paramètres SD_i et RD_i sont extensibles. Au moins l'entrée SD_1 sur le FB USEND et la sortie RD_1 sur le FB URCV doivent être présentes. Le nombre et le type de données des

entrées SD_i de l'instance USEND et des sorties RD_i de l'instance URCV correspondante doivent être compatibles.

Une instance USEND envoie des données à une instance URCV; cela signifie que ce sont des instances correspondantes si la valeur du paramètre ID référence le même canal de communication et si les valeurs des paramètres R_ID sont égales dans la portée du canal de communication.

NOTE Si le système de communication offre des canaux de communication qui prennent en charge les connexions à origine unique et destinataires multiples ou tous les destinataires, ces blocs fonctionnels peuvent être utilisés pour programmer une fonction d'acquisition de données entre un partenaire de communication et plusieurs autres.

Lorsqu'une erreur s'est produite, la sortie ERROR démarre un cycle pour indiquer une erreur et la sortie STATUS contient le code d'erreur.

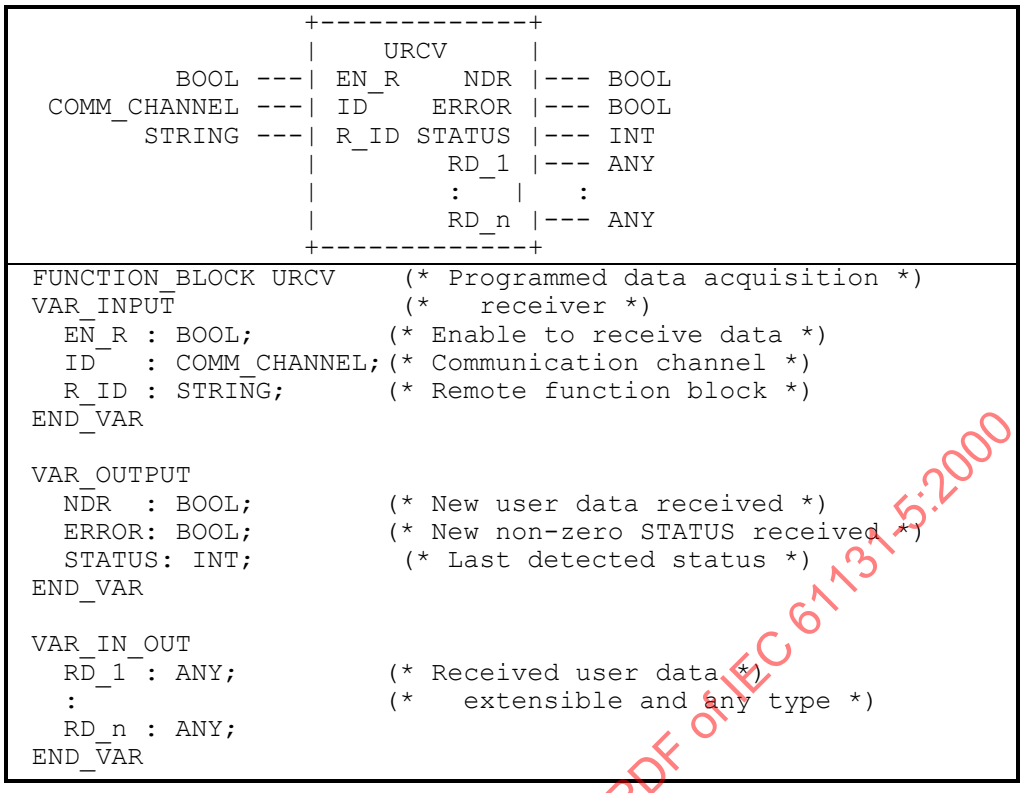



```
FUNCTION_BLOCK USEND    (* Programmed data acquisition *)
VAR_INPUT               (*    requester *)
  REQ  : BOOL R_EDGE; (* Request to send *)
  ID   : COMM_CHANNEL;(* Communication channel *)
  R_ID : STRING;      (* Remote function block *)
  SD_1 : ANY;         (* User data to send *)
  :                   (*   extensible and any type *)
  SD_n : ANY;
END_VAR

VAR_OUTPUT
  DONE : BOOL;        (* Function performed *)
  ERROR: BOOL;        (* New non-zero STATUS received *)
  STATUS: INT;         (* Last detected status *)
END_VAR
```

IEC 2267/2000

Figure 21 – Bloc fonctionnel USEND



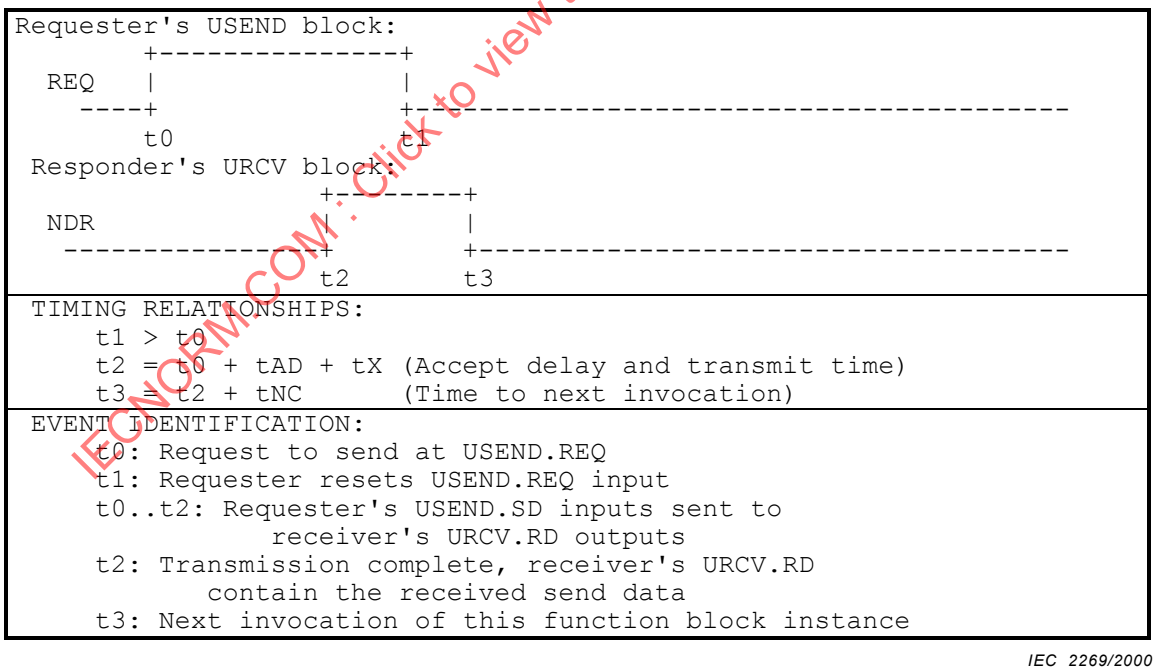

```
FUNCTION_BLOCK URCV     (* Programmed data acquisition *)
VAR_INPUT                (*   receiver *)
  EN_R : BOOL;          (* Enable to receive data *)
  ID   : COMM_CHANNEL;(* Communication channel *)
  R_ID : STRING;        (* Remote function block *)
END_VAR

VAR_OUTPUT
  NDR  : BOOL;          (* New user data received *)
  ERROR: BOOL;          (* New non-zero STATUS received *)
  STATUS: INT;           (* Last detected status *)
END_VAR

VAR_IN_OUT
  RD_1 : ANY;           (* Received user data *)
  :                     (*   extensible and any type *)
  RD_n : ANY;
END_VAR
```

IEC 2268/2000

Figure 22 – Bloc fonctionnel URCV




```
TIMING RELATIONSHIPS:
    t1 > t0
    t2 = t0 + tAD + tX (Accept delay and transmit time)
    t3 = t2 + tNC      (Time to next invocation)

EVENT IDENTIFICATION:
    t0: Request to send at USEND.REQ
    t1: Requester resets USEND.REQ input
    t0..t2: Requester's USEND.SD inputs sent to
                receiver's URCV.RD outputs
    t2: Transmission complete, receiver's URCV.RD
           contain the received send data
    t3: Next invocation of this function block instance
```

IEC 2269/2000

Figure 23 – Chronogramme des blocs fonctionnels USEND et URCV

Le diagramme d'état représenté sur la Figure 24 décrit l'algorithme du bloc fonctionnel USEND. Les Tableaux 31 et 32 décrivent les transitions de ce schéma d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel USEND.

