PUBLICLY
AVAILABLE
SPECIFICATION

IEC
PAS 62030

Pre-Standard

First edition
2004-11

Digital data communications for measurement and control – Fieldbus for use in industrial control systems –

Section 1: MODBUS® Application Protocol Specification V1.1a –

Section 2: Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification Version 1.0

Reference number
IEC/PAS 62030:2004(E)

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site (www.iec.ch)**

- **Catalogue of IEC publications**

  The on-line catalogue on the IEC web site (www.iec.ch/searchpub) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.

- **IEC Just Published**

  This summary of recently issued publications (www.iec.ch/online_news/ justpub) is also available by email. Please contact the Customer Service Centre (see below) for further information.

- **Customer Service Centre**

  If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

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

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

PUBLICLY
AVAILABLE
SPECIFICATION

IEC
PAS 62030

Pre-Standard

First edition
2004-11

Digital data communications for measurement and control – Fieldbus for use in industrial control systems –

Section 1: MODBUS® Application Protocol Specification V1.1a –

Section 2: Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification Version 1.0

© IEC 2004 — Copyright - all rights reserved

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE XG

For price, see current catalogue

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

CONTENTS

FOREWORD.. 5

Section 1 – MODBUS® Application Protocol Specification V1.1a 7

1 MODBUS 7

1.1 Introduction 7

1.1.1 Scope of this section.. 7

1.1.2 Normative references... 8

1.2 Abbreviations 8

1.3 Context 8

1.4 General description 9

1.4.1 Protocol description 9

1.4.2 Data Encoding 11

1.4.3 MODBUS data model 12

1.4.4 MODBUS Addressing model...13

1.4.5 Define MODBUS Transaction 14

1.5 Function Code Categories 16

1.5.1 Public Function Code Definition...17

1.6 Function codes descripitons 17

1.6.1 01 (0x01) Read Coils 17

1.6.2 02 (0x02) Read Discrete Inputs 19

1.6.3 03 (0x03) Read Holding Registers...21

1.6.4 04 (0x04) Read Input Registers..22

1.6.5 05 (0x05) Write Single Coil...23

1.6.6 06 (0x06) Write Single Register..24

1.6.7 07 (0x07) Read Exception Status (Serial Line only) 26

1.6.8 08 (0x08) Diagnostics (Serial Line only) 27

1.6.9 11 (0x0B) Get Comm Event Counter (Serial Line only).............................30

1.6.10 12 (0x0C) Get Comm Event Log (Serial Line only) 32

1.6.11 15 (0x0F) Write Multiple Coils 34

1.6.12 16 (0x10) Write Multiple registers...35

1.6.13 17 (0x11) Report Slave ID (Serial Line only)...37

1.6.14 20 / 6 (0x14 / 0x06) Read File Record 37

1.6.15 21 / 6 (0x15 / 0x06) Write File Record 39

1.6.16 22 (0x16) Mask Write Register 41

1.6.17 23 (0x17) Read/Write Multiple registers...43

1.6.18 24 (0x18) Read FIFO Queue 45

1.6.19 43 (0x2B) Encapsulated Interface Transport...46

1.6.20 43 / 13 (0x2B / 0x0D) CANopen General Reference Request and Response PDU 47

1.6.21 43 / 14 (0x2B / 0x0E) Read Device Identification 48

1.7 MODBUS Exception Responses...52

Annex A of Section 1 (informative) MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE ..54

A.1 INTRODUCTION 54

A.1.1 OBJECTIVES 54

A.1.2 CLIENT / SERVER MODEL..54

--`,````,,,```,,,`,,,``,`-`-`,,`,,`,`,,`---

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.1.3 REFERENCE DOCUMENTS 55
A.2 ABBREVIATIONS 55
A.3 CONTEXT 55
A.3.1 PROTOCOL DESCRIPTION 55
A.3.2 MODBUS FUNCTIONS CODES DESCRIPTION 57
A.4 FUNCTIONAL DESCRIPTION 58
A.4.1 MODBUS COMPONENT ARCHITECTURE MODEL 58
A.4.2 TCP CONNECTION MANAGEMENT 61
A.4.3 USE of TCP/IP STACK 65
A.4.4 COMMUNICATION APPLICATION LAYER 71
A.5 IMPLEMENTATION GUIDELINE 82
A.5.1 OBJECT MODEL DIAGRAM 83
A.5.2 IMPLEMENTATION CLASS DIAGRAM 87
A.5.3 SEQUENCE DIAGRAMS 89
A.5.4 CLASSES AND METHODS DESCRIPTION 92
Annex B of Section 1 (Informative) MODBUS RESERVED FUNCTION CODES, SUBCODES AND MEI TYPES 96
Annex C of Section 1 (Informative) CANOPEN GENERAL REFERENCE COMMAND 96
Section 2 – Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification Version 1.0 97
2 RTPS 97
2.1 Basic Concepts 97
2.1.1 Introduction 97
2.1.2 The RTPS Object Model 98
2.1.3 The Basic RTPS Transport Interface 99
2.1.4 Notational Conventions 100
2.2 Structure Definitions 101
2.2.1 Referring to Objects: the GUID 101
2.2.2 Building Blocks of RTPS Messages 102
2.3 RTPS Message Format 105
2.3.1 Overall Structure of RTPS Messages 105
2.3.2 Submessage Structure 105
2.3.3 How to Interpret a Message 106
2.3.4 Header 107
2.3.5 ACK 108
2.3.6 GAP 109
2.3.7 HEARTBEAT 110
2.3.8 INFO_DST 112
2.3.9 INFO_REPLY 112
2.3.10 INFO_SRC 113
2.3.11 INFO_TS 114
2.3.12 ISSUE 114
2.3.13 PAD 115
2.3.14 VAR 116
2.3.15 Versioning and Extensibility 117
2.4 RTPS and UDP/IPv4 118
2.4.1 Concepts 118
2.4.2 RTPS Packet Addressing 118
2.4.3 Possible Destinations for Specific Submessages 121

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.5 Attributes of Objects and Metatraffic 122
2.5.1 Concept 122
2.5.2 Wire Format of the ParameterSequence 124
2.5.3 ParameterID Definitions 125
2.5.4 Reserved Objects 126
2.5.5 Examples 130
2.6 Publish-Subscribe Protocol 132
2.6.1 Publication and Subscription Objects 132
2.6.2 Representation of User Data 137
2.7 CST Protocol 139
2.7.1 Object Model 139
2.7.2 Structure of the Composite State (CS) 140
2.7.3 CSTWriter 140
2.7.4 CSTReader 145
2.7.5 Overview of Messages used by CST 147
2.8 Discovery with the CST Protocol 149
2.8.1 Overview 149
2.8.2 Managers Keep Track of Their Managees 150
2.8.3 Inter-Manager Protocol 150
2.8.4 The Registration Protocol 151
2.8.5 The Manager-Discovery Protocol 152
2.8.6 The Application Discovery Protocol 152
2.8.7 Services Discovery Protocol 153

Annex A of Section 2 (informative) CDR for RTPS 155

A.1 Primitive Types 155
A.1.1 Semantics 155
A.1.2 Encoding 155
A.1.3 octet 155
A.1.4 boolean 156
A.1.5 unsigned short 156
A.1.6 short 156
A.1.7 unsigned long 156
A.1.8 long 156
A.1.9 unsigned long long 156
A.1.10 long long 156
A.1.11 float157
A.1.12 double 157
A.1.13 char 157
A.1.14 wchar 157

A.2 Constructed Types 157
A.2.1 Alignment 157
A.2.2 Identifiers 157
A.2.3 List of constructed types 157
A.2.4 Struct 158
A.2.5 Enumeration 158
A.2.6 Sequence 158
A.2.7 Array 158
A.2.8 String 158
A.2.9 Wstring 159

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

DIGITAL DATA COMMUNICATIONS FOR MEASUREMENT AND CONTROL – FIELDBUS FOR USE IN INDUSTRIAL CONTROL SYSTEMS –

Section 1: MODBUS®* Application Protocol Specification V1.1a – Section 2: Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification Version 1.0

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.

A PAS is a technical specification not fulfilling the requirements for a standard but made available to the public .

IEC-PAS 62030 has been processed by subcommittee 65C: Digital communications, of IEC technical committee 65: Industrial-process measurement and control.

The text of this PAS is based on the following document: | This PAS was approved for publication by the P-members of the committee concerned as indicated in the following document

Draft PAS	Report on voting
65C/341A/NP	65C/347/RVN

Following publication of this PAS, which is a pre-standard publication, the technical committee or subcommittee concerned will transform it into an International Standard.

* MODBUS is a trademark of Schneider Automation Inc.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

It is foreseen that, at a later date, the content of this PAS will be incorporated in the future new edition of the IEC 61158 series according to its structure.

This PAS shall remain valid for an initial maximum period of three years starting from 2004-11. The validity may be extended for a single three-year period, following which it shall be revised to become another type of normative document or shall be withdrawn.

--`,````,,,```,,,`,,,``,``-`-`,,`,,`,`,,`---
Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Overview

This PAS has been divided into two sections. Section 1 deals with MODBUS® Application Protocol Specification V1.1a while Section 2 covers the Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification Version 1.0.

It is intended that the content of this PAS will be incorporated in the future new editions of the various parts of IEC 61158 series according to the structure of this series.

Section 1 – MODBUS® Application Protocol Specification V1.1a

1 MODBUS

1.1 Introduction

1.1.1 Scope of this section

MODBUS is an application layer messaging protocol, positioned at level 7 of the OSI model, that provides client/server communication between devices connected on different types of buses or networks.

The industry's serial de facto standard since 1979, MODBUS continues to enable millions of automation devices to communicate. Today, support for the simple and elegant structure of MODBUS continues to grow. The Internet community can access MODBUS at a reserved system port 502 on the TCP/IP stack.

MODBUS is a request/reply protocol and offers services specified by **function codes**. MODBUS function codes are elements of MODBUS request/reply PDUs. The objective of this PAS is to describe the function codes used within the framework of MODBUS transactions.

MODBUS is an application layer messaging protocol for client/server communication between devices connected on different types of buses or networks.

It is currently implemented using:

- TCP/IP over Ethernet. See Annex A of Section 1: MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE.
- Asynchronous serial transmission over a variety of media (wire : EIA/TIA-232-E, EIA-422-A, EIA/TIA-485-A; fiber, radio, etc.)
- MODBUS PLUS, a high speed token passing network.

NOTE The "Specification" is Clause 1 of this PAS.

NOTE MODBUS Plus is not in this PAS.

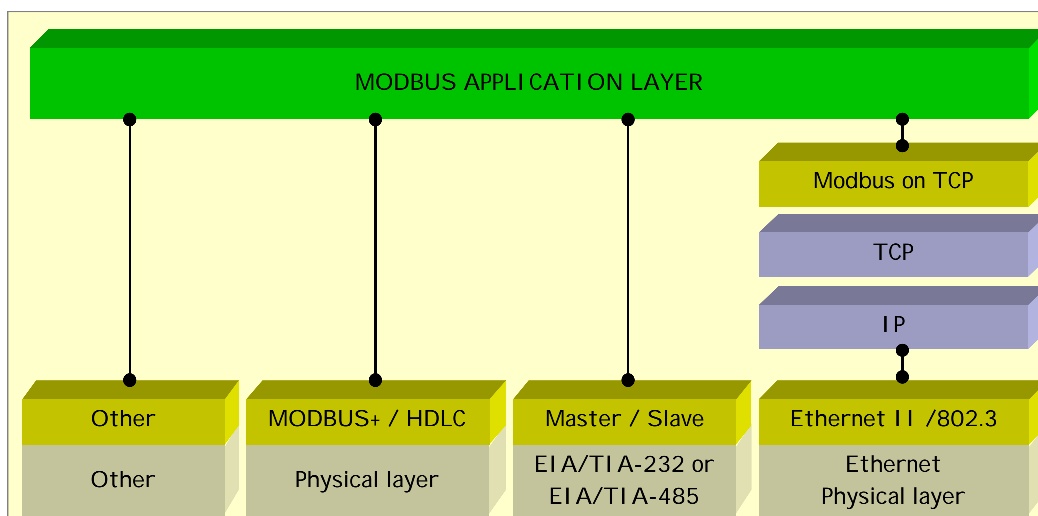


Figure 1 – MODBUS communication stack

This Figure 1 represents conceptually the MODBUS communication stack.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.1.2 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 61131 (all parts): Programmable controllers

EIA*/TIA**-232-E: *Interface between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary data Interchange*

EIA-422-A: *Electrical Characteristics-Balanced Voltage Digital Interface Circuit*

EIA/TIA-485-A: *Electrical Characteristics of Generators and Receivers for Use in balanced Digital Multipoint Systems*

RFC 791, *Interne Protocol*, Sep81 DARPA

1.2 Abbreviations

ADU Application Data Unit

HDLC High level Data Link Control

HMI Human Machine Interface

IETF Internet Engineering Task Force

I/O Input/Output

IP Internet Protocol

MAC Medium Access Control

MB MODBUS Protocol

MBAP MODBUS Application Protocol

PDU Protocol Data Unit

PLC Programmable Logic Controller

TCP Transport Control Protocol

1.3 Context

The MODBUS protocol allows an easy communication within all types of network architectures.

---

* EIA: Electronic Industries Alliance.

** TIA: Telecomunication Industry Association.

--`,````,,,```,,,`,,,``,`-`-`,,`,,`,`,,`---

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

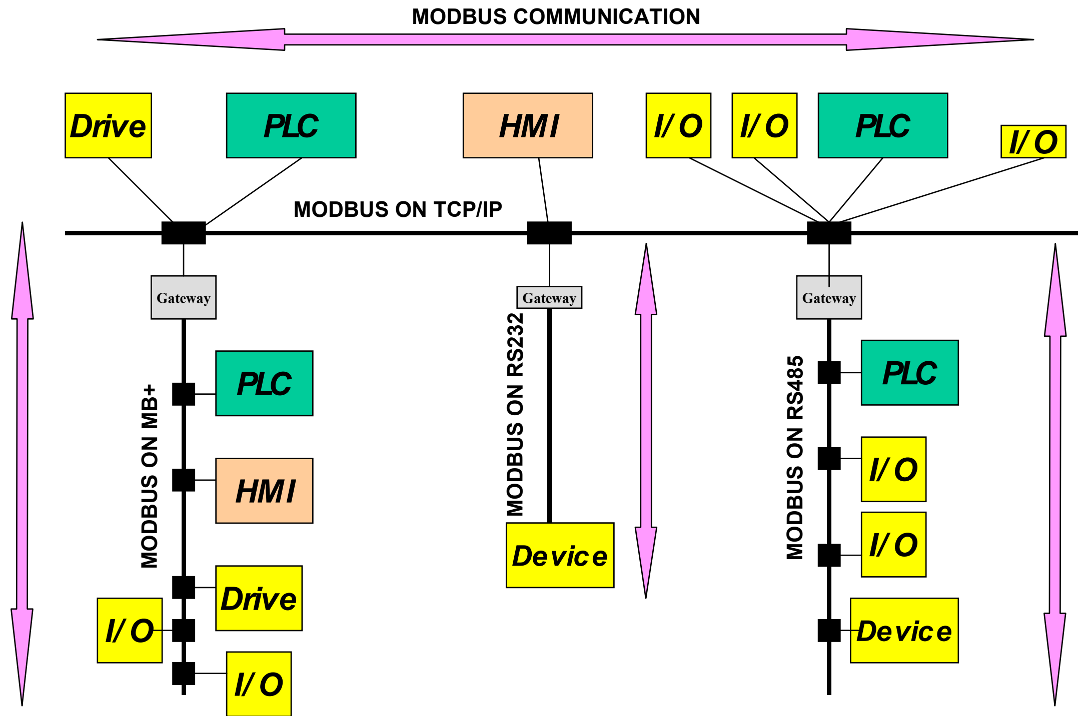


Figure 2 – Example of MODBUS Network Architecture

Every type of devices (PLC, HMI, Control Panel, Driver, Motion control, I/O Device…) can use MODBUS protocol to initiate a remote operation.

The same communication can be done as well on serial line as on an Ethernet TCP/IP networks. Gateways allow a communication between several types of buses or network using the MODBUS protocol.

1.4 General description

1.4.1 Protocol description

The MODBUS protocol defines a simple protocol data unit **(PDU)** independent of the underlying communication layers. The mapping of MODBUS protocol on specific buses or network can introduce some additional fields on the application data unit **(ADU)**.

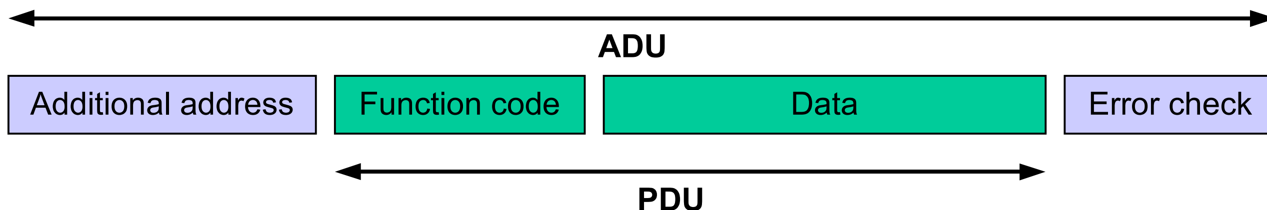


Figure 3 – General MODBUS frame

The MODBUS application data unit is built by the client that initiates a MODBUS transaction. The function indicates to the server what kind of action to perform. The MODBUS application protocol establishes the format of a request initiated by a client.

The function code field of a MODBUS data unit is coded in one byte. Valid codes are in the range of 1 ... 255 decimal (128 – 255 reserved for exception responses). When a message is sent from a Client to a Server device the function code field tells the server what kind of action to perform. Function code "0" is not valid.

Sub-function codes are added to some function codes to define multiple actions.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The data field of messages sent from a client to server devices contains additional information that the server uses to take the action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field.

The data field may be nonexistent (of zero length) in certain kinds of requests, in this case the server does not require any additional information. The function code alone specifies the action.

If no error occurs related to the MODBUS function requested in a properly received MODBUS ADU the data field of a response from a server to a client contains the data requested. If an error related to the MODBUS function requested occurs, the field contains an exception code that the server application can use to determine the next action to be taken.

For example a client can read the ON / OFF states of a group of discrete outputs or inputs or it can read/write the data contents of a group of registers.

When the server responds to the client, it uses the function code field to indicate either a normal (error-free) response or that some kind of error occurred (called an exception response). For a normal response, the server simply echoes to the request the original function code.

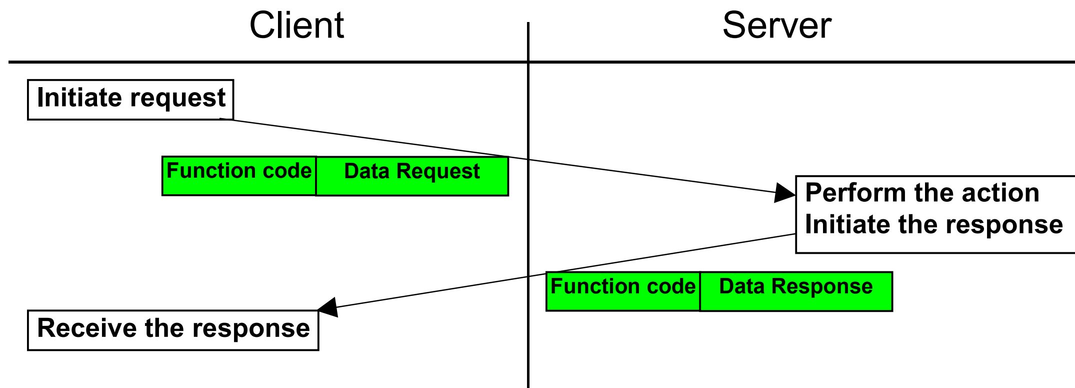


Figure 4 – MODBUS transaction (error free)

For an exception response, the server returns a code that is equivalent to the original function code from the request PDU with its most significant bit set to logic 1.

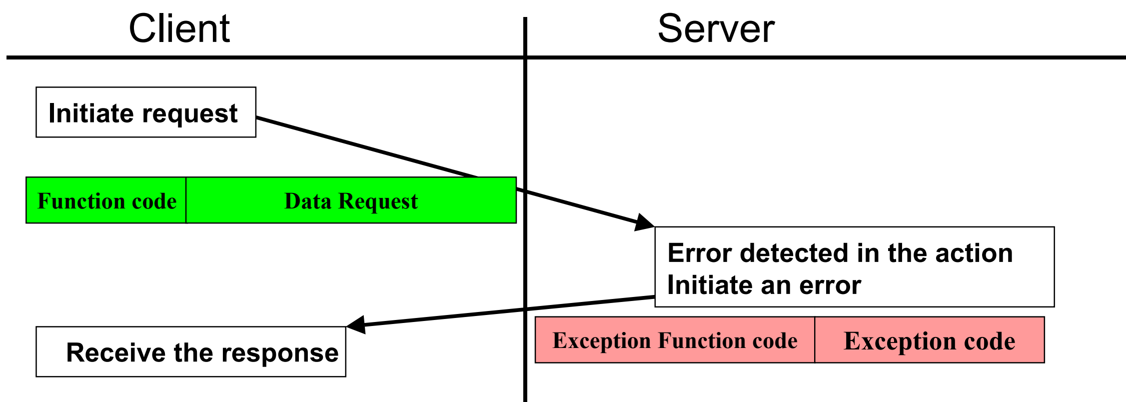


Figure 5 – MODBUS transaction (exception response)

NOTE It is desirable to manage a time out in order not to indefinitely wait for an answer which will perhaps never arrive.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The size of the MODBUS PDU is limited by the size constraint inherited from the first MODBUS implementation on Serial Line network (max. RS485 ADU = 256 bytes).

Therefore:

MODBUS **PDU for serial line communication =** 256 - Server adress (1 byte) - CRC (2 bytes) = **253 bytes**.

Consequently:

RS232 / RS485 **ADU** = 253 bytes + Server adress (1 byte) + CRC (2 bytes) = **256 bytes**.

TCP MODBUS **ADU** = 253 bytes + MBAP (7 bytes) = **260 bytes**.

The MODBUS protocol defines three PDUs. They are :

- MODBUS Request PDU, mb_req_pdu
- MODBUS Response PDU, mb_rsp_pdu
- MODBUS Exception Response PDU, mb_excep_rsp_pdu

The mb_req_pdu is defined as:

mb_req_pdu = {function_code, request_data}, where

function_code = [1 byte] MODBUS function code corresponding to the desired MODBUS function code or requested through the client API,

request_data = [n bytes] This field is function code dependent and usually contains information such as variable references, variable counts, data offsets, sub-function codes etc.

The mb_rsp_pdu is defined as:

mb_rsp_pdu = {function_code, response_data}, where

function_code = [1 byte] MODBUS function code

response_data = [n bytes] This field is function code dependent and usually contains information such as variable references, variable counts, data offsets, sub-function codes, etc.

The mb_excep_rsp_pdu is defined as:

mb_excep_rsp_pdu = {function_code, request_data}, where

exception-function_code = [1 byte] MODBUS function code + 0x80

exception_code = [1 byte] MODBUS Exception Code Defined in table "MODBUS Exception Codes" (see 1.7).

1.4.2 Data Encoding

- MODBUS uses a 'big-Endian' representation for addresses and data items. This means that when a numerical quantity larger than a single byte is transmitted, the most significant byte is sent first. So for example

Register size	value		
16 - bits	0x1234	the first byte sent is 0x12	then 0x34

NOTE For more details, see [1] in 1.1.2.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.4.3 MODBUS data model

MODBUS bases its data model on a series of tables that have distinguishing characteristics. The four primary tables are:

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

The distinctions between inputs and outputs, and between bit-addressable and word-addressable data items, do not imply any application behavior. It is perfectly acceptable, and very common, to regard all four tables as overlaying one another, if this is the most natural interpretation on the target machine in question.

For each of the primary tables, the protocol allows individual selection of 65536 data items, and the operations of read or write of those items are designed to span multiple consecutive data items up to a data size limit which is dependent on the transaction function code.

It's obvious that all the data handled via MODBUS (bits, registers) must be located in device application memory. But physical address in memory should not be confused with data reference. The only requirement is to link data reference with physical address.

MODBUS logical reference number, which are used in MODBUS functions, are unsigned integer indices starting at zero.

- **Implementation examples of MODBUS model**

The examples below show two ways of organizing the data in device. There are different organizations possible, but not all are described in this document. Each device can have its own organization of the data according to its application

Example 1 : Device having 4 separate blocks

The example below shows data organization in a device having digital and analog, inputs and outputs. Each block is separate because data from different blocks have no correlation. Each block is thus accessible with different MODBUS functions.

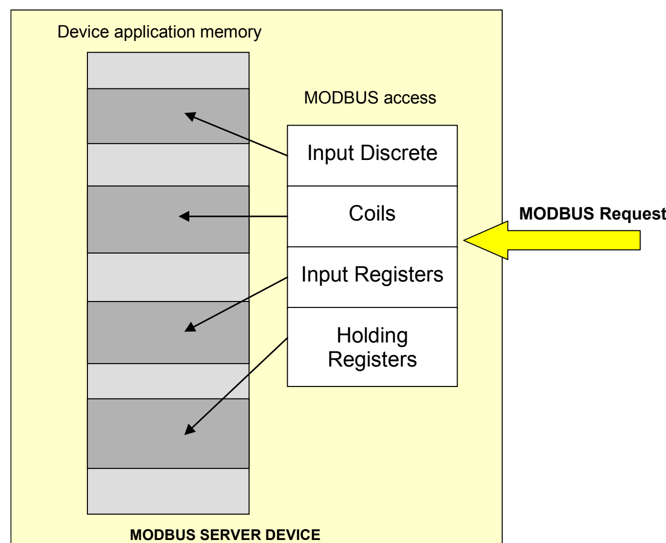


Figure 6 – MODBUS Data Model with separate block

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Example 2: Device having only 1 block

In this example, the device has only 1 data block. The same data can be reached via several MODBUS functions, either via a 16 bit access or via an access bit.

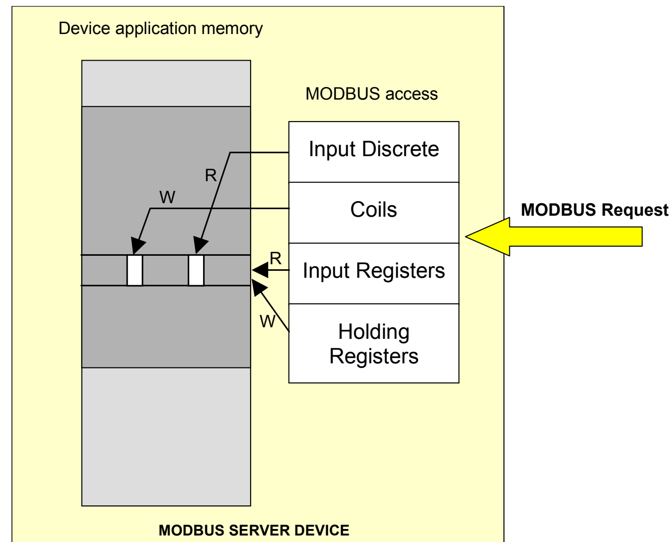


Figure 7 – MODBUS Data Model with only 1 block

1.4.4 MODBUS Addressing model

The MODBUS application protocol defines precisely PDU addressing rules.

In a MODBUS PDU each data is addressed from 0 to 65535.

It also defines clearly a MODBUS data model composed of 4 blocks that comprises several elements numbered from 1 to n.

In the MODBUS data Model each element within a data block is numbered from 1 to n.

Afterwards the MODBUS data model has to be bound to the device application (IEC-61131 object, or other application model).

The pre-mapping between the MODBUS data model and the device application is totally vendor device specific.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

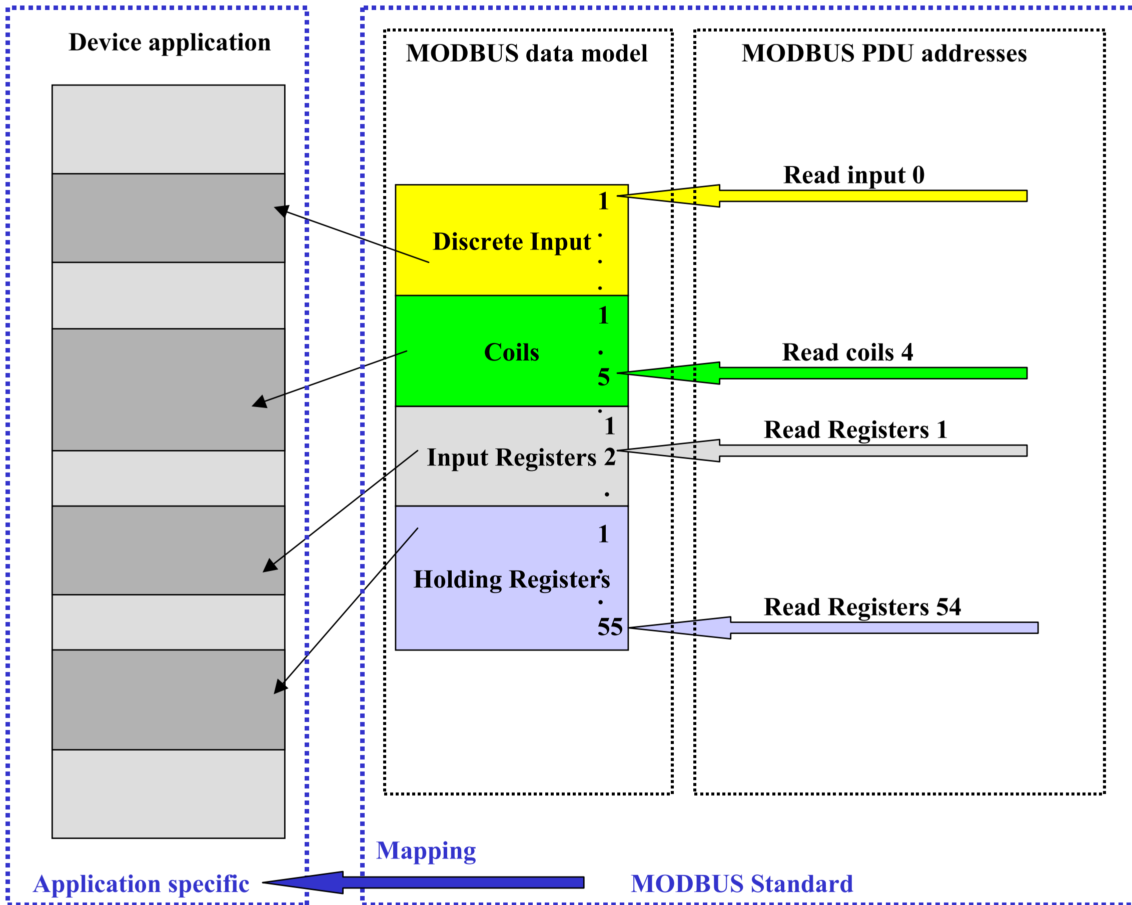


Figure 8 – MODBUS Addressing model

The previous figure shows that a MODBUS data numbered X is addressed in the MODBUS PDU X-1.

