IEC 62055-52

Edition 1.0 2008-05

INTERNATIONAL STANDARD

Electricity metering – Payment systems – Part 52: Standard transfer specification (STS) – Physical layer protocol for a two-way virtual token carrier for direct local connection

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: 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.

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC 62055-52

Edition 1.0 2008-05

INTERNATIONAL STANDARD

Electricity metering – Payment systems –
Part 52: Standard transfer specification (STS) – Physical layer protocol for a two-way virtual token carrier for direct local connection

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

PRICE CODE XA

ICS 35.100.20; 17.220.20; 91.140.50

ISBN 2-8318-9747-5

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ELECTRICITY METERING – PAYMENT SYSTEMS –

Part 52: Standard transfer specification (STS) – Physical layer protocol for a two-way virtual token carrier for direct local connection

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of 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, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). 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. 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 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 IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this International Standard may involve the use of a maintenance service concerning encryption key management and the stack of protocols on which the present International Standard IEC 62055-41 is based. [See Clause C.1 of IEC 62055-41.] The IEC takes no position concerning the evidence, validity and scope of this maintenance service.

The provider of the maintenance service has assured the IEC that he is willing to provide services under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the provider of the maintenance service is registered with the IEC. Information may be obtained from

Address: The STS Association, P.O. Box 868, Ferndale 2160, Republic of South Africa.
Tel: +27 11 789 1384
Fax: +27 11 789 1385
Email: email@sts.org.za
Website: http://www.sts.org.za

International Standard IEC 62055-52 has been prepared by working group 15, of IEC technical committee 13: Electrical energy measurement, tariff and load control.

IEC 62055-52 is complementary to, and should be read in conjunction with, IEC 62055-41.

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

FDIS	Report on voting
13/1424/FDIS	13/1428/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.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts of IEC 62055 series, published under the general title *Electricity metering – Payment systems,* can be found on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

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

A bilingual version of this publication may be issued at a later date.

INTRODUCTION

The IEC 62055 series covers payment systems, encompassing the customer information systems, point of sales systems, token carriers, payment meters and the respective interfaces that exist between these entities. At the time of preparation of this part, IEC 62055 comprised the following parts, under the general title *Electricity metering – Payment systems:*

Part 21: Framework for standardization

Part 31: Particular requirements – Static payment meters for active energy (classes 1 and 2)

Part 41: Standard transfer specification(STS) – Application layer protocol for one-way token carrier systems

Part 51: Standard transfer specification(STS) – Physical layer protocol for one-way numeric and magnetic card token carriers

Part 52: Standard transfer specification(STS) – Physical layer protocol for a two-way virtual token carrier for direct local connection

The *Part 4x series* specifies application layer protocols and the *Part 5x series* specifies physical layer protocols.

The protocol in this International Standard is based on the IEC 62056-21 communication protocol and has been simplified by removing features from the IEC 62056-21 protocol, which are not required for the current requirements of data exchange between the VTC07 client device and the payment meter server.

The main design objective in establishing the protocol has been the requirement to reduce the complexity of the software that is needed to implement this protocol in the payment meter. This directly relates to a smaller memory size that can be translated into a cost saving or the ability to include additional software features for a given memory size.

The Standard Transfer Specification (STS) is a secure message protocol that allows information to be carried between point of sale (POS) equipment and payment meters and it caters for several message types such as credit, configuration control, display and test instructions. It further specifies devices and Codes Of Practice that allows for the secure management (generation, storage, retrieval and transportation) of cryptographic keys used within the system.

The national electricity utility in South Africa (Eskom) first developed and published the STS in 1993 and transferred ownership to the STS Association in 1998 for management and further development.

Prior to the development of the STS, a variety of proprietary payment meters and POS equipment had been developed, which were however not compatible with each other. This gave rise to a definite need among the major users to move towards standardized solutions in addressing operational problems experienced where various types of payment meter and POS equipment had to be operated simultaneously. The Standard Transfer Specification was developed that would allow for the application and inter-operability of payment meters and POS equipment from multiple manufacturers in a payment metering installation.

The TokenCarrier is the physical medium used to transport information from a POS or the management system to the payment meter, or from the payment meter to the POS or management system. This part of IEC 62055 specifies a virtual token carrier as embodied in a direct local connection between a management device client and a payment meter server. It has been assigned identification code 07 by the STS Association and is also generally referred to as VTC07. New token carriers can be proposed as new work items through the National Committees or through the STS Association.

Although the main implementation of the STS is in the electricity supply industry, it inherently provides for the management of other utility services like water and gas. Future revisions of

the STS may allow for other token carrier technologies like smart cards and memory keys with two-way functionality.

The STS Association has established D-type liaison with working group 15 of IEC TC 13 for the development of standards within the scope of the STS, and is thus responsible for the maintenance of any such IEC standards that might be developed as a result of this liaison.

The STS Association is also registered with the IEC as a Registration Authority for providing maintenance services in support of the STS (see Clause C.1 of IEC 62055-41 for more information).

ELECTRICITY METERING – PAYMENT SYSTEMS –

Part 52: Standard transfer specification (STS) – Physical layer protocol for a two-way virtual token carrier for direct local connection

1 Scope

This part of IEC 62055 specifies a physical layer protocol of the STS for transferring units of credit and other management information between a client (typically a HHU) and a server (an STS-compliant electricity payment meter), typically over a direct local connection. It is complementary to the application layer protocol specified in IEC 62055-41 and should be used in conjunction with that standard.

This standard is not applicable to payment metering systems employing monetary-based tokens, complex tariffs and currency-mode accounting functions. It is only intended to support the STS functionality as defined in IEC 62055-41 and it does not support the additional functionality required for extended use that includes monetary-based tokens and complex meter functions such as tariffs, real time clocks and currency-mode accounting. If such extended use were required in the future, then it would need new work on this part of IEC 62055 as well as on IEC 62055-41.

It is intended for use across a range of payment meters developed by different manufacturers and to ensure compatibility between these products and other client devices.

It specifies a client/server communications protocol that:

- transfers STS-compliant tokens from a client device to a payment meter server;
- reads the result from the payment meter after transfer and execution of the token;
- transfers management data from the client device to the payment meter server;
- reads management data from the payment meter server and transfers same to the client device.

NOTE Although developed for payment systems for electricity, this standard can also be applied to other utility services, such as water and gas.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 60050-300, *International Electrotechnical Vocabulary – Electrical and electronic measurements and measuring instruments – Part 311: General terms relating to measurements – Part 312: General terms relating to electrical measurements – Part 313: Types of electrical measuring instruments – Part 314: Specific terms according to the type of instrument*

IEC 62051:1999, *Electricity metering – Glossary of terms*

IEC TR 62055-21:2005, *Electricity metering – Payment systems – Part 21: Framework for standardization*

IEC 62055-31:2005, *Electricity metering – Payment systems – Part 31: Particular requirements – Static payment meters for active energy (classes 1 and 2)*

IEC 62055-41, *Electricity metering – Payment systems – Part 41: Standard transfer specification – Application layer protocol for one-way token carrier systems*

IEC 62055-51, *Electricity metering – Payment systems Part 51: Standard transfer specification – Physical layer protocol for one-way numeric and magnetic card token carriers*

IEC 62056-21:2002, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 21: Direct local data exchange*

ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange*

STS 101-1, *Standard transfer specification (STS) – Interface specification – Physical layer mechanical and electrical interface for virtual token carriers*[1]

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 60050-300, IEC 62051, IEC 62055-31, IEC 62055-41 apply.

The term ASCII is used throughout the standard, which shall mean the 7-bit coded character set as defined in ISO/IEC 646.

3.2 Abbreviations

ACK	Acknowledge (ASCII code)
APDU	ApplicationProtocolDataUnit
ASCII	American Standard Code for Information Interchange
BCC	Block Check Character
Char	Character
CR	Carriage Return (ASCII code)
DL	Data Length
ETX	End Of Text (ASCII code)
FOIN	FunctionObjectIdentificationNumber
GPRS	General Packet Radio Service
GSM	Global System For Mobile Communication
hex	hexadecimal
HHU	Hand Held Unit

1 To be published

ID	Identification, Identifier
ISDN	Integrated Services Digital Network
ISO	International Standards Organisation
LAN	Local Area Network
LF	Line Feed (ASCII code)
ms	milli-second
NAK	Negative AcKnowledge (ASCII code)
OSI	Open Systems Interconnect
PLC	Power Line Carrier
POS	PointOfSale
PSTN	Public Switched Telephone Network
Ref	Reference clause
RID	Register IDentifier code
SOH	Start Of Header (ASCII code)
STS	Standard Transfer Specification
STX	Start Of Text (ASCII code)
TCDU	TokenCarrierDataUnit
VTC07	VirtualTokenCarrierType07
WAN	Wide Area Network

3.3 Notation and terminology

Throughout this standard, the following rules are observed regarding the naming of terms:

- entity names, data element names, function names and process names are treated as generic object classes and are given names in terms of phrases in which the words are capitalized and joined without spaces. Examples are: SupplyGroupCode as a data element name, TokenLockout as a function name and TransferCredit as a process name (see Note);
- direct (specific) reference to a named class of object uses the capitalized form, while general (non-specific) reference uses the conventional text, i.e. lower case form with spaces. An example of a direct reference is: “The SupplyGroupCode is linked to a group of meters”, while an example of a general reference is: “A supply group code links to a vending key”;
- attribute names of an object class uses the same convention as for the name of an object class, except that the first letter is in lower case format;
- object class names are capitalized, while names of attributes of a object class start with lower case;
- other terms use the generally accepted abbreviated forms like PSTN for Public Switched Telephone Network.