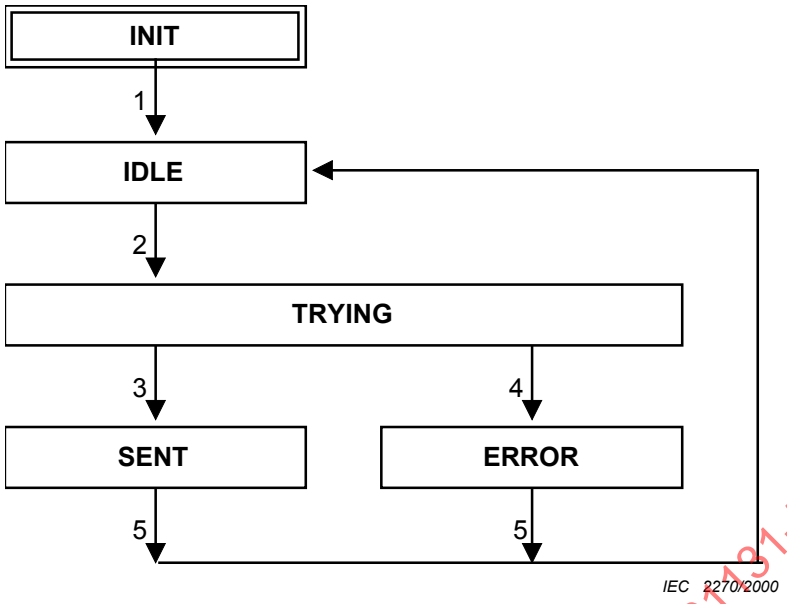


Figure 24 – Diagramme d'état du bloc fonctionnel USEND

Tableau 31 – Transitions du diagramme d'état USEND

Transition	Condition
1	Initialisation effectuée
2	Au montant sur l'entrée REQ
3	Le système de communication indique "Envoyé au partenaire de communication distant"
4	Le système de communication indique "Ne peut envoyer au partenaire de communication distant", ou autres problèmes de communication détectés
5	Après la prochaine invocation de cette instance

Tableau 32 – Table d'actions pour le diagramme d'état USEND

		Sorties du FB		
Etat	**Actions**	**DONE** [c]	**ERROR** [c]	**STATUS**
INIT [a]	Initialiser les sorties	0	0	0
IDLE	Aucune action	0	0	---
TRYING	Envoyer les données aux entrées SD_i au partenaire de communication distant	---	---	---
SENT	Effacer l'indication d'erreur	1	0	0
ERROR	Indiquer une erreur	0	1	[b]

--- indique les sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

Le diagramme d'état de la Figure 25 décrit l'algorithme du bloc fonctionnel URCV. Les Tableaux 33 et 34 décrivent les transitions de ce diagramme d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel URCV.

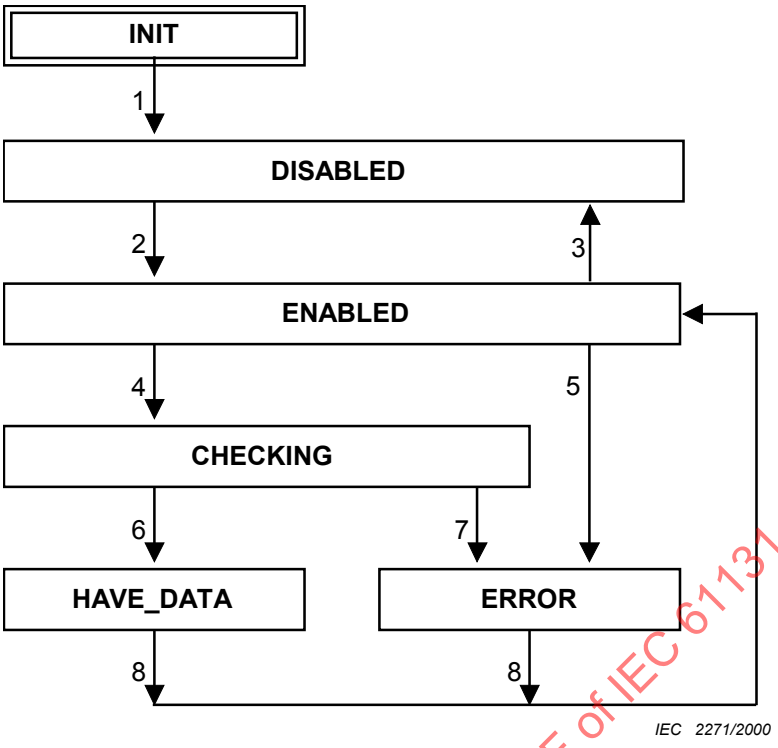


Figure 25 – Diagramme d'état du bloc fonctionnel URCV

Tableau 33 – Transitions du diagramme d'état URCV

Transition	Condition
1	Initialisation effectuée
2	EN_R = 1
3	EN_R = 0
4	Données reçues du partenaire de communication distant
5	Problèmes de communication détectés
6	Les types de données SD_i de USEND et RD_i de URCV correspondent
7	Les types de données SD_i de USEND et RD_i de URCV ne correspondent pas
8	Après la prochaine invocation de cette instance

Tableau 34 – Table d'actions pour le diagramme d'état URCV

		Sorties du FB			
Etat	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1..RD..n**
INIT [a]	Initialiser les sorties	0	0	0	Nul système
DISABLED	Aucune action	0	0	---	---
ENABLED	Aucune action	---	---	---	---
CHECKING	Vérifier la correspondance des types de données	---	---	---	---
HAVE_DATA	Déposer les données	1	0	0, 9	Nouvelles données
ERROR	Indiquer une erreur	0	1	[b]	---

--- indique les sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

7.5.2 Blocs fonctionnel BSEND/BRCV

Les instances correspondantes d'un bloc fonctionnel BSEND et d'un bloc fonctionnel BRCV produisent une instance de la fonction d'acquisition des données programmées de l'AP.

L'instance BSEND envoie les données à l'instance BRCV qui peut traiter ces données à l'aide de son programme application. Suite à une demande, l'instance BSEND prend les données de ses entrées SD_1 et les transmet à l'instance BRCV correspondante. Les données précédemment reçues sont écrasées. L'instance BRCV transfère les données reçues au programme application via ses sorties RD_1. Ceci se produit lorsque le programme application demande à son instance BSEND d'envoyer les données. L'instance BRCV transfère les données nouvellement reçues lorsqu'elle a terminé sa précédente requête et qu'elle obtient de nouvelles données. Elle informe le programme application de l'arrivée de nouvelles données.

L'entrée SD_1 de l'instance BSEND et la sortie RD_1 de l'instance BRCV doivent être toutes deux du type de données ANY et sont interprétées comme une séquence d'octets.

NOTE La représentation des données dans l'automate dépend généralement de la mise en œuvre. Les partenaires de communication doivent s'accorder sur l'interprétation des données lorsque des données d'un type autre qu'un tableau d'octets sont transférées. Ceci restreint l'interopérabilité des programmes qui utilisent ces blocs fonctionnels de communication.

L'instance BSEND doit envoyer autant d'octets parmi les données de l'entrée SD_1 qu'il y en a sur l'entrée LEN de l'instance BSEND. Si l'entrée LEN a la valeur 0, la variable entière de l'entrée SD_1 doit être transférée. La sortie RD_1 doit être capable de stocker les données envoyées. La sortie LEN de l'instance BRCV doit contenir le nombre d'octets reçus sur la sortie RD_1. Une erreur doit être signalée, si

– la valeur de l'entrée LEN de l'instance BSEND est supérieure à la longueur totale en octets de la variable connectée à l'entrée SD_1,

– ou la longueur totale en octets des données reçues est supérieure à la longueur totale en octets de la variable connectée à la sortie RD_1 de l'instance BRCV.

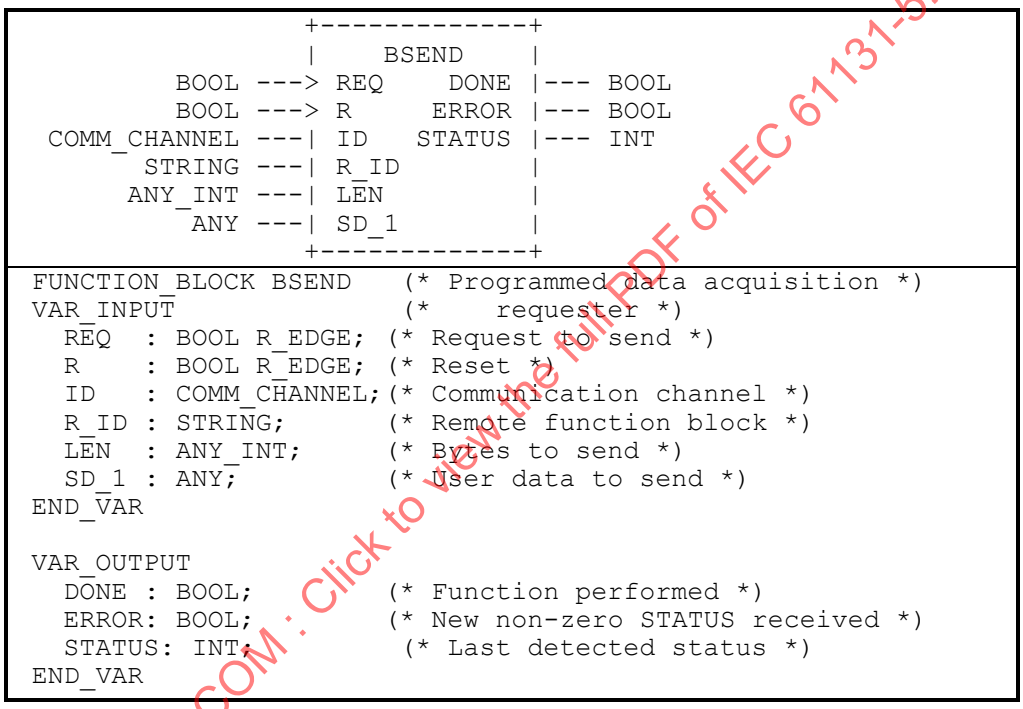
La phase de transfert des données de l'instance BSEND commence au front montant sur l'entrée REQ. Elle se termine lorsque la sortie DONE ou ERROR est égale à 1. La sortie DONE doit être mise à 1 pour un cycle lorsque le bloc entier des données à envoyer est complètement transmis. Pendant la phase de transfert des données, l'intégrateur peut

restreindre l'accès à la variable contenant les données à envoyer. La sortie RD_1 de l'instance BRCV est valide lorsque la sortie NDR a la valeur 1.