1.4.5 Define MODBUS Transaction

The following state diagram describes the generic processing of a MODBUS transaction in server side.

NOTE In this PAS, a normal response is the function code its specific data.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

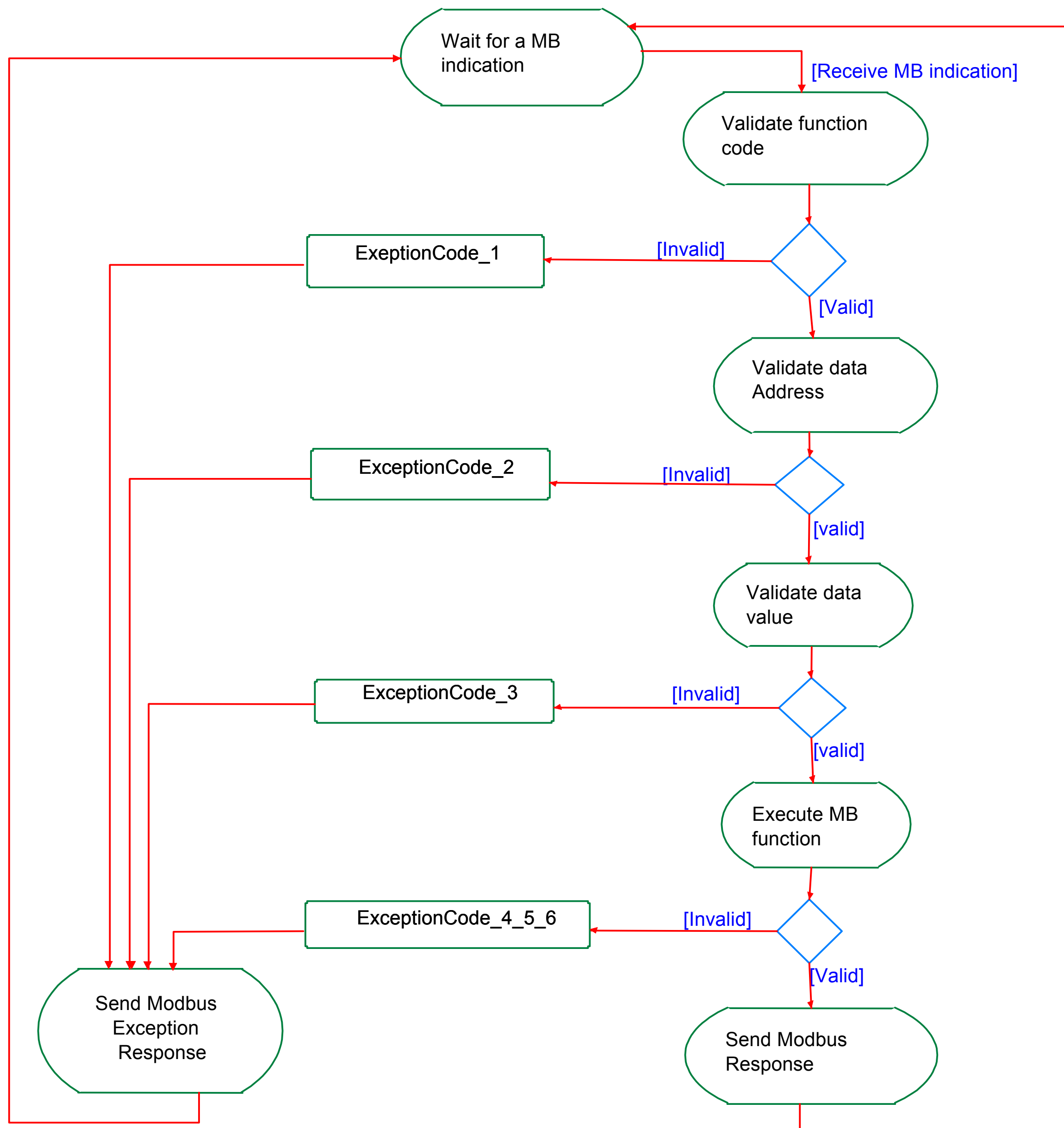


Figure 9 – MODBUS Transaction state diagram

Once the request has been processed by a server, a MODBUS response using the adequate MODBUS server transaction is built.

Depending on the result of the processing two types of response are built :

- A positive MODBUS response :
  - the response function code = the request function code

- A MODBUS Exception response (see 1.7):
  - the objective is to provide to the client relevant information concerning the error detected during the processing ;
  - the exception function code = the request function code + 0x80 ;
  - an exception code is provided to indicate the reason of the error.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.5 Function Code Categories

There are three categories of MODBUS Functions codes. They are :

Public Function Codes

- Are well defined function codes ,
- guaranteed to be unique,
- validated by the MODBUS-IDA.org community,
- publicly documented
- have available conformance test,
- includes both defined public assigned function codes as well as unassigned function codes reserved for future use.

User-Defined Function Codes

- there are two ranges of user-defined function codes, ie 65 to 72 and from 100 to 110 decimal.
- user can select and implement a function code that is not supported by the specification.
- there is no guarantee that the use of the selected function code will be unique
- if the user wants to re-position the functionality as a public function code, he must initiate an RFC to introduce the change into the public category and to have a new public function code assigned.
- MODBUS Organization, Inc expressly reserves the right to develop the proposed RFC.

Reserved Function Codes

- Function Codes currently used by some companies for legacy products and that are not available for public use.

NOTE The reader should refer to Annex B: MODBUS RESERVED FUNCTION CODES, SUBCODES AND MEI TYPES.

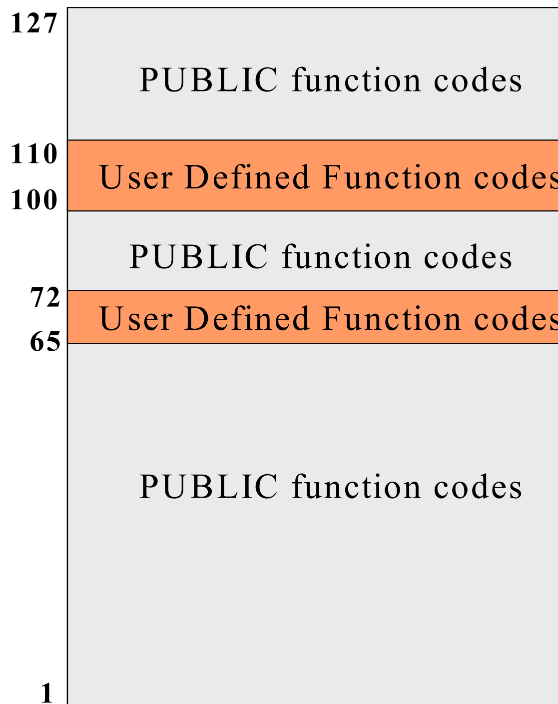


Figure 10 – MODBUS Function Code Categories

NOTE This Figure 10 MODBUS Function Code Categories represents the range where reserved function codes may reside.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.5.1 Public Function Code Definition

				Function Codes			
				code	*Sub code*	*(hex)*	*Section*
Data Access	**Bit access**	Physical Discrete Inputs	Read Discrete Inputs	**02**		**02**	1.6.2
		Internal Bits Or Physical coils	Read Coils	**01**		**01**	1.6.1
			Write Single Coil	**05**		**05**	1.6.5
			Write Multiple Coils	**15**		**0F**	1.6.11
	16 bits access	Physical Input Registers	Read Input Register	**04**		**04**	1.6.4
		Internal Registers Or Physical Output Registers	Read Holding Registers	**03**		**03**	1.6.3
			Write Single Register	**06**		**06**	1.6.6
			Write Multiple Registers	**16**		**10**	1.6.12
			Read/Write Multiple Registers	**23**		**17**	1.6.17
			Mask Write Register	**22**		**16**	1.6.16
			Read FIFO queue	**24**		**18**	1.6.18
	File record access		Read File record	**20**	**6**	**14**	1.6.14
			Write File record	**21**	**6**	**15**	1.6.15
Diagnostics			Read Exception status	**07**		**07**	1.6.7
			Diagnostic	**08**	**00-18,20**	**08**	1.6.8
			Get Com event counter	**11**		**OB**	1.6.9
			Get Com Event Log	**12**		**0C**	1.6.10
			Report Slave ID	**17**		**11**	1.6.13
			Read device Identification	**43**	**14**	**2B**	1.6.21
Other			Encapsulated Interface Transport	**43**	**13,14**	**2B**	1.6.19
			CANopen General Reference	**43**	**13**	**2B**	1.6.20

1.6 Function codes descripitons

1.6.1 01 (0x01) Read Coils

This function code is used to read from 1 to 2000 contiguous status of coils in a remote device. The Request PDU specifies the starting address, ie the address of the first coil specified, and the number of coils. In the PDU Coils are addressed starting at zero. Therefore coils numbered 1-16 are addressed as 0-15.

The coils in the response message are packed as one coil per bit of the data field. Status is indicated as 1= ON and 0= OFF. The LSB of the first data byte contains the output addressed in the query. The other coils follow toward the high order end of this byte, and from low order to high order in subsequent bytes.

If the returned output quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Request

Function code	1 Byte	**0x01**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of coils	2 Bytes	1 to 2000 (0x7D0)

Response

Function code	1 Byte	**0x01**
Byte count	1 Byte	**N***
Coil Status	**n** Byte	n = N or N+1

***N** = Quantity of Outputs / 8, if the remainder is different of 0 ⇒ N = N+1

Error

Function code	1 Byte	**Function code + 0x80**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to read discrete outputs 20–38:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**01**	Function	**01**
Starting Address Hi	**00**	Byte Count	**03**
Starting Address Lo	**13**	Outputs status 27-20	**CD**
Quantity of Outputs Hi	**00**	Outputs status 35-28	**6B**
Quantity of Outputs Lo	**13**	Outputs status 38-36	**05**

The status of outputs 27–20 is shown as the byte value CD hex, or binary 1100 1101. Output 27 is the MSB of this byte, and output 20 is the LSB.

By convention, bits within a byte are shown with the MSB to the left, and the LSB to the right. Thus the outputs in the first byte are ‘27 through 20’, from left to right. The next byte has outputs ‘35 through 28’, left to right. As the bits are transmitted serially, they flow from LSB to MSB: 20 . . . 27, 28 . . . 35, and so on.

In the last data byte, the status of outputs 38-36 is shown as the byte value 05 hex, or binary 0000 0101. Output 38 is in the sixth bit position from the left, and output 36 is the LSB of this byte. The five remaining high order bits are zero filled.

NOTE The five remaining bits (toward the high order end) are zero filled.

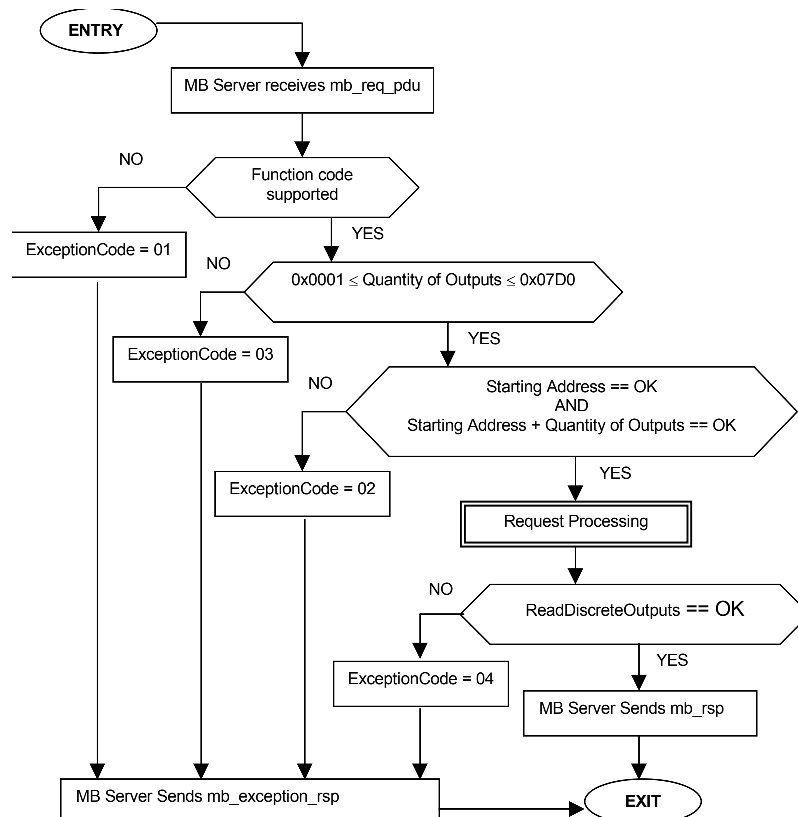


Figure 11 – Read Coils state diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.6.2 02 (0x02) Read Discrete Inputs

This function code is used to read from 1 to 2000 contiguous status of discrete inputs in a remote device. The Request PDU specifies the starting address, ie the address of the first input specified, and the number of inputs. In the PDU Discrete Inputs are addressed starting at zero. Therefore Discrete inputs numbered 1-16 are addressed as 0-15.

The discrete inputs in the response message are packed as one input per bit of the data field. Status is indicated as 1= ON; 0= OFF. The LSB of the first data byte contains the input addressed in the query. The other inputs follow toward the high order end of this byte, and from low order to high order in subsequent bytes.

If the returned input quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Request

Function code	1 Byte	**0x02**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Inputs	2 Bytes	1 to 2000 (0x7D0)

Response

Function code	1 Byte	**0x02**
Byte count	1 Byte	**N***
Input Status	**N*** x 1 Byte	

*__N__ = Quantity of Inputs / 8 if the remainder is different of 0 ⇒ N = N+1

Error

Error code	1 Byte	**0x82**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to read discrete inputs 197 – 218:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**02**	Function	**02**
Starting Address Hi	**00**	Byte Count	**03**
Starting Address Lo	**C4**	Inputs Status 204-197	**AC**
Quantity of Inputs Hi	**00**	Inputs Status 212-205	**DB**
Quantity of Inputs Lo	**16**	Inputs Status 218-213	**35**

The status of discrete inputs 204–197 is shown as the byte value AC hex, or binary 1010 1100. Input 204 is the MSB of this byte, and input 197 is the LSB.

The status of discrete inputs 218–213 is shown as the byte value 35 hex, or binary 0011 0101. Input 218 is in the third bit position from the left, and input 213 is the LSB.

NOTE The two remaining bits (toward the high order end) are zero filled.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

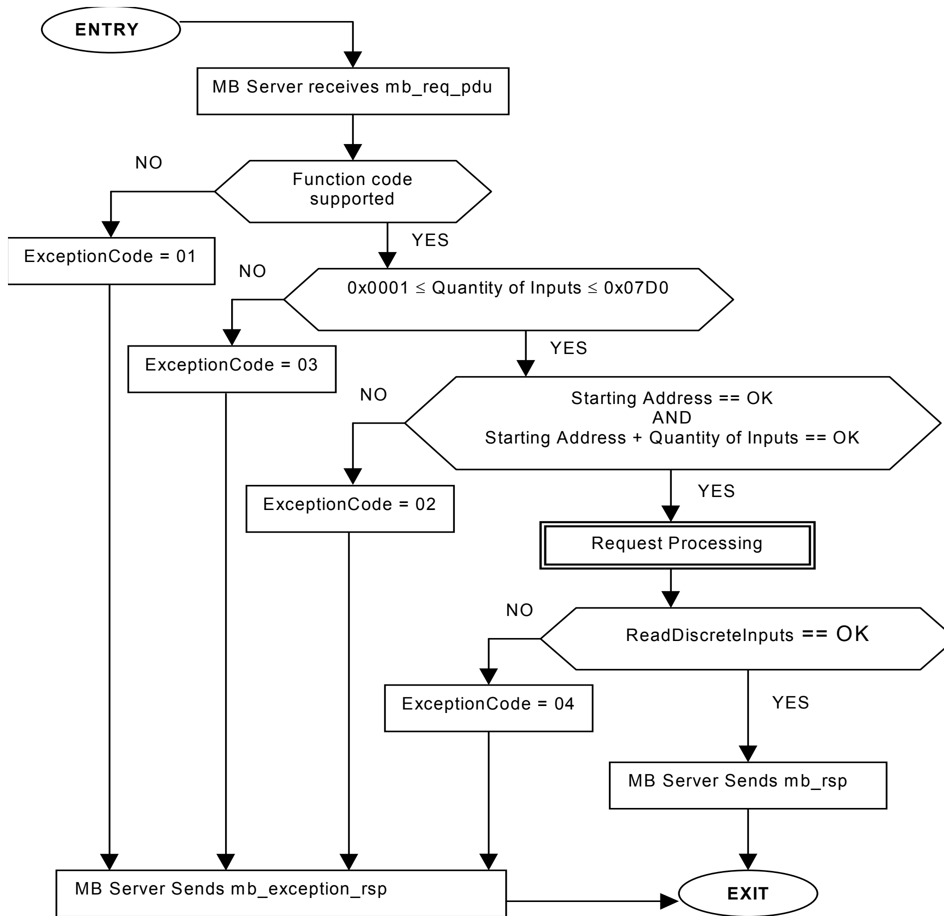


Figure 12 – Read Discrete Inputs state diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.6.3 03 (0x03) Read Holding Registers

This function code is used to read the contents of a contiguous block of holding registers in a remote device. The Request PDU specifies the starting register address and the number of registers. In the PDU Registers are addressed starting at zero. Therefore registers numbered 1-16 are addressed as 0-15.

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Request

Function code	1 Byte	**0x03**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 125 (0x7D)

Response

Function code	1 Byte	**0x03**
Byte count	1 Byte	2 x **N***
Register value	**N*** x 2 Bytes	

*__N__ = Quantity of Registers

Error

Error code	1 Byte	**0x83**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to read registers 108 – 110:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**03**	Function	**03**
Starting Address Hi	**00**	Byte Count	**06**
Starting Address Lo	**6B**	Register value Hi (108)	**02**
No. of Registers Hi	**00**	Register value Lo (108)	**2B**
No. of Registers Lo	**03**	Register value Hi (109)	**00**
		Register value Lo (109)	**00**
		Register value Hi (110)	**00**
		Register value Lo (110)	**64**

The contents of register 108 are shown as the two byte values of 02 2B hex, or 555 decimal. The contents of registers 109–110 are 00 00 and 00 64 hex, or 0 and 100 decimal, respectively.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

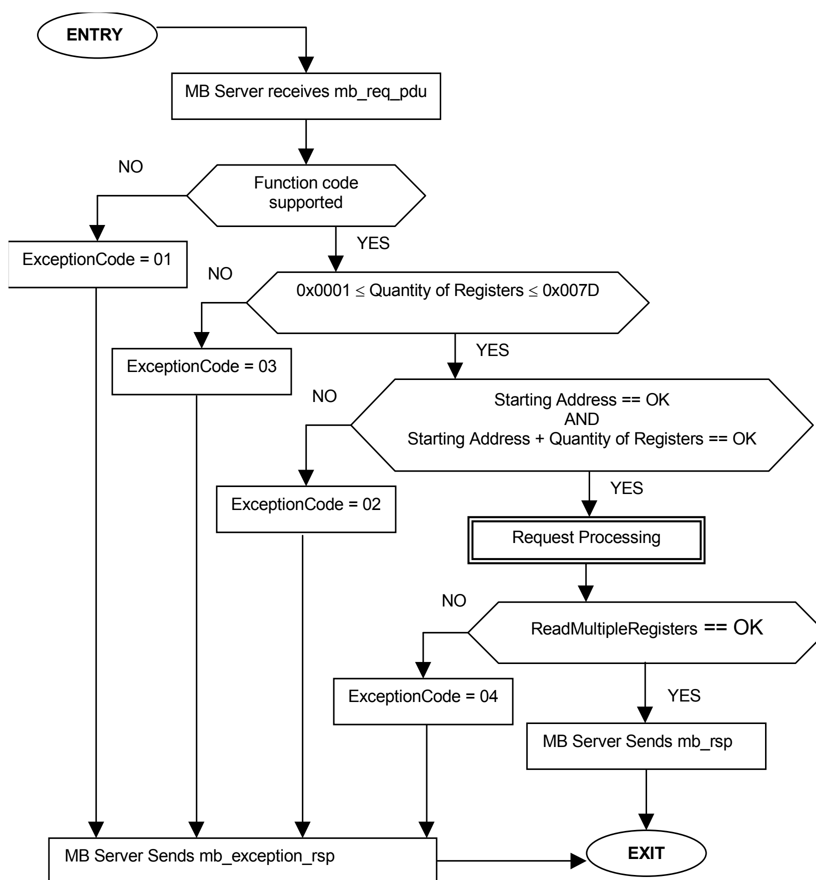


Figure 13 – Read Holding Registers state diagram

1.6.4 04 (0x04) Read Input Registers

This function code is used to read from 1 to approx. 125 contiguous input registers in a remote device. The Request PDU specifies the starting register address and the number of registers. In the PDU Registers are addressed starting at zero. Therefore input registers numbered 1-16 are addressed as 0-15.

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Request

Function code	1 Byte	**0x04**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Input Registers	2 Bytes	0x0001 to 0x007D

Response

Function code	1 Byte	**0x04**
Byte count	1 Byte	2 x **N***
Input Registers	**N*** x 2 Bytes	

*__N__ = Quantity of Input Registers

Error

Error code	1 Byte	**0x84**
Exception code	1 Byte	01 or 02 or 03 or 04

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Here is an example of a request to read input register 9:

Request		Response	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**04**	Function	**04**
Starting Address Hi	**00**	Byte Count	**02**
Starting Address Lo	**08**	Input Reg. 9 Hi	**00**
Quantity of Input Reg. Hi	**00**	Input Reg. 9 Lo	**0A**
Quantity of Input Reg. Lo	**01**		

The contents of input register 9 are shown as the two byte values of 00 0A hex, or 10 decimal.

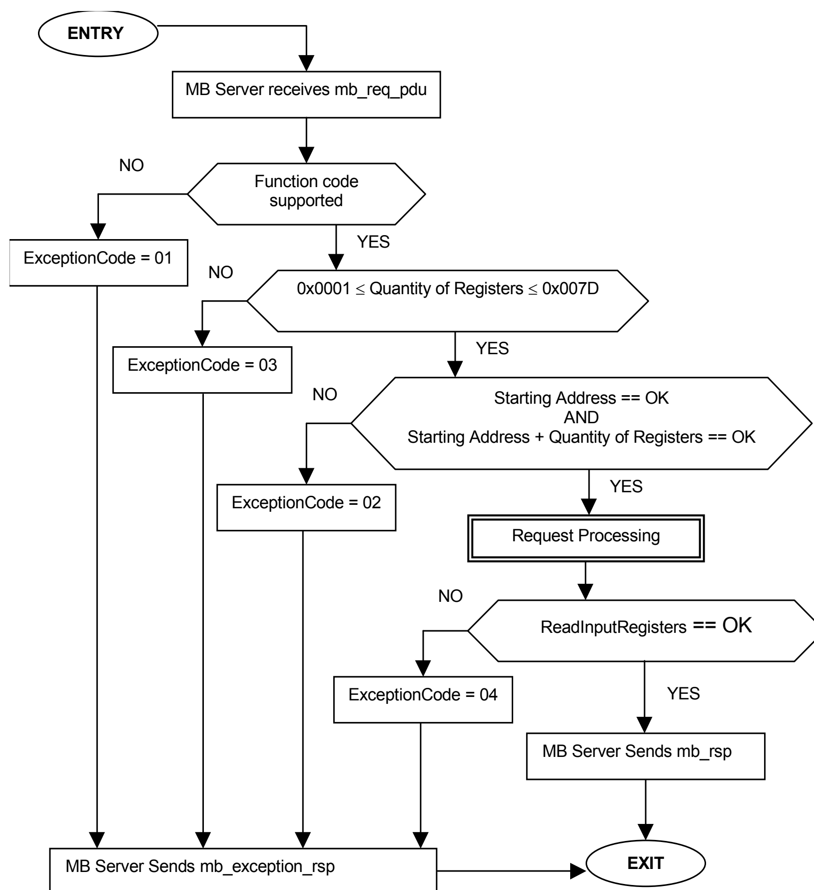


Figure 14 – Read Input Registers state diagram

1.6.5 05 (0x05) Write Single Coil

This function code is used to write a single output to either ON or OFF in a remote device.

The requested ON/OFF state is specified by a constant in the request data field. A value of FF 00 hex requests the output to be ON. A value of 00 00 requests it to be OFF. All other values are illegal and will not affect the output.

The Request PDU specifies the address of the coil to be forced. Coils are addressed starting at zero. Therefore coil numbered 1 is addressed as 0. The requested ON/OFF state is specified by a constant in the Coil Value field. A value of 0XFF00 requests the coil to be ON. A value of 0X0000 requests the coil to be off. All other values are illegal and will not affect the coil.

The normal response is an echo of the request, returned after the coil state has been written.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Request

Function code	1 Byte	**0x05**
Output Address	2 Bytes	0x0000 to 0xFFFF
Output Value	2 Bytes	0x0000 or 0xFF00

Response

Function code	1 Byte	**0x05**
Output Address	2 Bytes	0x0000 to 0xFFFF
Output Value	2 Bytes	0x0000 or 0xFF00

Error

Error code	1 Byte	**0x85**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to write Coil 173 ON:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**05**	Function	**05**
Output Address Hi	**00**	Output Address Hi	**00**
Output Address Lo	**AC**	Output Address Lo	**AC**
Output Value Hi	**FF**	Output Value Hi	**FF**
Output Value Lo	**00**	Output Value Lo	**00**

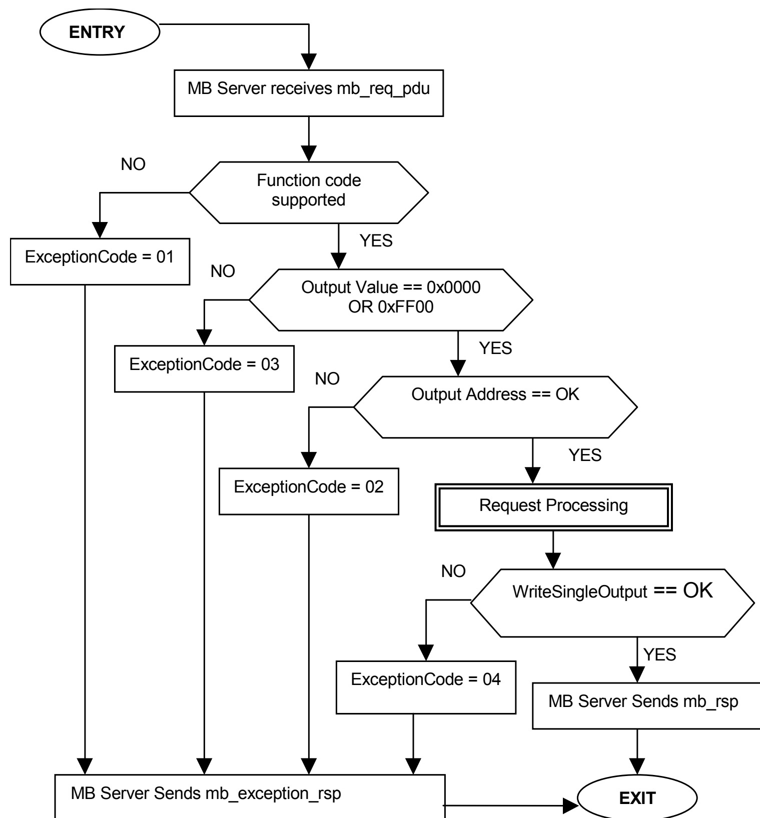


Figure 15 – Write Single Output state diagram

1.6.6 06 (0x06) Write Single Register

This function code is used to write a single holding register in a remote device.

The Request PDU specifies the address of the register to be written. Registers are addressed starting at zero. Therefore register numbered 1 is addressed as 0.

The normal response is an echo of the request, returned after the register contents have been written.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Request

Function code	1 Byte	**0x06**
Register Address	2 Bytes	0x0000 to 0xFFFF
Register Value	2 Bytes	0x0000 or 0xFFFF

Response

Function code	1 Byte	**0x06**
Register Address	2 Bytes	0x0000 to 0xFFFF
Register Value	2 Bytes	0x0000 or 0xFFFF

Error

Error code	1 Byte	**0x86**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to write register 2 to 00 03 hex:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**06**	Function	**06**
Register Address Hi	**00**	Register Address Hi	**00**
Register Address Lo	**01**	Register Address Lo	**01**
Register Value Hi	**00**	Register Value Hi	**00**
Register Value Lo	**03**	Register Value Lo	**03**

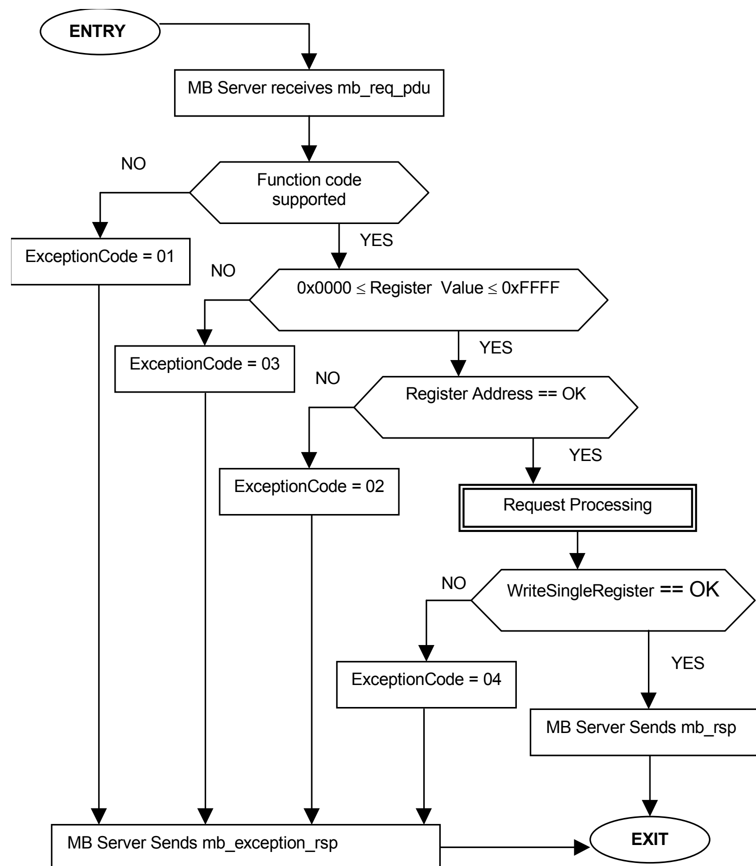


Figure 16 – Write Single Register state diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.6.7 07 (0x07) Read Exception Status (Serial Line only)

This function code is used to read the contents of eight Exception Status outputs in a remote device.