NOTE The notation used for naming of objects has been aligned with the so called “camel-notation” used in the common information model (CIM) standards prepared by TC 57, in order to facilitate future harmonization and integration of payment system standards with the CIM standards.

3.4 Numbering conventions

In this standard, the representation of numbers in binary strings uses the convention that the least significant bit is to the right, and the most significant bit is to the left.

Numbering of bit positions start with bit position 0, which corresponds to the least significant bit of a binary number.

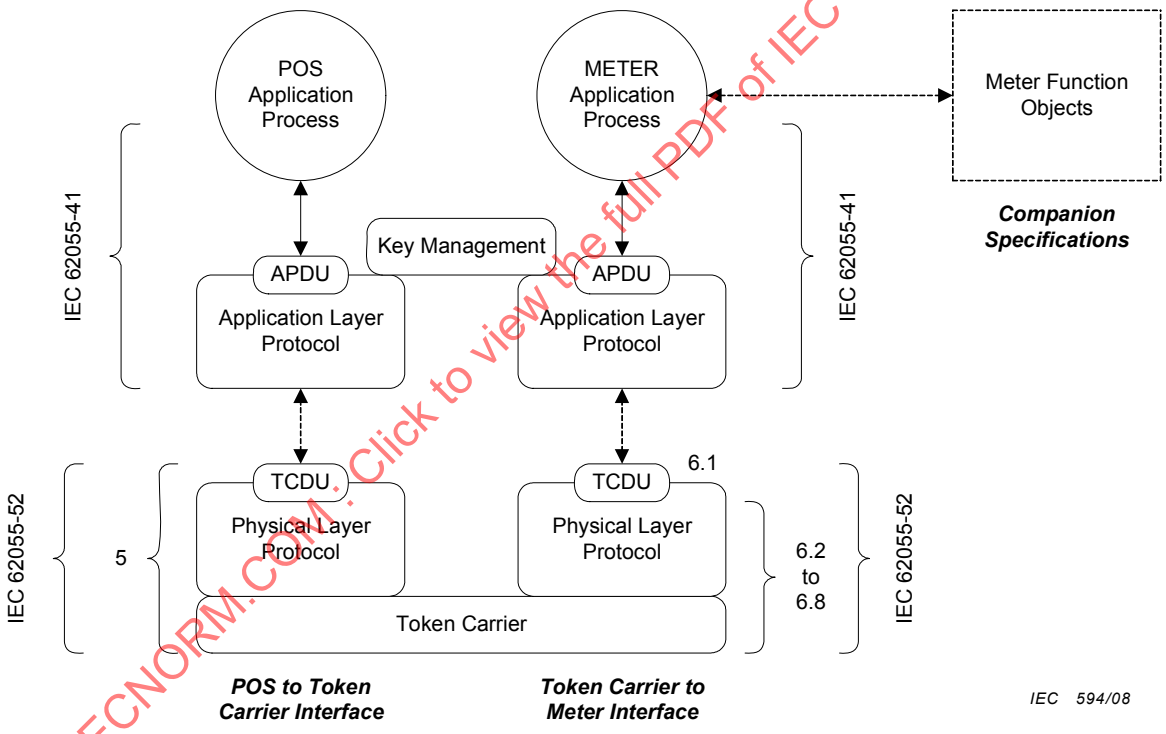
Numbers are generally in decimal format, unless otherwise indicated. Any digit without an indicator signifies decimal format.

Binary digit values range from 0-1.

Decimal digit values range from 0-9.

Hexadecimal digit values range from 0-9, A-F and are indicated by “hex”.

4 STS protocol reference model



Key:

APDU ApplicationLayerDataUnit; data interface to the application layer protocol

TCDU TokenCarrierDataUnit; data interface to the physical layer protocol

Relevant clause number references in this standard are indicated adjacent to each box

Figure 1 – Physical layers of the STS protocol stack

The STS is a secure data transfer protocol between a POS and a payment meter using a token carrier as the transfer medium. The application layer protocol deals with tokens, encryption processes and functions and is specified in IEC 62055-41, while the physical layer protocol deals with the actual encoding of the token data onto various types of token carriers.

This part of IEC 62055 specifies a physical layer protocol that deals with the actual encoding of the token data onto a virtual token carrier comprising of a direct local connection between a client and a server (payment meter) using a serial communications protocol, and operates in conjunction with the application layer protocol specified in IEC 62055-41(see Figure 1).

Examples of other types of virtual token carriers are: PSTN modem, ISDN modem, GSM modem, GPRS modem, Radio modem, PLC modem, Infra-red, LAN and WAN connections and direct local connection, which might be specified in the future in other parts of the IEC 62055-5x series.

A more complete description of the STS reference model and data flows from the POSApplicationProcess to the MeterApplicationProcess may be found in Clause 5 of IEC 62055-41.

The protocol defines a generic write and read message structure that allows for a client to read data from or write data to a payment meter by reference to a virtual register table as a logical interface to actual registers or functions. This standard defines a RegisterTable interface class (see 6.8.1) and a Register interface class (see 6.8.2) for a MeterFunctionObject, which gets defined in a companion specification. Companion specifications are not normative to IEC 62055-52 and are administered by the STS Association (see 6.9).

5 POSToTokenCarrierInterface: Physical layer protocol

The client interface to the virtual token carrier is not defined in detail in this standard, but it shall generally complement the requirements given in the relevant parts of Clause 6.

In practice, the client device is typically a mobile HHU that connects to the payment meter by means of a direct local connection, but it is also possible for the connection to be extended to a remote management system by means of suitable interposing modem devices linked over any appropriate communications medium. Such extended remote connection is not specifically covered in this standard, but in essence it simply means an extension of the physical medium.

6 TokenCarrierToMeterInterface: Physical layer protocol

6.1 TCDU

6.1.1 General

The TCDU is the data interface between the physical layer protocol and the application layer protocol and comprises the following data elements as given in Table 1.

Table 1 – Data elements in the TCDU

Element	Format	Reference
TokenData	66-bit binary	6.1.2
AuthenticationResult	Boolean	6.1.3
ValidationResult	Boolean	6.1.4
TokenResult	Boolean	6.1.5

6.1.2 TokenData

This is the 66-bit binary format of the token data as decoded from the TokenCarrier. It is the same data element as is presented to the TCDU at the POSToTokenCarrierInterface (see 6.4.3 to 6.4.5 of IEC 62055-41).

6.1.3 AuthenticationResult

This is a status indicator to the physical layer protocol to convey the result from the initial authentication checks. See also 7.1.3 of IEC 62055-41 for definition of the AuthenticationResult values.

These data elements are also included in the TokenStatus register (see 6.8.3.7) for access by the client.

6.1.4 ValidationResult

This is a status indicator to the physical layer protocol to convey the result from the initial validation checks. See also 7.1.4 of IEC 62055-41 for definition of the ValidationResult values.

These data elements are also included in the TokenStatus register (see 6.8.3.7) for access by the client.

6.1.5 TokenResult

This is a status indicator from the MeterApplicationProcess to convey the result after processing the Token so that the physical layer protocol can take the appropriate action. See also 7.1.5 of IEC 62055-41 for the definition of the TokenResult values.

These data elements are also included in the TokenStatus register (see 6.8.3.7) for access by the client.

6.2 Physical connection and signal interfaces

6.2.1 Interface options

The payment meter manufacturer may implement any of the options given in 6.2.2 to 6.2.4, but the actual implementation shall be agreed between the manufacturer and the utility.

Further options may be given in future revisions to this standard.

6.2.2 Option 1: Optical interface

The physical connection and signal interfaces shall comply with the requirements given in 4.3 of IEC 62056-21.

6.2.3 Option 2: Current loop interface

The physical connection and signal interfaces shall comply with the requirements given in 4.1 of IEC 62056-21.

6.2.4 Option 3: Voltage interface

The physical connection and signal interfaces shall comply with the requirements given in STS 101-1.

6.3 Character transmission

6.3.1 Transmission type

Asynchronous serial transmission - half duplex.

6.3.2 Transmission format

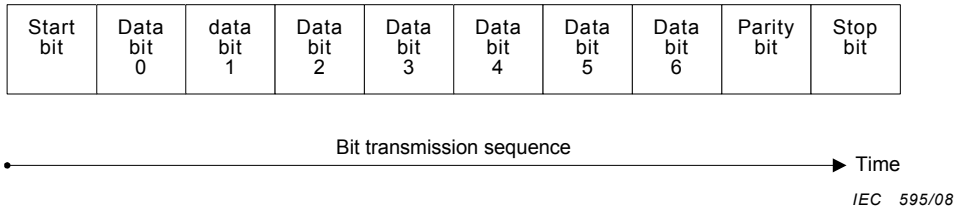


Figure 2 – Character transmission format

The character transmission format comprises 1 start bit, 7 data bits, 1 even parity bit, and 1 stop bit, as shown in Figure 2.

The bit-encoding of the 7-bit ASCII character is given in Table 2.

Table 2 – Bit-encoding of a 7-bit character code

Bit	Value	Context
Start bit	0	Signifies the start of a character serial bit stream
Data bit 0	0 – 1	Least significant bit of the character code
Data bit 1	0 – 1	Character code
Data bit 2	0 – 1	Character code
Data bit 3	0 – 1	Character code
Data bit 4	0 – 1	Character code
Data bit 5	0 – 1	Character code
Data bit 6	0 – 1	Most significant bit of the character code
Parity bit	0	If data bits 0 to 6 contain an even number of binary 1's
	1	If data bits 0 to 6 contain an odd number of binary 1's
Stop bit	1	Signifies the end of a character serial bit stream

6.3.3 Transmission speed

Baud rate = 2 400 bits per second.

The server shall not support Baud rate negotiation or switchover.