Une instance BSEND envoie des données à une instance BRCV; cela signifie que ce sont des instances correspondantes si la valeur du paramètre ID référence le même canal de communication et si les valeurs des paramètres R_ID sont égales dans la portée de ce canal de communication.

Le transfert de données doit être annulé lorsqu'un front montant est détecté sur l'entrée R de l'instance BSEND. Si le transfert de données est annulé, les sorties ERROR et STATUS de l'instance BRCV sont définies. Dans ce cas, les valeurs des sorties RD_1 et LEN ne sont pas définies.

Lorsqu'une erreur s'est produite, la sortie ERROR démarre un cycle pour indiquer une erreur et la sortie STATUS contient le code d'erreur.

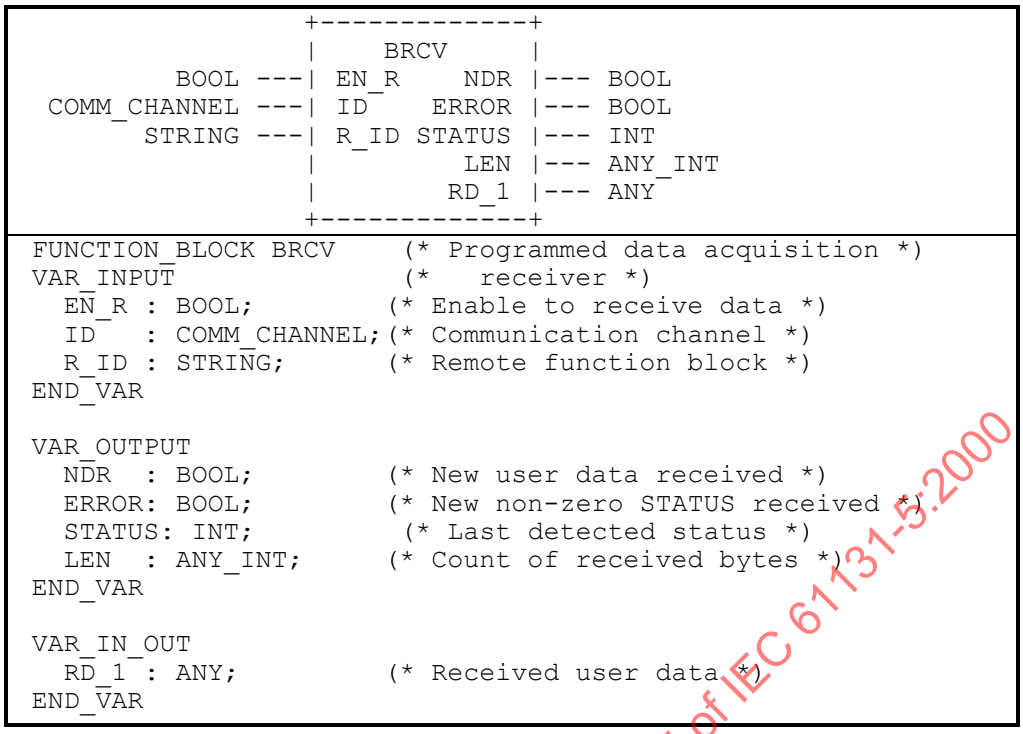



```
FUNCTION_BLOCK BSEND    (* Programmed data acquisition *)
VAR_INPUT               (*    requester *)
  REQ  : BOOL R_EDGE; (* Request to send *)
  R    : BOOL R_EDGE; (* Reset *)
  ID   : COMM_CHANNEL;(* Communication channel *)
  R_ID : STRING;      (* Remote function block *)
  LEN  : ANY_INT;     (* Bytes to send *)
  SD_1 : ANY;         (* User data to send *)
END_VAR

VAR_OUTPUT
  DONE : BOOL;        (* Function performed *)
  ERROR: BOOL;        (* New non-zero STATUS received *)
  STATUS: INT;         (* Last detected status *)
END_VAR
```

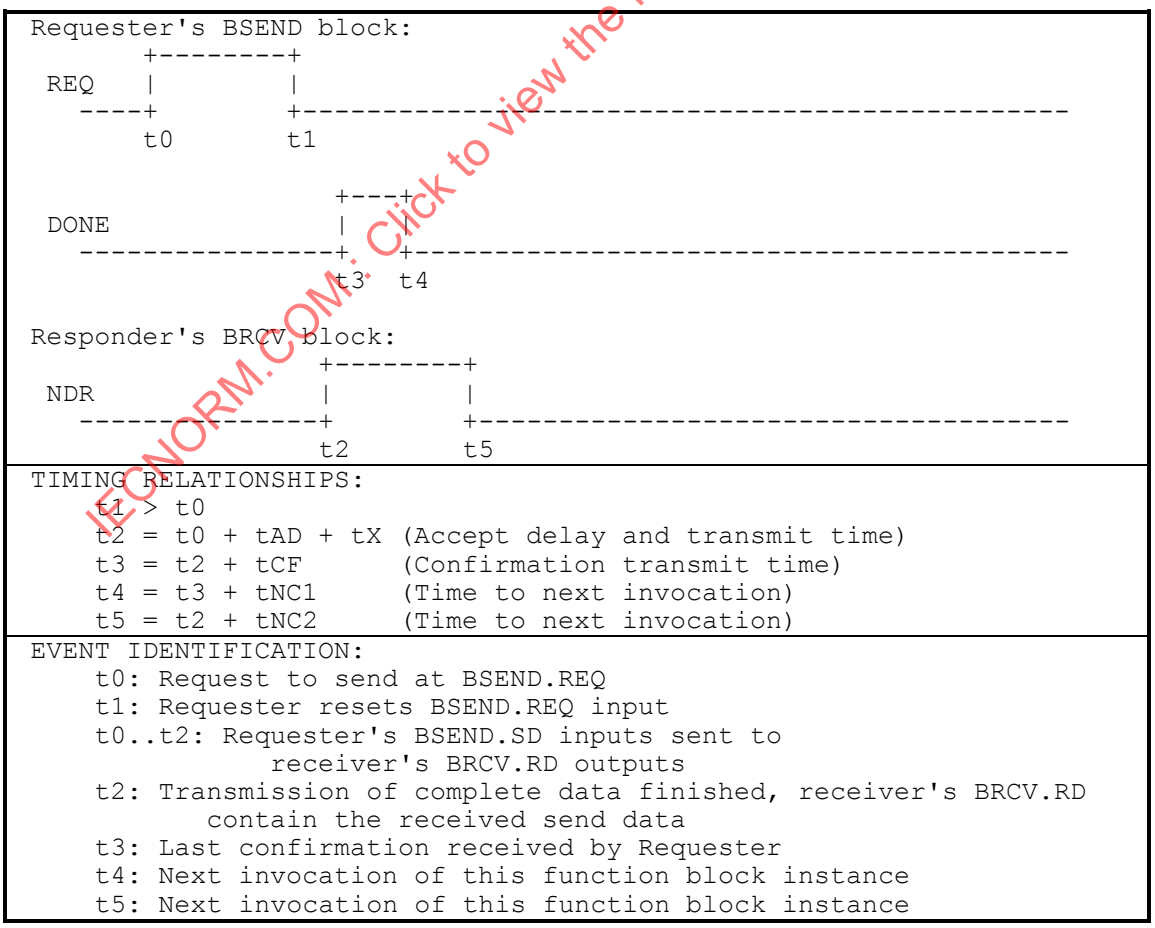
IEC 2272/2000

Figure 26 – Bloc fonctionnel BSEND



IEC 2273/2000

Figure 27 – Bloc fonctionnel BRCV



IEC 2274/2000

Figure 28 – Chronogramme des blocs fonctionnels BSEND et BRCV

Le diagramme d'état représenté sur la Figure 29 décrit l'algorithme du bloc fonctionnel BSEND. Les Tableaux 35 et 36 décrivent les transitions de ce diagramme d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel BSEND.