The function provides a simple method for accessing this information, because the Exception Output references are known (no output reference is needed in the function).

The normal response contains the status of the eight Exception Status outputs. The outputs are packed into one data byte, with one bit per output. The status of the lowest output reference is contained in the least significant bit of the byte.

The contents of the eight Exception Status outputs are device specific.

Request

Function code	1 Byte	**0x07**

Response

Function code	1 Byte	**0x07**
Output Data	1 Byte	0x00 to 0xFF

Error

Error code	1 Byte	**0x87**
Exception code	1 Byte	01 or 04

Here is an example of a request to read the exception status:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**07**	Function	**07**
		Output Data	**6D**

In this example, the output data is 6D hex (0110 1101 binary). Left to right, the outputs are OFF–ON–ON–OFF–ON–ON–OFF–ON. The status is shown from the highest to the lowest addressed output.

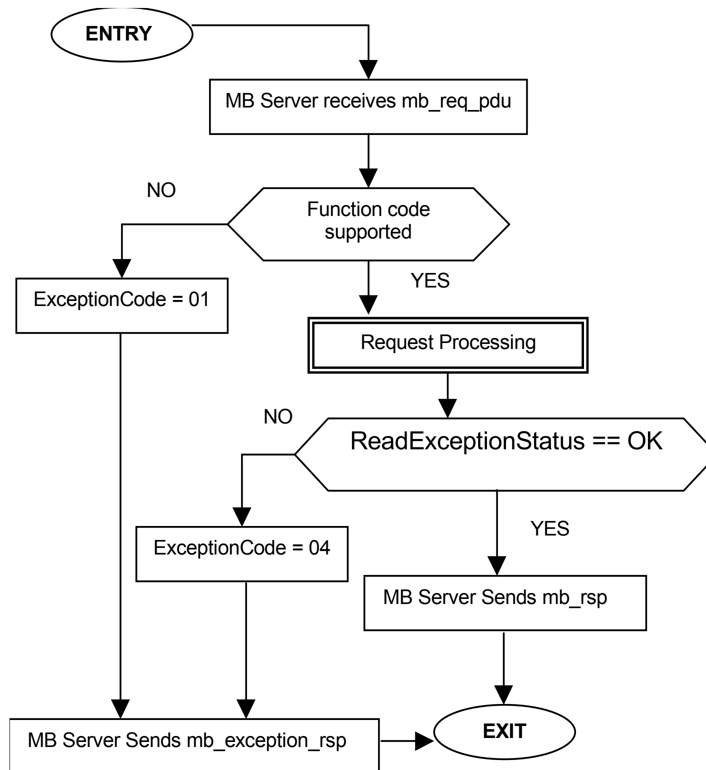


Figure 17 – Read Exception Status state diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.6.8 08 (0x08) Diagnostics (Serial Line only)

MODBUS function code 08 provides a series of tests for checking the communication system between a client (Master) device and a server (Slave), or for checking various internal error conditions within a server.

The function uses a two–byte sub-function code field in the query to define the type of test to be performed. The server echoes both the function code and sub-function code in a normal response. Some of the diagnostics cause data to be returned from the remote device in the data field of a normal response.

In general, issuing a diagnostic function to a remote device does not affect the running of the user program in the remote device. User logic, like discrete and registers, is not accessed by the diagnostics. Certain functions can optionally reset error counters in the remote device.

A server device can, however, be forced into 'Listen Only Mode' in which it will monitor the messages on the communications system but not respond to them. This can affect the outcome of your application program if it depends upon any further exchange of data with the remote device. Generally, the mode is forced to remove a malfunctioning remote device from the communications system.

The following diagnostic functions are dedicated to serial line devices.

The normal response to the Return Query Data request is to loopback the same data. The function code and sub-function codes are also echoed.

Request

Function code	1 Byte	**0x08**
Sub-function	2 Bytes	
Data	N x 2 Bytes	

Response

Function code	1 Byte	**0x08**
Sub-function	2 Bytes	
Data	N x 2 Bytes	

Error

Error code	1 Byte	**0x88**
Exception code	1 Byte	01 or 03 or 04

1.6.8.1 Sub-function codes supported by the serial line devices

Here the list of sub-function codes supported by the serial line devices. Each sub-function code is then listed with an example of the data field contents that would apply for that diagnostic.

Sub-function code		Name
Hex	Dec	
00	00	Return Query Data
01	01	Restart Communications Option
02	02	Return Diagnostic Register
03	03	Change ASCII Input Delimiter
04	04	Force Listen Only Mode
	05.. 09	RESERVED
0A	10	Clear Counters and Diagnostic Register
0B	11	Return Bus Message Count
0C	12	Return Bus Communication Error Count
0D	13	Return Bus Exception Error Count
0E	14	Return Slave Message Count
0F	15	Return Slave No Response Count
10	16	Return Slave NAK Count
11	17	Return Slave Busy Count
12	18	Return Bus Character Overrun Count
13	19	RESERVED
14	20	Clear Overrun Counter and Flag
N.A.	21 … 65535	RESERVED

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

00 Return Query Data

The data passed in the request data field is to be returned (looped back) in the response. The entire response message should be identical to the request.

Sub-function	Data Field (Request)	Data Field (Response)
00 00	Any	Echo Request Data

01 Restart Communications Option

The remote device serial line port must be initialized and restarted, and all of its communications event counters are cleared. If the port is currently in Listen Only Mode, no response is returned. This function is the only one that brings the port out of Listen Only Mode. If the port is not currently in Listen Only Mode, a normal response is returned. This occurs before the restart is executed.

When the remote device receives the request, it attempts a restart and executes its power–up confidence tests. Successful completion of the tests will bring the port online.

A request data field contents of FF 00 hex causes the port's Communications Event Log to be cleared also. Contents of 00 00 leave the log as it was prior to the restart.

Sub-function	Data Field (Request)	Data Field (Response)
00 01	00 00	Echo Request Data
00 01	FF 00	Echo Request Data

02 Return Diagnostic Register

The contents of the remote device's 16–bit diagnostic register are returned in the response.

Sub-function	Data Field (Request)	Data Field (Response)
00 02	00 00	Diagnostic Register Contents

03 Change ASCII Input Delimiter

The character 'CHAR' passed in the request data field becomes the end of message delimiter for future messages (replacing the default LF character). This function is useful in cases of a Line Feed is not required at the end of ASCII messages.

Sub-function	Data Field (Request)	Data Field (Response)
00 03	CHAR 00	Echo Request Data

04 Force Listen Only Mode

Forces the addressed remote device to its Listen Only Mode for MODBUS communications. This isolates it from the other devices on the network, allowing them to continue communicating without interruption from the addressed remote device. No response is returned.

When the remote device enters its Listen Only Mode, all active communication controls are turned off. The Ready watchdog timer is allowed to expire, locking the controls off. While the device is in this mode, any MODBUS messages addressed to it or broadcast are monitored, but no actions will be taken and no responses will be sent.

The only function that will be processed after the mode is entered will be the Restart Communications Option function (function code 8, sub-function 1).

Sub-function	Data Field (Request)	Data Field (Response)
00 04	00 00	No Response Returned

10 (0A Hex) Clear Counters and Diagnostic Register

The goal is to clear all counters and the diagnostic register. Counters are also cleared upon power–up.

Sub-function	Data Field (Request)	Data Field (Response)
00 0A	00 00	Echo Request Data

11 (0B Hex) Return Bus Message Count

The response data field returns the quantity of messages that the remote device has detected on the communications system since its last restart, clear counters operation, or power–up.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Sub-function	Data Field (Request)	Data Field (Response)
00 0B	00 00	Total Message Count

12 (0C Hex) Return Bus Communication Error Count

The response data field returns the quantity of CRC errors encountered by the remote device since its last restart, clear counters operation, or power–up.

Sub-function	Data Field (Request)	Data Field (Response)
00 0C	00 00	CRC Error Count

13 (0D Hex) Return Bus Exception Error Count

The response data field returns the quantity of MODBUS exception responses returned by the remote device since its last restart, clear counters operation, or power–up.

Exception responses are described and listed in 1.7 .

Sub-function	Data Field (Request)	Data Field (Response)
00 0D	00 00	Exception Error Count

14 (0E Hex) Return Slave Message Count

The response data field returns the quantity of messages addressed to the remote device, or broadcast, that the remote device has processed since its last restart, clear counters operation, or power–up.

Sub-function	Data Field (Request)	Data Field (Response)
00 0E	00 00	Slave Message Count

15 (0F Hex) Return Slave No Response Count

The response data field returns the quantity of messages addressed to the remote device for which it has returned no response (neither a normal response nor an exception response), since its last restart, clear counters operation, or power–up.

Sub-function	Data Field (Request)	Data Field (Response)
00 0F	00 00	Slave No Response Count

16 (10 Hex) Return Slave NAK Count

The response data field returns the quantity of messages addressed to the remote device for which it returned a Negative Acknowledge (NAK) exception response, since its last restart, clear counters operation, or power–up. Exception responses are described and listed in section 1.7 .

Sub-function	Data Field (Request)	Data Field (Response)
00 10	00 00	Slave NAK Count

17 (11 Hex) Return Slave Busy Count

The response data field returns the quantity of messages addressed to the remote device for which it returned a Slave Device Busy exception response, since its last restart, clear counters operation, or power–up.

Sub-function	Data Field (Request)	Data Field (Response)
00 11	00 00	Slave Device Busy Count

18 (12 Hex) Return Bus Character Overrun Count

The response data field returns the quantity of messages addressed to the remote device that it could not handle due to a character overrun condition, since its last restart, clear counters operation, or power–up. A character overrun is caused by data characters arriving at the port faster than they can be stored, or by the loss of a character due to a hardware malfunction.

Sub-function	Data Field (Request)	Data Field (Response)
00 12	00 00	Slave Character Overrun Count

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

20 (14 Hex) Clear Overrun Counter and Flag

Clears the overrun error counter and reset the error flag.

Sub-function	Data Field (Request)	Data Field (Response)
00 14	00 00	Echo Request Data

1.6.8.2 Example and state diagram

Here is an example of a request to remote device to Return Query Data. This uses a sub-function code of zero (00 00 hex in the two–byte field). The data to be returned is sent in the two–byte data field (A5 37 hex).

Request		Response	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**08**	Function	**08**
Sub-function Hi	**00**	Sub-function Hi	**00**
Sub-function Lo	**00**	Sub-function Lo	**00**
Data Hi	**A5**	Data Hi	**A5**
Data Lo	**37**	Data Lo	**37**

The data fields in responses to other kinds of queries could contain error counts or other data requested by the sub-function code.

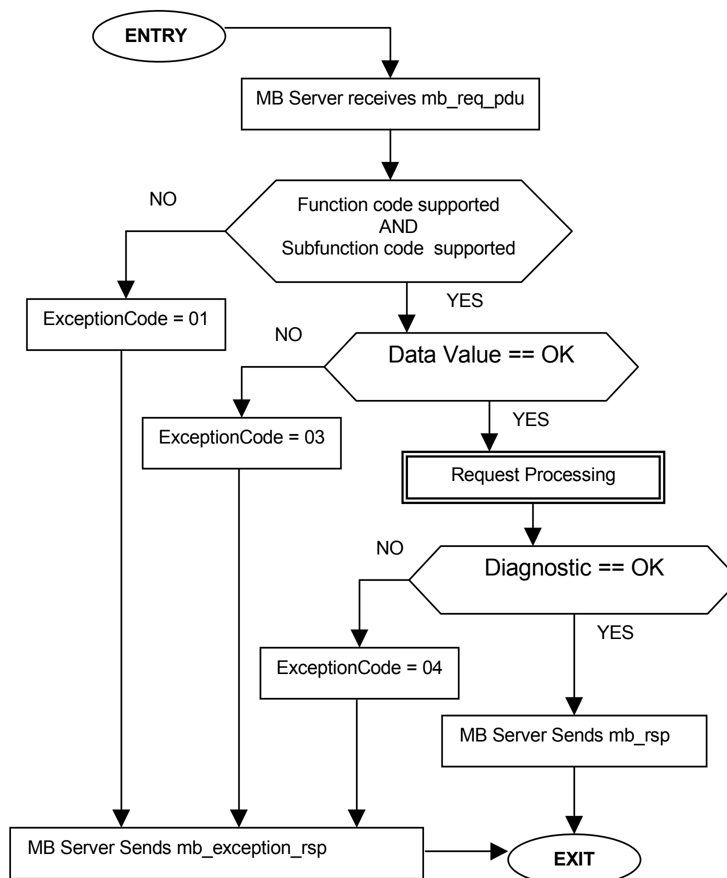


Figure 18 – Diagnostic state diagram

1.6.9 11 (0x0B) Get Comm Event Counter (Serial Line only)

This function code is used to get a status word and an event count from the remote device's communication event counter.

By fetching the current count before and after a series of messages, a client can determine whether the messages were handled normally by the remote device.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The device's event counter is incremented once for each successful message completion. It is not incremented for exception responses, poll commands, or fetch event counter commands.

The event counter can be reset by means of the Diagnostics function (code 08), with a sub-function of Restart Communications Option (code 00 01) or Clear Counters and Diagnostic Register (code 00 0A).

The normal response contains a two–byte status word, and a two–byte event count. The status word will be all ones (FF FF hex) if a previously–issued program command is still being processed by the remote device (a busy condition exists). Otherwise, the status word will be all zeros.

Request

Function code	1 Byte	**0x0B**

Response

Function code	1 Byte	**0x0B**
Status	2 Bytes	0x0000 to 0xFFFF
Event Count	2 Bytes	0x0000 to 0xFFFF

Error

Error code	1 Byte	**0x8B**
Exception code	1 Byte	01 or 04

Here is an example of a request to get the communications event counter in remote device:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**0B**	Function	**0B**
		Status Hi	**FF**
		Status Lo	**FF**
		Event Count Hi	**01**
		Event Count Lo	**08**

In this example, the status word is FF FF hex, indicating that a program function is still in progress in the remote device. The event count shows that 264 (01 08 hex) events have been counted by the device.

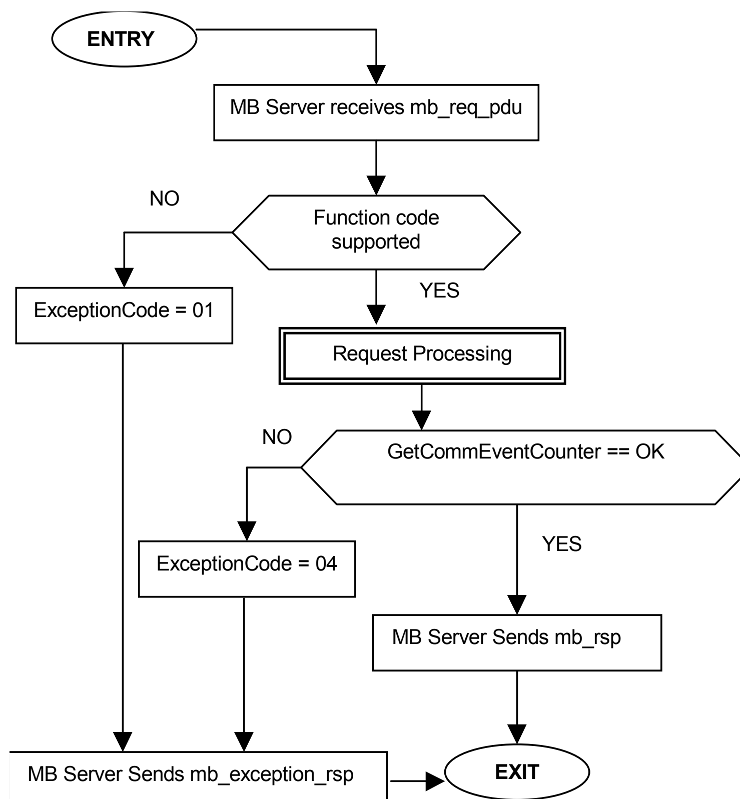


Figure 19 – Get Comm Event Counter state diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.6.10 12 (0x0C) Get Comm Event Log (Serial Line only)

This function code is used to get a status word, event count, message count, and a field of event bytes from the remote device.

The status word and event counts are identical to that returned by the Get Communications Event Counter function (11, 0B hex).

The message counter contains the quantity of messages processed by the remote device since its last restart, clear counters operation, or power–up. This count is identical to that returned by the Diagnostic function (code 08), sub-function Return Bus Message Count (code 11, 0B hex).

The event bytes field contains 0-64 bytes, with each byte corresponding to the status of one MODBUS send or receive operation for the remote device. The remote device enters the events into the field in chronological order. Byte 0 is the most recent event. Each new byte flushes the oldest byte from the field.

The normal response contains a two–byte status word field, a two–byte event count field, a two–byte message count field, and a field containing 0-64 bytes of events. A byte count field defines the total length of the data in these four fields.

Request

Function code	1 Byte	**0x0C**

Response

Function code	1 Byte	**0x0C**
Byte Count	1 Byte	**N***
Status	2 Bytes	0x0000 to 0xFFFF
Event Count	2 Bytes	0x0000 to 0xFFFF
Message Count	2 Bytes	0x0000 to 0xFFFF
Events	(**N**-6) x 1 Byte	

***N** = Quantity of Events + 3 x 2 Bytes, (Length of Status, Event Count and Message Count)

Error

Error code	1 Byte	**0x8C**
Exception code	1 Byte	01 or 04

Here is an example of a request to get the communications event log in remote device:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**0C**	Function	**0C**
		Byte Count	**08**
		Status Hi	**00**
		Status Lo	**00**
		Event Count Hi	**01**
		Event Count Lo	**08**
		Message Count Hi	**01**
		Message Count Lo	**21**
		Event 0	**20**
		Event 1	**00**

In this example, the status word is 00 00 hex, indicating that the remote device is not processing a program function. The event count shows that 264 (01 08 hex) events have been counted by the remote device. The message count shows that 289 (01 21 hex) messages have been processed.

The most recent communications event is shown in the Event 0 byte. Its content (20 hex) show that the remote device has most recently entered the Listen Only Mode.

The previous event is shown in the Event 1 byte. Its contents (00 hex) show that the remote device received a Communications Restart.

The layout of the response's event bytes is described below.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

What the Event Bytes Contain

An event byte returned by the Get Communications Event Log function can be any one of four types. The type is defined by bit 7 (the high–order bit) in each byte. It may be further defined by bit 6. This is explained below.

- **Remote device MODBUS Receive Event**

  The remote device stores this type of event byte when a query message is received. It is stored before the remote device processes the message. This event is defined by bit 7 set to logic '1'. The other bits will be set to a logic '1' if the corresponding condition is TRUE. The bit layout is:

Bit	Contents
0	Not Used
1	Communication Error
2	Not Used
3	Not Used
4	Character Overrun
5	Currently in Listen Only Mode
6	Broadcast Received
7	1

- **Remote device MODBUS Send Event**

  The remote device stores this type of event byte when it finishes processing a request message. It is stored if the remote device returned a normal or exception response, or no response. This event is defined by bit 7 set to a logic '0', with bit 6 set to a '1'. The other bits will be set to a logic '1' if the corresponding condition is TRUE. The bit layout is:

Bit	Contents
0	Read Exception Sent (Exception Codes 1-3)
1	Slave Abort Exception Sent (Exception Code 4)
2	Slave Busy Exception Sent (Exception Codes 5-6)
3	Slave Program NAK Exception Sent (Exception Code 7)
4	Write Timeout Error Occurred
5	Currently in Listen Only Mode
6	1
7	0

- **Remote device Entered Listen Only Mode**

  The remote device stores this type of event byte when it enters the Listen Only Mode. The event is defined by a content of 04 hex.

- **Remote device Initiated Communication Restart**

  The remote device stores this type of event byte when its communications port is restarted. The remote device can be restarted by the Diagnostics function (code 08), with sub-function Restart Communications Option (code 00 01).

  That function also places the remote device into a 'Continue on Error' or 'Stop on Error' mode. If the remote device is placed into 'Continue on Error' mode, the event byte is added to the existing event log. If the remote device is placed into 'Stop on Error' mode, the byte is added to the log and the rest of the log is cleared to zeros.

  The event is defined by a content of zero.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

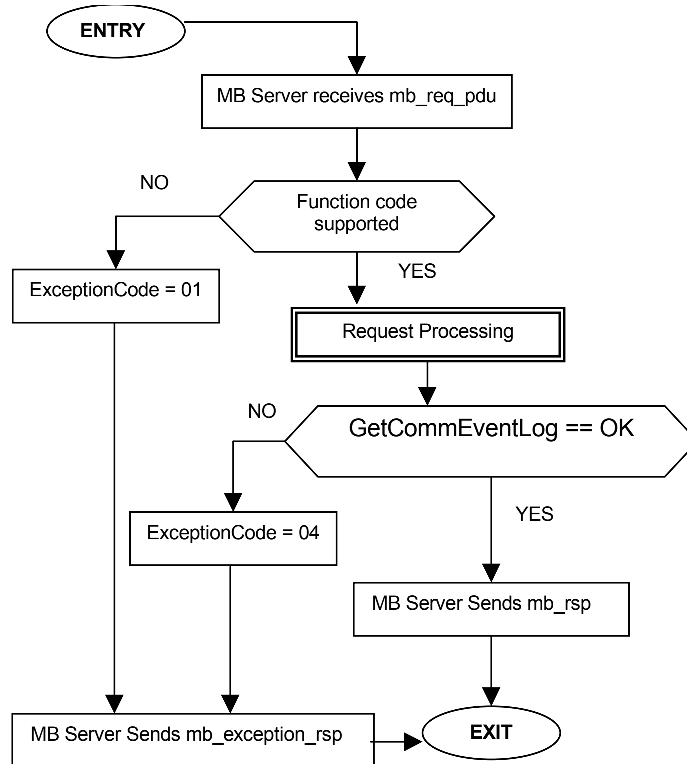


Figure 20 – Get Comm Event Log state diagram

1.6.11 15 (0x0F) Write Multiple Coils

This function code is used to force each coil in a sequence of coils to either ON or OFF in a remote device. The Request PDU specifies the coil references to be forced. Coils are addressed starting at zero. Therefore coil numbered 1 is addressed as 0.

The requested ON/OFF states are specified by contents of the request data field. A logical '1' in a bit position of the field requests the corresponding output to be ON. A logical '0' requests it to be OFF.

The normal response returns the function code, starting address, and quantity of coils forced.

Request PDU

Function code	1 Byte	**0x0F**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Outputs	2 Bytes	0x0001 to 0x07B0
Byte Count	1 Byte	**N***
Outputs Value	**N*** x 1 Byte	

***N** = Quantity of Outputs / 8, if the remainder is different of 0 ⇒ N = N+1

Response PDU

Function code	1 Byte	**0x0F**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Outputs	2 Bytes	0x0001 to 0x07B0

Error

Error code	1 Byte	**0x8F**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to write a series of 10 coils starting at coil 20:

The request data contents are two bytes: CD 01 hex (1100 1101 0000 0001 binary). The binary bits correspond to the outputs in the following way:

Bit:	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	1
Output:	27	26	25	24	23	22	21	20	–	–	–	–	–	–	29	28

The first byte transmitted (CD hex) addresses outputs 27-20, with the least significant bit addressing the lowest output (20) in this set.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The next byte transmitted (01 hex) addresses outputs 29-28, with the least significant bit addressing the lowest output (28) in this set. Unused bits in the last data byte should be zero–filled.

Request		Response	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**0F**	Function	**0F**
Starting Address Hi	**00**	Starting Address Hi	**00**
Starting Address Lo	**13**	Starting Address Lo	**13**
Quantity of Outputs Hi	**00**	Quantity of Outputs Hi	**00**
Quantity of Outputs Lo	**0A**	Quantity of Outputs Lo	**0A**
Byte Count	**02**		
Outputs Value Hi	**CD**		
Outputs Value Lo	**01**		

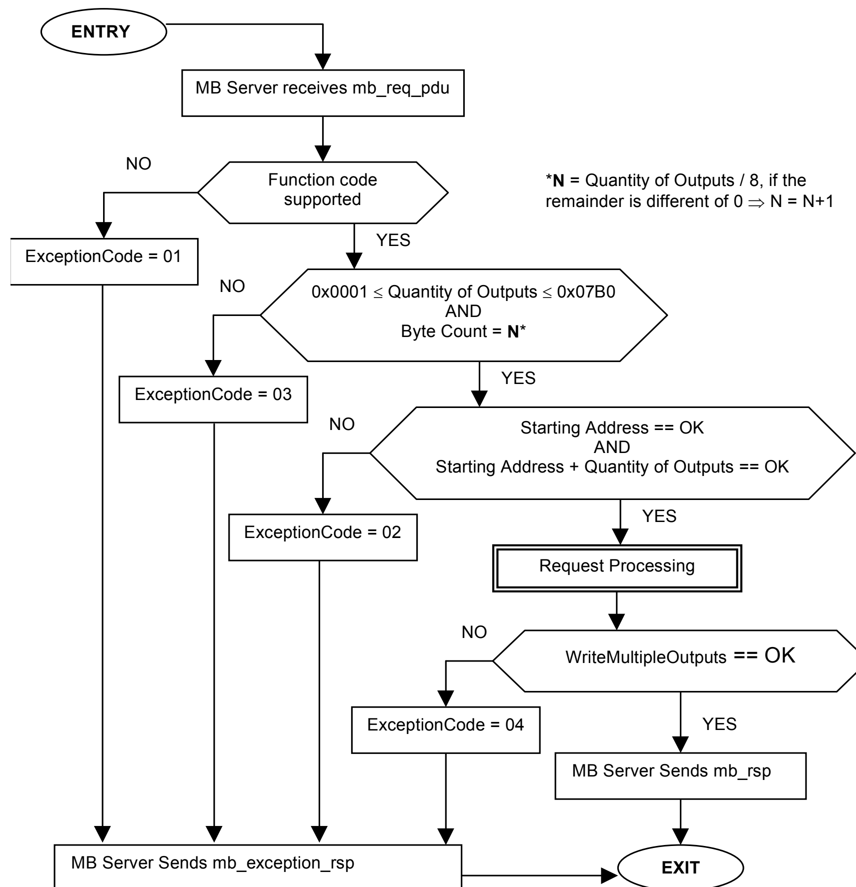


Figure 21 – Write Multiple Outputs state diagram

1.6.12 16 (0x10) Write Multiple registers

This function code is used to write a block of contiguous registers (1 to approx. 120 registers) in a remote device.

The requested written values are specified in the request data field. Data is packed as two bytes per register.

The normal response returns the function code, starting address, and quantity of registers written.

Request

Function code	1 Byte	**0x10**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	0x0001 to 0x0078
Byte Count	1 Byte	2 x **N***
Registers Value	**N*** x 2 Bytes	value

***N** = Quantity of Registers

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Response

Function code	1 Byte	**0x10**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 123 (0x7B)

Error

Error code	1 Byte	**0x90**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to write two registers starting at 2 to 00 0A and 01 02 hex:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**10**	Function	**10**
Starting Address Hi	**00**	Starting Address Hi	**00**
Starting Address Lo	**01**	Starting Address Lo	**01**
Quantity of Registers Hi	**00**	Quantity of Registers Hi	**00**
Quantity of Registers Lo	**02**	Quantity of Registers Lo	**02**
Byte Count	**04**		
Registers Value Hi	**00**		
Registers Value Lo	**0A**		
Registers Value Hi	**01**		
Registers Value Lo	**02**		

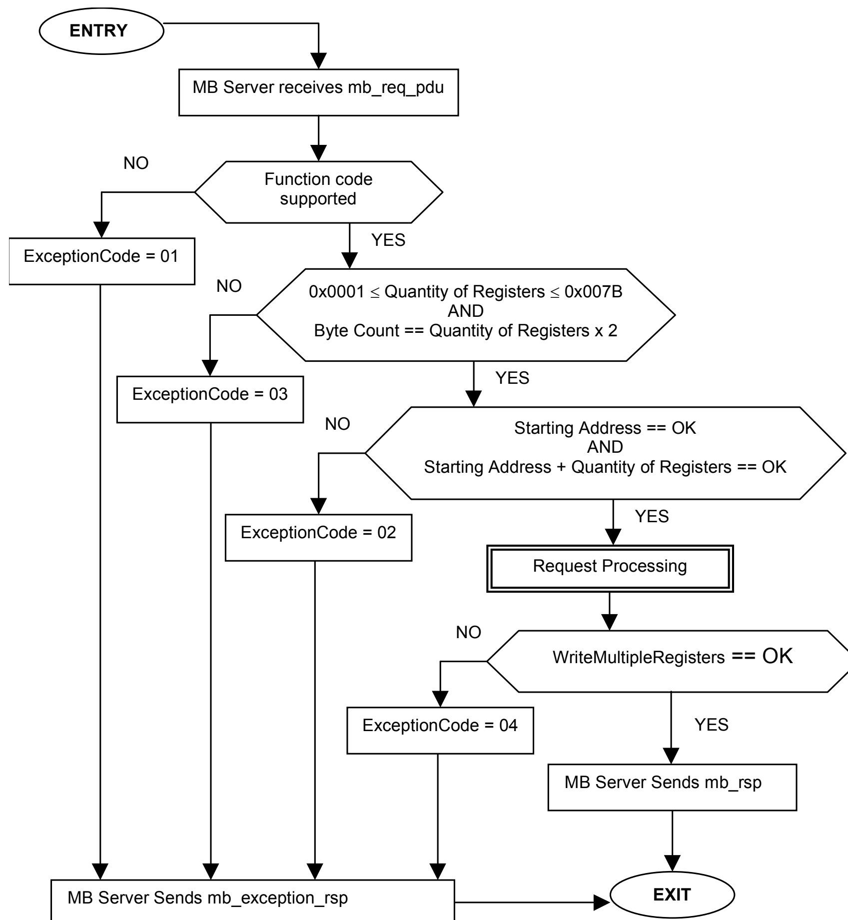


Figure 22 – Write Multiple Registers state diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.6.13 17 (0x11) Report Slave ID (Serial Line only)

This function code is used to read the description of the type, the current status, and other information specific to a remote device.