6.3.4 Character encoding

The format of application data is defined in specifications applicable to the particular data element. For the purposes of this standard, all application data elements that are to be transported over the VTC07 virtual token carrier shall be presented in binary, hexadecimal or decimal format before character encoding.

Binary format data shall be encoded as follows:

Binary format data is split into 4-bit nibbles and each nibble is converted into its 7-bit ASCII representation before transmission. The most significant nibble is transmitted first. During reception by the client the inverse process is applied to reconstruct the data into its original format. In the case where the data format is not a whole number of 4-bit nibbles, then the binary number shall be right justified and padded on the left with leading binary zeros to make a whole number of nibbles. An example is given in Table 3.

Table 3 – Character encoding example of a 14-bit binary number

Operation	Result	7-bit ASCII
Binary number starting value	11001011011110	
Split into 4-bit nibbles	11 0010 1101 1110	
Padded on the left with zeros to make a whole number of nibbles	**00**11 0010 1101 1110	
1st nibble to convert and transmit	0011	0110011
2nd nibble to convert and transmit	0010	0110010
3rd nibble to convert and transmit	1101	1000100
4th nibble to convert and transmit	1110	1000101

Hexadecimal format data shall be encoded as follows:

Hexadecimal format data is split into hexadecimal digits and each digit is converted into its 7-bit ASCII representation before transmission. The most significant digit is transmitted first. During reception by the client the inverse process is applied to reconstruct the data into its original format. An example is given in Table 4.

Table 4 – Character encoding example of a 4-digit hexadecimal number

Operation	Result	7-bit ASCII
Hexadecimal number starting value	12AF	
Split into 4 hexadecimal digits	1 2 A F	
1st digit to convert and transmit	1	0110001
2nd digit to convert and transmit	2	0110010
3rd digit to convert and transmit	A	1000001
4th digit to convert and transmit	F	1000110

Decimal format data shall be encoded as follows:

Decimal format data is split into decimal digits and each digit is converted into its 7-bit ASCII representation before transmission. The most significant digit is transmitted first. During reception by the client the inverse process is applied to reconstruct the data into its original format. An example is given in Table 5.

Table 5 – Character encoding example of a 4-digit decimal number

Operation	Result	7-bit ASCII
Decimal number starting value	3059	
Split into 4 decimal digits	3 0 5 9	
1st digit to convert and transmit	3	0110011
2nd digit to convert and transmit	0	0110000
3rd digit to convert and transmit	5	0110101
4th digit to convert and transmit	9	0111001

6.4 Message syntax definitions

6.4.1 General

The various messages supported are given below. Reference numbers below each field refer to the listing in 6.5, defining the detailed definition for each field.

Each field represents one or more 7-bit ASCII characters after encoding of the data (see 6.3.4).

6.4.2 IDRequest message

The client sends the IDRequest message to request the server to produce identification information that will help the client determine which server type it is dealing with. This is normally the first message the server receives in a communication session.

/	?	!	CR	LF
(1)	(2)	(3)	(4)	(5)

See also 6.6.2 for more information on the processing of this message.

6.4.3 IDResponse message

The server sends the IDResponse message to the client in response to receiving the IDRequest message from the client. The server identifies its payment meter type by sending the manufacturer code and software version code of the server.

/	M	X	V	CR	LF
(1)	(22)	(6)	(7)	(4)	(5)

See also 6.6.2 for more information on the processing of this message.

6.4.4 ReadCommand message

The client reads a dataset from a particular Register (see 6.8.2) in the payment meter by sending a ReadCommand to the server.

SOH	R	STX	RID	DL	ETX	BCC
(8)	(9)	(10)	(11)	(12)	(13)	(14)

NOTE BCC is calculated on fields (9) to (13).

See also 6.6.3 for more information on the processing of this message.

6.4.5 WriteCommand message

The client writes a dataset to a particular Register (see 6.8.2) in the payment meter by sending a WriteCommand to the server.

SOH	W	STX	RID	(	D	)	ETX	BCC
(8)	(15)	(10)	(11)	(16)	(17)	(18)	(13)	(14)

NOTE BCC is calculated on fields (15) to (13)

See also 6.6.4 for more information on the processing of this message.

6.4.6 BreakCommand message

The client cancels all previous messages that have not yet been executed by the server by sending a BreakCommand to the server.

SOH	B	ETX	BCC
(8)	(19)	(13)	(14)

NOTE BCC is calculated on fields (19) to (13)

See also 6.6.5 for more information on the processing of this message.

6.4.7 ACK: Acknowledge message

The server returns a positive response by sending the Acknowledge message.

ACK
(20)

See also 6.6.4 and 6.6.5 for more information on the processing of this message.

6.4.8 NAK: NegativeAcknowledge message

The server returns a negative response by sending the NegativeAcknowledge message.

NAK
(21)

See also 6.6.1 to 6.6.5 and 6.7.2 for more information on the processing of this message.

6.4.9 Data message

The server sends a Data message in response to a ReadCommand from the client.

STX	(	D	)	ETX	BCC
(10)	(16)	(17)	(18)	(13)	(14)

NOTE BCC is calculated on fields (16) to (13)

See also 6.6.3 for more information on the processing of this message.

6.5 Message field definitions

Each message field comprises one or more 7-bit ASCII characters. The possible value for each character in a field is given in Table 6 below.

NOTE Although this standard is based on IEC 62056-21, the character values in Table 6 are given in decimal format, whereas the equivalent values in IEC 62056-21 are given in hexadecimal format.

Table 6 – Message field definitions

Field	Field number	No. characters	Character value	Context	Reference
/	(1)	1	47	Start character ASCII "/" ; Right-slash	6.4.2 6.4.3
?	(2)	1	63	IDRequest command character ASCII "?" ; Question mark	6.4.2
!	(3)	1	33	End character ASCII "!" ; Exclamation mark	6.4.2
CR	(4)	1	13	Completion character ASCII CR ; Carriage return	6.4.2 6.4.3
LF	(5)	1	10	Completion character ASCII LF ; Line feed	6.4.2 6.4.3
X	(6)	2	48-57	Manufacturer code ASCII "0" – "9" Range 00 – 99 2-digit decimal representation of the manufacturer code as defined in 6.1.2.3.2 of IEC 62055-41	6.4.3
V	(7)	4	48-57 and 65-70	SoftwareVersion ASCII "0" – "9" and "A" – "F" 4-digit representation of the payment meter software version. Interpretation of the digits are manufacturer-specific (see also 6.8.3.5)	6.4.3
SOH	(8)	1	1	Header character ASCII SOH ; Start of header	6.4.4 6.4.5 6.4.6
R	(9)	1	82	ReadCommand character ASCII "R" ;	6.4.4
STX	(10)	1	2	Frame start character ASCII STX ; Start of text	6.4.4 6.4.5 6.4.9
RID	(11)	4	48-57 and 65-70	registerID ASCII "0" – "9" and "A" – "F" 4-digit register identifier field in hexadecimal format. It is the identifier of the register from which data must be read, or to which data must be written. See also 6.8 for definition of the RegisterTable	6.4.4 6.4.5

Field	Field number	No. characters	Character value	Context	Reference
DL	(12)	1	48-57 and 65-70	Data length ASCII “0” – “9” and “A” – “F” The server shall interpret the value of this field in accordance with the particular definition for each Register (see 6.8.2) in a RegisterTable (see 6.8.1) instance. NOTE In legacy payment meters, this control data element was used to specify the number of bytes to be read from a register during execution of a ReadCommand message. It was also constrained to a maximum value of 10	6.4.4
ETX	(13)	1	3	Frame end character ASCII ETX ; End of text	6.4.4 6.4.5 6.4.6 6.4.9
BCC	(14)	1	0-127	Block check character This is a calculated value and may result in any one of the values within the complete 7-bit ASCII range. The BCC is calculated from the first character immediately following the first SOH or STX character in the message up to and including the ETX character, which terminates the message frame. The BCC is calculated by performing a 7-bit character-wise logical XOR and is placed immediately following the ETX character	6.4.4 6.4.5 6.4.6 6.4.9
W	(15)	1	87	WriteCommand character ASCII “W“ ;	6.4.5
(	(16)	1	40	Open data block character ASCII “(“ ; Left-parenthesis	6.4.5 6.4.9
D	(17)	x	48-57 and 65-70	Dataset ASCII “0” – “9” and “A” – “F” Data field representing the data to be written (WriteCommand message) or read (Data message) NOTE In legacy payment meters, the dataset was limited to 10 bytes maximum (20 characters)	6.4.5 6.4.9
)	(18)	1	41	Close data block character ASCII “)“ ; Right-parenthesis	6.4.5 6.4.9
B	(19)	1	66	BreakCommand character ASCII “B“ ;	6.4.6
ACK	(20)	1	6	Acknowledge character ASCII ACK ; Acknowledge	6.4.7
NAK	(21)	1	21	Negative acknowledge character ASCII NAK ; Negative acknowledge	6.4.8
M	(22)	1	77	Manufacturer information follows ASCII “M” ; Indicates that manufacturer details are contained in the next 6 characters of the message	6.4.3

The D field (17) and the DL field (12) for each Register instance shall be defined in a companion specification (see 6.8.1 and 6.9).

6.6 Physical layer protocol functions

6.6.1 Server protocol flow diagram

The server protocol flow diagram is shown in Figure 3.