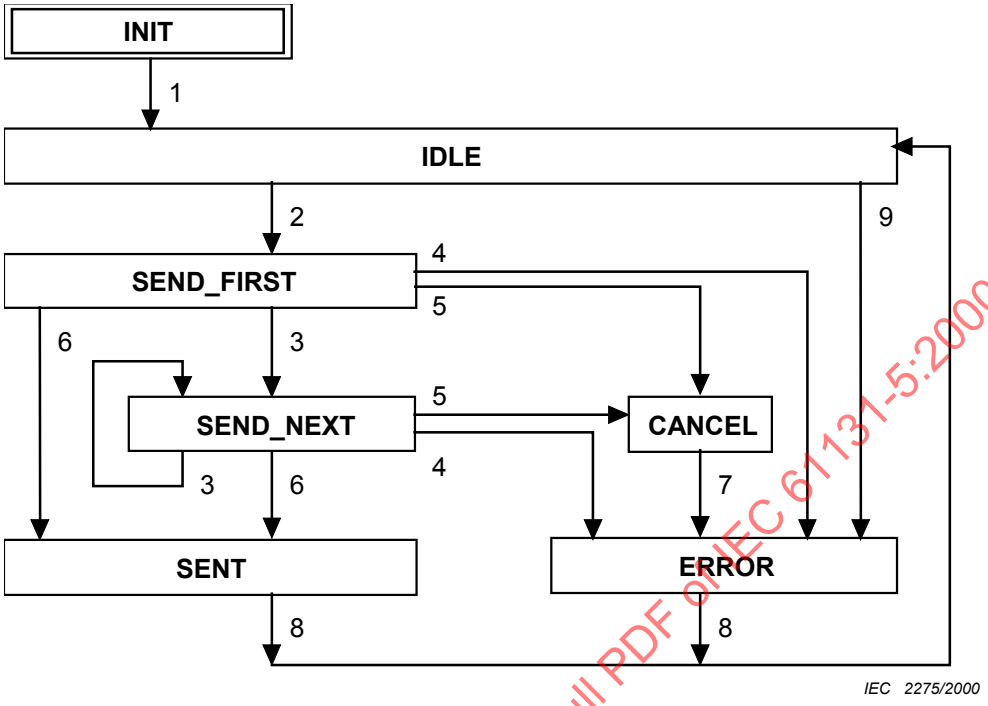


Figure 29 – Diagramme d'état du bloc fonctionnel BSEND

Tableau 35 – Transitions du diagramme d'état BSEND

Transition	Condition
1	Initialisation effectuée
2	Au front montant sur l'entrée REQ
3	Confirmation positive reçue et d'autres données à envoyer
4	Confirmation négative reçue ou problèmes de communication détectés
5	Au front montant sur l'entrée R
6	Confirmation positive reçue et plus aucune donnée à envoyer
7	Immédiat
8	Après la prochaine invocation de cette instance
9	Problèmes de communication détectés

Tableau 36 – Table d'actions pour le diagramme d'état BSEND

		Sorties du FB		
Etat	**Actions**	**DONE** [c]	**ERROR** [c]	**STATUS**
INIT [a]	Initialiser les sorties	0	0	0
IDLE	Aucune action	0	0	---
SEND_FIRST	Envoyer le premier bloc de données de l'entrée SD_1 au partenaire de communication distant, envoyer au total un maximum de LEN octets	---	–1	---
SEND_NEXT	Envoyer le bloc de données suivant de l'entrée SD_1 au partenaire de communication distant, envoyer au total un maximum de LEN octets	---	---	---
SENT	Effacer l'indication d'erreur	1	0	0
CANCEL	Cesser le transfert de données	---	---	---
ERROR	Indiquer une erreur	0	1	[b]

--- indique les sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

Le diagramme d'état représenté sur la Figure 30 décrit l'algorithme du bloc fonctionnel BRCV. Les Tableaux 37 et 38 décrivent les transitions de ce schéma d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel BRCV.

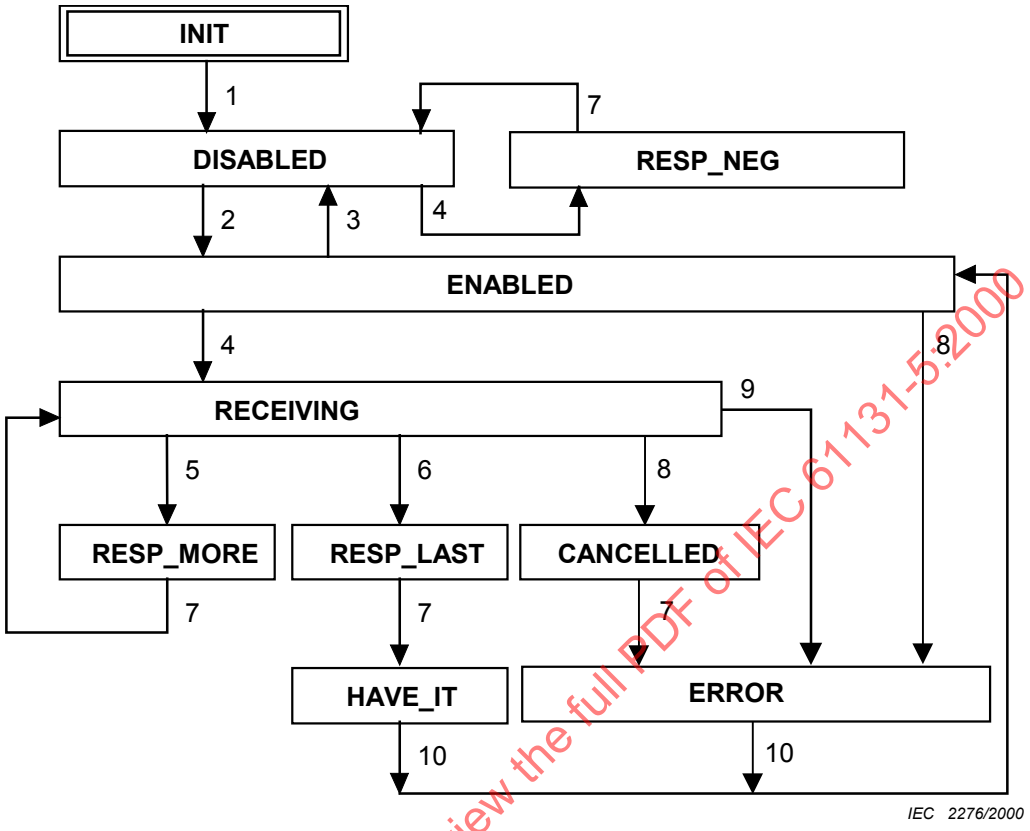


Figure 30 – Diagramme d'état du bloc fonctionnel BRCV

Tableau 37 – Transitions du diagramme d'état BRCV

Transition	Condition
1	Initialisation effectuée
2	EN_R = 1
3	EN_R = 0
4	Données reçues du partenaire de communication distant
5	D'autres données à suivre est vrai
6	D'autres données à suivre est faux
7	Immédiat
8	Problèmes de communication détectés
9	Indication reçue d'annuler le transfert de données
10	Après la prochaine invocation de cette instance

Tableau 38 – Table d'actions pour le diagramme d'état BRCV

		Sorties du FB			
Etat	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1, LEN**
INIT [a]	Initialiser les sorties	0	0	0	Nul système
DISABLED	Aucune action	0	0	---	---
RESP_NEG	Envoyer une réponse négative	---	---	---	---
ENABLED	Aucune action	---	---	---	---
RECEIVING	Vérifier que les données peuvent être stockées à l'index donné	---	---	[d]	Nouvelles données [d]
RESP_MORE	Envoyer une réponse positive	---	---	---	---
RESP_LAST	Envoyer une réponse positive	---	---	---	---
CANCELLED	Envoyer une réponse positive	---	---	5	---
HAVE_IT	Déposer les données	1	0	0	Nouvelles données
ERROR	Indiquer une erreur	0	1	[b]	---

--- indique les sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

[d] Les nouvelles données peuvent être placées dans la sortie RD_1, au cas où la sortie STATUS doit être définie à –1.

7.6 Contrôle paramétrique

La fonction de communication de contrôle paramétrique de l'AP utilise le bloc fonctionnel WRITE.

Une instance d'un bloc fonctionnel WRITE produit une instance de la fonction de contrôle paramétrique de l'AP.

Le paramètre ID identifie le canal de communication vis-à-vis du partenaire de communication distant.

Les entrées VAR_i du bloc fonctionnel WRITE contiennent une chaîne qui peut être interprétée par le partenaire de communication distant comme son identifiant de variable (chemin d'accès). Les entrées SD_i référencent les valeurs à écrire pour les variables identifiées par les entrées VAR_i. Le partenaire de communication distant écrit les valeurs pour ces variables. Les paramètres VAR_i et SD_i sont extensibles. Au moins VAR_1 et SD_1 doivent être présents.

Chaque variable du partenaire de communication distant doit être du même type de données que le type programmé aux entrées SD_i de l'instance WRITE.