The format of a normal response is shown in the following example. The data contents are specific to each type of device.

Request

Function code	1 Byte	**0x11**

Response

Function code	1 Byte	**0x11**
Byte Count	1 Byte	
Slave ID	*device specific*	
Run Indicator Status	1 Byte	0x00 = OFF, 0xFF = ON
Additional Data		

Error

Error code	1 Byte	**0x91**
Exception code	1 Byte	01 or 04

Here is an example of a request to report the ID and status:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**11**	Function	**11**
		Byte Count	Device Specific
		Slave ID	Device Specific
		Run Indicator Status	0x00 or 0xFF
		Additional Data	Device Specific

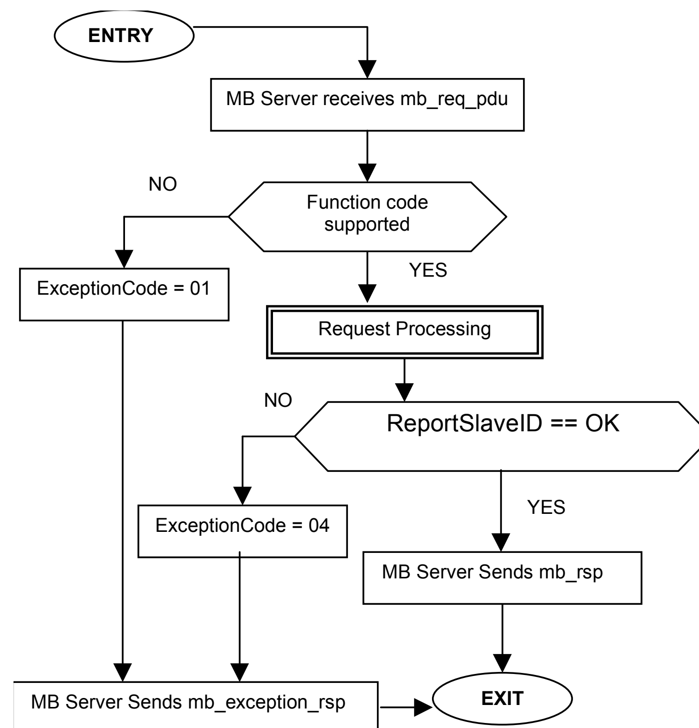


Figure 23 – Report slave ID state diagram

1.6.14 20 / 6 (0x14 / 0x06) Read File Record

This function code is used to perform a file record read. All Request Data Lengths are provided in terms of number of bytes and all Record Lengths are provided in terms of registers.

A file is an organization of records. Each file contains 10000 records, addressed 0000 to 9999 decimal or 0X0000 to 0X270F. For example, record 12 is addressed as 12.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The function can read multiple groups of references. The groups can be separating (non-contiguous), but the references within each group must be sequential.

Each group is defined in a separate 'sub-request' field that contains 7 bytes:

The reference type: 1 byte (must be specified as 6)

The File number: 2 bytes

The starting record number within the file: 2 bytes

The length of the record to be read: 2 bytes.

The quantity of registers to be read, combined with all other fields in the expected response, must not exceed the allowable length of the MODBUS PDU : 253 bytes.

The normal response is a series of 'sub-responses', one for each 'sub-request'. The byte count field is the total combined count of bytes in all 'sub-responses'. In addition, each 'sub-response' contains a field that shows its own byte count.

Request

Function code	1 Byte	**0x14**
Byte Count	1 Byte	0x07 to 0xF5 bytes
Sub-Req. x, Reference Type	1 Byte	06
Sub-Req. x, File Number	2 Bytes	0x0000 to 0xFFFF
Sub-Req. x, Record Number	2 Bytes	0x0000 to 0x270F
Sub-Req. x, Register Length	2 Bytes	**N**
Sub-Req. x+1, ...		

Response

Function code	1 Byte	**0x14**
Resp. data Length	1 Byte	0x07 to 0xF5
Sub-Req. x, File Resp. length	1 Byte	0x07 to 0xF5
Sub-Req. x, Reference Type	1 Byte	6
Sub-Req. x, Record Data	**N** x 2 Bytes	
Sub-Req. x+1, ...		

Error

Error code	1 Byte	**0x94**
Exception code	1 Byte	01 or 02 or 03 or 04 or 08

Here is an example of a request to read two groups of references from remote device:

- Group 1 consists of two registers from file 4, starting at register 1 (address 0001).
- Group 2 consists of two registers from file 3, starting at register 9 (address 0009).

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**14**	Function	**14**
Byte Count	**0E**	Resp. Data length	**0C**
Sub-Req. 1, Ref. Type	**06**	Sub-Req. 1, File resp. length	**05**
Sub-Req. 1, File Number Hi	**00**	Sub-Req. 1, Ref. Type	**06**
Sub-Req. 1, File Number Lo	**04**	Sub-Req. 1, Record. Data Hi	**0D**
Sub-Req. 1, Record number Hi	**00**	Sub-Req. 1, Record. Data Lo	**FE**
Sub-Req. 1, Record number Lo	**01**	Sub-Req. 1, Record. Data Hi	**00**
Sub-Req. 1, Record Length Hi	**00**	Sub-Req. 1, Record. Data Lo	**20**
Sub-Req. 1, Record Length Lo	**02**	Sub-Req. 2, File resp. length	**05**
Sub-Req. 2, Ref. Type	**06**	Sub-Req. 2, Ref. Type	**06**
Sub-Req. 2, File Number Hi	**00**	Sub-Req. 2, Record. Data Hi	**33**
Sub-Req. 2, File Number Lo	**03**	Sub-Req. 2, Record. Data Lo	**CD**
Sub-Req. 2, Record number Hi	**00**	Sub-Req. 2, Record. Data Hi	**00**
Sub-Req. 2, Record number Lo	**09**	Sub-Req. 2, Record. Data Lo	**40**
Sub-Req. 2, Record Length Hi	**00**		
Sub-Req. 2, Record Length Lo	**02**		

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

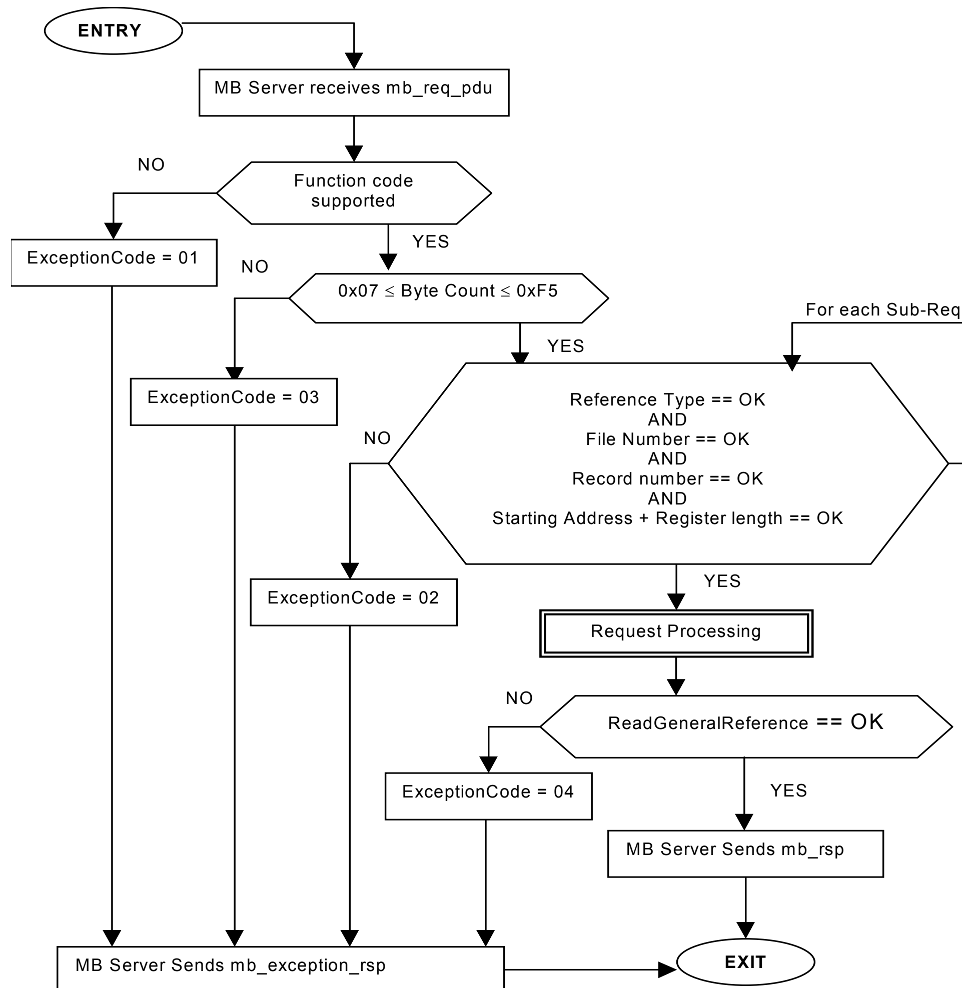


Figure 24 – Read File Record state diagram

1.6.15 21 / 6 (0x15 / 0x06) Write File Record

This function code is used to perform a file record write. All Request Data Lengths are provided in terms of number of bytes and all Record Lengths are provided in terms of the number of 16-bit words.

A file is an organization of records. Each file contains 10000 records, addressed 0000 to 9999 decimal or 0X0000 to 0X270F. For example, record 12 is addressed as 12.

The function can write multiple groups of references. The groups can be separate, ie non–contiguous, but the references within each group must be sequential.

Each group is defined in a separate 'sub-request' field that contains 7 bytes plus the data:

The reference type: 1 byte (must be specified as 6)

The file number: 2 bytes

The starting record number within the file: 2 bytes

The length of the record to be written: 2 bytes

The data to be written: 2 bytes per register.

The quantity of registers to be written, combined with all other fields in the request, must not exceed the allowable length of the MODBUS PDU : 253bytes.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The normal response is an echo of the request.

Request

Function code	1 Byte	**0x15**
Request data length	1 Byte	0x07 to 0xF5
Sub-Req. x, Reference Type	1 Byte	06
Sub-Req. x, File Number	2 Bytes	0x0000 to 0xFFFF
Sub-Req. x, Record Number	2 Bytes	0x0000 to 0x270F
Sub-Req. x, Record length	2 Bytes	**N**
Sub-Req. x, Record data	**N** x 2 Bytes	
Sub-Req. x+1, ...		

Response

Function code	1 Byte	**0x15**
Response Data length	1 Byte	
Sub-Req. x, Reference Type	1 Byte	06
Sub-Req. x, File Number	2 Bytes	0x0000 to 0xFFFFF
Sub-Req. x, Record number	2 Bytes	0x0000 to 0xFFFFF
Sub-Req. x, Record length	2 Bytes	0x0000 to 0xFFFFF **N**
Sub-Req. x, Record Data	**N** x 2 Bytes	
Sub-Req. x+1, ...		

Error

Error code	1 Byte	**0x95**
Exception code	1 Byte	01 or 02 or 03 or 04 or 08

Here is an example of a request to write one group of references into remote device:

- The group consists of three registers in file 4, starting at register 7 (address 0007).

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**15**	Function	**15**
Request Data length	**0D**	Request Data length	**0D**
Sub-Req. 1, Ref. Type	**06**	Sub-Req. 1, Ref. Type	**06**
Sub-Req. 1, File Number Hi	**00**	Sub-Req. 1, File Number Hi	**00**
Sub-Req. 1, File Number Lo	**04**	Sub-Req. 1, File Number Lo	**04**
Sub-Req. 1, Record number Hi	**00**	Sub-Req. 1, Record number Hi	**00**
Sub-Req. 1, Record number Lo	**07**	Sub-Req. 1, Record number Lo	**07**
Sub-Req. 1, Record length Hi	**00**	Sub-Req. 1, Record length Hi	**00**
Sub-Req. 1, Record length Lo	**03**	Sub-Req. 1, Record length Lo	**03**
Sub-Req. 1, Record Data Hi	**06**	Sub-Req. 1, Record Data Hi	**06**
Sub-Req. 1, Record Data Lo	**AF**	Sub-Req. 1, Record Data Lo	**AF**
Sub-Req. 1, Record Data Hi	**04**	Sub-Req. 1, Record Data Hi	**04**
Sub-Req. 1, Record Data Lo	**BE**	Sub-Req. 1, Record Data Lo	**BE**
Sub-Req. 1, Record Data Hi	**10**	Sub-Req. 1, Record Data Hi	**10**
Sub-Req. 1, Reg. Data Lo	**0D**	Sub-Req. 1, Reg. Data Lo	**0D**

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

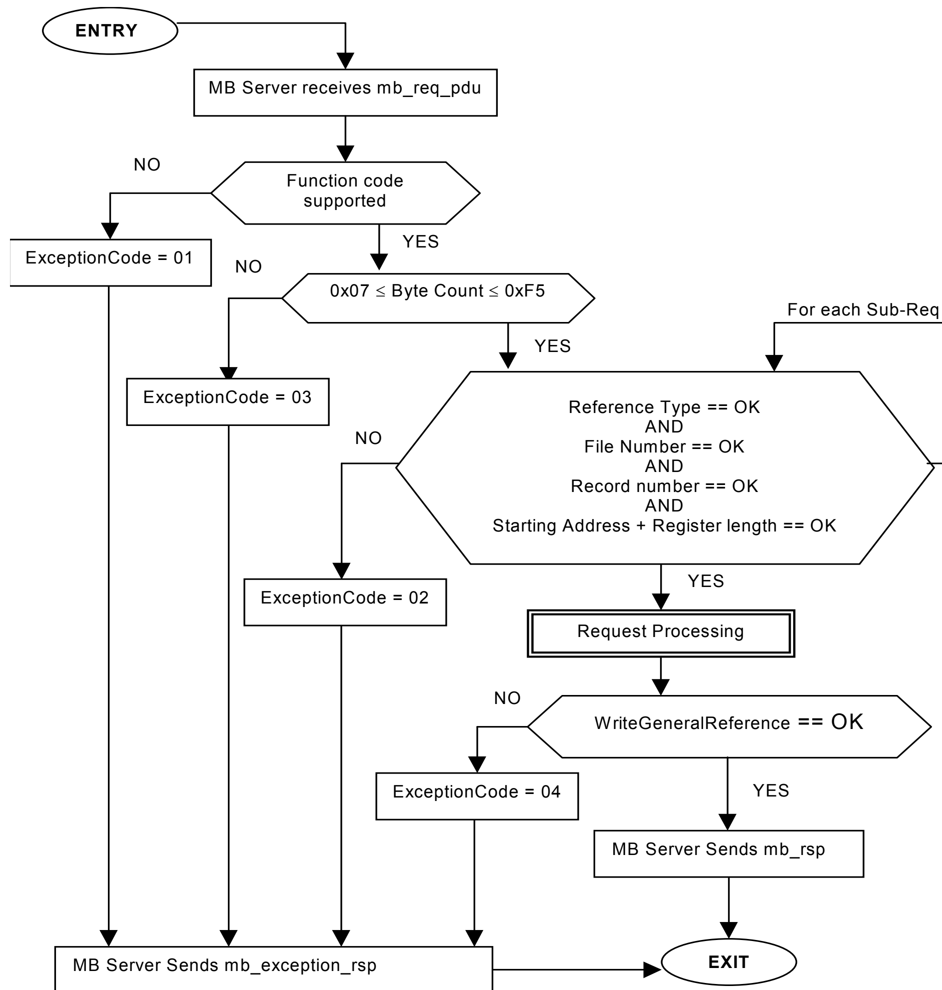


Figure 25 – Write File Record state diagram

1.6.16 22 (0x16) Mask Write Register

This function code is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the register's current contents. The function can be used to set or clear individual bits in the register.

The request specifies the holding register to be written, the data to be used as the AND mask, and the data to be used as the OR mask. Registers are addressed starting at zero. Therefore registers 1-16 are addressed as 0-15.

The function's algorithm is:

Result = (Current Contents AND And_Mask) OR (Or_Mask AND (NOT And_Mask))

For example:

	Hex	**Binary**	
Current Contents =		12	0001 0010
And_Mask =	F2	1111 0010	
Or_Mask =	25	0010 0101	
(NOT And_Mask) =	0D	0000 1101	
Result =	17	0001 0111	

NOTE 1 That if the Or_Mask value is zero, the result is simply the logical ANDing of the current contents and And_Mask. If the And_Mask value is zero, the result is equal to the Or_Mask value.

NOTE 2 The contents of the register can be read with the Read Holding Registers function (function code 03). They could, however, be changed subsequently as the controller scans its user logic program.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The normal response is an echo of the request. The response is returned after the register has been written.

Request

Function code	1 Byte	**0x16**
Reference Address	2 Bytes	0x0000 to 0xFFFF
And_Mask	2 Bytes	0x0000 to 0xFFFF
Or_Mask	2 Bytes	0x0000 to 0xFFFF

Response

Function code	1 Byte	**0x16**
Reference Address	2 Bytes	0x0000 to 0xFFFF
And_Mask	2 Bytes	0x0000 to 0xFFFF
Or_Mask	2 Bytes	0x0000 to 0xFFFF

Error

Error code	1 Byte	**0x96**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a Mask Write to register 5 in remote device, using the above mask values.

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**16**	Function	**16**
Reference address Hi	**00**	Reference address Hi	**00**
Reference address Lo	**04**	Reference address Lo	**04**
And_Mask Hi	**00**	And_Mask Hi	**00**
And_Mask Lo	**F2**	And_Mask Lo	**F2**
Or_Mask Hi	**00**	Or_Mask Hi	**00**
Or_Mask Lo	**25**	Or_Mask Lo	**25**

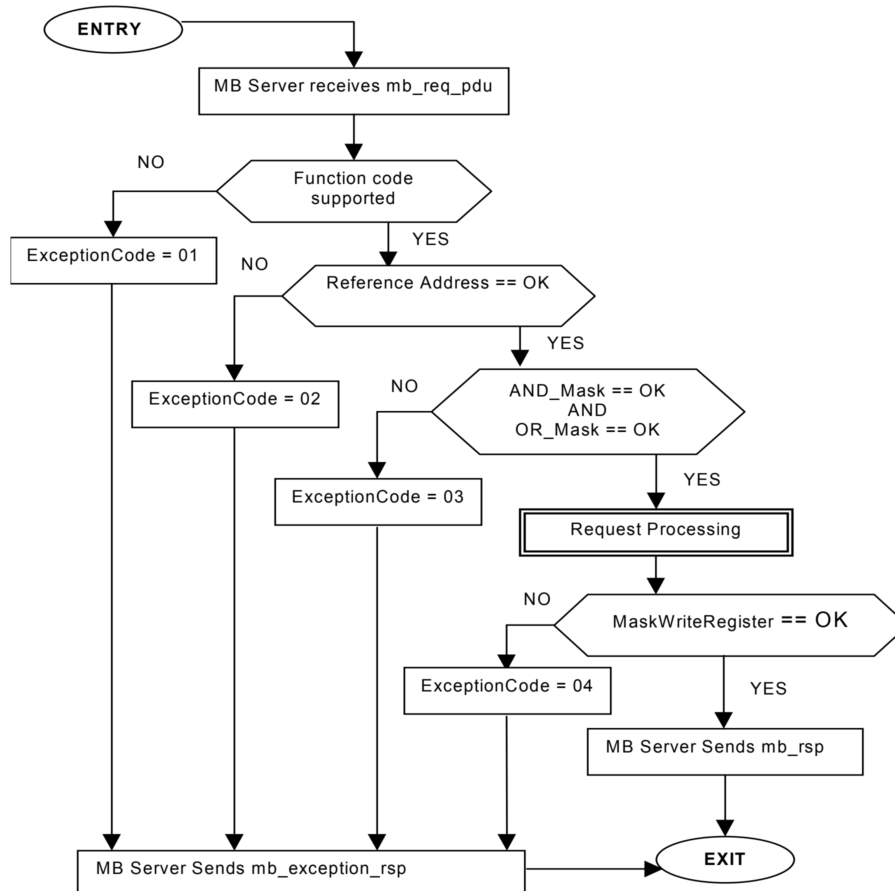


Figure 26 – Mask Write Holding Register state diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dhabi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.6.17 23 (0x17) Read/Write Multiple registers

This function code performs a combination of one read operation and one write operation in a single MODBUS transaction. The write operation is performed before the read.

Holding registers are addressed starting at zero. Therefore holding registers 1-16 are addressed in the PDU as 0-15.

The request specifies the starting address and number of holding registers to be read as well as the starting address, number of holding registers, and the data to be written. The byte count specifies the number of bytes to follow in the write data field.

The normal response contains the data from the group of registers that were read. The byte count field specifies the quantity of bytes to follow in the read data field.

Request

Function code	1 Byte	**0x17**
Read Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity to Read	2 Bytes	0x0001 to approx. 0x0076
Write Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity to Write	2 Bytes	0x0001 to approx. 0X0076
Write Byte Count	1 Byte	2 x **N***
Write Registers Value	**N***x 2 Bytes	

***N** = Quantity to Write

Response

Function code	1 Byte	**0x17**
Byte Count	1 Byte	2 x **N'***
Read Registers value	**N'*** x 2 Bytes	

***N'** = Quantity to Read

Error

Error code	1 Byte	**0x97**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of a request to read six registers starting at register 4, and to write three registers starting at register 15:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**17**	Function	**17**
Read Starting Address Hi	**00**	Byte Count	**0C**
Read Starting Address Lo	**03**	Read Registers value Hi	**00**
Quantity to Read Hi	**00**	Read Registers value Lo	**FE**
Quantity to Read Lo	**06**	Read Registers value Hi	**0A**
Write Starting Address Hi	**00**	Read Registers value Lo	**CD**
Write Starting address Lo	**0E**	Read Registers value Hi	**00**
Quantity to Write Hi	**00**	Read Registers value Lo	**01**
Quantity to Write Lo	**03**	Read Registers value Hi	**00**
Write Byte Count	**06**	Read Registers value Lo	**03**
Write Registers Value Hi	**00**	Read Registers value Hi	**00**
Write Registers Value Lo	**FF**	Read Registers value Lo	**0D**
Write Registers Value Hi	**00**	Read Registers value Hi	**00**
Write Registers Value Lo	**FF**	Read Registers value Lo	**FF**
Write Registers Value Hi	**00**		
Write Registers Value Lo	**FF**		

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

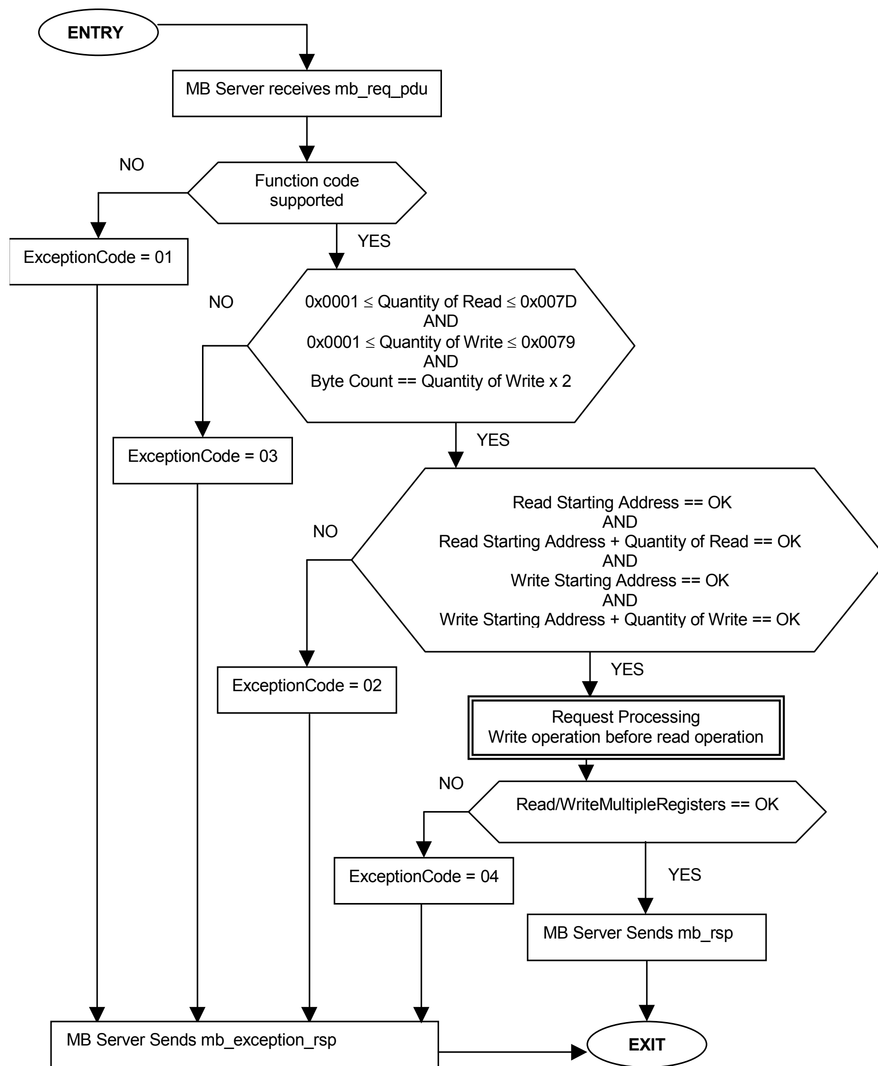


Figure 27 – Read/Write Multiple Registers state diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.6.18 24 (0x18) Read FIFO Queue

This function code allows to read the contents of a First-In-First-Out (FIFO) queue of register in a remote device. The function returns a count of the registers in the queue, followed by the queued data. Up to 32 registers can be read: the count, plus up to 31 queued data registers. The queue count register is returned first, followed by the queued data registers.

The function reads the queue contents, but does not clear them.

In a normal response, the byte count shows the quantity of bytes to follow, including the queue count bytes and value register bytes (but not including the error check field).

The queue count is the quantity of data registers in the queue (not including the count register).

If the queue count exceeds 31, an exception response is returned with an error code of 03 (Illegal Data Value).

Request

Function code	1 Byte	**0x18**
FIFO Pointer Address	2 Bytes	0x0000 to 0xFFFF

Response

Function code	1 Byte	**0x18**
Byte Count	2 Bytes	
FIFO Count	2 Bytes	≤ 31
FIFO Value Register	**N*** x 2 Bytes	

*__N__ = FIFO Count

Error

Error code	1 Byte	**0x98**
Exception code	1 Byte	01 or 02 or 03 or 04

Here is an example of Read FIFO Queue request to remote device. The request is to read the queue starting at the pointer register 1246 (0x04DE):

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**18**	Function	**18**
FIFO Pointer Address Hi	**04**	Byte Count Hi	**00**
FIFO Pointer Address Lo	**DE**	Byte Count Lo	**06**
		FIFO Count Hi	**00**
		FIFO Count Lo	**02**
		FIFO Value Register Hi	**01**
		FIFO Value Register Lo	**B8**
		FIFO Value Register Hi	**12**
		FIFO Value Register Lo	**84**

In this example, the FIFO pointer register (1246 in the request) is returned with a queue count of 2. The two data registers follow the queue count. These are:

1247 (contents 440 decimal -- 0x01B8); and 1248 (contents 4740 -- 0x1284).

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

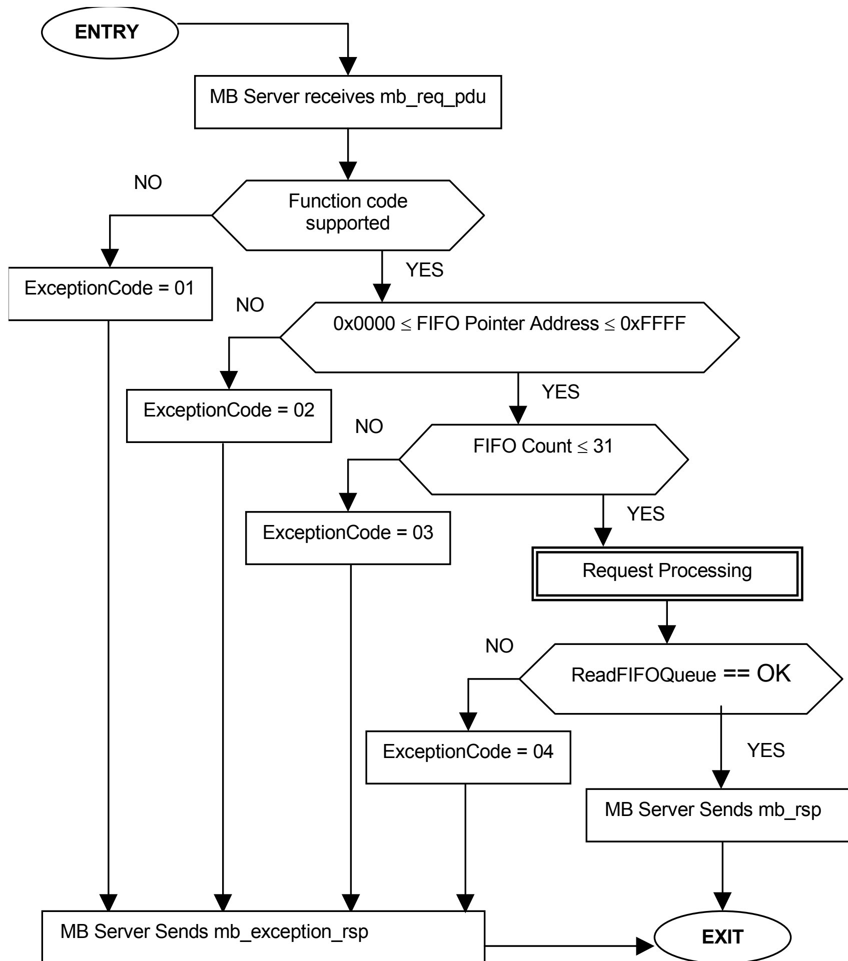


Figure 28 – Read FIFO Queue state diagram

1.6.19 43 (0x2B) Encapsulated Interface Transport

NOTE The user should refer to Annex B: MODBUS RESERVED FUNCTION CODES, SUBCODES AND MEI TYPES.