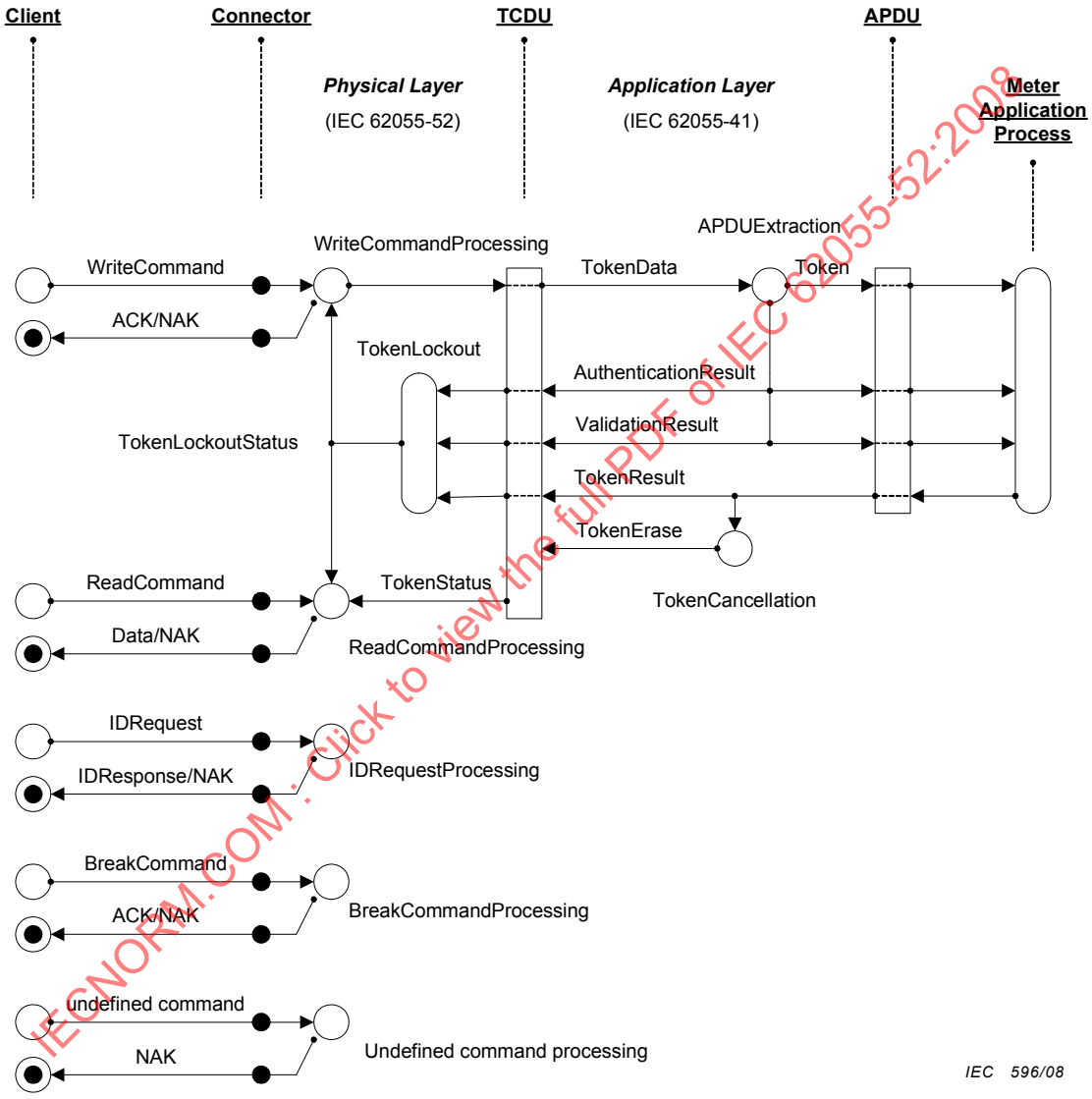


Figure 3– Server protocol flow diagram

The client may send messages in any sequence and the server shall process each message in the order that they are received, except for the BreakCommand message (see 6.6.5).

A communication session shall constitute one message pair, initiated as a request from the client and ending with a response from the server.

The client may recover from a session error condition by means of time-out (see 6.6.2 to 6.6.7, and 6.7).

The server shall support the request messages given in Table 7.

Table 7 – Request messages supported by the server

Request message	Reference
IDRequest	6.4.2
ReadCommand	6.4.4
WriteCommand	6.4.5
BreakCommand	6.4.6

The server shall support the response messages given in Table 8.

Table 8 – Response messages supported by the server

Response message	Reference
IDResponse	6.4.3
ACK	6.4.7
NAK	6.4.8
Data	6.4.9

The server shall support the functions given in Table 9.

Table 9 – Functions supported by the server

Function	Reference
IDRequestProcessing function	6.6.2
ReadCommandProcessing function	6.6.3
WriteCommandProcessing function	6.6.4
BreakCommandProcessing function	6.6.5
TokenLockout function	6.6.7

6.6.2 IDRequestProcessing function

The IDRequestProcessing function locates in the physical layer protocol of the payment meter server, which receives and processes IDRequest messages (see 6.4.2) from a client.

The server shall process an IDRequest message in accordance with the server state diagram given in Figure A.1.

The server shall respond with a NAK under any one of the following conditions:

- the value of the parity bit in the transmission format is not in accordance with 6.3.2 in any of the received characters; the server shall also set the ParityError code in the ServerStatus register;
- the maximum time allowed between characters has been exceeded in accordance with the values given in Table 11; the server shall also set the CharacterTimeoutError code in the ServerStatus register;
- the client has sent more characters than the server could receive; the server shall also set the CharacterOverflowError code in the ServerStatus register;
- the received message syntax and structure does not comply with the definition as given in 6.4.2; the server shall also set the MessageSyntaxError code in the ServerStatus register;

- an error that is not defined in this standard has occurred during the receipt of the message; the server shall also set the UndefinedTransmissionError code in the ServerStatus register.

For each of the above conditions, the value of the ServerStatus register (see 6.8.3.4) shall be set to the corresponding code given in Table 20.

If none of the above conditions are true, the server shall first execute the command, then respond with an IDResponse message and then set the CommandExecuted code in the ServerStatus register.

After execution of the command, the server shall respond with an IDResponse message to the client (see 6.4.3).

6.6.3 ReadCommandProcessing function

The ReadCommandProcessing function locates in the physical layer protocol of the payment meter server, which receives and processes ReadCommand messages (see 6.4.4) from a client.

To “read” information from any defined Register in the payment meter, the client may send a ReadCommand message with the appropriate registerID value in the RID field in accordance with that defined in the RegisterTable (see 6.8).

The server shall process a ReadCommand message in accordance with the server state diagram given in Figure A.2.

The server shall respond with a NAK under any one of the following conditions:

- the value of the parity bit in the transmission format is not in accordance with 6.3.2 in any of the received characters; the server shall also set the ParityError code in the ServerStatus register;
- the maximum time allowed between characters has been exceeded in accordance with the values given in Table 11; the server shall also set the CharacterTimeoutError code in the ServerStatus register;
- the client has sent more characters than the server could receive; the server shall also set the CharacterOverflowError code in the ServerStatus register;
- the received message syntax and structure does not comply with the definition as given in 6.4.4; the server shall also set the MessageSyntaxError code in the ServerStatus register;
- the BCC field in the message (see 6.4.4) does not match the calculated value from the received message; the server shall also set the BCCError code in the ServerStatus register;
- an error that is not defined in this standard has occurred during the receiving phase of the message; the server shall also set the UndefinedTransmissionError code in the ServerStatus register.

For each of the above conditions, the value of the ServerStatus register (see 6.8.3.4) shall be set to the corresponding code given in Table 20.

If none of the above conditions are true, then the server shall proceed to “read” the dataset from the Register indicated in the RID field of the message.

During the “reading” from the Register, the server shall respond with a NAK under any one of the following conditions:

- the registerID as received in the RID field of the message does not correspond to a supported register in the server; the server shall also set the registerIDInvalid code in the ServerStatus register:
- the Register is busy and cannot process the “read” request at this point in time; the server shall also set the RegisterBusy code in the ServerStatus register;
- the Register does not have the R attribute defined as True (see 6.8.2), thus it cannot be “read”; the server shall also set the RegisterReadProtected code in the ServerStatus register;
- the Register is supported, but the function that it is associated with has been disabled in the payment meter; the server shall also set the FunctionDisabled code in the ServerStatus register;
- an error that is not defined in this standard has occurred during the “reading” phase of the Register; the server shall also set the UndefinedReadingError code in the ServerStatus register.

For each of the above conditions, the value of the ServerStatus register (see 6.8.3.4) shall be set to the corresponding code given in Table 20.

If none of the above conditions are true, the server shall first execute the “read” from the Register; then respond with the Data message and then set the CommandExecuted code in the ServerStatus register.

After execution of the command, the server shall respond with the Data message to the client (see 6.4.9).

Additional processing related to token entry is given in 6.8.3.7 for the TokenStatus register.

6.6.4 WriteCommandProcessing function

The WriteCommandProcessing function locates in the physical layer protocol of the payment meter server, which receives and processes WriteCommand messages (see 6.4.5) from a client.

To “write” information into any defined Register in the payment meter, the client may send a WriteCommand message with the dataset in the D field and the appropriate registerID value in the RID field in accordance with that defined in the RegisterTable (see 6.8).

The server shall process a WriteCommand message in accordance with the server state diagram given in Figure A.3.

The server shall respond with a NAK under any one of the following conditions:

- the value of the parity bit in the transmission format is not in accordance with 6.3.2 in any of the received characters; the server shall also set the ParityError code in the ServerStatus register;
- the maximum time allowed between characters has been exceeded in accordance with the values given in Table 11; the server shall also set the CharacterTimeoutError code in the ServerStatus register;
- the client has sent more characters than the server could receive; the server shall also set the CharacterOverflowError code in the ServerStatus register;
- the received message syntax and structure does not comply with the definition as given in 6.4.5; the server shall also set the MessageSyntaxError code in the ServerStatus register;
- the BCC field in the message (see 6.4.5) does not match the calculated value from the received message; the server shall also set the BCCError code in the ServerStatus register;

- an error that is not defined in this standard has occurred during the receiving phase of the message; the server shall also set the UndefinedTransmissionError code in the ServerStatus register.