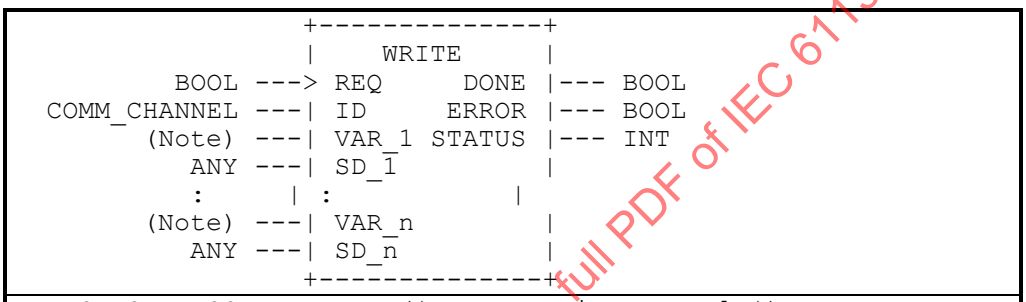
Si le partenaire de communication distant est un AP, les variables avec chemin d'accès et les variables à représentation directe peuvent être accédées à l'aide d'un bloc fonctionnel WRITE. Les variables avec chemin d'accès sont référencées dans la construction de VAR_ACCESS des langages de programmation de l'AP (voir 2.7.1 de la CEI 61131-3). Le nom d'accès spécifié dans cette construction doit être utilisé comme identifiant de la variable dans l'entrée

VAR_i. Lorsqu'une variable à représentation directe doit être accédée à l'aide d'un bloc fonctionnel WRITE, l'entrée VAR_i doit contenir la représentation directe, par exemple %IW17, sous forme de chaîne. Il est possible de mélanger l'accès aux variables avec chemin d'accès et à représentation directe dans une seule invocation d'instance d'un bloc fonctionnel WRITE.

Lorsqu'une variable doit être écrite via un chemin d'accès, qui est déclaré dans un programme (voir 2.5.3 de la CEI 61131-3), la fonction REMOTE_VAR doit être utilisée. Le nom de l'instance du programme doit être utilisé sur l'entrée SC_ID, le nom de la variable sur l'entrée NAME; par exemple pour lire la variable AB12 du programme DO7, la fonction REMOTE_VAR doit être invoquée à l'aide de REMOTE_VAR (2, "DO7", "AB12", "").

Lorsqu'un sous-élément d'une variable structurée ou d'un élément d'un tableau doit être écrit, l'entrée SUB de la fonction REMOTE_VAR doit être utilisée pour identifier ce sous-élément ou cet élément dans les entrées VAR_i.

Lorsqu'une erreur s'est produite, la sortie ERROR démarre un cycle pour indiquer une erreur et la sortie STATUS contient le code d'erreur.




```
FUNCTION_BLOCK WRITE   (* Parametric control *)
VAR_INPUT
  REQ  : BOOL R_EDGE; (* Request *)
  ID   : COMM_CHANNEL;(* Communication channel *)
  VAR_1: (Note);      (* Identifier and *)
  SD_1 : ANY;         (*   value of variable to be set *)
  :                   (*   extensible and any type *)
  VAR_n: (Note);
  SD_n : ANY;
END_VAR

VAR_OUTPUT
  DONE : BOOL;        (* Function performed *)
  ERROR: BOOL;        (* New non-zero STATUS received *)
  STATUS: INT;         (* Last detected status *)
END_VAR
```

NOTE Les entrées VAR_i sont extensibles et peuvent être du type STRING ou du type de données VAR_ADDR.

IEC 2277/2000

Figure 31 – Bloc fonctionnel WRITE

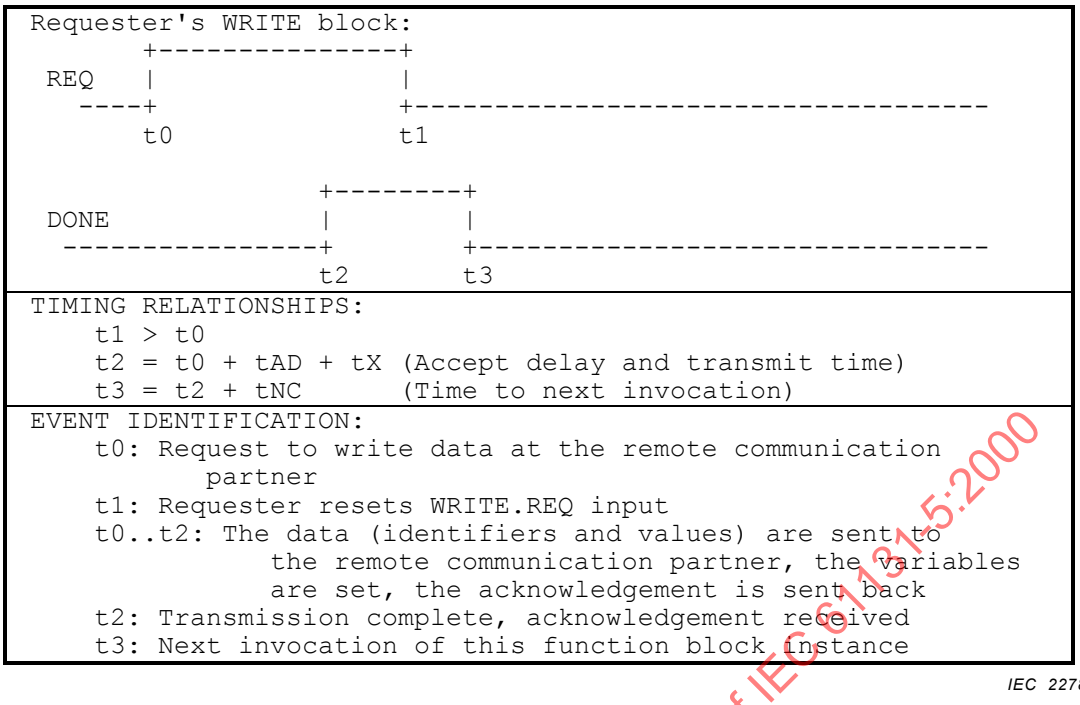


Figure 32 – Chronogramme du bloc fonctionnel WRITE

Le diagramme d'état représenté sur la Figure 33 décrit l'algorithme du bloc fonctionnel WRITE. Les Tableaux 39 et 40 décrivent les transitions de ce diagramme d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel WRITE.

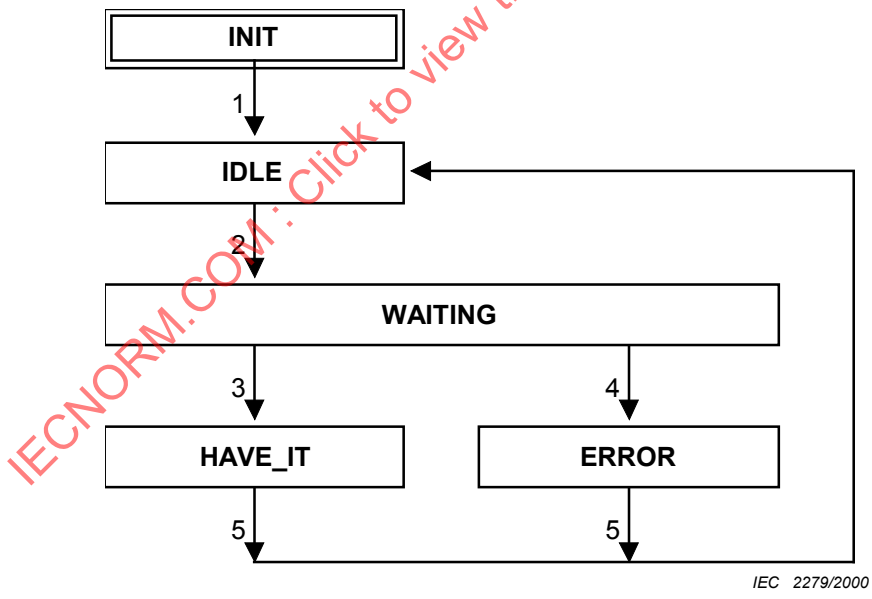


Figure 33 – Diagramme d'état du bloc fonctionnel WRITE

Tableau 39 – Transitions du diagramme d'état WRITE

Transition	Condition
1	Initialisation effectuée
2	Au front montant sur l'entrée REQ
3	Réponse positive du partenaire de communication distant
4	Réponse négative du partenaire de communication distant ou autres problèmes de communication détectés
5	Après la prochaine invocation de cette instance

Tableau 40 – Table d'actions pour le diagramme d'état WRITE

		Sorties du FB		
Etat	**Actions**	**DONE** [c]	**ERROR** [c]	**STATUS**
INIT [a]	Initialiser les sorties	0	0	0
IDLE	Aucune action	0	0	---
WAITING	Demander d'écrire les variables dans le partenaire de communication distant	---	---	–1
HAVE_IT	Indiquer un succès	1	0	0
ERROR	Indiquer une erreur	0	1	[b]

--- indique les sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

7.7 Contrôle verrouillé

La fonction de communication de contrôle verrouillé de l'AP utilise les blocs fonctionnels SEND et RCV.

Les instances correspondantes d'un bloc fonctionnel SEND et d'un RCV produisent une fonction de contrôle verrouillé de l'AP. Deux instances du bloc fonctionnel SEND et RCV correspondent si les valeurs des paramètres ID référencent le même canal de communication et si les valeurs des paramètres R_ID sont égales sur la portée de ce canal de communication.