Function Code 43 and its MEI Type 14 for Device Identification is one of two Encapsulated Interface Transport currently available in this PAS. The following function codes and MEI Types shall not be part of the IEC published Specification derived from this PAS and these function codes and MEI Types are specifically reserved: 43/0-12 and 43/15-255.

The MODBUS Encapsulated Interface (MEI)Transport is a mechanism for tunneling service requests and method invocations, as well as their returns, inside MODBUS PDUs.

The primary feature of the MEI Transport is the encapsulation of method invocations or service requests that are part of a defined interface as well as method invocation returns or service responses.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

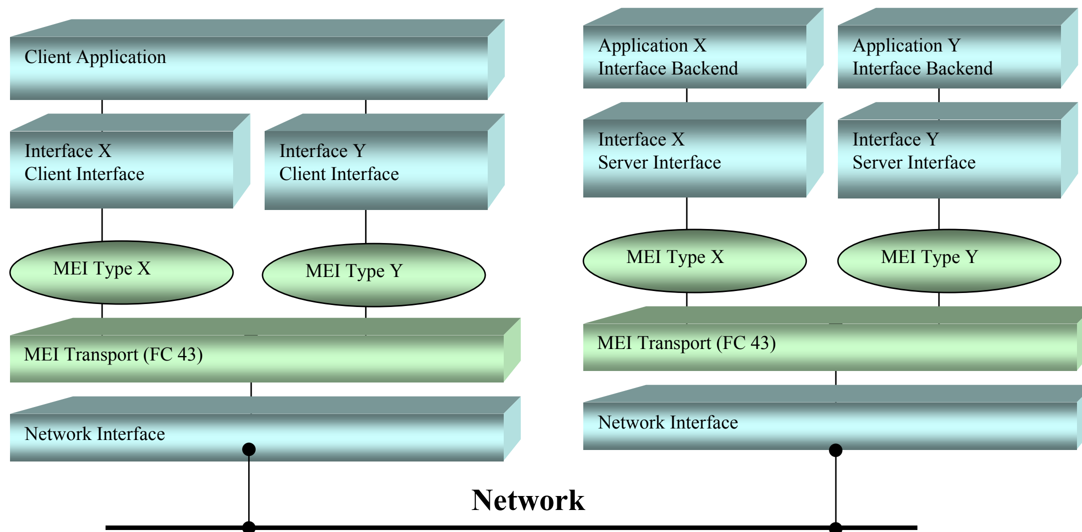


Figure 29 – MODBUS encapsulated Interface Transport

The **Network Interface** can be any communication stack used to send MODBUS PDUs, such as TCP/IP, or serial line.

A **MEI Type** is a MODBUS Assigned Number and therefore will be unique, the value between 0 to 255 are Reserved according to Annex B except for MEI Type 13 and MEI Type 14.

The MEI Type is used by MEI Transport implementations to dispatch a method invocation to the indicated interface.

Since the MEI Transport service is interface agnostic, any specific behavior or policy required by the interface must be provided by the interface, e.g. MEI transaction processing, MEI interface error handling, etc.

Request

Function code	1 Byte	**0x2B**
MEI Type*	1 Byte	0x0E
MEI type specific data	n Bytes	

* MEI = MODBUS Encapsulated Interface

Response

Function code	1 Byte	**0x2B**
MEI Type	1 byte	0x0E
MEI type specific data	n Bytes	

Error

Function code	1 Byte	**0xAB :** **Fc 0x2B + 0x80**
MEI Type	1 Byte	0x0E
Exception code	1 Byte	01, 02, 03, 04

As an example see Read device identification request.

1.6.20 43 / 13 (0x2B / 0x0D) CANopen General Reference Request and Response PDU

The CANopen General reference Command is an encapsulation of the services that will be used to access (read from or write to) the entries of a CAN-Open Device Object Dictionary as well as controlling and monitoring the server device, the CANopen system, and devices.

The MEI Type 13 (0x0D) is a MODBUS Assigned Number licensed to CiA for the CANopen General Reference.

The system is intended to work within the limitations of existing MODBUS networks. Therefore, the information needed to query or modify the object dictionaries in the system is

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

mapped into the format of a MODBUS message. The command will have the 256 Byte limitation in both the Request and the Response message.

Informative: Please refere to Annex C for a reference to a specification that provides information on MEI Type 13.

1.6.21 43 / 14 (0x2B / 0x0E) Read Device Identification

This function code allows reading the identification and additional information relative to the physical and functional description of a remote device, only.

The Read Device Identification interface is modeled as an address space composed of a set of addressable data elements. The data elements are called objects and an object Id identifies them.

The interface consists of 3 categories of objects :

- Basic Device Identification. All objects of this category are mandatory : VendorName, Product code, and revision number.
- Regular Device Identification. In addition to Basic data objects, the device provides additional and optional identification and description data objects. All of the objects of this category are defined in the standard but their implementation is optional .
- Extended Device Identification. In addition to regular data objects, the device provides additional and optional identification and description private data about the physical device itself. All of these data are device dependent.

Object Id	Object Name / Description	Type	M/O	category
0x00	VendorName	ASCII String	**Mandatory**	**Basic**
0x01	ProductCode	ASCII String	**Mandatory**	
0x02	MajorMinorRevision	ASCII String	**Mandatory**	
0x03	VendorUrl	ASCII String	Optional	**Regular**
0x04	ProductName	ASCII String	Optional	
0x05	ModelName	ASCII String	Optional	
0x06	UserApplicationName	ASCII String	Optional	
0x07 … 0x7F	*Reserved*		Optional	
0x80 … 0xFF	*Private objects may be* ***optionally*** *defined.* *The range [0x80 – 0xFF] is Product dependant.*	device dependant	Optional	**Extended**

Request

Function code	1 Byte	**0x2B**
MEI Type*	1 Byte	0x0E
Read Device ID code	1 Byte	01 / 02 / 03 / 04
Object Id	1 Byte	0x00 to 0xFF

* MEI = MODBUS Encapsulated Interface

Response

Function code	1 Byte	**0x2B**
MEI Type	1 byte	0x0E
Read Device ID code	1 Byte	01 / 02 / 03 / 04
Conformity level	1 Byte	
More Folows	1 Byte	00 / FF
Next Object Id	1 Byte	Object ID number
Number of objects	1 Byte	
List Of		
Object ID	1 Byte	
Object length	1 Byte	
Object Value	Object length	Depending on the object ID

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Error

Function code	1 Byte	**0xAB :** **Fc 0x2B + 0x80**
MEI Type	1 Byte	0x0E
Exception code	1 Byte	01, 02, 03, 04

Request parameters description :

A MODBUS Encapsulated Interface assigned number 14 identifies the Read identification request.

The paremeter " Read Device ID code " allows to define four access types :

01: request to get the basic device identification (stream access)
02: request to get the regular device identification (stream access)
03: request to get the extended device identification (stream access)
04: request to get one specific identification object (individual access)

An exception code 03 is sent back in the response if the Read device ID code is illegal.

In case of a response that does not fit into a single response, several transactions (request/response) must be done. The Object Id byte gives the identification of the first object to obtain. For the first transaction, the client must set the Object Id to 0 to obtain the beginning of the device identification data. For the following transactions, the client must set the Object Id to the value returned by the server in its previous response.

Remark : An object is indivisible, therefore any object must have a size consistent with the size of transaction response.

If the Object Id does not match any known object, the server responds as if object 0 were pointed out (restart at the beginning).

In case of an individual access: ReadDevId code 04*,* the Object Id in the request gives the identification of the object to obtain, and if the Object Id doesn't match to any known object, the server returns an exception response with exception code = 02 (Illegal data address).

If the server device is asked for a description level (readDevice Code) higher that its conformity level , It must respond in accordance with its actual conformity level.

Response parameter description :

Function code : Function code 43 (decimal) 0x2B (hex)

MEI Type 14 (0x0E) MEI Type assigned number for Device Identification Interface

ReadDevId code : Same as request ReadDevId code : 01, 02, 03 or 04

Conformity Level Identification conformity level of the device and type of supported access
01 : basic identification (stream access only)
02 : regular identification (stream access only)
03 : extended identification (stream access only)
81 : basic identification (stream access and individual access)
82 : regular identification (stream access and individual access)
83 : extended identification (stream access and individual access)

More Follows ***In case of ReadDevId codes 01, 02 or 03 (stream access),***
If the identification data doesn't fit into a single response, several request/response transactions may be required.
00 : no more Object are available
FF : other identification Object are available and further MODBUS transactions are required
In case of ReadDevId code 04 (individual access),
this field must be set to 00.

Next Object Id If "MoreFollows = FF", identification of the next Object to be asked for.
If "MoreFollows = 00", must be set to 00 (useless)

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Number Of Objects	Number of identification Object returned in the response (for an individual access, Number Of Objects = 1)
Object0.Id	Identification of the first Object returned in the PDU (stream access) or the requested Object (individual access)
Object0.Length	Length of the first Object in byte
Object0.Value	Value of the first Object (Object0.Length bytes)
…	
ObjectN.Id	Identification of the last Object (within the response)
ObjectN.Length	Length of the last Object in byte
ObjectN.Value	Value of the last Object (ObjectN.Length bytes)

Example of a Read Device Identification request for "Basic device identification" : In this example all information are sent in one response PDU.

Request		**Response**	
Field Name	*Value*	*Field Name*	*Value*
Function	**2B**	Function	**2B**
MEI Type	**0E**	MEI Type	**0E**
Read Dev Id code	**01**	Read Dev Id Code	**01**
Object Id	**00**	Conformity Level	**01**
		More Follows	**00**
		NextObjectId	**00**
		Number Of Objects	**03**
		Object Id	**00**
		Object Length	**16**
		Object Value	**" Company identification"**
		Object Id	**01**
		Object Length	**0D**
		Object Value	**" Product code XX"**
		Object Id	**02**
		Object Length	**05**
		Object Value	**"V2.11"**

In case of a device that required several transactions to send the response the following transactions is intiated.

First transaction :

Request		**Response**	
Field Name	*Value*	*Field Name*	*Value*
Function	**2B**	Function	**2B**
MEI Type	**0E**	MEI Type	**0E**
Read Dev Id code	**01**	Read Dev Id Code	**01**
Object Id	**00**	Conformity Level	**01**
		More Follows	**FF**
		NextObjectId	**02**
		Number Of Objects	**03**
		Object Id	**00**
		Object Length	**16**
		Object Value	**" Company identification"**
		Object Id	**01**
		Object Length	**1C**
		Object Value	**" Product code XXXXXXXXXXXXXXXX"**

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Second transaction :

Request		Response	
Field Name	*Value*	*Field Name*	*Value*
Function	**2B**	Function	**2B**
MEI Type	**0E**	MEI Type	**0E**
Read Dev Id code	**01**	Read Dev Id Code	**01**
Object Id	**02**	Conformity Level	**01**
		More Follows	**00**
		NextObjectId	**00**
		Number Of Objects	**03**
		Object Id	**02**
		Object Length	**05**
		Object Value	**"V2.11"**

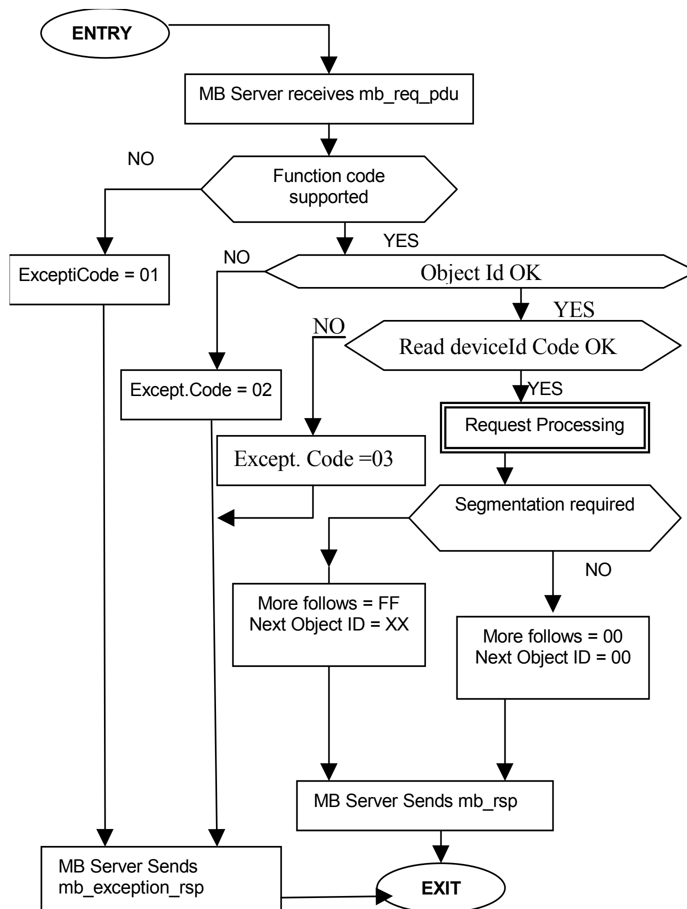


Figure 30 – Read Device Identification state diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

1.7 MODBUS Exception Responses

When a client device sends a request to a server device it expects a normal response. One of four possible events can occur from the master's query:

- If the server device receives the request without a communication error, and can handle the query normally, it returns a normal response.
- If the server does not receive the request due to a communication error, no response is returned. The client program will eventually process a timeout condition for the request.
- If the server receives the request, but detects a communication error (parity, LRC, CRC, ...), no response is returned. The client program will eventually process a timeout condition for the request.
- If the server receives the request without a communication error, but cannot handle it (for example, if the request is to read a non–existent output or register), the server will return an exception response informing the client of the nature of the error.

The exception response message has two fields that differentiate it from a normal response:

Function Code Field: In a normal response, the server echoes the function code of the original request in the function code field of the response. All function codes have a most–significant bit (MSB) of 0 (their values are all below 80 hexadecimal). In an exception response, the server sets the MSB of the function code to 1. This makes the function code value in an exception response exactly 80 hexadecimal higher than the value would be for a normal response.

With the function code's MSB set, the client's application program can recognize the exception response and can examine the data field for the exception code.

Data Field: In a normal response, the server may return data or statistics in the data field (any information that was requested in the request). In an exception response, the server returns an exception code in the data field. This defines the server condition that caused the exception.

Example of a client request and server exception response

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Function	**01**	Function	**81**
Starting Address Hi	**04**	Exception Code	**02**
Starting Address Lo	**A1**		
Quantity of Outputs Hi	**00**		
Quantity of Outputs Lo	**01**		

In this example, the client addresses a request to server device. The function code (01) is for a Read Output Status operation. It requests the status of the output at address 1245 (04A1 hex). Note that only that one output is to be read, as specified by the number of outputs field (0001).

If the output address is non–existent in the server device, the server will return the exception response with the exception code shown (02). This specifies an illegal data address for the slave.

A listing of exception codes begins on the next page.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

MODBUS Exception Codes		
Code	**Name**	**Meaning**
01	ILLEGAL FUNCTION	The function code received in the query is not an allowable action for the server (or slave). This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server (or slave) is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values.
02	ILLEGAL DATA ADDRESS	The data address received in the query is not an allowable address for the server (or slave). More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed, a request with offset 96 and length 5 will generate exception 02.
03	ILLEGAL DATA VALUE	A value contained in the query data field is not an allowable value for server (or slave). This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does NOT mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the MODBUS protocol is unaware of the significance of any particular value of any particular register.
04	SLAVE DEVICE FAILURE	An unrecoverable error occurred while the server (or slave) was attempting to perform the requested action.
05	ACKNOWLEDGE	Specialized use in conjunction with programming commands. The server (or slave) has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the client (or master). The client (or master) can next issue a Poll Program Complete message to determine if processing is completed.
06	SLAVE DEVICE BUSY	Specialized use in conjunction with programming commands. The server (or slave) is engaged in processing a long–duration program command. The client (or master) should retransmit the message later when the server (or slave) is free.
08	MEMORY PARITY ERROR	Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server (or slave) attempted to read record file, but detected a parity error in the memory. The client (or master) can retry the request, but service may be required on the server (or slave) device.
0A	GATEWAY PATH UNAVAILABLE	Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Annex A of Section 1
(informative)

MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE

A.1 INTRODUCTION

A.1.1 OBJECTIVES

The objective of this document is to present the MODBUS messaging service over TCP/IP , in order to provide reference information that helps software developers to implement this service. The encoding of all MODBUS function codes are not described in this document, for this information please read Part of this Specification.

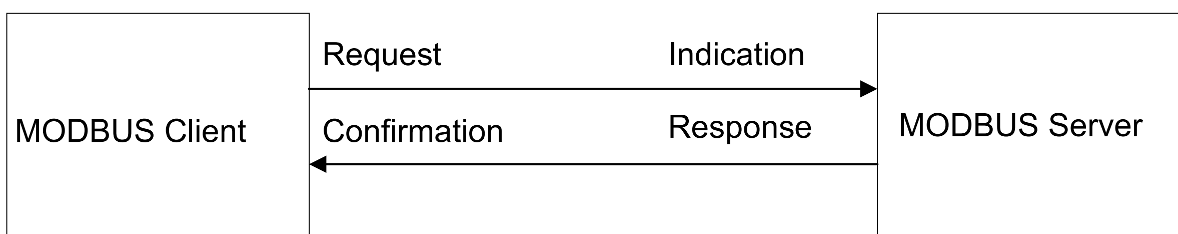
This document gives accurate and comprehensive description of a MODBUS messaging service implementation. Its purpose is to facilitate the interoperability between the devices using the MODBUS messaging service.

This document comprises mainly three parts:
- An overview of the MODBUS over TCP/IP protocol
- A functional description of a MODBUS client, server and gateway implementation
- An implementation guideline that proposes the object model of an MODBUS implementation example.

A.1.2 CLIENT / SERVER MODEL

The MODBUS messaging service provides a Client/Server communication between devices connected on an Ethernet TCP/IP network.
This client / server model is based on four type of messages:

- **MODBUS Request,**
- **MODBUS Confirmation,**
- **MODBUS Indication,**
- **MODBUS Response**



A MODBUS Request is the message sent on the network by the Client to initiate a transaction,

A MODBUS Indication is the Request message received on the Server side,

A MODBUS Response is the Response message sent by the Server,

A MODBUS Confirmation is the Response Message received on the Client side

The MODBUS messaging services (Client / Server Model) are used for real time information exchange:

- between two device applications,
- between device application and other device,
- between HMI/SCADA applications and devices,
- between a PC and a device program providing on line services.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.1.3 REFERENCE DOCUMENTS

Before reading this annex, it would be useful to consult the following:

1) Clause 1 of Section 1.
2) RFC 1122 Requirements for Internet Hosts – Communication Layers

A.2 ABBREVIATIONS

ADU	Application Data Unit
IETF	Internet Engineering Task Force
IP	Internet Protocol
MAC	Medium Access Control
MB	MODBUS
MBAP	MODBUS Application Protocol
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
TCP	Transport Control Protocol
BSD	Berkeley Software Distribution
MSL	Maximum Segment Lifetime

A.3 CONTEXT

A.3.1 PROTOCOL DESCRIPTION

A.3.1.1 General communication architecture

A communicating system over MODBUS TCP/IP may include different types of device:

- A MODBUS TCP/IP Client and Server devices connected to a TCP/IP network
- The Interconnection devices like bridge, router or gateway for interconnection between the TCP/IP network and a serial line sub-network which permit connections of MODBUS Serial line Client and Server end devices.

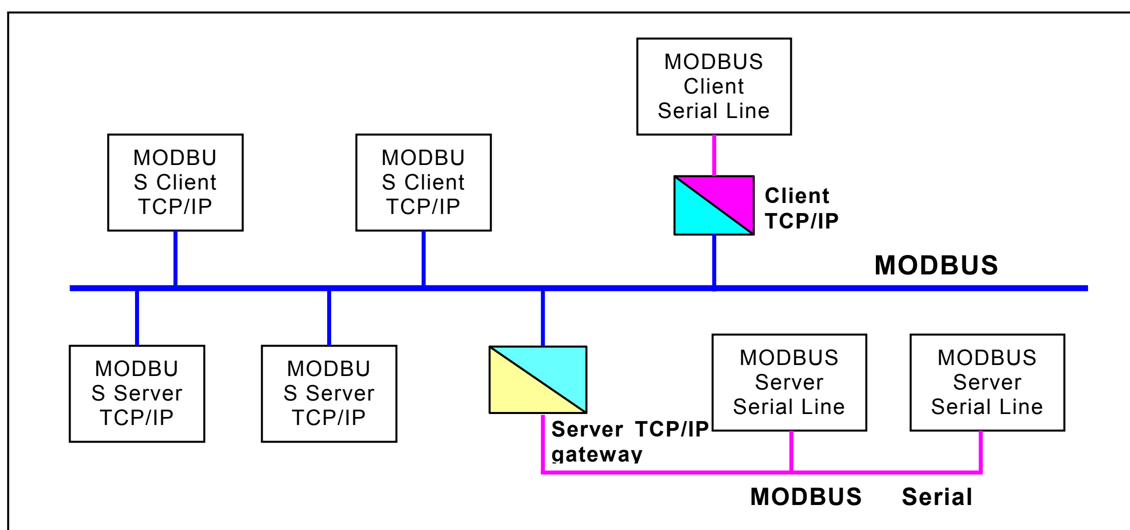


Figure A.1 – MODBUS TCP/IP communication architecture

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The MODBUS protocol defines a **simple Protocol Data Unit (PDU)** independent of the underlying communication layers. The mapping of MODBUS protocol on specific buses or networks can introduce some additional fields on the **Application Data Unit (ADU)**.

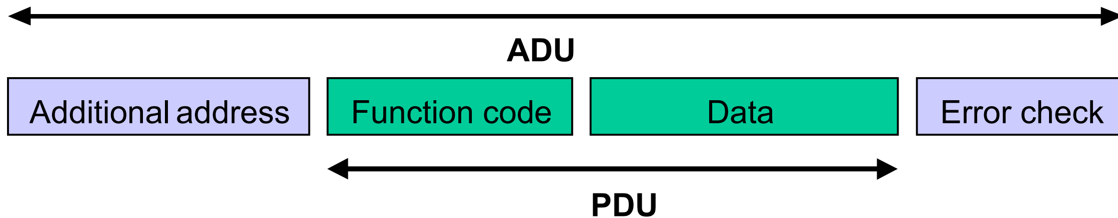


Figure A.2 – General MODBUS frame

The client that initiates a MODBUS transaction builds the MODBUS Application Data Unit. The function code indicates to the server which kind of action to perform.

A.3.1.2 MODBUS On TCP/IP Application Data Unit

This subclause describes the encapsulation of a MODBUS request or response when it is carried on a MODBUS TCP/IP network.

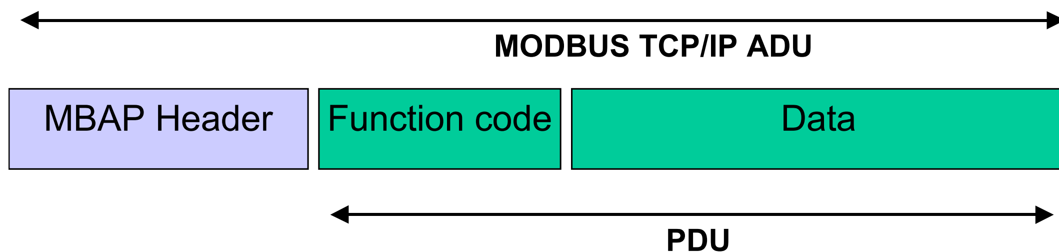


Figure A.3 – MODBUS request/response over TCP/IP

A dedicated header is used on TCP/IP to identify the MODBUS Application Data Unit. It is called the MBAP header (MODBUS Application Protocol header).

This header provides some differences compared to the MODBUS RTU application data unit used on serial line:

- The MODBUS ‘slave address’ field usually used on MODBUS Serial Line is replaced by a single byte ‘Unit Identifier’ within the MBAP Header. The ‘Unit Identifier’ is used to communicate via devices such as bridges, routers and gateways that use a single IP address to support multiple independent MODBUS end units.
- All MODBUS requests and responses are designed in such a way that the recipient can verify that a message is finished. For function codes where the MODBUS PDU has a fixed length, the function code alone is sufficient. For function codes carrying a variable amount of data in the request or response, the data field includes a byte count.
- When MODBUS is carried over TCP, additional length information is carried in the MBAP header to allow the recipient to recognize message boundaries even if the message has been split into multiple packets for transmission. The existence of explicit and implicit length rules, and use of a CRC-32 error check code (on Ethernet) results in an infinitesimal chance of undetected corruption to a request or response message.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.3.1.3 MBAP Header description

The MBAP Header contains the following fields:

Fields	Length	Description -	Client	Server
Transaction Identifier	2 Bytes	Identification of a MODBUS Request / Response transaction.	Initialized by the client	Recopied by the server from the received request
Protocol Identifier	2 Bytes	0 = MODBUS protocol	Initialized by the client	Recopied by the server from the received request
Length	2 Bytes	Number of following bytes	Initialized by the client (request)	Initialized by the server (Response)
Unit Identifier	1 Byte	Identification of a remote slave connected on a serial line or on other buses.	Initialized by the client	Recopied by the server from the received request

The header is 7 bytes long:

- **Transaction Identifier** - It is used for transaction pairing, the MODBUS server copies in the response the transaction identifier of the request.
- **Protocol Identifier** – It is used for intra-system multiplexing. The MODBUS protocol is identified by the value 0.
- **Length** - The length field is a byte count of the following fields, including the Unit Identifier and data fields.
- **Unit Identifier** – This field is used for intra-system routing purpose. It is typically used to communicate to a MODBUS or a MODBUS+ serial line slave through a gateway between an Ethernet TCP-IP network and a MODBUS serial line. This field is set by the MODBUS Client in the request and must be returned with the same value in the response by the server.

All MODBUS/TCP ADU are sent via TCP on registered port 502.

Remark : the different fields are encoded in Big-endian.

A.3.2 MODBUS FUNCTIONS CODES DESCRIPTION

Standard function codes used on MODBUS application layer protocol are described in details of Clause 1 of this Specification.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.4 FUNCTIONAL DESCRIPTION

The MODBUS Component Architecture presented here is a general model including both MODBUS Client and Server Components and usable on any device.

Some devices may only provide the server or the client component.

In the first part of this clause, a brief overview of the MODBUS messaging service component architecture is given, followed by a description of each component presented in the architectural model.

A.4.1 MODBUS COMPONENT ARCHITECTURE MODEL

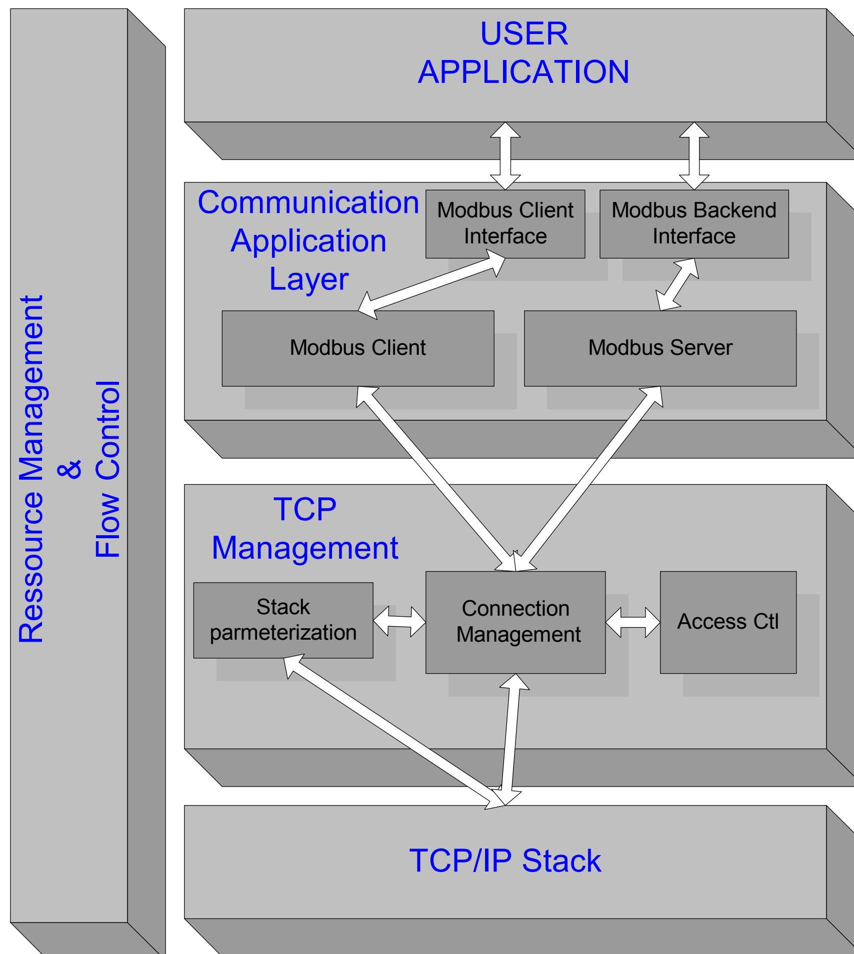


Figure A.4 – MODBUS Messaging Service Conceptual Architecture

- **Communication Application Layer**

A MODBUS device may provide a client and/or a server MODBUS interface.