For each of the above conditions, the value of the ServerStatus register (see 6.8.3.4) shall be set to the corresponding code given in Table 20.

If none of the above conditions are true, then the server shall proceed to "write" the dataset to the Register indicated in the RID field of the message.

During the "writing" to the Register the server shall respond with a NAK under any one of the following conditions:

- the registerID as received in the RID field of the message does not correspond to a supported register in the server; the server shall also set the RegisterIDInvalid code in the ServerStatus register;
- the Register is busy and cannot process the "write" command at this point in time; the server shall also set the RegisterBusy code in the ServerStatus register;
- the Register does not have the W attribute defined as True (see 6.8.2), thus it cannot be "written" to; the server shall also set the RegisterWriteProtected code in the ServerStatus register;
- the Register is supported, but the function that it is associated with has been disabled in the payment meter; the server shall also set the FunctionDisabled code in the ServerStatus register;
- in the case of "writing" to the BinaryTokenEntry register (see 6.8.3.6), or any other functionally equivalent register that may be defined in a companion specification, and the TokenLockoutStatus indication (see 6.6.7) from the TokenLockout function is true; the server shall also set the TokenLockout code in the ServerStatus register;
- an error that is not defined in this standard has occurred during the "writing" phase of the Register; the server shall also set the UndefinedWritingError code in the ServerStatus register.

If none of the above conditions are true, the server shall first respond with an ACK, then proceed to execute the "write" to the Register and then set the CommandExecuted code in the ServerStatus register.

The additional processing, specifically related to the loading of tokens into the payment meter, is described in 6.8.3.6.

An ACK or NAK response shall not be conditioned on the outcome after execution of a WriteCommand message, but shall merely indicate that the message was successfully received.

Application-specific feedback to the client after execution of the WriteCommand message shall be by means of setting of appropriate data elements in a suitably defined register, which can be read by the client.

6.6.5 BreakCommandProcessing function

The BreakCommandProcessing function locates in the physical Layer Protocol of the payment meter server, which receives and processes BreakCommand messages (see 6.4.6) from a client.

The server shall process a BreakCommand message in accordance with the server state diagram given in Figure A.4.

The server shall respond with a NAK under any one of the following conditions:

- the value of the parity bit in the transmission format is not in accordance with 6.3.2 in any of the received characters; the server shall also set the ParityError code in the ServerStatus register;
- the maximum time allowed between characters has been exceeded in accordance with the values given in Table 11; the server shall also set the CharacterTimeoutError code in the ServerStatus register;
- the client has sent more characters than the server could receive; the server shall also set the CharacterOverflowError code in the ServerStatus register;
- the received message syntax and structure does not comply with the definition as given in 6.4.6; the server shall also set the MessageSyntaxError code in the ServerStatus register,
- the BCC field in the message (see 6.4.6) does not match the calculated value from the received message; the server shall also set the BCCError code in the ServerStatus register;
- an error that is not defined in this standard has occurred during the receiving phase of the message; the server shall also set the UndefinedTransmissionError code in the ServerStatus register.

For each of the above conditions, the value of the ServerStatus register (see 6.8.3.4) shall be set to the corresponding code given in Table 20.

If none of the above conditions are true, the server shall first respond with an ACK, then proceed to execute the “break” command and then set the CommandExecuted code in the ServerStatus register.

An ACK or NAK response shall not be conditioned on the outcome after execution of a BreakCommand message, but shall merely indicate that the message was successfully received.

A BreakCommand message shall always have the highest priority and become the next in line to be executed by the server, irrespective of any other messages that may be waiting in a queue, pending execution.

Upon execution of a BreakCommand message, the server shall terminate all pending messages and wait for the completion of any currently active processes, which were initiated by previous command messages from the client. See also 6.7.1 to 6.7.3 for handling of error conditions.

6.6.6 Undefined command processing

In the case where the server receives a command that is not defined in this standard, it shall process such command in accordance with the requirements of this subclause.

The server shall respond with a NAK under any one of the following conditions:

- the value of the parity bit in the transmission format is not in accordance with 6.3.2 in any of the received characters; the server shall also set the ParityError code in the ServerStatus register;
- the maximum time allowed between characters has been exceeded in accordance with the values given in Table 11; the server shall also set the CharacterTimeoutError code in the ServerStatus register;
- the client has sent more characters than the server could receive; the server shall also set the CharacterOverflowError code in the ServerStatus register;

- the received message syntax and structure does not comply with the definition as given in 6.4.2, 6.4.4, 6.4.5 or 6.4.6; the server shall also set the MessageSyntaxError code in the ServerStatus register,

For each of the above conditions, the value of the ServerStatus register (see 6.8.3.4) shall be set to the corresponding code given in Table 20.

The server shall only respond with a NAK message to the client.

6.6.7 TokenLockout function

The TokenLockout function locates in the physical layer protocol of the payment meter server and monitors, the results of token processing in the application layer protocol or the MeterApplicationProcess by means of the ValidationResult, AuthenticationResult and TokenResult fields in the TCDU (see Figure 3). The monitoring result is indicated by TokenLockoutStatus.

The TokenLockout function defined in this protocol shall only affect the processing of tokens being entered by means of the virtual token carrier interface as defined in this standard and shall not influence the processing of tokens being entered by means of other token carrier interfaces that may also be present in the payment meter.

For the purpose of this function, short-codes shall not be treated the same as tokens. Short-codes shall not influence the TokenLockout function and valid short-codes shall not be affected by a lockout condition.

NOTE A short-code is a code that, when presented to a payment meter, may invoke the same action as that of a Class 1 token as defined in IEC 62055-41, except that it may comprise fewer numerical digits than that of the token. Maintenance personnel generally use short-codes to invoke test or display sequences in the payment meter.

In the event where any tokens are rejected in succession under any one of the token rejection conditions given in 8.2 of IEC 62055-41, further entry of any subsequent token shall be locked out for a period of time after each such rejection.

A maximum lockout time of approximately 60s to 120 s shall be reached within at least 10 successive rejections.

A recommended method is for the lockout time to start with a small value and then progressively increase with each successive rejection. This approach is preferred, as it is more tolerant to clients that experience difficulty with token entry under poor communications conditions.

Other methods may be implemented, but the maximum lockout time shall be approximately 60 s to 120 s.

The lockout condition shall be cleared and the token entry process shall restore to normal operation upon the first acceptance of a Class 0 or a Class 2 token in terms of the conditions given in 8.2 of IEC 62055-41.

The remaining lockout time value in seconds shall be made available in the TokenLockoutTimeRemaining register (see 6.8.3.8) that may be read by means of a ReadCommand message with the appropriate registerID value set in the RID field.

6.7 Server timing requirements

6.7.1 Inter-message and inter-character timing

The inter-message timing responses are shown in Figure 4.

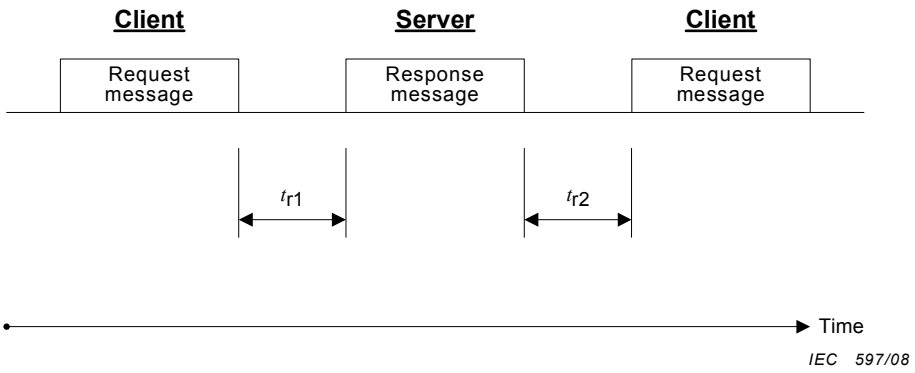


Figure 4 – Inter-message timing responses

The server shall respond to a request message within the time constraints given for t_{r1} in Table 10.

The server shall be ready to receive a request message after sending the last response message within the time constraints given for t_{r2} in Table 10.

Table 10 – Server timing requirements to respond to a request message

t_{r1}	t_{r2}
20 ms ≤ t_{r1} ≤ 1 500 ms	20 ms ≤ t_{r2} ≤ 1 500 ms

If the server does not respond to a request within the inter-message timing requirements as given in Table 10, then the client may assume that an error has occurred.

The time between two characters in any character sequence shall be within the time constraints given for t_a in Table 11.

Table 11 – Inter-character timing requirements

t_a
t_a ≤ 1 500 ms

If the server detects that the inter-character timing exceeds the limits given in Table 11, then this shall constitute a transmission error as defined in 6.7.2.

6.7.2 Transmission error timing

A transmission error occurs at the server when it detects an error in the message while it is still busy receiving it from the client and constitutes any one of the following conditions:

- the value of the parity bit in the transmission format is not in accordance with 6.3.2 in any of the received characters; this condition results in the ParityError code being set in the ServerStatus register;
- the maximum time allowed between characters has been exceeded in accordance with the values given in Table 11; this condition results in the CharacterTimeoutError code being set in the ServerStatus register;
- the client has sent more characters than the server could receive; this condition results in the CharacterOverflowError code being set in the ServerStatus register;

- the received message syntax and structure does not comply with the definitions as given in 6.4; this condition results in the MessageSyntaxError code being set in the ServerStatus register;
- the BCC field in the message (see 6.4) does not match the calculated value from the received message; this condition results in the BCCError code being set in the ServerStatus register;
- an error that is not defined in this standard has occurred during the receiving phase of the message; this condition results in the UndefinedTransmissionError code being set in the ServerStatus register.