L'instance SEND demande à l'instance RCV d'exécuter une opération de l'application et d'informer l'instance SEND du résultat de l'opération. Cette action présente deux aspects: la synchronisation du programme application des instances SEND et RCV et l'échange d'informations entre elles. Cette fonction peut être utilisée pour effectuer un appel de procédure distant entre un programme application et un autre.

La fonction de contrôle verrouillé est demandée par un front montant sur l'entrée REQ de l'instance du bloc fonctionnel SEND. Suite à une demande, l'instance SEND prend les données de ses entrées SD_i et les transmet à l'instance RCV correspondante.

Lorsque l'entrée EN_R de l'instance RCV a la valeur 1, elle est autorisée à recevoir les données provenant de l'instance SEND correspondante et à exécuter l'opération prévue du programme application. Lorsque l'instance RCV reçoit les données de l'instance SEND, elle transfère les données reçues au programme application via ses sorties RD_i. La sortie NDR de l'instance RCV démarre un cycle pour indiquer que de nouvelles données ont été reçues. Après l'exécution de l'opération prévue du programme application, les données de résultat sont

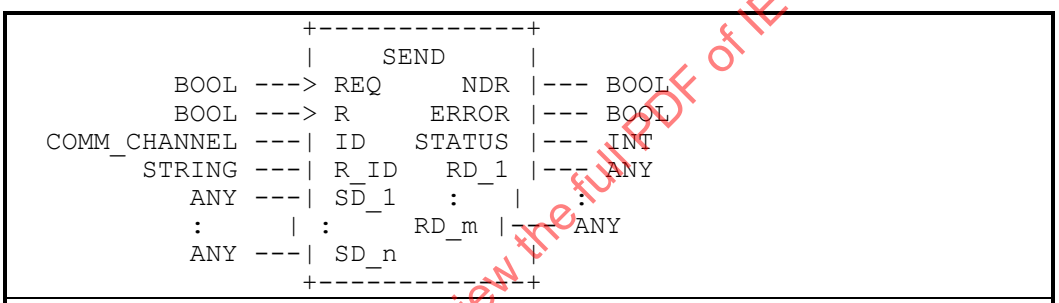
prises via les entrées SD_i et la réponse est initiée sur un front montant de l'entrée RESP de l'instance RCV.

Lorsque l'instance SEND reçoit cette réponse, elle transfère les données reçues à ses sorties RD_i. La sortie NDR de l'instance SEND impulse un cycle pour indiquer que de nouvelles données sont prêtes.

Un front montant sur l'entrée R de l'instance SEND réinitialise l'instance SEND ainsi que l'instance RCV correspondante.

Le nombre et le type de données des entrées SD_i de l'instance SEND et des sorties RD_i de l'instance RCV correspondante doivent être compatibles. Ceci est requis également pour les entrées SD_i de l'instance RCV et les sorties RD_i de l'instance SEND. Les entrées SD_i et les sorties RD_i des blocs fonctionnels SEND et RCV sont extensibles. Les données d'envoi ou les données de réponse, ou les deux, peuvent être vides, c'est-à-dire que l'utilisateur n'a programmé aucune entrée SD_i ni sortie RD_i correspondante.

Si les données reçues ne correspondent pas aux sorties RD_i du bloc fonctionnel RCV ou si une erreur se produit, la sortie ERROR démarre un cycle pour indiquer une erreur et la sortie STATUS contient le code d'erreur.



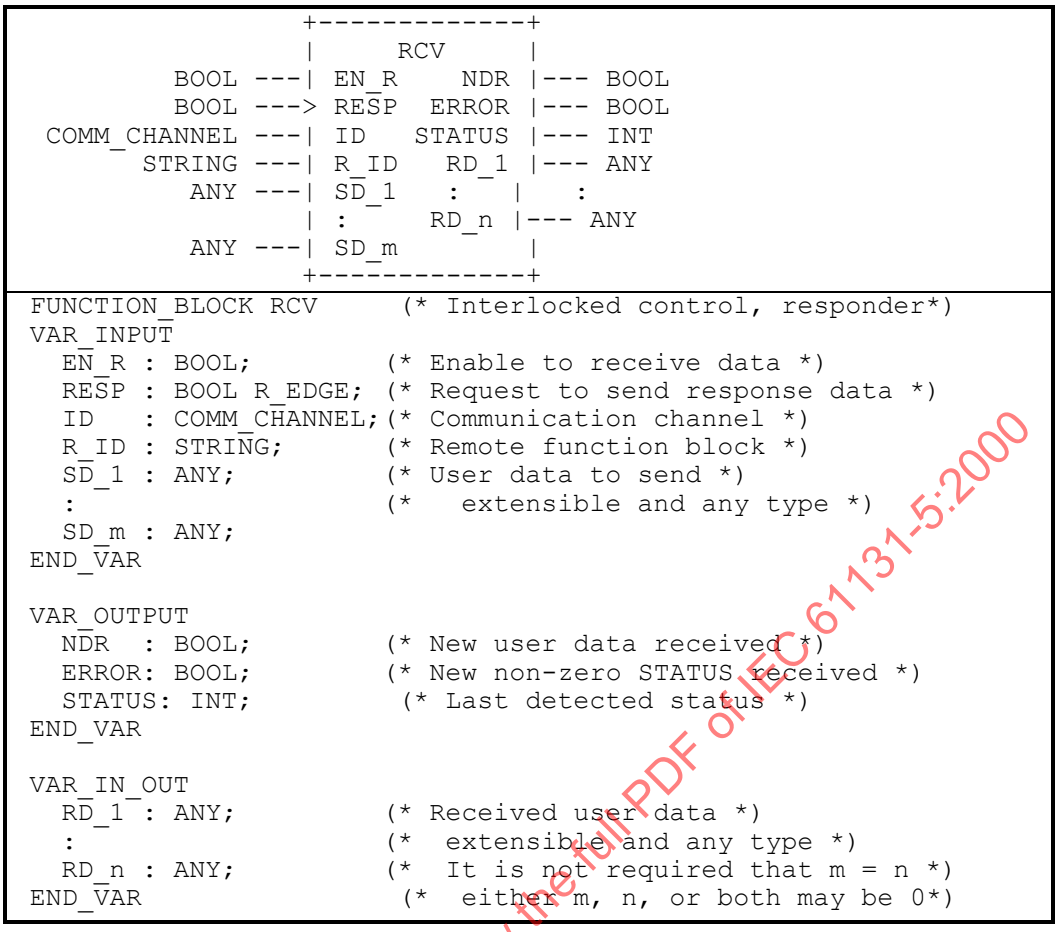

```
FUNCTION_BLOCK SEND     (* Interlocked control, requester*)
VAR_INPUT
  REQ   : BOOL R_EDGE; (* Request to send with raising edge*)
  R     : BOOL R_EDGE; (* Reset local, remote function *)
  ID    : COMM_CHANNEL;(* Communication channel *)
  R_ID  : STRING;      (* Remote function block *)
  SD_1  : ANY;         (* User data to send *)
  :                    (*   extensible and any type *)
  SD_n  : ANY;
END_VAR

VAR_OUTPUT
  NDR   : BOOL;        (* New user data received *)
  ERROR: BOOL;         (* New non-zero STATUS received *)
  STATUS: INT;          (* Last detected status *)
END_VAR

VAR_IN_OUT
  RD_1  : ANY;         (* Received user data *)
  :                    (*   extensible and any type *)
  RD_m  : ANY;         (*   It is not required that m = n *)
END_VAR                 (*   either m, n, or both may be 0*)
```