A MODBUS backend interface can be provided allowing indirectly the access to user application objects.
Four areas can compose this interface: input discrete, output discrete (coils), input registers and output registers. A pre-mapping between this interface and the user application data has to be done (local issue).

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dhabi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

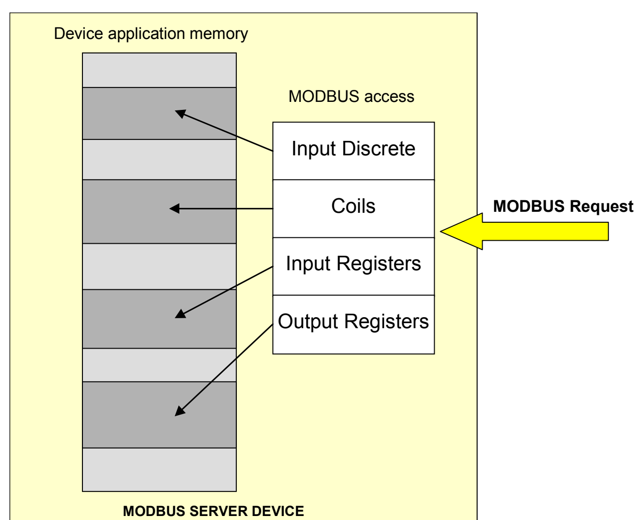


Figure A.5– MODBUS Data Model with separate blocks

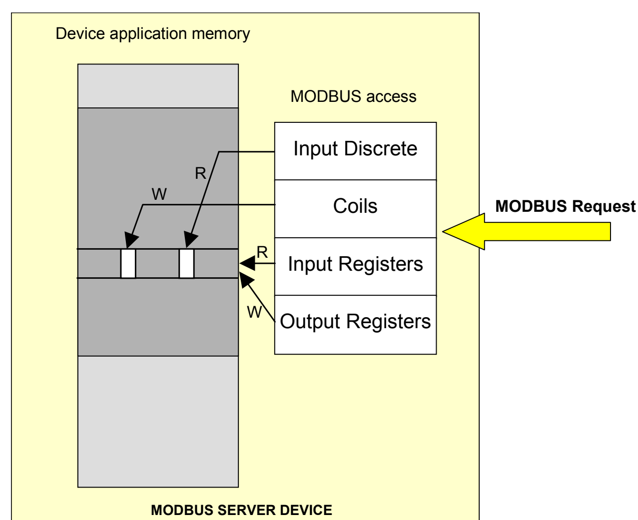


Figure A.6 –MODBUS Data Model with only 1 block

- MODBUS Client

The MODBUS Client allows the user application to explicitly control information exchange with a remote device. The MODBUS Client builds a MODBUS request from parameter contained in a demand sent by the user application to the MODBUS Client Interface.
The MODBUS Client uses a MODBUS transaction whose management includes waiting for and processing of a MODBUS confirmation.

- MODBUS Client Interface

The MODBUS Client Interface provides an interface enabling the user application to build the requests for various MODBUS services including access to MODBUS application objects. The MODBUS Client interface (API) is not part of this PAS, although an example is described in the implementation model.

- MODBUS Server

On reception of a MODBUS request this module actives a local action to read, to write or to achieve some other actions. The processing of these actions is done totally transparently for the application programmer. The main MODBUS server functions are to wait for a MODBUS request on 502 TCP port, to treat this request and then to build a MODBUS response depending on device context.

- MODBUS Backend Interface

The MODBUS Backend Interface is an interface from the MODBUS Server to the user application in which the application objects are defined.

NOTE The Backend Interface is not defined in this Specification

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

- **TCP Management layer**

NOTE The TCP/IP discussion in this Annex A is based in part upon reference RFC 1122 to assist the user in implementing Clause 1 of this section 1 over TCP/IP.

One of the main functions of the messaging service is to manage communication establishment and ending and to manage the data flow on established TCP connections.

➢ Connection Management

A communication between a client and server MODBUS Module requires the use of a TCP connection management module. It is in charge to manage globally messaging TCP connections.

Two possibilities are proposed for the connection management. Either the user application itself manages TCP connections or the connection management is totally done by this module and therefore it is transparent for the user application. The last solution implies less flexibility.

The **listening TCP port 502 is reserved for MODBUS** communications. It is mandatory to listen by default on that port. However, some markets or applications might require that another port is dedicated to MODBUS over TCP. For that reason, it is highly recommended that the clients and the servers give the possibility to the user to parameterize the MODBUS over TCP port number. **It is important to note that even if another TCP server port is configured for MODBUS service in certain applications, TCP server port 502 must still be available in addition to any application specific ports.**

➢ Access Control Module

In certain critical contexts, accessibility to internal data of devices must be forbidden for undesirable hosts. That is why a security mode is needed and security process may be implemented if required.

- **TCP/IP Stack layer**

The TCP/IP stack can be parameterized in order to adapt the data flow control, the address management and the connection management to different constraints specific to a product or to a system. Generally the BSD socket interface is used to manage the TCP connections.

▪ **Resource management and Data flow control**

In order to equilibrate inbound and outbound messaging data flow between the MODBUS client and the server, data flow control mechanism is provided in all layers of MODBUS messaging stack.
The resource management and flow control module is first based on TCP internal flow control added with some data flow control in the data link layer and also in the user application level.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.4.2 TCP CONNECTION MANAGEMENT

A.4.2.1 Connections management Module

A.4.2.1.1 General description

A MODBUS communication requires the establishment of a TCP connection between a Client and a Server.
The establishment of the connection can be activated either explicitly by the User Application module or automatically by the TCP connection management module.
In the first case an application-programming interface has to be provided in the user application module to manage completely the connection. This solution provides flexibility for the application programmer but it requires a good expertise on TCP/IP mechanism.
In the second case the TCP connection management is completely hidden to the user application that only sends and receives MODBUS messages. The TCP connection management module is in charge to establish a new TCP connection when it is required.
The definition of the number of TCP client and server connections is not on the scope of this document (value n in this document). Depending on the device capacities the number of TCP connections can be different.

Implementation Rules :

1) Without explicit user requirement, it is recommended to implement the automatic TCP connection management

2) It is recommended to keep the TCP connection opened with a remote device and not to open and close it for each MODBUS/TCP transaction,
Remark: However the MODBUS client must be capable of accepting a close request from the server and closing the connection. The connection can be reopened when required.

3) It is recommended for a MODBUS Client to open a minimum of TCP connections with a remote MODBUS server (with the same IP address). One connection per application could be a good choice.

4) Several MODBUS transactions can be activated simultaneously on the same TCP Connection.
Remark: If this is done then the MODBUS transaction identifier must be used to uniquely identify the matching requests and responses.

5) In case of a bi-directional communication between two remote MODBUS entities (each of them is client and server), it is necessary to open separate connections for the client data flow and for the server data flow.

6) A TCP frame must transport only one MODBUS ADU. It is advised against sending multiple MODBUS requests or responses on the same TCP PDU

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

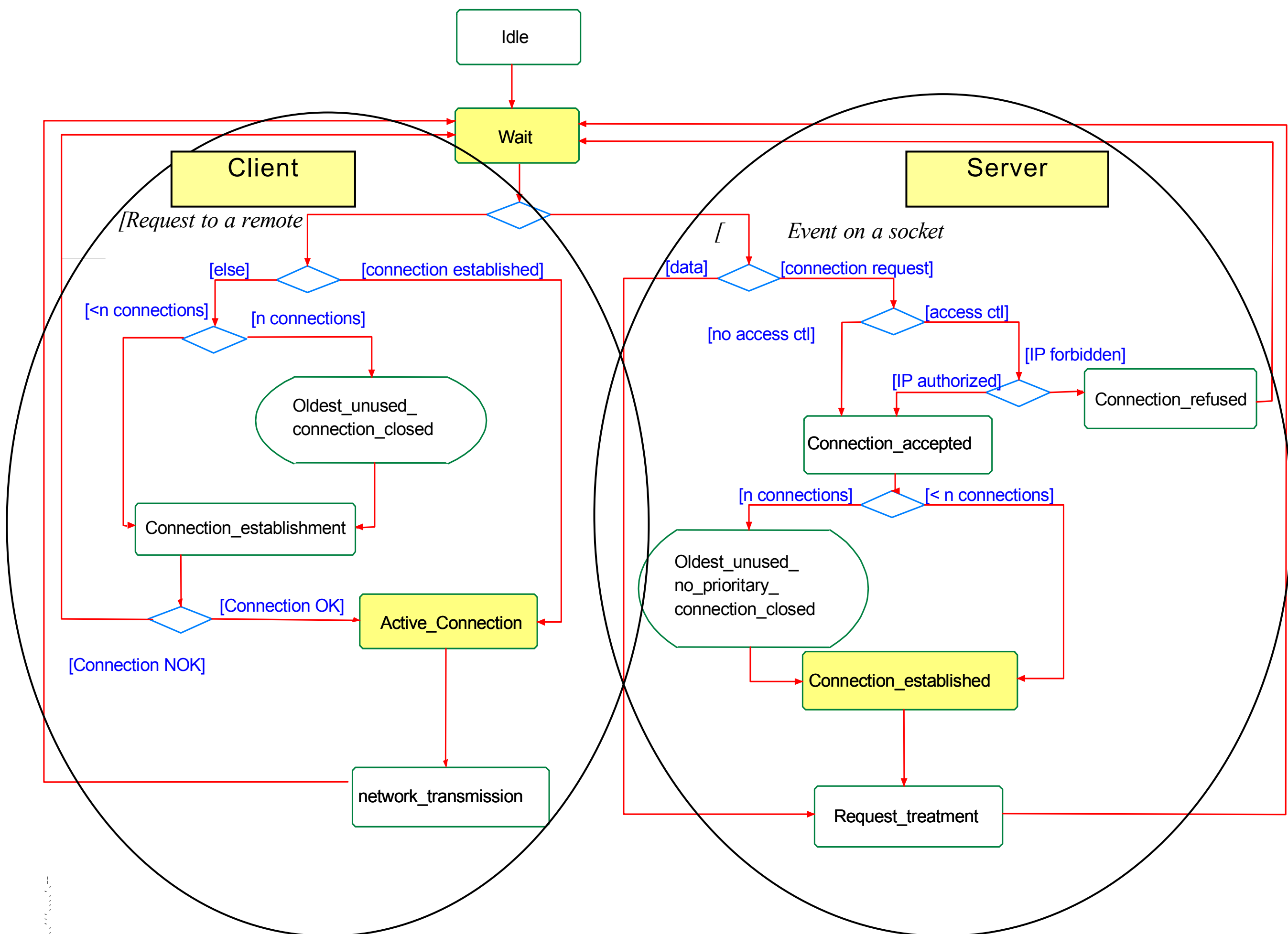


Figure A.7 – TCP connection management activity diagram

1. **Explicit TCP connection management**

   The user application module is in charge of managing all the TCP connections: active and passive establishment, connection ending, etc. This management is done for all MODBUS communication between a client and a server. The BSD Socket interface is used in the user application module to manage the TCP connection. This solution offers a total flexibility but it implies that the application programmer has sufficient TCP knowledge.
   A limit of number of client and server connections has to be configured taking into account the device capabilities and requirement.

2. **Automatic TCP connection management**

   The TCP connection management is totally transparent for the user application module. The connection management module may accept a sufficient number of client and server connections.
   Nevertheless a mechanism must be implemented in case of exceeding the number of authorized connection. In such a case we recommend to close the oldest unused connection.
   A connection with a remote partner is established at the first packet received from a remote client or from the local user application. This connection will be closed if a termination arrived from the network or decided locally on the device. On reception of a connection request, the access control option can be used to forbid device accessibility to unauthorized clients.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The TCP connection management module uses the Stack interface (usually BSD Socket interface) to communicate with the TCP/IP stack.

In order to maintain compatibility between system requirements and server resources, the TCP management will maintain 2 pools of connection.

- The first pool (**priority connection pool**) is made of connections that are never closed on a local initiative. A configuration must be provided to set this pool up. The principle to be implemented is to associate a specific IP address with each possible connection of this pool. The devices with such IP addresses are said to be “marked”. Any new connection that is requested by a marked device must be accepted, and will be taken from the priority connection pool. It is also necessary to configure the maximum number of Connections allowed for each remote device to avoid that the same device uses all the connections of the priority pool.
- The second pool (**non-priority connection pool**) contains connections for non marked devices. The rule that takes over here is to close the oldest connection when a new connection request arrives from a non-marked device and when there is no more connection available in the pool.

A configuration might be optionally provided to assign the number of connections available in each pool. However (It is not mandatory) the designers can set the number of connections at design time if required.

A.4.2.1.2 Connection management description

- **Connection establishment :**

The MODBUS messaging service must provide a listening socket on Port 502, which permits to accept new connection and to exchange data with other devices.
When the messaging service needs to exchange data with a remote server, it must open a new client connection with a remote Port 502 in order to exchange data with this distant. The local port must be higher than 1024 and different for each client connection.

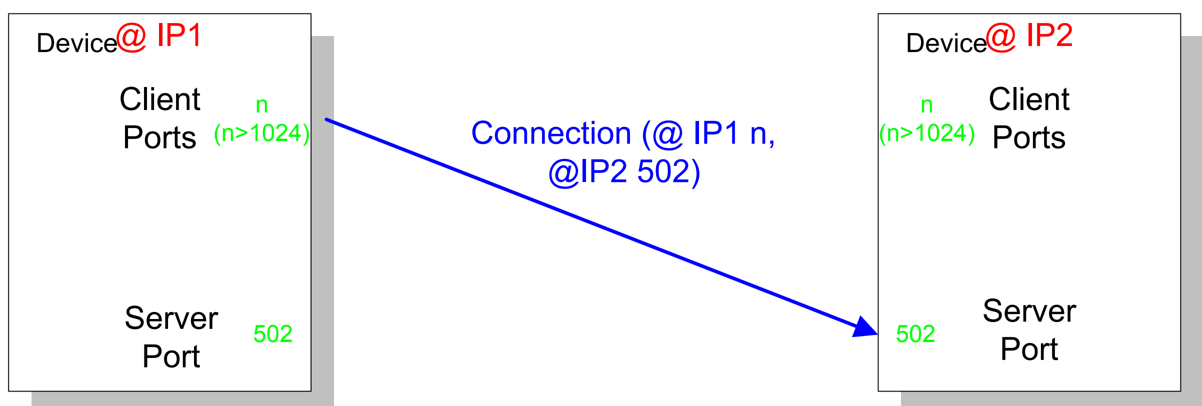


Figure A.8 – MODBUS TCP connection establishment

If the number of client and server connections is greater than the number of authorized connections the oldest unused connection is closed. The access control mechanism can be activated to check if the IP address of the remote client is authorized. If not the new connection is refused.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

- **MODBUS data transfer**

A MODBUS request has to be sent on the right TCP connection already opened. The IP address of the remote is used to find the TCP connection. In case of multiple TCP connections opened with the same remote, one connection has to be chosen to send the MODBUS message, different choice criteria can be used like the oldest one, the first one. The connection has to maintain open during all the MODBUS communications. As described in the following chapters a client can initiate several MODBUS transactions with a server without waiting the ending of the previous one.

- **Connection closing**

When the MODBUS communications are ended between a Client and a Server, the client has to initiate a connection closing of the connection used for these communications.

A.4.2.2 Impact of Operating Modes on the TCP Connection

Some Operating Modes (communication break between two operational End Points, Crash and Reboot on one of the End Point, …) may have impacts on the TCP Connections. A connection can be seen closed or aborted on one side without the knowledge of the other side. The connection is said to be **"half-open"**.
This chapter describes the behavior for each main Operating Modes. It is assumed that the **Keep Alive** TCP mechanism is used on both end points (See A4.3.2)

A.4.2.2.1 Communication break between two operational end points:

The origin of the communication break can be the disconnection of the Ethernet cable on the Server side. The expected behavior is:

- If no packet is currently sent on the connection:
  The communication break will not be seen if it lasts less than the Keep Alive timer value. If the communication break lasts more than the Keep Alive timer value, an error is returned to the TCP Management layer that can reset the connection.
- If Some packets are sent before and after the disconnection:
  The TCP retransmission algorithms (Jacobson's, Karn's algorithms and exponential backoff See 4.3.2) are activated. This may lead to a stack TCP layer Reset of the Connection before the Keep Alive timer is over.

A.4.2.2.2 Crash and Reboot of the Server end point

After the crash and Reboot of the Server, the connection is "**half-open"** on Client side. The expected behavior is:

- If no packet is sent on the half-open connection:
  The TCP half-open connection is seen opened from the Client side as long as the Keep Alive timer is not over. After that an error is returned to the TCP Management layer that can reset the connection.
- If some packets are sent on the half-open connection:
  The Server receives data on a connection that doesn't exist anymore. The stack TCP layer sends a Reset to close the half-open connection on the Client side

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.4.2.2.3 Crash and Reboot of the Client

After the crash and Reboot of the Client, the connection is "**half-open"** on Server side. The expected behavior is:

- No packet is sent on the half-open connection:
  The TCP half-open connection is seen opened from the Server side as long as the Keep Alive timer is not over. After that an error is returned to the TCP Management layer that can reset the connection.
- If the Client opens a new connection before the Keep Aliver timer is over :
  Two cases has to be studied:
  - The connection opening has the **same characteristics than the half-open connection** on the server side (same source and destination Ports, same source and destination IP Addresses), therefore the connection opening will fail at the TCP stack level after the Time-Out on Connection Establishment (75s on most of Berkeley implementations). To avoid this long Time-Out during which it is not possible to communicate, it is advised to ensure that different source port numbers than the previous one are used for a connection opening after a reboot on the client side.

  - The connection opening has **not the same characteristics as the half-open connection** on server side (different source Ports, same destination Port, same source and destination IP Address), therefore the connection is opened at the stack TCP level and signaled to the Server TCP Management layer.
    If the Server TCP Management layer only supports one connection from a remote Client IP Address, it can close the old half-opened connection and use the new one.
    If the Server TCP Management layer supports several connections from a remote Client IP Address, the new connection stays opened and the old one also stays half-opened until the expiration of the Keep Alive Timer that will return an error to the TCP Management layer. After that the TCP Management layer will be able to Reset the old connection.

A.4.2.3 Access Control Module

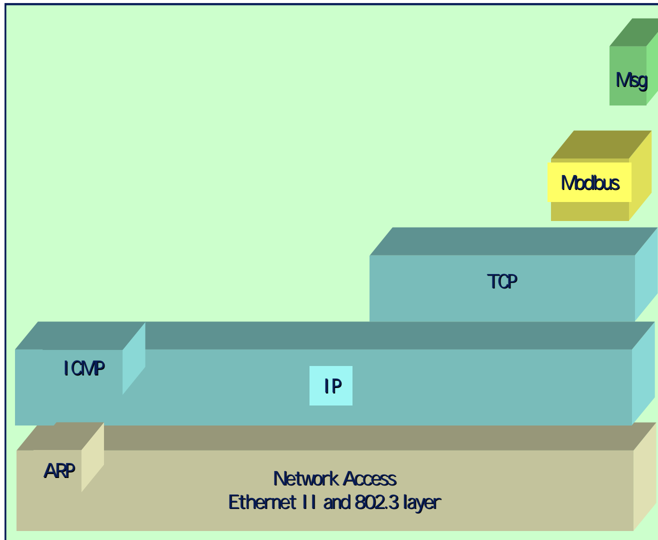
The goal of this module is to check every new connection and using a list of authorized remote IP addresses the module can authorize or forbid a remote Client TCP connection.

In critical context the application programmer needs to choose the Access Control mode in order to secure its network access. In such a case he needs to Authorize/forbid access for each remote @IP. The user needs to provide a list of IP addresses and to specify for each IP address if it's authorized or not. By default, on security mode, the IP addresses not configured by the user are forbidden. Therefore with the access control mode a connection coming from an unknown IP address is closed.

A.4.3 USE of TCP/IP STACK

A TCP/IP stack provides an interface to manage connections, to send and receive data, and also to do some parameterizations in order to adapt the stack behavior to the device or system constraints.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST



The goal of this clause is to give an overview of the Stack interface and also some information concerning the parameterization of the stack. This overview focuses on the features used by the MODBUS Messaging.

For more information, the advice is to read the RFC 1122 that provides guidance for vendors and designers of Internet communication software. It enumerates standard protocols that a host connected to the Internet must use as well as an explicit set of requirements and options.
The stack interface is generally based on the BSD (Berkeley Software Distribution) Interface that is described in this document.

A.4.3.1 Use of BSD Socket interface

Remark : some TCP/IP stack proposes other types of interface for performance issues. A MODBUS client or server can use these specific interfaces, but this use will be not described in Clause 1.

A socket is an endpoint of communication. It is the basic building block for communication. A MODBUS communication is executed by sending and receiving data through sockets. The TCPIP library provides only stream sockets using TCP and providing a connection-based communication service.
The Sockets are created via the **socket ()** function. A socket number is returned, which is then used by the creator to access the socket. Sockets are created without addresses (IP address and port number). Until a port is bound to a socket, it cannot be used to receive data.
The **bind ()** function is used to bind a port number to a socket. The **bind ()** creates an association between the socket and the port number specified.
In order to initiate a connection, the client must issue the **connect ()** function specifying the socket number, the remote IP address and the remote listening port number (active connection establishment).
In order to complete a connection, the server must issue the **accept ()** function specifying the socket number that was specified in the prior **listen ()** call (passive connection establishment). A new socket is created with the same properties as the initial one. This new socket is connected to the client's socket, and its number is returned to the server. The initial socket is thereby free for other clients that might want to connect with the server.

After the establishment of the TCP connection the data can be transferred. The **send()** and **recv()** functions are designed specifically to be used with sockets that are already connected.

The **setsockopt ()** function allows a socket's creator to associate options with a socket. These options modify the behavior of the socket. The description of these options is given in A.4.3.2.

The **select ()** function allows the programmer to test events on all sockets.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The **shutdown ()** function allows a socket user to disable send () and/or receive () on the socket.

Once a socket is no longer needed, its socket descriptor can be discarded by using the **close ()** function.

Figure A.9: MODBUS Exchanges describes a full MODBUS communication between a client and a s server. The Client establishes the connection and sends 3 MODBUS requests to the server without waiting the response of the first one. After receiving all the responses the Client closes the connection properly.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

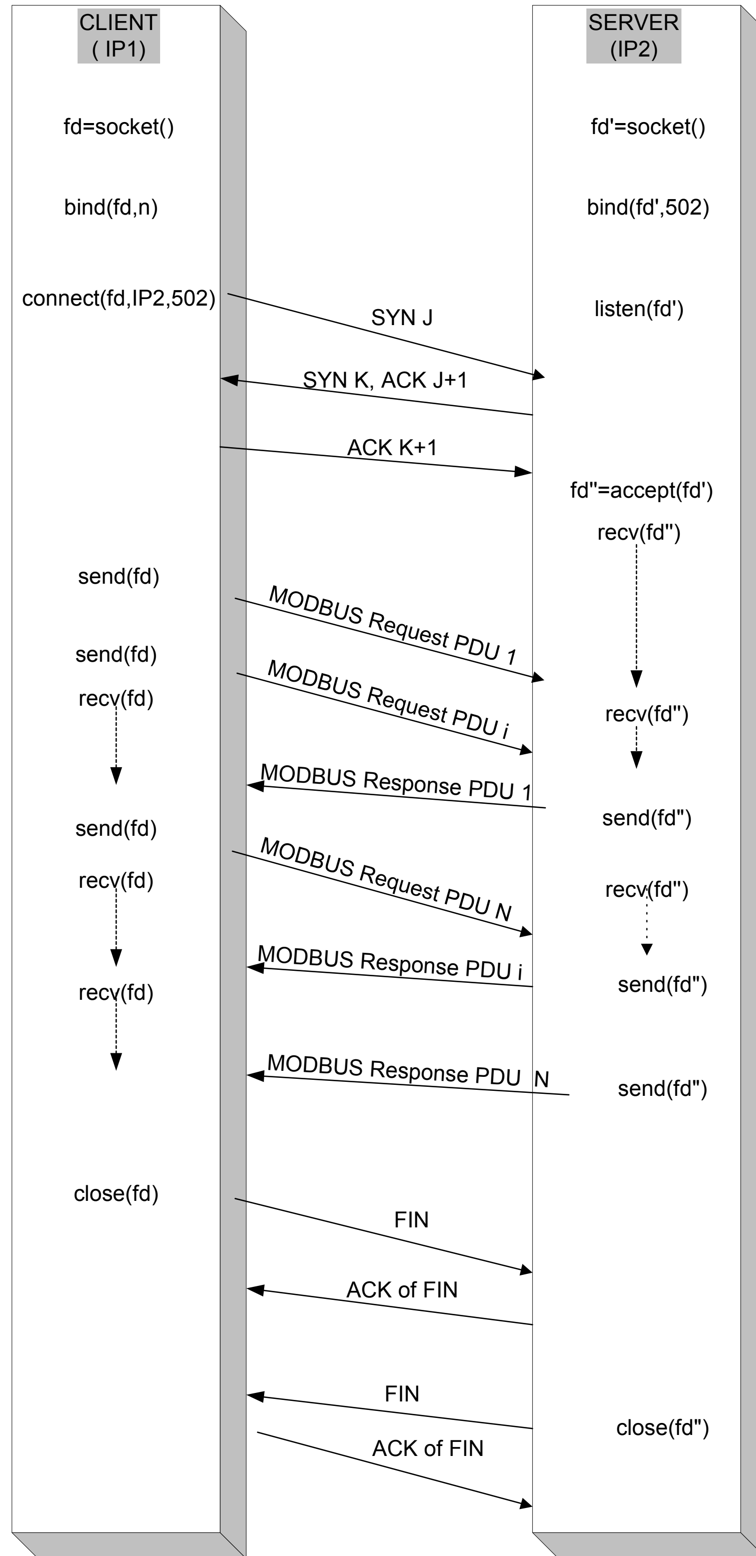


Figure A.9 – MODBUS Exchanges

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.4.3.2 TCP layer parameterization

Some parameters of the TCP/IP stack can be adjusted to adapt its behavior to the product or system constraints. The following parameters can be adjusted in the TCP layer:

- **Parameters for each connection:**

SO-RCVBUF, SO-SNDBUF:

These parameters allow setting the high water mark for the Send and the Receive Socket. They can be adjusted for flow control management. The size of the received buffer is the maximum size advertised window for that connection. Socket buffer sizes must be increased in order to increase performances. Nevertheless these values must be smaller than internal driver resources in order to close the TCP window before exhausting internal driver resources.

The received buffer size depends on the TCP Windows size, the TCP Maximum segment size and the time needed to absorb the incoming frames. With a Maximum Segment Size of 300 bytes (a MODBUS request needs a maximum of 256 bytes + the MBAP header size), if we need 3 frames buffering, the socket buffer size value can be adjusted to 900 bytes. For biggest needs and best-scheduled time, the size of the TCP window may be increased.

TCP-NODELAY:

Small packets (called tinygrams) are normally not a problem on LANs, since most LANs are not congested, but these tinygrams can lead to congestion on wide area networks. A simple solution, called the "NAGLE algorithm", is to collect small amounts of data and sends them in a single segment when TCP acknowledgments of previous packets arrive.
In order to have better real-time behavior it is recommended to send small amounts of data directly without trying to gather them in a single segment. That is why it is recommended to force the TCP-NODELAY option that disables the "NAGLE algorithm" on client and server connections.

SO-REUSEADDR:

When a MODBUS server closes a TCP connection initialized by a remote client, the local port number used for this connection cannot be reused for a new opening while that connection stay in the "Time-wait" state (during two MSL : Maximum Segment Lifetime).
It is recommended specifying the SO-REUSEADDR option for each client and server connection to bypass this restriction. This option allows the process to assign itself a port number that is part of a connection that is in the 2MSL wait for client and listening socket.

SO-KEEPALIVE:

By default on TCP/IP protocol no data are sent across an idle TCP connection. Therefore if no process at the ends of a TCP connection is sending data to the other, nothing is exchanged between the two TCP modules. This assumes that either the client application or the server application uses timers to detect inactivity in order to close a connection.
It is recommended to enable the KEEPALIVE option on both client and server connection in order to poll the other end to know if the distant has either crashed and is down or crashed and rebooted.
Nevertheless we must keep on mind that enabling KEEPALIVE can cause perfectly good connections to be dropped during transient failures, that it consumes unnecessary bandwidth on the network if the keep alive timer is too short.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

- **Parameters for the whole TCP layer:**

  Time Out on establishing a TCP Connection:

  Most Berkeley-derived systems set a time limit of 75 seconds on the establishment of a new connection, this default value should be adapted to the real time constraint of the application.

  Keep Alive parameters:

  The default idle time for a connection is 2 hours. Idles times in excess of this value trigger a keep alive probe. After the first keep alive probe, a probe is sent every 75 seconds for a maximum number of times unless a probe response is received.
  The maximum number of keep Alive probes sent out on an idle connection is 8. If no probe response is received after sending out the maximum number of keep Alive probes,TCP signal an error to the application that can decide to close the connection

  Time-out and retransmission parameters:

  A TCP packet is retransmitted if its lost has been detecting. One way to detect the lost is to manage a Retransmission Time-Out (RTO) that expires if no acknowledgement have been received from the remote side.
  TCP manages a dynamic estimation of the RTO. For that purpose a Round-Trip Time (RTT) is measured after the send of every packet that is not a retransmission. The Round-Trip Time (RTT) is the time taken for a packet to reach the remote device and to get back an acknowledgement to the sending device. The RTT of a connection is calculated dynamically, nevertheless if TCP cannot get an estimate within 3 seconds, the default value of the RTT is set to 3 seconds.
  If the RTO has been estimated, it applies to the next packet sending. If the acknowledgement of the next packet is not received before the estimated RTO expiration, the **Exponential BackOff** is activated. A maximum number of retransmissions of the same packet is allowed during a certain amount of time. After that if no acknowledgement has been received, the connection is aborted.
  The maximum number of retransmissions and the maximum amount of time before the abort of the connection (tcp_ip_abort_interval) can be set up on some stacks.