The values for the error codes in the ServerStatus register are given in Table 20.

The timing requirements for the detection of a transmission error are given in Figure 5.

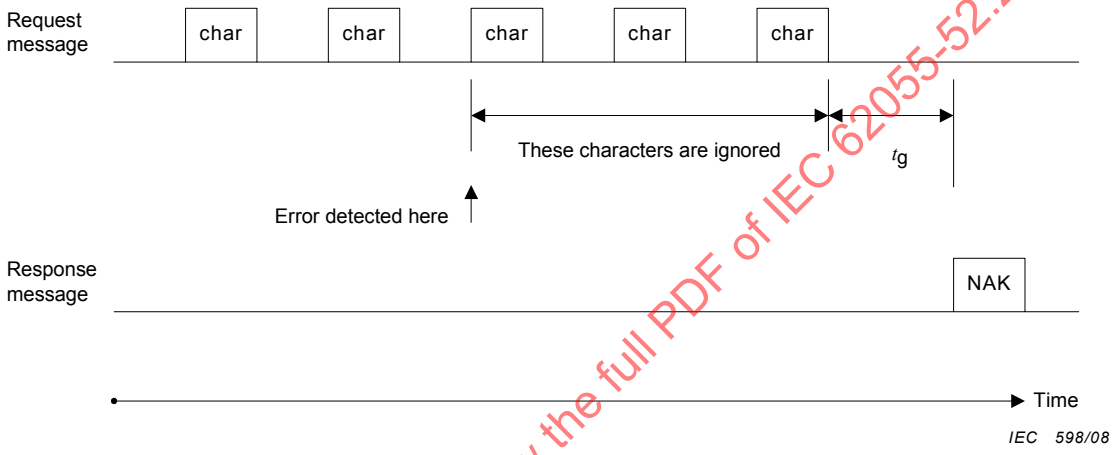


Figure 5 –Transmission error timing

If the server detects a transmission error during receipt of a request message, it shall ignore the rest of the message.

The server shall then wait for a period equal to t_g given in Table 12, during which no characters are received before transmitting a NAK response.

Table 12 – Transmission error recovery wait period

t_g
1500 ms

6.7.3 Message execution timing

The message execution timing is not specified in this standard, but the server shall be ready to receive a new request message within the timing constraints imposed by 6.7.1 and 6.7.2.

In the case where the server needs to exceed the constraints imposed by 6.7.1 and 6.7.2 in order to execute the message, it may make use of the RegisterBusy code in the ServerStatus register (see 6.8.3.4). See also 6.6.3 to 6.6.4 for an example of how the RegisterBusy status code may be utilized.

6.8 RegisterTable

6.8.1 RegisterTable interface class

The generic format for RegisterTable is given in Table 13.

Table 13 – Generic format for RegisterTable

Attributes	Range	Context
name	**{RegisterTableName}**	The registered name of this FunctionObject in the companion specification.
tableID	FOIN	FunctionObjectIdentificationNumber as registered in the companion specification
Data elements		**External data interface to the FunctionObject instance** **(from the TokenCarrier perspective)**
registerArray	Array of simple or complex registers; quantity between 1 – 65536 Data elements are defined per Register instance (see 6.8.2)	Array of Register instances defined in the companion specification. There may be up to 65536 entries in the table. Each Register instance may have a simple or complex dataset structure
Methods		**External service interface to the FunctionObject instance** **(from the TokenCarrier perspective)**
none	Methods are defined per Register instance (see 6.8.2) The RegisterTable has no operational methods	The table is virtual and cannot be accessed
Operation		**Internal functionality of the RegisterTable**
none	Operations are defined per Register instance (see 6.8.2) The RegisterTable has no operational functionality	The table merely defines a list of Register definitions, each of which provides a logical interface to the actual payment meter registers
Association		**Support services provided by other FunctionObjects**
none	Associations are defined per Register instance (see 6.8.2)	The RegisterTable does not make use of other specified services for it's functioning

This interface class provides the generic format for defining a RegisterTable as an instance of a MeterFunctionObject as defined in a companion specification (see 6.9).

Each instance of a RegisterTable, together with each Register, is defined in a companion specification.

Each RegisterTable is uniquely identified by means of its FOIN as allocated by the STS Association and as registered in a companion specification.

Each Register instance (see 6.8.2) defined in the RegisterTable provides a logical interface to an actual register or a function in the payment meter.

6.8.2 Register interface class

The generic format for Register is given in Table 14.

Table 14 – Generic format for Register

Attributes	Range	Context
registerName	**{RegisterName}**	The name of the Register instance
registerID	0000 – FFFF hex	Identification number of the Register instance as entered into the RegisterTable
format	Register instance-specific	Presentation format of the Data Element to the Register interface
readWrite	R, W	Read and Write permission granted to a client accessing the Register at the interface
confidentiality	True, False	If = True, the dataset passing across the n interface is to be treated as confidential and may not be publicly displayed. See also Table 7 of IEC 62055-21.
authentication	True, False	If = True, the client must authenticate itself to the server before permission is granted to access the register
Data elements		**External data interface to the Register instance** **(from the TokenCarrier perspective)**
dataset	Register instance-specific; simple or complex structure	Actual data that is passed across the Register interface and that a client may access by means of a ReadCommand or a WriteCommand
Methods		**External service interface to the Register instance** **(from the TokenCarrier perspective)**
ReadCommand (RID, DL)	Returns the value of the dataset data element RID is the registerID of the Register from which the dataset must be read DL dictates which part of the dataset should be returned (this is Register instance –specific)	The client may retrieve data from the payment meter via the Register interface by sending a ReadCommand message to the Register.
WriteCommand (RID, D)	Loads the D value into the dataset data element RID is the registerID of the Register to which the dataset must be written D is the dataset to be written	The client may load data into the payment meter via the Register interface by sending a WriteCommand message to the Register
Operation		
Virtual recording	Keeps virtual record of the dataset data element	This object defines a virtual Register, which provides a logical interface to the actual payment meter register or function
Association	**Internal functionality of the Register**	**Support services provided by other FunctionObjects**
x	Register instance-specific	The register may make use of other specified services for it's functioning

This interface class provides the generic format for defining a Register instance within a RegisterTable (see 6.8.1).

Each instance of a Register is defined within a RegisterTable instance (see 6.8.1) and provides a logical interface to an actual register or a function in the payment meter.

Each Register instance is uniquely identified by means of its registerID within a RegisterTable instance and in this way each Register instance definition can be globally identified by a combination of the FOIN plus the registerID.

Each such Register instance definition is specified together with the RegisterTable instance definition in a companion specification (see 6.9).

Proprietary Register definitions are allowed in a RegisterTable, but such defined registerID values shall remain entirely under the management and control of the payment meter manufacturer.

A client may use a ReadCommand message (see 6.4.4) addressed to a particular registerID to retrieve a dataset from the payment meter via the Register interface.

A client may use a WriteCommand message (see 6.4.5) addressed to a particular registerID to load a dataset into the payment meter via the Register interface.

Reading of DispenserKey values in any form or by any means from the payment meter shall not be permitted.

Changing the values of STS-defined data elements in the payment meter by any means other than STS-defined processes shall not be permitted.

6.8.3 Predefined Registers and registerID values

6.8.3.1 General requirements

Where the confidentiality attribute of a Register instance is defined as = True, then the content passing over such Register interface shall be protected against unauthorised access by appropriate techniques (see also Table 7 of IEC 62055-21).

All RegisterTable instances shall, as a minimum, provide for the specific Register instances and registerID values as given in Table 15.

All Register instances shall update the ServerStatus register (see Annex A), except for the ServerStatus register, which shall not update itself.

The processing of the BinaryTokenEntry register, or any other functionally equivalent register defined in a companion specification, shall also update the TokenStatus register.

Table 15 – Predefined Registers and registerID values

registerID	registerName	Reference
2000 hex	ProtocolVersion	6.8.3.2
2001 hex	TableID	6.8.3.3
2002 hex	ServerStatus	6.8.3.4
(see NOTE)	SoftwareVersion	6.8.3.5
(see NOTE)	BinaryTokenEntry	6.8.3.6
(see NOTE)	TokenStatus	6.8.3.7
(see NOTE)	TokenLockoutTimeRemaining	6.8.3.8
NOTE The registerID values for these registers are defined in the corresponding companion specification for a RegisterTable instance.		

6.8.3.2 ProtocolVersion register

The instance format for ProtocolVersion register is given in Table 16.

Table 16 – Instance format for ProtocolVersion register

Attributes	Range	Context
name	**ProtocolVersion**	Register name in this table
registerID	2000 hex	Register identifier in this table
format	8-bit binary	Presentation format to the interface
readWrite	R	Read only
confidentiality	False	The content of this register is not confidential.
authentication	False	Client authentication not required
Data elements		**External data interface to the Register instance (from the TokenCarrier perspective)**
protocolVersion	8-bit binary value; range 0 – 255 As defined in Table 17	Actual data that is passed across the Register interface and that a client may access by means of a ReadCommand
Methods		**External service interface to the Register instance (from the TokenCarrier perspective)**
ReadCommand (RID, DL)	Returns the value of the protocolVersion data element RID = registerID of Register to be read DL = ignored by server	The data may be retrieved from the payment meter via the Register interface by sending a ReadCommand message to the Register with the registerID value in the RID field. The DL field is ignored
Operation		**Internal functionality of the Register**
Virtual recording	Keeps virtual record of the value of the protocolVersion data element	This object defines a virtual Register, which provides a logical interface to the actual payment meter register or function
Association		**Support services provided by other FunctionObjects**
none	x	The register does not make use of other specified services for its functioning

This Register serves to maintain compatibility between publications of IEC 62055-52 and publications of companion specifications for RegisterTable instances and shall always be assigned to a registerID value as given in Table 15.