IEC 2280/2000

Figure 34 – Bloc fonctionnel SEND



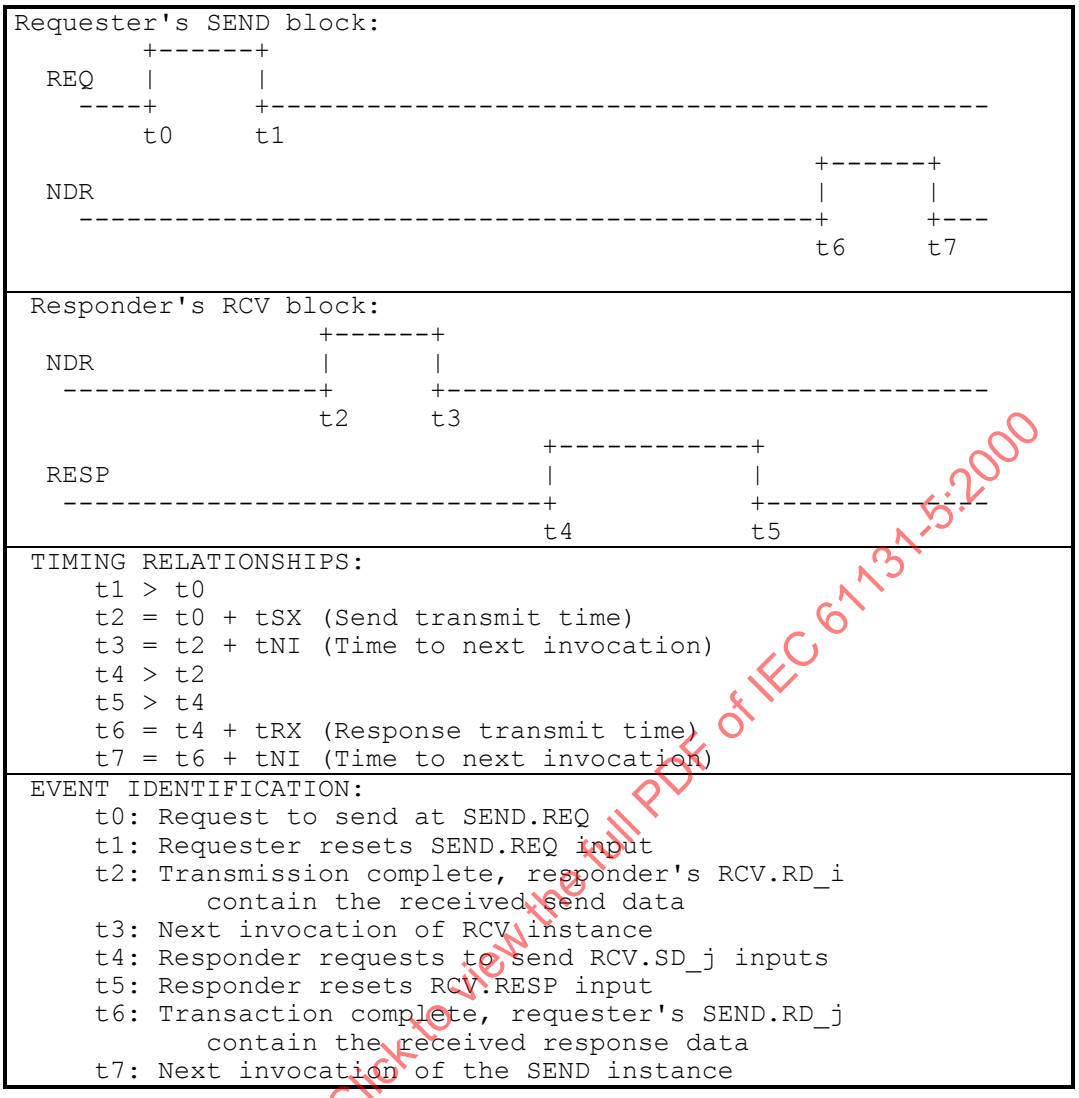

```
FUNCTION_BLOCK RCV     (* Interlocked control, responder*)
VAR_INPUT
  EN_R : BOOL;         (* Enable to receive data *)
  RESP : BOOL R_EDGE;  (* Request to send response data *)
  ID   : COMM_CHANNEL;(* Communication channel *)
  R_ID : STRING;       (* Remote function block *)
  SD_1 : ANY;          (* User data to send *)
  :                    (*   extensible and any type *)
  SD_m : ANY;
END_VAR

VAR_OUTPUT
  NDR  : BOOL;         (* New user data received *)
  ERROR: BOOL;         (* New non-zero STATUS received *)
  STATUS: INT;          (* Last detected status *)
END_VAR

VAR_IN_OUT
  RD_1 : ANY;          (* Received user data *)
  :                    (*  extensible and any type *)
  RD_n : ANY;          (*  It is not required that m = n *)
END_VAR                 (*  either m, n, or both may be 0*)
```

IEC 2281/2000

Figure 35 – Bloc fonctionnel RCV



IEC 2282/2000

Figure 36 – Chronogramme des blocs fonctionnels SEND et RCV

Le diagramme d'état représenté sur la Figure 37 décrit l'algorithme du bloc fonctionnel SEND. Les Tableaux 41 et 42 décrivent les transitions de ce diagramme d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel SEND.

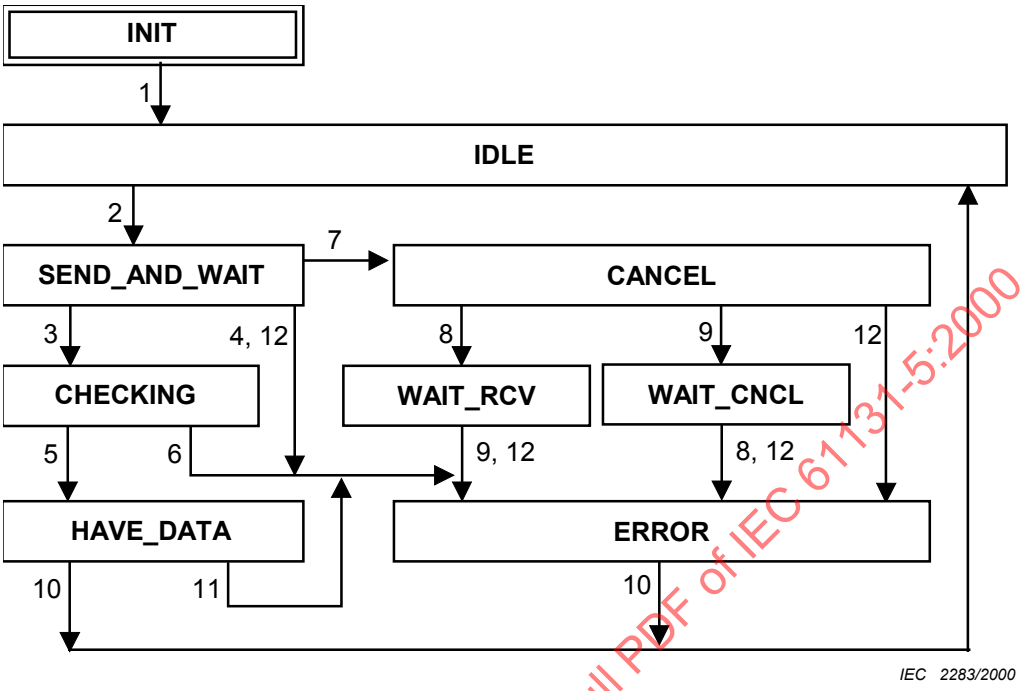


Figure 37 – Diagramme d'état du bloc fonctionnel SEND

Tableau 41 – Transitions du diagramme d'état SEND

Transition	Condition
1	Initialisation effectuée
2	Au front montant sur l'entrée REQ
3	Réponse positive de RCV
4	Réponse négative de RCV
5	Les types des données reçues et de RD_1 à RD_m correspondent
6	Les types des données reçues et de RD_1 à RD_m ne correspondent pas
7	Au front montant sur l'entrée R
8	Réponse positive ou négative à la demande de réinitialisation
9	Réponse positive ou négative de RCV
10	Après la prochaine invocation de l'instance
11	Problèmes de ressources locales détectés
12	Problèmes de communication détectés

Tableau 42 – Tableau d'actions pour le diagramme d'état SEND

		Sorties du FB			
Etat	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1 ... RD_m**
INIT [a]	Initialiser les sorties	0	0	0	Nul système
IDLE	Aucune action	0	0	---	---
SEND_AND_WAIT	Envoyer les données à RCV, évaluer tout d'abord la transition 7	---	---	–1	---
CHECKING	Vérifier la correspondance des types de données	---	---	---	---
HAVE_DATA	Déposer les données dans l'instance	1	0	0	Nouvelles données de RCV
ERROR	Indiquer une erreur	0	1	[b]	---
CANCEL	Demander de réinitialiser RCV	---	---	5	---
WAIT_RCV	Aucune action	---	---	---	---
WAIT_CNCL	Aucune action	---	---	---	---

--- indique des sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

Le diagramme d'état représenté sur la Figure 38 décrit l'algorithme du bloc fonctionnel RCV. Les Tableaux 43 et 44 décrivent les transitions de ce diagramme d'état et les actions à réaliser dans les états ainsi que les réglages des sorties du bloc fonctionnel RCV.