  Some retransmission algorithms are defined in TCP standards :

  - The **Jacobson's RTO estimation algorithm** is used to estimate the Retransmission Time-Out (RTO),
  - The **Karn's algorithm** says that the RTO estimation should not be done on a retransmitted segment,
  - The **Exponential BackOff** defines that the retransmission time-out is doubled for each retransmission with an upper limit of 64 seconds.
  - The **fast retransmission algorithm** allows retransmitting after the reception of three duplicate acknowledgments. This algorithm is advised because on a LAN it may lead to a quicker detection of the lost of a packet than waiting for the RTO expiration.

  The use of these algorithms is recommended for a MODBUS implementation.

A.4.3.3 IP layer parameterization

A.4.3.3.1 IP Parameters

The following parameters must be configured in the IP layer of a MODBUS implementation :

- Local IP Address : the IP address can be part of a Class A, B or C.

--`,````,,,```,,,`,,,``,`,``-`-`,,`,,`,`,,`---

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dhabi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

- Subnet Mask, : Subnetting an IP Network can be done for a variety of reasons : use of different physical media (such as Ethernet, WAN, etc.), more efficient use of network addresses, and the capability to control network traffic. The Subnet Mask has to be consistent with the IP address class of the local IP address.

- Default Gateway: The IP address of the default gateway has to be on the same subnet as the local IP address. The value 0.0.0.0 is forbidden. If no gateway is to be defined then this value is to be set to either 127.0.0.1 or the Local IP address.

Remark : The MODBUS messaging service does not require the fragmentation function in the IP layer.

The local IP End Point shall be configured with a **local IP Address** and with **a Subnet Mask** and a **Default Gateway** (different from 0.0.0.0) .

A.4.4 COMMUNICATION APPLICATION LAYER

A.4.4.1 MODBUS Client

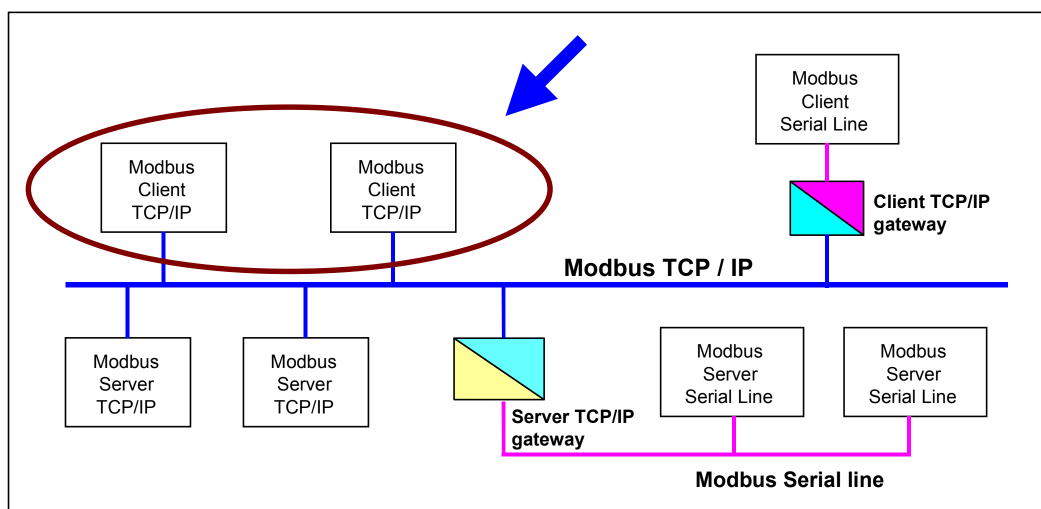


Figure A.10 – MODBUS Client unit

A.4.4.1.1 MODBUS client design

The definition of MODBUS/TCP protocol allows a simple design of a client. The following activity diagram describes the main treatments that are processed by a client to send a MODBUS request and to treat a MODBUS response.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

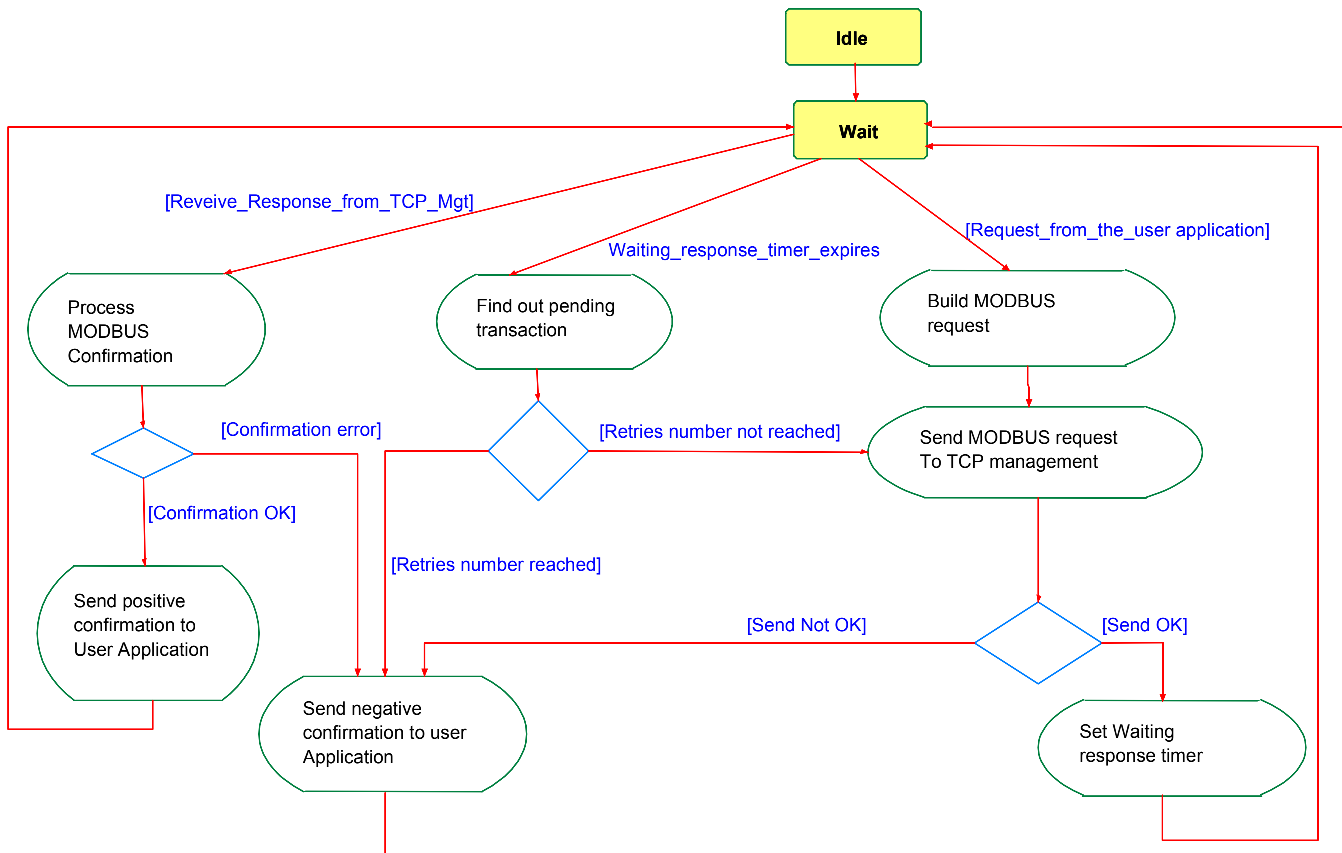


Figure A.11 – MODBUS Client Activity Diagram

A MODBUS client can receive three events:

- A new demand from the user application to send a request, in this case a MODBUS request has to be encoded and be sent on the network using the TCP management component service. The lower layer (TCP management module) can give back an error due to a TCP connection error, or some other errors.
- A response from the TCP management, in this case the client has to analyze the content of the response and send a confirmation to the user application
- The expiration of a Time out due to a non-response. A new retry can be sent on the network or a negative confirmation can be sent to the User Application. *Remark : These retries are initiated by the MODBUS client, some other retries can also be done by the TCP layer in case of TCP acknowledge lack.*

A.4.4.1.2 Build a MODBUS Request

Following the reception of a demand from the user application, the client has to build a MODBUS request and to send it to the TCP management.
Building the MODBUS request can be split in several sub-tasks:

- The instantiation of a MODBUS transaction that enables the Client to memorize all required information in order to bind later the response to the request and to send the confirmation to the user application.
- The encoding of the MODBUS request (PDU + MPAB header). The user application that initiates the demand has to provide all required information which enables the Client to encode the request. The MODBUS PDU is encoded according to part 1 of this Specification. (MB function code, associated parameters and application data). All fields of the MBAP header are filled. Then, the MODBUS request ADU is built prefixing the PDU with the MBAP header

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

- The sending of the MODBUS request ADU to the TCP management module which is in charge of finding the right TCP socket towards the remote Server. In addition to the MODBUS ADU the Destination IP address must also be passed.

The following activity diagram describes, more deeply than in Figure A.11 MODBUS Client Activity Diagram, the request building phase.

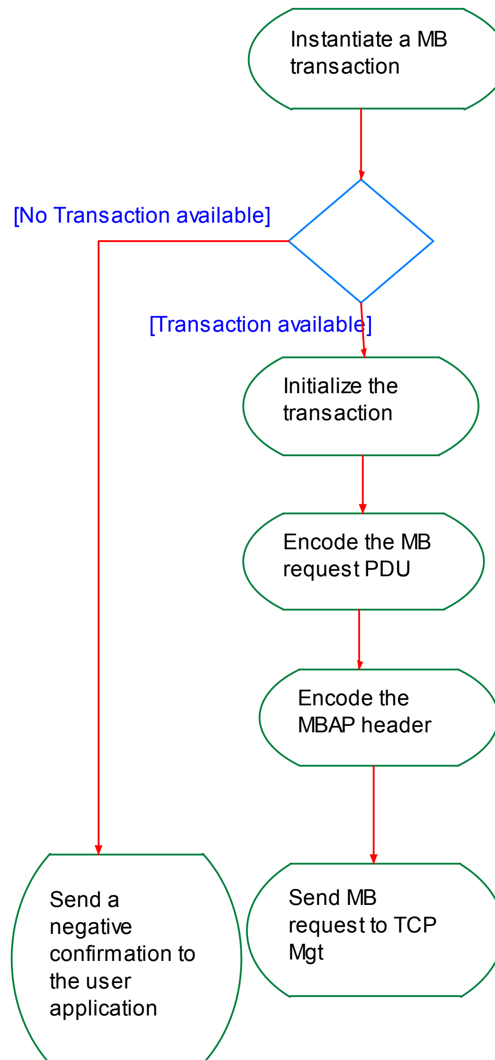


Figure A.12 – Request building activity diagram

The following example describes the MODBUS request ADU encoding for reading the register # 5 in a remote server :

The following example describes the MODBUS request ADU encoding for reading 1 word at the address 05 in a remote server :

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

♦ MODBUS Request ADU encoding :

	Description	Size	Example
MBAP Header	Transaction Identifier Hi	1	0x15
	Transaction Identifier Lo	1	0x01
	Protocol Identifier	2	0x0000
	Length	2	0x0006
	Unit Identifier	1	0xFF
MODBUS request	*Function Code (*)*	*1*	*0x03*
	Starting Address	*2*	*0x0004*
	Quantity of Registers	*2*	*0x0001*

(*) See Clause 1 of this PAS.

- **Transaction Identifier**

The transaction identifier is used to associate the future response with the request. So, at a time, on a TCP connection, this identifier must be unique. There are several manners to use the transaction identifier:

- For example, it can be used as a simple "TCP sequence number" with a counter which is incremented at each request.
- It can also be judiciously used as a smart index or pointer to identify a transaction context in order to memorize the current remote server and the pending MODBUS request.

Normally, on MODBUS serial line a client must send one request at a time. This means that the client must wait for the answer to the first request before sending a second request. On TCP/MODBUS, several requests can be sent without waiting for a confirmation to the same server. The MODBUS/TCP to MODBUS serial line gateway is in charge of ensuring compatibility between these two behaviors.
The number of requests accepted by a server depends on its capacity in term of number of resources and size of the TCP windows. In the same way the number of transactions initialized simultaneously by a client depends also on its resource capacity. This implementation parameter is called "**NumberMaxOfClientTransaction**" and must be described as one of the MODBUS client features. Depending of the device type this parameter can take a value from 1 to 16.

- **Unit Identifier**

This field is used for routing purpose when addressing a device on a MODBUS or MODBUS+ serial line sub-network. In that case, the “Unit Identifier” carries the MODBUS slave address of the remote device:

If the MODBUS server is connected to a MODBUS+ or MODBUS Serial Line sub-network and addressed through a bridge or a gateway, the MODBUS Unit identifier is necessary to identify the slave device connected on the sub-network behind the bridge or the gateway. The destination IP address identifies the bridge itself and the bridge uses the MODBUS Unit identifier to forward the request to the right slave device.
The MODBUS slave device addresses on serial line are assigned from 1 to 247 (decimal). Address 0 is used as broadcast address.

On TCP/IP, the MODBUS server is addressed using its IP address; therefore, the MODBUS Unit Identifier is useless. The value 0xFF has to be used.

When addressing a MODBUS server connected directly to a TCP/IP network, it’s recommended not using a significant MODBUS slave address in the “Unit Identifier” field. In the event of a re-allocation of the IP addresses within an automated system and if a IP address previously assigned to a MODBUS server is then assigned to a gateway, using a significant slave address may cause trouble because of a bad routing by the gateway. Using a non-

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

significant slave address, the gateway will simply discard the MODBUS PDU with no trouble. 0xFF is recommended for the “Unit Identifier" as non-significant value.
Remark : The value 0 is also accepted to communicate directly to a MODBUS/TCP device.

A.4.4.1.3 Process MODBUS Confirmation

When a response frame is received on a TCP connection, the Transaction Identifier carried in the MBAP header is used to associate the response with the original request previously sent on that TCP connection:

- If the Transaction Identifier does not refer to any MODBUS pending transaction, the response must be discarded.
- If the Transaction Identifier refers to a MODBUS pending transaction, the response must be parsed in order to send a MODBUS Confirmation to the User Application (positive or negative confirmation)

Parsing the response consists in verifying the MBAP Header and the MODBUS PDU response:

- **MBAP Header**
  After the verification of the Protocol Identifier that must be 0x0000, the length gives the size of the MODBUS response.
  If the response comes from a MODBUS server device directly connected to the TCP/IP network, the TCP connection identification is sufficient to unambiguously identify the remote server. Therefore, the Unit Identifier carried in the MBAP header is not significant (value 0xFF) and must be discarded.
  If the remote server is connected on a Serial Line sub-network and the response comes from a bridge, a router or a gateway, then the Unit Identifier (value != 0xFF) identifies the remote MODBUS server which has originally sent the response.

- **MODBUS Response PDU**
  The function code must be verified and the MODBUS response format analyzed according to the MODBUS Application Protocol:
  - if the function code is the same as the one used in the request, and if the response format is correct, then the MODBUS response is given to the user application as a **Positive Confirmation**.
  - If the function code is a MODBUS exception code (Function code + 80H), the MODBUS exception response is given to the user application as a **Positive Confirmation**.
  - If the function code is different from the one used in the request (=non expected function code), or if the format of the response is incorrect, then an error is signaled to the user application using a **Negative Confirmation**.

Remark: A positive confirmation is a confirmation that the command was received and responded to by the server. It does not imply that the server was able to successfully act on the command (failure to successfully act on the command is indicated by the MODBUS Exception response).

The following activity diagram describes, more deeply than in Figure A.11: MODBUS Client Activity Diagram, the confirmation processing phase.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

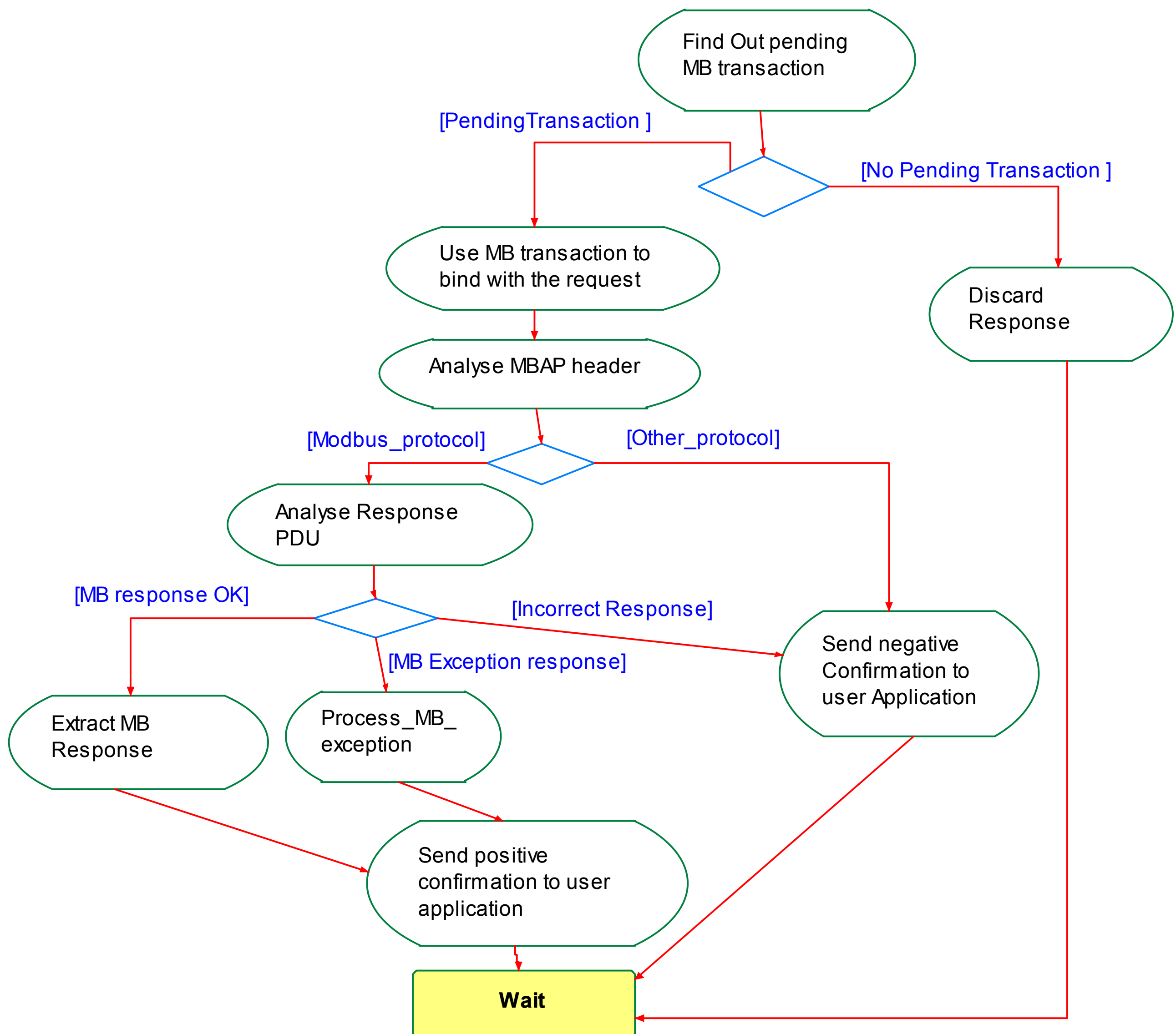


Figure A.13 – Process MODBUS Confirmation activity diagram

A.4.4.1.4 Time-out managing

There is deliberately NO specification of required response time for a transaction over MODBUS/TCP.
This is because MODBUS/TCP is expected to be used in the widest possible variety of communication situations, from I/O scanners expecting sub-millisecond timing to long distance radio links with delays of several seconds.
From a client perspective, the timeout must take into account the expected transport delays across the network, to determine a 'reasonable' response time. Such transport delays might be milliseconds for a switched Ethernet, or hundreds of milliseconds for a wide area network connection.
In turn, any 'timeout' time used at a client to initiate an application retry should be larger than the expected maximum 'reasonable' response time. If this is not followed, there is a potential for excessive congestion at the target device or on the network, which may in turn cause further errors. This is a characteristic, which should always be avoided.
So in practice, the client timeouts used in high performance applications are always likely to be somewhat dependent on network topology and expected client performance.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Applications which are not time critical can often leave timeout values to the normal TCP defaults, which will report communication failure after several seconds on most platforms.

A.4.4.2 MODBUS Server

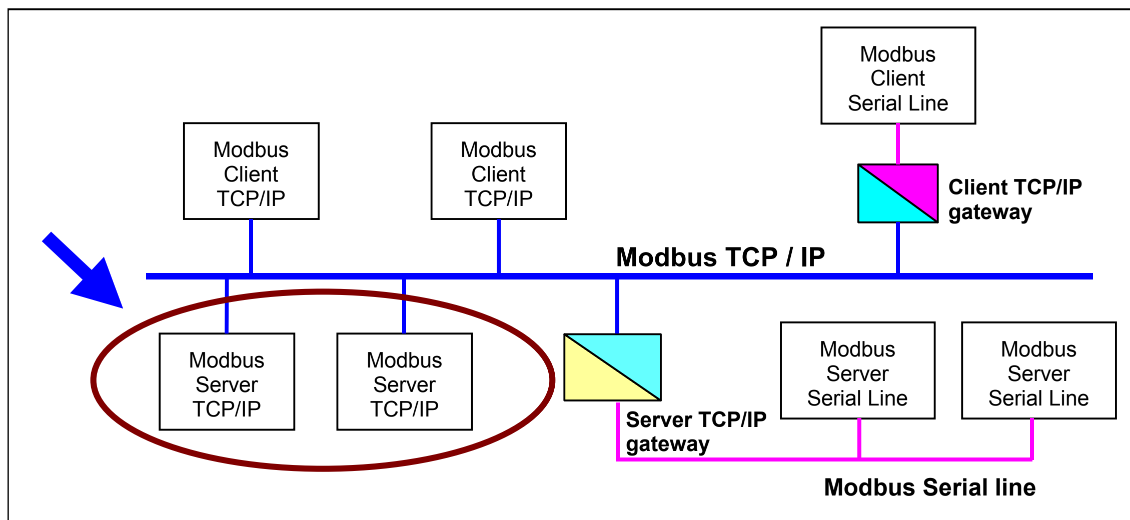


Figure A.14 – MODBUS Server unit

The role of a MODBUS server is to provide access to application objects and services to remote MODBUS clients.

Different kind of access may be provided depending on the user application :

- simple access like get and set application objects attributes
- advanced access in order to trigger specific application services

The MODBUS server has:

- To map application objects onto readable and writable MODBUS objects, in order to get or set application objects attributes.
- To provide a way to trigger services onto application objects.

In run time the MODBUS server has to analyze a received MODBUS request, to process the required action, and to send back a MODBUS response.

NOTE The application objects and services of the Backend Interface are to obtain the requested data based upon the function code, and the User is responsible.

A.4.4.2.1 MODBUS Server Design

The MODBUS Server design depends on both :

- the kind of access to the application objects (simple access to attributes or advanced access to services)
- the kind of interaction between the MODBUS server and the user application (synchronous or asynchronous).

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The following activity diagram describes the main treatments that are processed by the Server to obtain a MODBUS request from TCP Management, then to analyze the request, to process the required action, and to send back a MODBUS response.

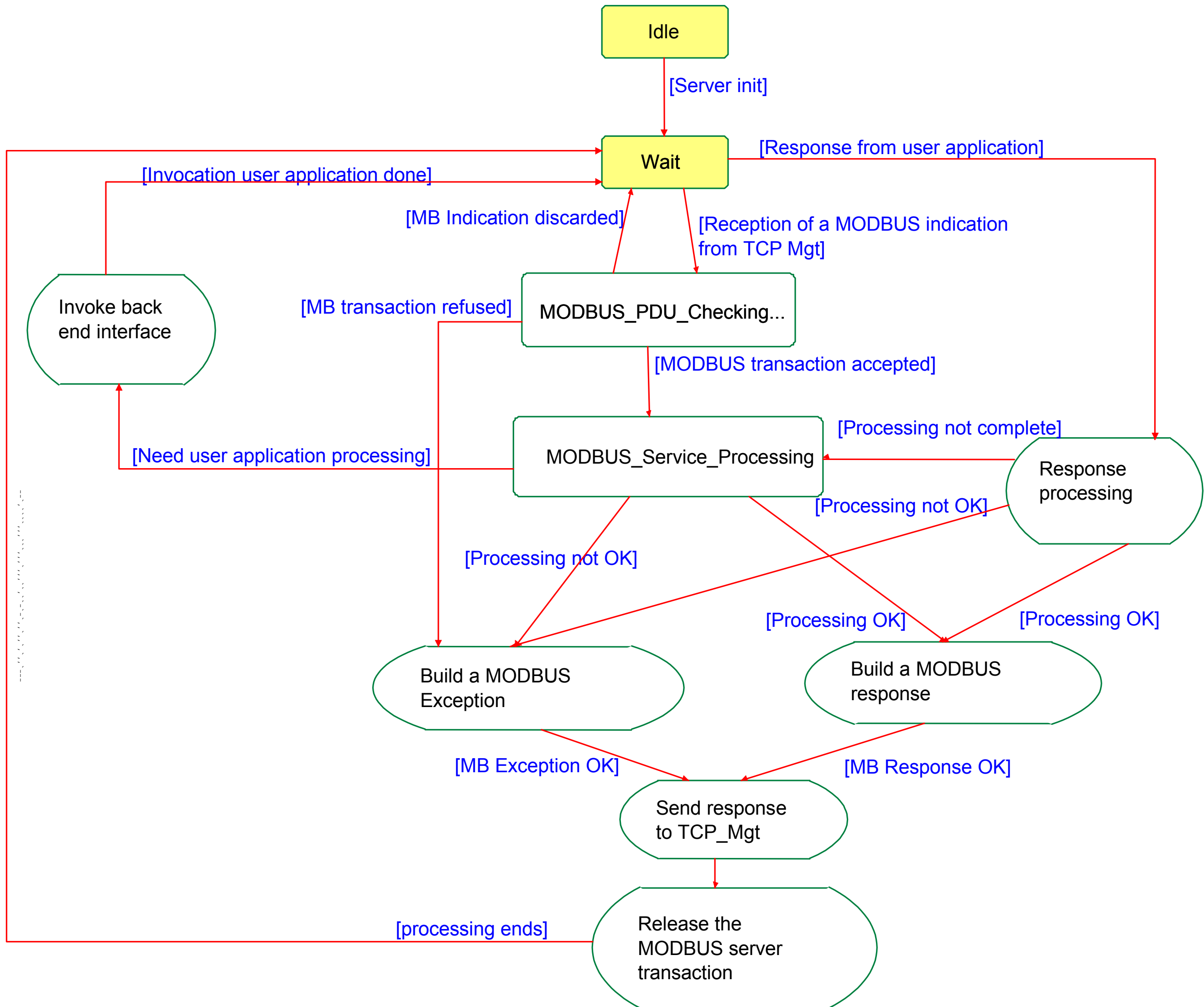


Figure A.15 – Process MODBUS Indication activity diagram

As shown in the previous activity diagram:

- Some services can be immediately processed by the MODBUS Server itself, with no direct interaction with the User Application ;
- Some services may require also interacting explicitly with the User Application to be processed ;
- Some other advanced services require invoking a specific interface called MODBUS Back End service. For example, a User Application service may be triggered using a sequence of several MODBUS request/response transactions according to a User Application level protocol. The Back End service is responsible for the correct processing of all individual MODBUS transactions in order to execute the global User Application service.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A more complete description is given in the following chapters.

The MODBUS server can accept to serve simultaneously several MODBUS requests. The maximum number of simultaneous MODBUS requests the server can accept is one of the main characteristics of a MODBUS server. This number depends on the server design and its processing and memory capabilities. This implementation parameter is called "**NumberMaxOfSeverTransaction**" and must be described as one of the MODBUS server features. It may have a value from 1 to 16 depending on the device capabilities.

The behavior and the performance of the MODBUS server are significantly affected by the "NumberMaxOfTransaction" parameter. Particularly, it's important to note that the number of concurrent MODBUS transactions managed may affect the response time of a MODBUS request by the server.

A.4.4.2.2 MODBUS PDU Checking

The following diagram describes the MODBUS PDU Checking activity.

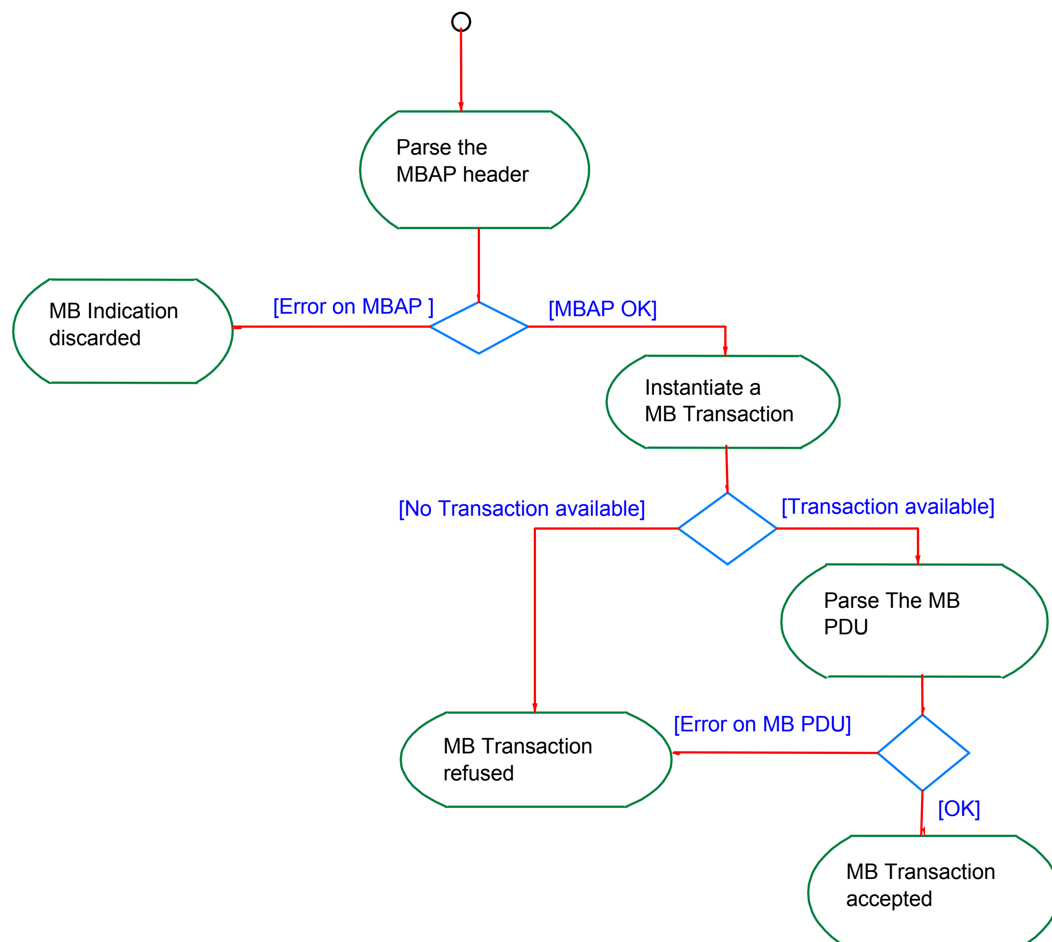


Figure A.16 – MODBUS PDU Checking activity diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The MODBUS PDU Checking function consists of first parsing the MBAP Header. The Protocol Identifier field has to be checked :

- If it is different from MODBUS protocol type, the indication is simply discarded.
- If it is correct (= MODBUS protocol type; value 0x00), a MODBUS transaction is instantiated.