All companion specifications for RegisterTable instances shall include a ProtocolVersion register instance as given here and allocate one of the defined values given in Table 17.

A new value shall be added to Table 17 with every revised publication of IEC 62055-52 if it introduces a change in functionality or specification to the physical layer protocol.

Any protocol change in IEC 62055-52 shall ensure that backward compatibility is maintained, so that a client is always capable of reading from this register in exactly the same way as before. This will ensure that a client is always able to determine which protocol version the payment meter is capable of supporting.

Table 17 – Defined protocolVersion values

Value	Context
0	Reserved for future assignment
1	For legacy payment meters that have not implemented the version 2 or higher server protocol. A client that wants to exchange data with these payment meters requires a proprietary manufacturer-specific RegisterTable
2	For payment meters that implement the server protocol specified in this version of IEC 62055-52. A client that wants to exchange data with these payment meters requires the relevant companion specification
3 – 255	Reserved for future assignment
NOTE Legacy payment meters that have implemented the version 1 protocol, will return a NAK to a client if it tries to read data from the ProtocolVersion register. By that response, the client is able to determine that it is dealing with a legacy payment meter.	

If the server responds with a NAK then it is an implied value of 1, in which case the client may then assume that it requires a proprietary manufacturer-specific RegisterTable and that it should use the version 1 protocol.

If the server responds with a value in the range 2 to 255, the client uses the relevant protocol version and RegisterTable instance defined in a companion specification. The server shall never respond with a value of 0 or 1.

6.8.3.3 TableID register

The instance format for TableID register is given in Table 18.

Table 18 – Instance format for TableID register

Attributes	Range	Context
name	**TableID**	Register name in this table
registerID	2001 hex	Register identifier in this table
format	22-bit binary	Presentation format to the interface
readWrite	R	Read only
confidentiality	False	The content of this register is not confidential.
authentication	False	Client authentication not required
Data elements		**External data interface to the Register instance (from the TokenCarrier perspective)**
tableID	22-bit binary FOIN as defined by the STS Association	Actual data that is passed across the Register interface and that a client may access by means of a ReadCommand
Methods		**External service interface to the Register instance (from the TokenCarrier perspective)**
ReadCommand (RID, DL)	Returns the value of the tableID data element RID = registerID of Register to be read DL = ignored by server	The data may be retrieved from the payment meter via the Register interface by sending a ReadCommand message to the Register with the registerID value in the RID field. The DL field is ignored
Operation		**Internal functionality of the Register**
Virtual recording	Keeps virtual record of the value of the tableID data element	This object defines a virtual Register, which provides a logical interface to the actual payment meter register or function

Attributes	Range	Context
Association		**Support services provided by other FunctionObjects**
none	x	The register does not make use of other specified services for its functioning

NOTE Legacy payment meters that have not implemented this IEC 62055-52 standard, will return a NAK to a client if it tries to read data from the TableID register. By that response, the client is able to determine that it is dealing with a legacy payment meter and thus a proprietary RegisterTable is required from the manufacturer.

This Register serves to maintain compatibility between publications of IEC 62055-52 and publications of companion specifications for RegisterTable instances and shall always be assigned a fixed registerID value as given in Table 15.

All companion specifications for RegisterTable instances shall include a TableID register instance as given here and shall define a value equal to the FOIN as allocated by the STS Association in the companion specification for the RegisterTable instance that is implemented in the particular payment meter. See also Clause 7 for more information on the maintenance of the FOIN values.

The value of the TableID register shall always be set to the FOIN.

If new registers are added to a RegisterTable, then the FOIN shall also change.

Any protocol change in IEC 62055-52 shall ensure that backward compatibility is maintained, so that a client is always capable of reading from this register in exactly the same way as before. This will ensure that a client is always able to determine which RegisterTable version the payment meter has implemented.

If the server responds with a NAK then it is an implied invalid FOIN value, in which case the client may then assume that it requires a proprietary manufacturer-specific RegisterTable.

If the server responds with a valid FOIN value, the client may use the relevant RegisterTable instance defined in a companion specification. The server shall never respond with an invalid FOIN value.

6.8.3.4 ServerStatus register

The instance format for ServerStatus register is given in Table 19.

Table 19 – Instance format for ServerStatus register

Attributes	Range	Context
registerName	**ServerStatus**	Register name in this table
registerID	2002 hex	Register identifier in this table
format	8-bit binary	Presentation format to the interface
readWrite	R	Read only
confidentiality	False	The content of this register is not confidential.
authentication	False	Client authentication not required
Data elements		**External data interface to the Register instance** **(from the TokenCarrier perspective)**

Attributes	Range	Context
serverStatus	8-bit binary value; range 0 -255 as defined in Table 20	Actual data that is passed across the Register interface and that a client may access by means of a ReadCommand Indicates the state of the server with respect to processing of a message in the physical layer protocol
Methods ReadCommand (RID, DL)	Returns the value of the serverStatus data element RID = registerID of Register to be read DL = ignored by server	**External service interface to the Register instance** **(from the TokenCarrier perspective)** The data may be retrieved from the payment meter via the Register interface by sending a ReadCommand message to the Register with the registerID value in the RID field. The DL field is ignored
Operation Virtual recording	Keeps virtual record of the value of the serverStatus data element	**Internal functionality of the Register** This object defines a virtual Register, which provides a logical interface to the actual payment meter register or function. Keeps virtual record of the state of the server physical layer protocol of the payment meter
Association none	x	**Support services provided by other FunctionObjects** The register does not make use of other specified services for its functioning

Table 20 – Defined serverStatus values

Value	Status name	Context	Reference
0	Reserved	Reserved for future assignment	
1	ParityError	A parity error was detected in a received character	6.7.2
2	CharacterTimeoutError	The client is sending characters too slowly	6.7.2
3	CharacterOverflowError	The client is sending more characters than the server can receive	6.7.2
4	MessageSyntaxError	The structure of the message is not in accordance with one of the defined request messages (see 6.4.2, 6.4.4, 6.4.5 and 6.4.6)	6.7.2
5	BCCError	The BCC field does not match the calculated value from the message received	6.7.2
6	UndefinedTransmissionError	Any other error condition that is not defined in this RegisterTable, which relates to the receiving of the message	6.7.2
7	RegisterIDInvalid	The value of the RID field in the message contains a registerID that is not supported in the payment meter	6.6.3 6.6.4
8	RegisterBusy	The Register that is being written to of being read from is currently busy and cannot respond to the request message at this point in time. The client should wait and try later	6.6.3 6.6.4
9	RegisterWriteProtected	The Register W attribute is not defined as = True. A value cannot be written to it	6.6.4
10	RegisterReadProtected	The Register R attribute is not defined as = True. A value cannot be read from it	6.6.3

Value	Status name	Context	Reference
11	FunctionDisabled	The Register is supported, but the function that it interfaces to has been disabled. For example: the MaximumPowerLimit function in the payment meter might be present, but it may be in a disabled state. In this case, a read command to the register will return the value as set in the payment meter, but will also indicate the status value = 11 in the ServerStatus register	6.6.3 6.6.4
12	TokenLockout	Token lockout is active. This only applies in the case of a write command to BinaryTokenEntry register or a write command to any other functionally equivalent register defined in a companion specification	6.6.7
13	UndefinedReadingError	Any other error condition that is not defined in this table, which relates to the execution of the ReadCommand message	6.6.3
14	UndefinedWritingError	Any other error condition that is not defined in this table, which relates to the execution of the WriteCommand message	6.6.4
15	CommandExecuted	The message has been successfully delivered to the destination Register, where it will be further processed	6.6.2 to 6.6.5
16 – 255	Reserved	Reserved for future assignment	

6.8.3.5 SoftwareVersion register

The instance format for SoftwareVersion register is given in Table 21.

Table 21 – Instance format for SoftwareVersion register

Attributes	Range	Context
registerName	**SoftwareVersion**	Register name in this table
registerID	0000 – 1FFF and 2003 – FFFF hex	Register identifier in this table
format	4 hex digits, 0 – 9 and A – F	Presentation format to the interface
readWrite	R	Read only
confidentiality	False	The content of this register is not confidential.
authentication	False	Client authentication not required
Data elements		**External data interface to the Register instance** **(from the TokenCarrier perspective)**
softwareVersion	4-digit software version as defined by the manufacturer	Actual data that is passed across the Register interface and that a client may access by means of a ReadCommand
Methods		**External service interface to the Register instance** **(from the TokenCarrier perspective)**
ReadCommand (RID, DL)	Returns the value of the softwareVersion data element RID = registerID of Register to be read DL = ignored by server	The data may be retrieved from the payment meter via the Register interface by sending a ReadCommand message to the Register with the registerID value in the RID field. The DL field is ignored
Operation		**Internal functionality of the Register**
Virtual recording	Keeps virtual record of the value of the softwareVersion data element	This object defines a virtual Register, which provides a logical interface to the actual payment meter register or function

Attributes	Range	Context
Association		**Support services provided by other FunctionObjects**
none	x	The register does not make use of other specified services for its functioning

This Register serves to identify the version of software that is implemented in the payment meter and shall be assigned to a registerID value as given in a particular RegisterTable instance defined in a companion specification.

All companion specifications for RegisterTable instances shall include a softwareVersion register instance as given here.

Administration of softwareVersion register values and interpretation of the application data is manufacturer-specific and may be formatted as decimal or hexadecimal digits.

This is the same value that returns in the IDResponse message (see 6.4.3).