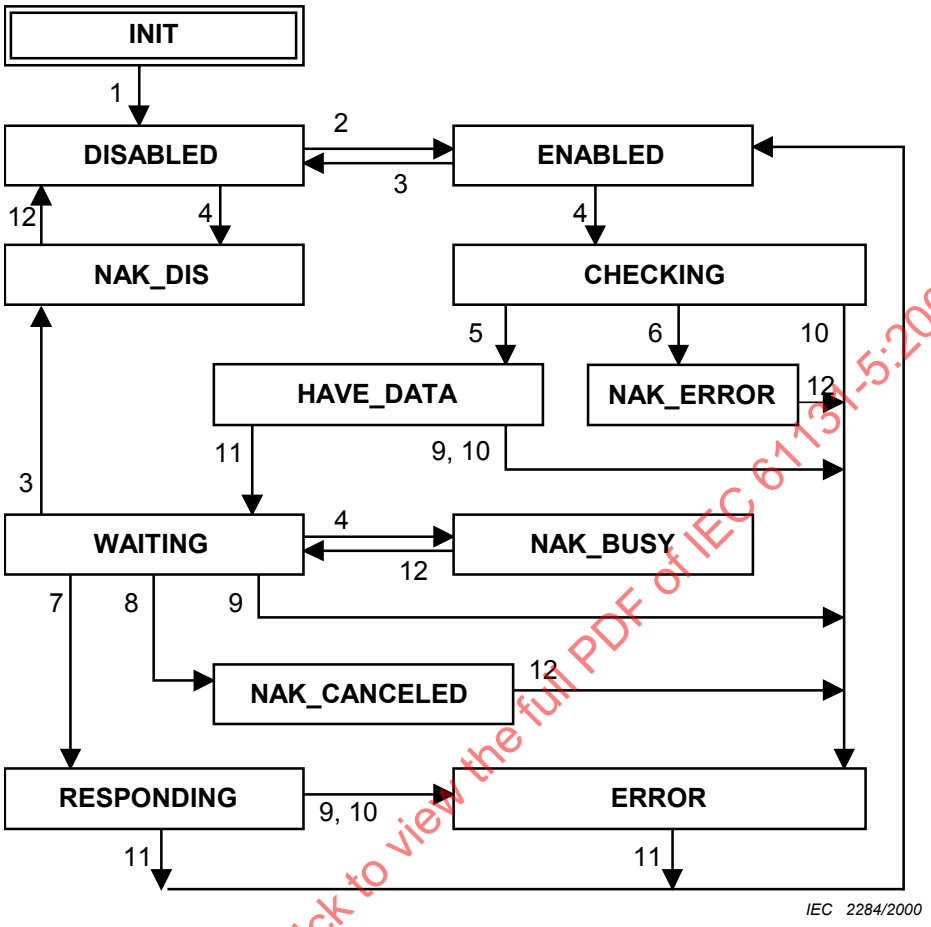


Figure 38 – Diagramme d'état du bloc fonctionnel RCV

Tableau 43 – Transitions du diagramme d'état RCV

Transition	Condition
1	Initialisation effectuée
2	EN_R = 1
3	EN_R = 0
4	Lorsque des données sont reçues depuis SEND
5	Les types des données reçues et de RD_1 à RD_n correspondent
6	Les types des données reçues et de RD_1 à RD_n ne correspondent pas
7	Front montant de l'entrée RESP
8	Lorsque la réinitialisation est demandée par SEND
9	Problèmes de communication détectés
10	Problèmes de ressources locales détectés
11	Après la prochaine invocation de l'instance
12	Vrai, c.-à-d. que la condition est vide

Tableau 44 – Tableau d'actions pour le diagramme d'état RCV

		Sorties du FB			
Etat	**Actions**	**NDR [c]**	**ERROR [c]**	**STATUS**	**RD_1 ... RD_n**
INIT [a]	Initialiser les sorties	0	0	0	Nul système
DISABLED	Aucune action	0	0	---	---
ENABLED	Aucune action	0	0	---	---
CHECKING	Vérifier la correspondance des types de données	---	---	---	---
HAVE_DATA	Déposer les données	1	0	0	Nouvelles données de SEND
NAK_ERROR	Envoyer une réponse négative à SEND	---	---	---	---
WAITING	Aucune action	0	0	---	---
NAK_BUSY	Envoyer une réponse négative à SEND	---	---	---	---
NAK_CANCELED	Envoyer une réponse positive à la demande de réinitialisation puis envoyer une réponse négative à SEND	---	---	---	---
RESPONDING	Envoyer une réponse positive avec données d'utilisateur à SEND	---	---	---	---
ERROR	Indiquer une erreur	0	1	[b]	---
NAK_DIS	Envoyer une réponse négative à SEND	---	---	---	---

--- indique les sorties de FB "non modifiées".

[a] INIT est l'état de démarrage à froid.

[b] Le code d'erreur est placé dans la sortie du statut.

[c] Voir Figure 10.

7.8 Rapport d'alarme programmé

La fonction de communication de rapport d'alarme programmé de l'AP utilise les blocs fonctionnels NOTIFY et ALARM.

Une instance du bloc fonctionnel NOTIFY ou une instance du bloc fonctionnel ALARM produit une instance de la fonction de rapport d'alarme programmé de l'AP.

Un AP peut être programmé au moyen du bloc fonctionnel ALARM afin de signaler un message d'alarme avec la capacité d'acquittement. Il peut également être programmé au moyen du bloc fonctionnel NOTIFY afin de signaler un message d'alarme sans capacité d'acquittement.

Le front montant sur l'entrée EVENT signale un message d'alarme d'une apparition d'événement, le front descendant sur l'entrée EVENT signale la disparition de cet événement. La cause de l'événement peut être donnée avec l'entrée EV_ID et sa sévérité peut être donnée à l'entrée SEVERITY des blocs fonctionnels. La sévérité doit être comprise entre 0 et 127 compris. 0 doit représenter la sévérité la plus élevée, 64 une sévérité normale et 127 la sévérité la plus faible. Des données supplémentaires aux entrées SD_i peuvent servir à préciser les détails de l'événement survenu.

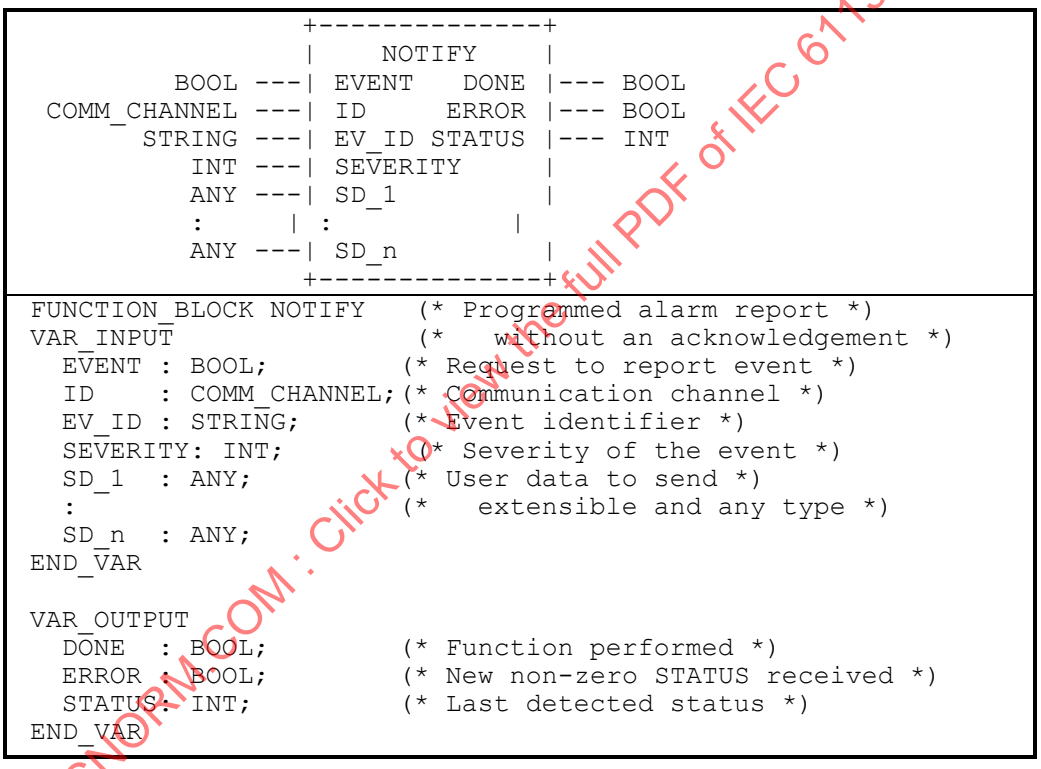
La sortie ACK_UP du bloc fonctionnel ALARM doit contenir une indication de l'acquittement ou non de l'arrivée de l'événement par le partenaire de communication distant. La sortie ACK_DN

doit contenir cette indication pour le départ de l'événement. Seul l'acquittement de l'événement le plus récent doit définir les sorties d'acquittement. Le même comportement doit être vrai pour le front descendant sur l'entrée EVENT et la sortie ACK_DN.

Le paramètre ID identifie le canal de communication vis-à-vis du partenaire de communication distant. Si le système de communication offre des canaux de communication qui prennent en charge les connexions d'une origine unique vers plusieurs destinataires ou vers tous les destinataires, ces blocs fonctionnels peuvent être utilisés pour programmer une fonction de rapport d'alarme entre un AP et plusieurs autres. Dans ce cas, le premier acquittement valide définit la sortie d'acquittement ALARM appropriée.

Les données envoyées peuvent fournir des informations supplémentaires dans le message d'alarme. Les données envoyées peuvent être vides, c'est-à-dire qu'aucune entrée SD_i n'est programmée.

Lorsqu'une erreur s'est produite, la sortie ERROR démarre un cycle pour indiquer une erreur et la sortie STATUS contient le code d'erreur.




```
FUNCTION_BLOCK NOTIFY    (* Programmed alarm report *)
VAR_INPUT                (*   without an acknowledgement *)
  EVENT : BOOL;         (* Request to report event *)
  ID    : COMM_CHANNEL;(* Communication channel *)
  EV_ID : STRING;       (* Event identifier *)
  SEVERITY: INT;         (* Severity of the event *)
  SD_1  : ANY;          (* User data to send *)
  :                     (*   extensible and any type *)
  SD_n  : ANY;
END_VAR

VAR_OUTPUT
  DONE  : BOOL;         (* Function performed *)
  ERROR : BOOL;         (* New non-zero STATUS received *)
  STATUS: INT;          (* Last detected status *)
END_VAR
```

IEC 2285/2000

Figure 39 – Bloc fonctionnel NOTIFY