This value may also be used as authentication data for messages operating on proprietary registers that are defined in a RegisterTable instance.

6.8.3.6 BinaryTokenEntry register

The instance format for BinaryTokenEntry register is given in Table 22.

Table 22 – Instance format for BinaryTokenEntry register

Attributes	Range	Context
registerName	**BinaryTokenEntry**	Register name in this table
registerID	0000 – 1FFF and 2003 – FFFF hex	Register identifier in this table
format	66-bit binary	Presentation format to the interface
readWrite	W	Write only
confidentiality	False	The content of this register is not confidential.
authentication	False	Client authentication not required
Data elements		**External data interface to the Register instance** **(from the TokenCarrier perspective)**
binaryTokenEntry	66-bit binary Defined in 6.1.2 as TokenData in the TCDU	Actual data that is passed across the Register interface and that a client may access by means of a WriteCommand
Methods		**External service interface to the Register instance** **(from the TokenCarrier perspective)**
WriteCommand (RID, D)	Loads the 66-bit binary number into the binaryTokenEntry data element. RID = registerID of Register to be written D = 66-bit binary number	The data may be loaded into the payment meter via the Register interface by sending a WriteCommand message to the Register with the registerID value in the RID field and the dataset in the D field
Operation		**Internal functionality of the Register**
Virtual token entry	Provides a virtual point of entry for a token to the payment meter as another token carrier interface	This object defines a virtual Register, which provides a logical interface to the actual payment meter register or function

Attributes	Range	Context
Association		**Support services provided by other FunctionObjects**
none	x	The register does not make use of other specified services for its functioning

This Register serves to provide a means of loading a token into the payment meter via the virtual token carrier and shall be assigned to a registerID value as given in a particular RegisterTable instance defined in a companion specification.

All companion specifications for RegisterTable instances shall include a binaryTokenEntry register instance as given here.

The client may load a token into the payment meter by sending a WriteCommand message (see 6.4.5) with the 66-bit binary TokenData (see 6.1.2) in the D field and the registerID value for the BinaryTokenEntry register in the RID field of the message.

The received 66-bit value is then transferred to the TokenData field of the TCDU, which is further processed by the application layer protocol (see 7.2 of IEC 62055-41) and also by the MeterApplicationProcess (see Clause 8 of IEC 62055-41), where it is executed. The result is then returned in the TokenResult field of the TCDU (see also 6.8.3.7 for the reading of the result from the TokenStatus register).

At the same time when the 66-bit binary number is transferred to the TCDU, the TokenStatusNotReady code (see Table 24) shall be set in the TokenStatus register.

The requirements for token acceptance and rejection are described in 8.2 of IEC 62055-41, while the requirements for indication of token processing results are described in 8.3 of IEC 62055-41.

6.8.3.7 TokenStatus register

The instance format for TokenStatus register is given in Table 23.

Table 23 – Instance format for TokenStatus register

Attributes	Range	Context
registerName	**TokenStatus**	Register name in this table
registerID	0000 – 1FFF and 2003 – FFFF hex	Register identifier in this table
format	8-bit binary	Presentation format to the interface
readWrite	R	Read only
confidentiality	False	The content of this register is not confidential.
authentication	False	Client authentication not required
Data elements		**External data interface to the Register instance** **(from the TokenCarrier perspective)**
tokenStatus	8-bit binary value; range 0 – 255 As defined in Table 24	Actual data that is passed across the Register interface and that a client may access by means of a ReadCommand. Indicates the result after processing of a token that was entered by a client via the VTC07 interface

Attributes	Range	Context
Methods ReadCommand (RID, DL)	 Returns the value of the tokenStatus data element RID = registerID of Register to be read DL = ignored by server	**External service interface to the Register instance** **(from the TokenCarrier perspective)** The data may be retrieved from the payment meter via the Register interface by sending a ReadCommand message to the Register with the registerID value in the RID field. The DL field is ignored
Operation Virtual recording	 Keeps virtual record of the value of the tokenStatus data element	**Internal functionality of the Register** This object defines a virtual Register, which provides a logical interface to the actual payment meter register or function. Keeps virtual record of the result of the processing of a token entered via the VTC07 connection
Association none	 x	**Support services provided by other FunctionObjects** The register does not make use of other specified services for its functioning

This Register serves to provide feedback to the client with respect to the status of a token that is being processed by the payment meter after being entered via the virtual token carrier VTC07 and shall be assigned to a registerID value as given in a particular RegisterTable instance defined in a companion specification.

All companion specifications for RegisterTable instances shall include a TokenStatus register instance as given here.

After the MeterApplicationProcess has executed the instruction on a token that was entered by means of a WriteCommand message, the application layer protocol returns the result in the TCDU.

The TokenStatus register shall be updated as soon as the result has been returned in the TCDU.

The client may then read the result by sending a ReadCommand message with the registerID value for the TokenStatus register in the RID field.

The tokenStatus value is calculated from the ValidationResult, AuthenticationResult, TokenResult and TokenLockoutStatus data fields in the TCDU.

The defined values for tokenStatus are given in Table 24.

Table 24 – Defined tokenStatus values

Value	Context	Reference
0	Reserved for future assignment	
1	Accept Token is accepted The Accept attribute in TokenResult in the TCDU is True	6.1.5
2	1stKCT This is a Set1stSectionDecoderKey token that has been entered The 1stKCT attribute in TokenResult in the TCDU is True	6.1.5

Value	Context	Reference
3	2ndKCT This is a Set2ndSectionDecoderKey token that has been entered. The 2ndKCT attribute in TokenResult in the TCDU is True.	6.1.5
4	OverflowError Acceptance of this token would cause a register in the payment meter to overflow. For example: the available credit in the accounting register or the power limit register. The OverflowError attribute in TokenResult in the TCDU is True.	6.1.5
5	KeyTypeError Indicates that an attempt is being made to change the DecoderKey from one key type to another which is in violation of the key change rules. The KeyTypeError attribute in TokenResult in the TCDU is True	6.1.5
6	FormatError One or more data elements in the token does not comply with the required format for that element The FormatError attribute in TokenResult in the TCDU is True	
7	RangeError One or more data elements in the token have a value that is outside of the defined range of values defined in the application for that element For example A data parameter value on the token is larger than the meter application process function can handle The RangeError attribute in TokenResult in the TCDU is True	6.1.5
8	FunctionError Function is not implemented in the meter application process For example: ClearCredit token is being entered for a water register, but payment meter has only implemented an electricity credit register. Token has a SubClass value that is not supported by the meter application process. The FunctionError attribute in TokenResult in the TCDU is True	6.1.5
9	OldError Token is old (expired) The OldError attribute in ValidationResult in the TCDU is True	6.1.4
10	UsedError Token has already been used (duplicate) The UsedError attribute in ValidationResult in the TCDU is True	6.1.4
11	KeyExpiredError Meter key has expired The KeyExpiredError attribute in ValidationResult in the TCDU is True	6.1.4
12	DDTKError The Decoder has a DDTK value in the DKR; a TransferCredit token may not be processed by the MeterApplicationProcess in accordance with the key type rules The DDTKError attribute in ValidationResult in the TCDU is True	6.1.4
13	CRCError The CRC value in the token is different to the CRC value as calculated from the data in the token The CRCError attribute in AuthenticationResult in the TCDU is True	6.1.3

Value	Context	Reference
14	MfrCodeError The MfrCode value in the Class 1 token does not match the MfrCode value for the Decoder The MfrCodeError attribute in AuthenticationResult in the TCDU is True	6.1.3
15	TokenLockoutStatus Token lockout is active The TokenLockoutStatus in the TokenLockout function is True	6.6.7
16	TokenStatusNotReady Indicates that processing of the last token entered via a virtual token carrier VTC07 Register (see 6.8.3.6 for example) has not been completed. The tokenStatus is only updated on completion and the client should wait and try again later to read the TokenStatus register	6.8.3.6
17 – 255	Reserved for future assignment	

NOTE The sequence in which the payment meter will perform the tests and will detect the errors are in the same order as 6.1.3, 6.1.4 and then 6.1.5. From Figure 20 in 7.2.1 of IEC 62055-41, it can be seen that AuthenticationResult appears first, then ValidationResult and lastly, TokenResult. Depending on the existing status, the test for TokenLockoutStatus in 6.6.7 may appear first or last.

The requirements for token acceptance and rejection are described in 8.2 of IEC 62055-41, while the requirements for indication of token processing results are described in 8.3 of IEC 62055-41.

6.8.3.8 TokenLockoutTimeRemaining register

The instance format for TokenLockoutTimeRemaining register is given in Table 25.

Table 25 – Instance format for TokenLockoutTimeRemaining register

Attributes	Range	Context
registerName	**TokenLockoutTimeRemaining**	Register name in this table
registerID	0000 – 1FFF and 2003 – FFFF hex	Register identifier in this table
format	16-bit binary	Presentation format to the interface
readWrite	R	Read only
confidentiality	False	The content of this register is not confidential.
authentication	False	Client authentication not required
Data elements		**External data interface to the Register instance** **(from the TokenCarrier perspective)**
tokenLockoutTimeRemaining	16-bit binary value; 0 – 65535 Number of seconds	Actual data that is passed across the Register interface and that a client may access by means of a ReadCommand Indicates the number of seconds remaining if the token lockout is active (see 6.6.7)
Methods		**External service interface to the Register instance** **(from the TokenCarrier perspective)**
ReadCommand (RID, DL)	Returns the value of the tokenLockoutTimeRemaining Data Element RID = registerID of Register to be read DL = ignored by server	The data may be retrieved from the payment meter via the Register interface by sending a ReadCommand message to the Register with the registerID value in the RID field. The DL field is ignored
Operation		**Internal functionality of the Register**