The maximum number of MODBUS transactions the server can instantiated is defined by the "NumberMaxOfTransaction" parameter (A system or a configuration parameter).

In the case of no more transaction is available, the server builds a MODBUS exception response (Exception code 6 : Server Busy).

If a MODBUS transaction is available, it's initialized in order to memorize the following information:

- The TCP connection identifier used to send the indication (given by the TCP Management)
- The MODBUS Transaction ID (given in MBAP Header)
- The Unit Identifier (given in MBAP Header)

Then the MODBUS PDU is parsed. The function code is first controlled :

- in case of invalidity a MODBUS exception response is built (Exception code 1 : Invalid function).
- If the function code is accepted, the server initiates the "MODBUS Service processing" activity.

A.4.4.2.3 MODBUS service processing

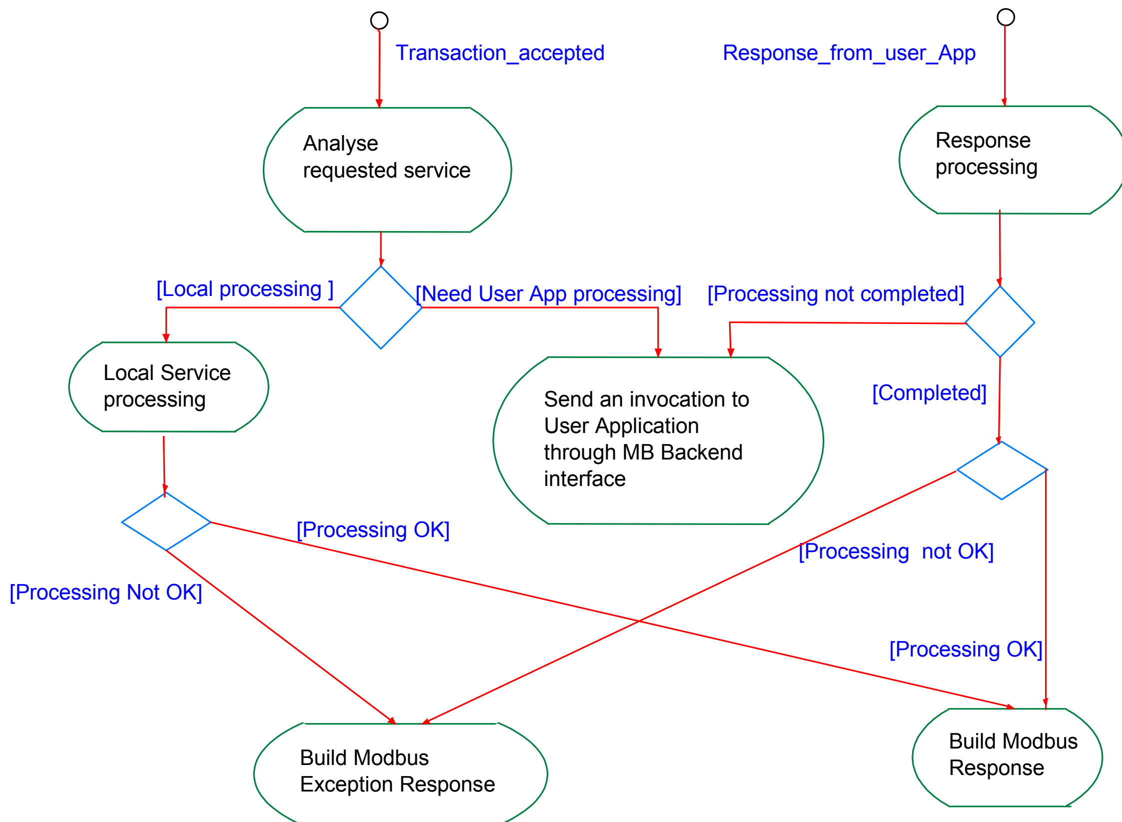


Figure A.17 – MODBUS service processing activity diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The processing of the required MODBUS service can be done in different ways depending on the device software and hardware architecture as described in the hereafter examples :

- Within a compact device or a mono-thread architecture where the MODBUS server can access directly to the user application data, the required service can be processed "locally" by the server itself without invoking the Back End service. The processing is done according to Clause 1 of this PAS. In case of an error, a MODBUS exception response is built.
- Within a modular multi-processor device or a multi-thread architecture where the "communication layers" and the "user application layer" are 2 separate entities, some trivial services can be processed completely by the Communication entity while some others can require a cooperation with the User Application entity using the Back End service.

To interact with the User Application, the MODBUS Backend service must implement all appropriate mechanisms in order to handle User Application transactions and to manage correctly the User Application invocations and associated responses.

A.4.4.2.4 User Application Interface (Backend Interface)

Several strategies can be implemented in the MODBUS Backend service to achieve its job although they are not equivalent in terms of user network throughput, interface bandwidth usage, response time, or even design workload.

The MODBUS Backend service will use the appropriate interface to the user application :

- Either a physical interface based on a serial link, or a dual-port RAM scheme, or a simple I/O line, or a logical interface based on messaging services provided by an operating system.
- The interface to the User Application may be synchronous or asynchronous.

The MODBUS Backend service will also use the appropriate design pattern to get/set objects attributes or to trigger services. In some cases, a simple "gateway pattern" will be adequate. In some other cases, the designer will have to implement a "proxy pattern" with a corresponding caching strategy, from a simple exchange table history to more sophisticated replication mechanisms.

The MODBUS Backend service has the responsibility to implement the protocol transcription in order to interact with the User Application. Therefore, it can have to implement mechanisms for packet fragmentation/reconstruction, data consistency guarantee, and synchronization whatever is required.

A.4.4.2.5 MODBUS Response building

Once the request has been processed, the MODBUS server has to build the response using the adequate MODBUS server transaction and has to send it to TCP management component.
Depending on the result of the processing two types of response can be built :

- A positive MODBUS response :
  - The response function code = The request function code

- A MODBUS Exception response :
  - The objective is to provide to the client relevant information concerning the error detected during the processing ;
  - The response function code = the request function code + 0x80 ;
  - The exception code is provided to indicate the reason of the error.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

▪

Exception Code	MODBUS name	Comments
01	Illegal Function Code	The function code is unknown by the server
02	Illegal Data Address	Dependant on the request
03	Illegal Data Value	Dependant on the request
04	Server Failure	The server failed during the execution
05	Acknowledge	The server accepted the service invocation but the service requires a relatively long time to execute. The server therefore returns only an acknowledgement of the service invocation receipt.
06	Server Busy	The server was unable to accept the MB Request PDU. The client application has the responsibility of deciding if and when to re-send the request.
0A	Gateway problem	Gateway paths not available.
0B	Gateway problem	The targeted device failed to respond. The gateway generates this exception

The MODBUS response PDU must be prefixed with the MBAP header which is built using data memorized in the transaction context.

- **Unit Identifier**
  The Unit Identifier is copied as it was given within the received MODBUS request and memorized in the transaction context.
- **Length**
  The server calculates the size of the MODBUS PDU plus the Unit Identifier byte. This value is set in the "Length" field.
- **Protocol Identifier**
  The Protocol Identifier field is set to 0x0000 (MODBUS protocol), as it was given within the received MODBUS request.
- **Transaction Identifier**
  This field is set to the "Transaction Identifier" value that was associated with the original request and memorized in the transaction context.

Then the MODBUS response must be returned to the right MODBUS Client using the TCP connection memorized in the transaction context. When the response is sent, the transaction context must be free.

A.5 IMPLEMENTATION GUIDELINE

The objective of this clause is to propose an example of a messaging service implementation.
The model describes below can be used as a guideline during a client or a server implementation of a MODBUS messaging service.

NOTE The messaging service implementation is the responsibility of the User.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.5.1 OBJECT MODEL DIAGRAM

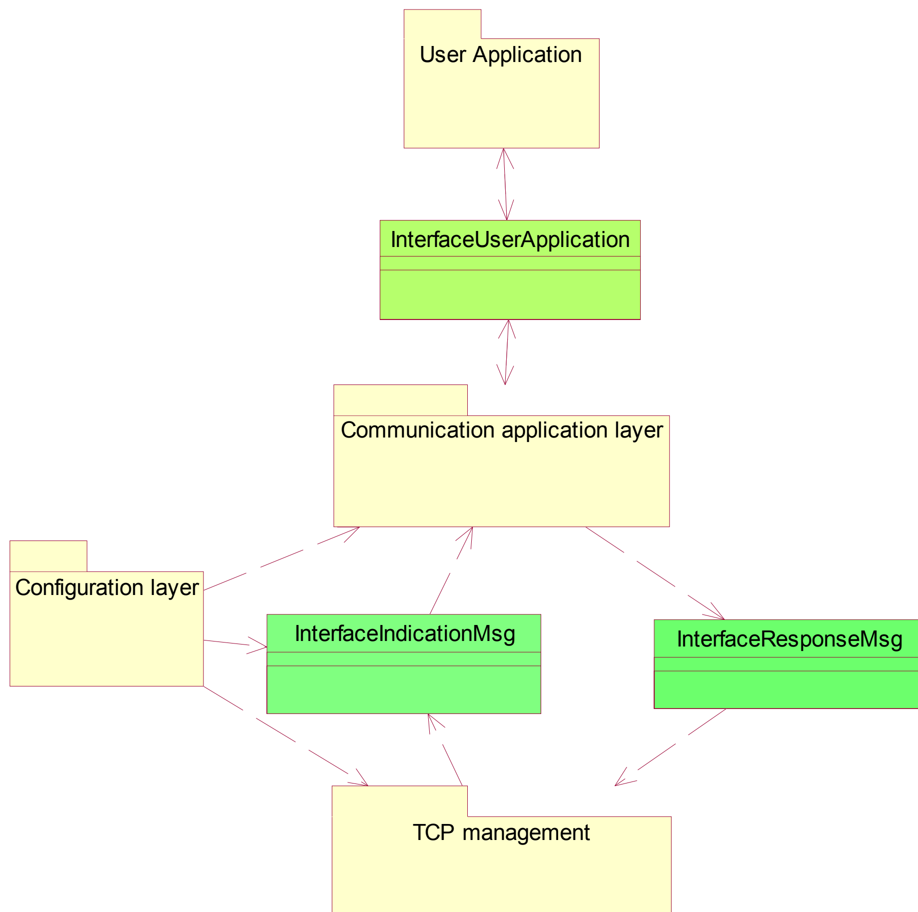


Figure A.18 – MODBUS Messaging Service Object Model Diagram

Four main packages composes the Object Model Diagram:

- **The Configuration layer** which configures and manages operating modes of components of other packages
- **The TCP Management** which interfaces the TCP/IP stack and the communication application layer managing TCP connection. It implies the management of socket interface.
- **The Communication application layer** which is composed by the MODBUS client on one side and the MODBUS server on the other side. This package is linked with the user application.

The User application which corresponds to the device application, it is completely dependent on the device and therefore it will be not part of this Specification.

This model is independent of implementation choices like the type of OS, the memory management, etc. In order to guarantee this independence generic Interface layers are used between the TCP management layer and the communication layer and between the communication layer and the user application layer.

Different implementations of this interface can be realized by the User: Pipe between two tasks, shared memory, serial link interface, procedural call, etc.

Some assumptions have to be taken to defined the hereafter implementation model :

- Static memory management
- Synchronous treatment of the server
- One task to process the receptions on all sockets.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.5.1.1 TCP management package

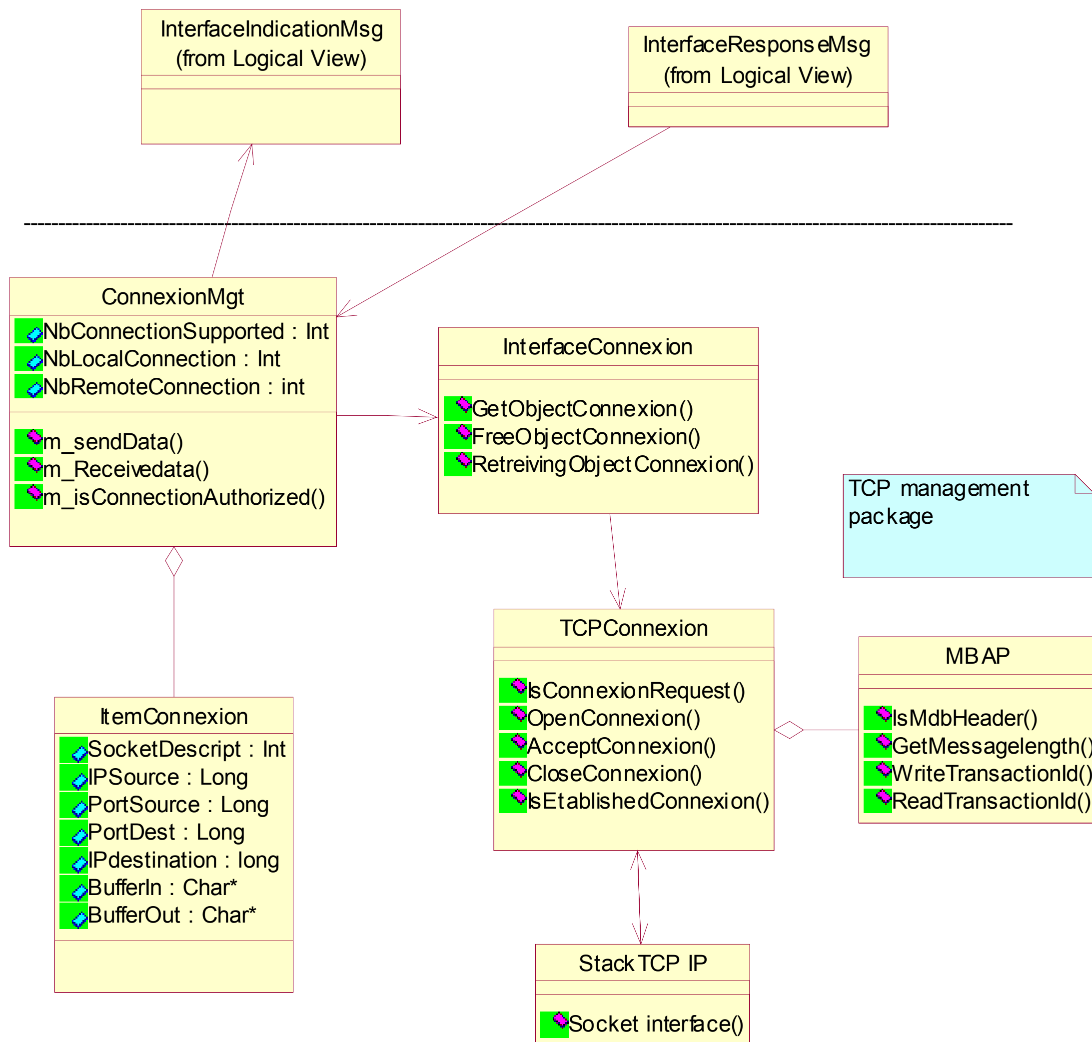


Figure A.19 – MODBUS TCP management package

The TCP management package comprises the following classes :

CInterfaceConnexion: The role of this class consists in managing memory pool for connections.

CItemConnexion: This class contains all information needed to describe a connection.

CTCPConnexion:, This class provides methods for managing automatically a TCP connection (Interface socket is provided by **CStackTCP_IP**).

CConnexionMngt: This class manages all connections and send query/response to MODBUS Server/MODBUS Client through **CinterfaceIndicationMsg** and **CInterfaceResponseMsg**. This class also treats the Access control for the connection opening.

CMBAP: This class provides methods for reading/writing/analyzing the MODBUS MBAP.

CStackTCP_IP: This class Implements socket services and provides parameterization of the stack.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.5.1.2 Configuration layer package

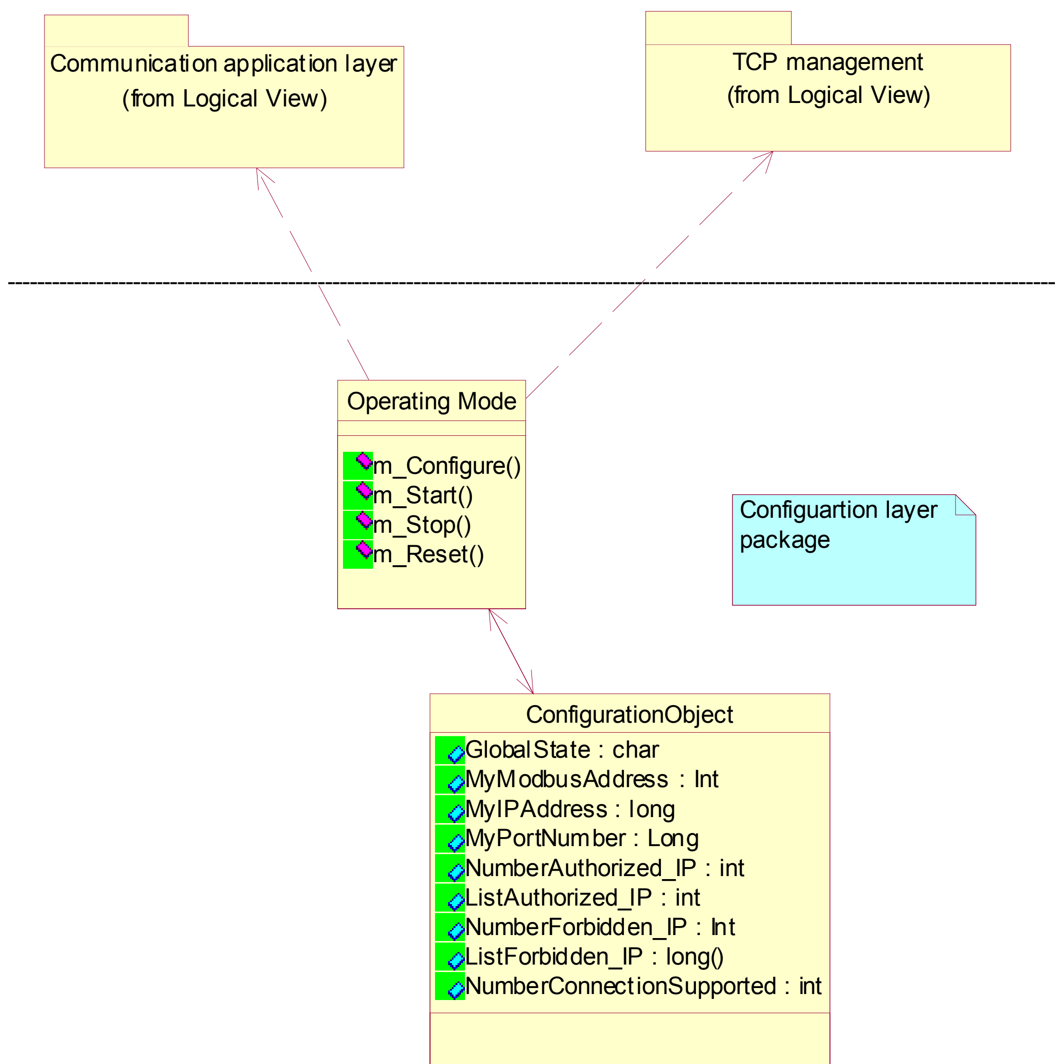


Figure A.20 – MODBUS Configuration layer package

The Configuration layer package comprises the following classes :

TConfigureObject: This class groups all data needed for configuring each other component. This structure is filled by the method m_Configure from the class **CoperatingMode**. Each class needing to be configured gets its own configuration data from this object. The configuration data is implementation dependent therefore the list of attributes of this class is provided as an example.

COperatingMode: The role of this class is to fill the **TConfigureObject** (according to the user configuration) and to manage the operating modes of the classes described below:

- CMODBUSServer
- CMODBUSClient
- CconnexionMngt

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.5.1.3 Communication layer package

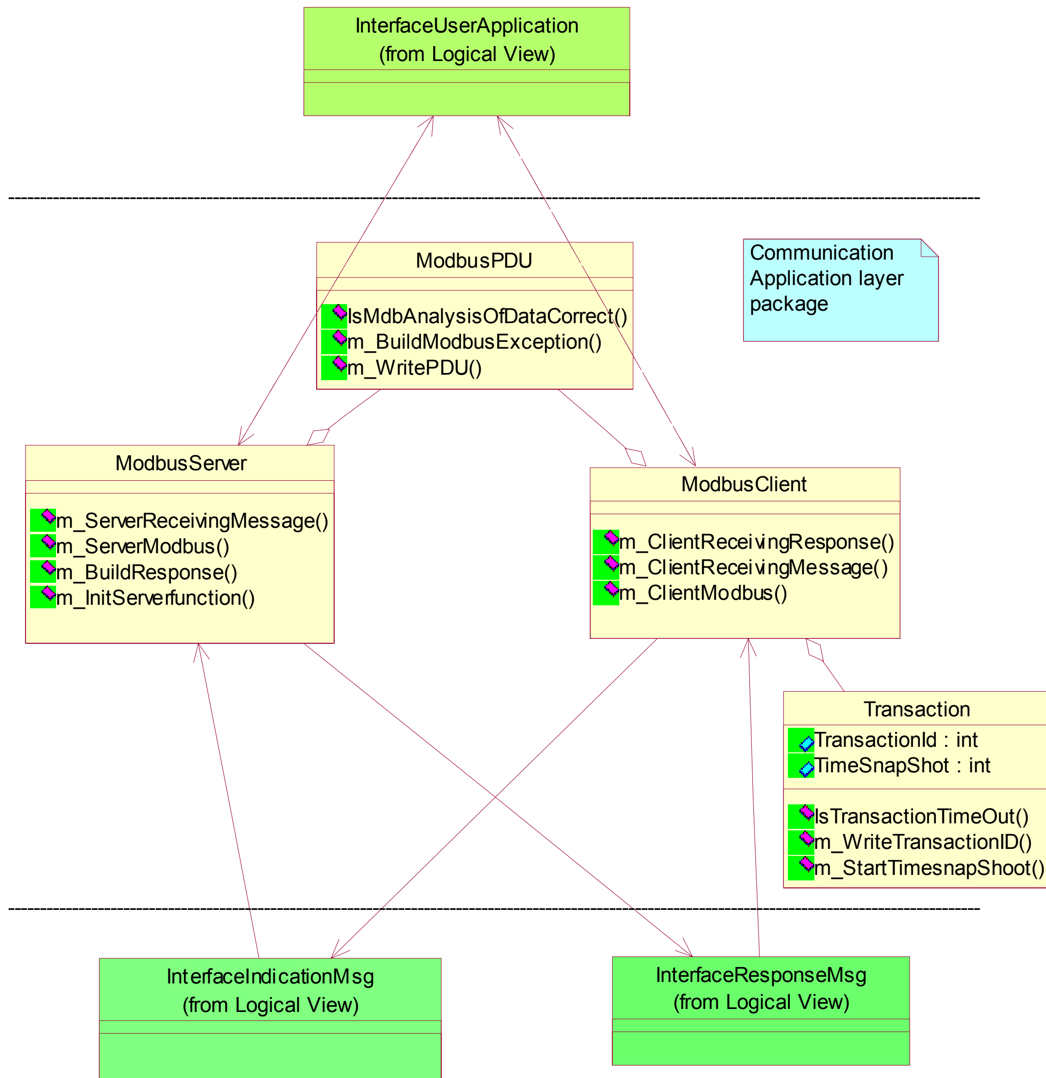


Figure A.21 –MODBUS Communication Application layer package

--`,````,,,```,,,`,,,`,`-`-`,,`,,`,`,,`---

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The Communication Application layer package comprises the following classes :

CMODBUSServer: MODBUS query is received from class **CInterfaceIndicationMsg** (by the method m_ServerReceivingMessage). The role of this class is to build the MODBUS response or the MODBUS Exception according the query (incoming from network). This class implements the Graph State of MODBUS server. Response can be built only if class **COperatingMode** has sent both user configuration and right operating modes.

CMODBUSClient: MODBUS query is read from class **CInterfaceUserApplication**, The client task receives query by the method m_ClientReceivingMessage. This class implements the State Graph of MODBUS client and manages transaction for linking query with response (from network). Query can be sent over network only if class **CoperatingMode** has sent both user configuration and right operating modes.

CTransaction: This class implements methods and structures for managing transactions.

A.5.1.4 Interface classes

CInterfaceUserApplication: This class represents the interface with the user application, it provides two methods to access to the user data. In a real implementation this method can be implemented in different way depending of the hardware and software device capabilities (equivalent to an end-driver, example access to PCMCIA, shared memory, etc).

CInterfaceIndicationMsg: This Interface class is proposed for sending query from Network to the MODBUS Server, and for sending response from Network for the Client. This class interfaces TCPManagement and 'Communication Application Layer' packages (From Network). The implementation of this class is device dependent.

CInterfaceResponseMsg: This Interface class is used for receiving response from the Server and for sending query from the client to the Network. This class interfaces packages 'Communication Application Layer' and package 'TCPManagement' (To Network). The implementation of this class is device dependent.

A.5.2 IMPLEMENTATION CLASS DIAGRAM

The following Class Diagram describes the complete diagram of a proposal implementation.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

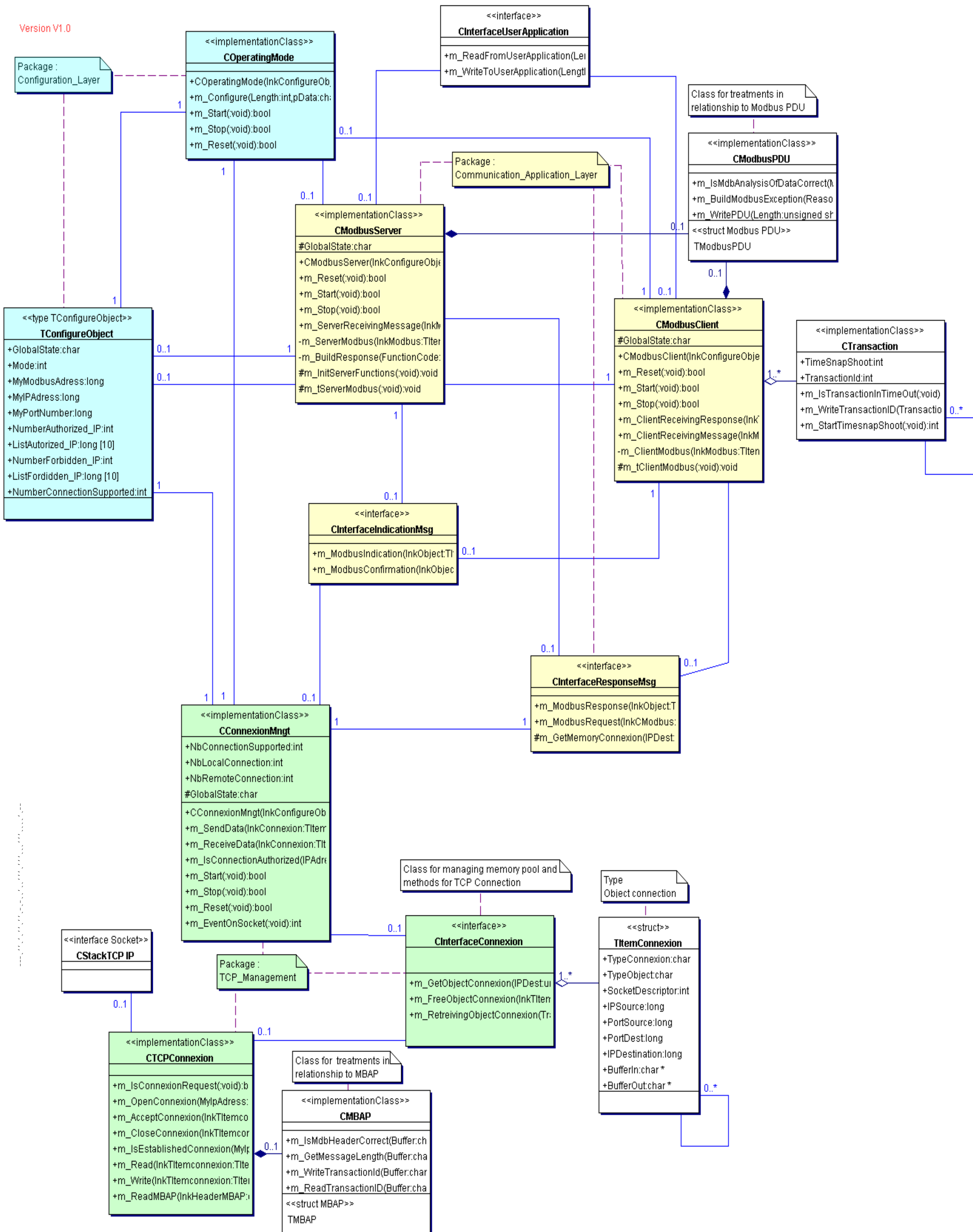


Figure A.22 – Class Diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.5.3 SEQUENCE DIAGRAMS

Two Sequence diagrams are described hereafter are an example in order to illustrate a Client MODBUS transaction and a Server MODBUS transaction.

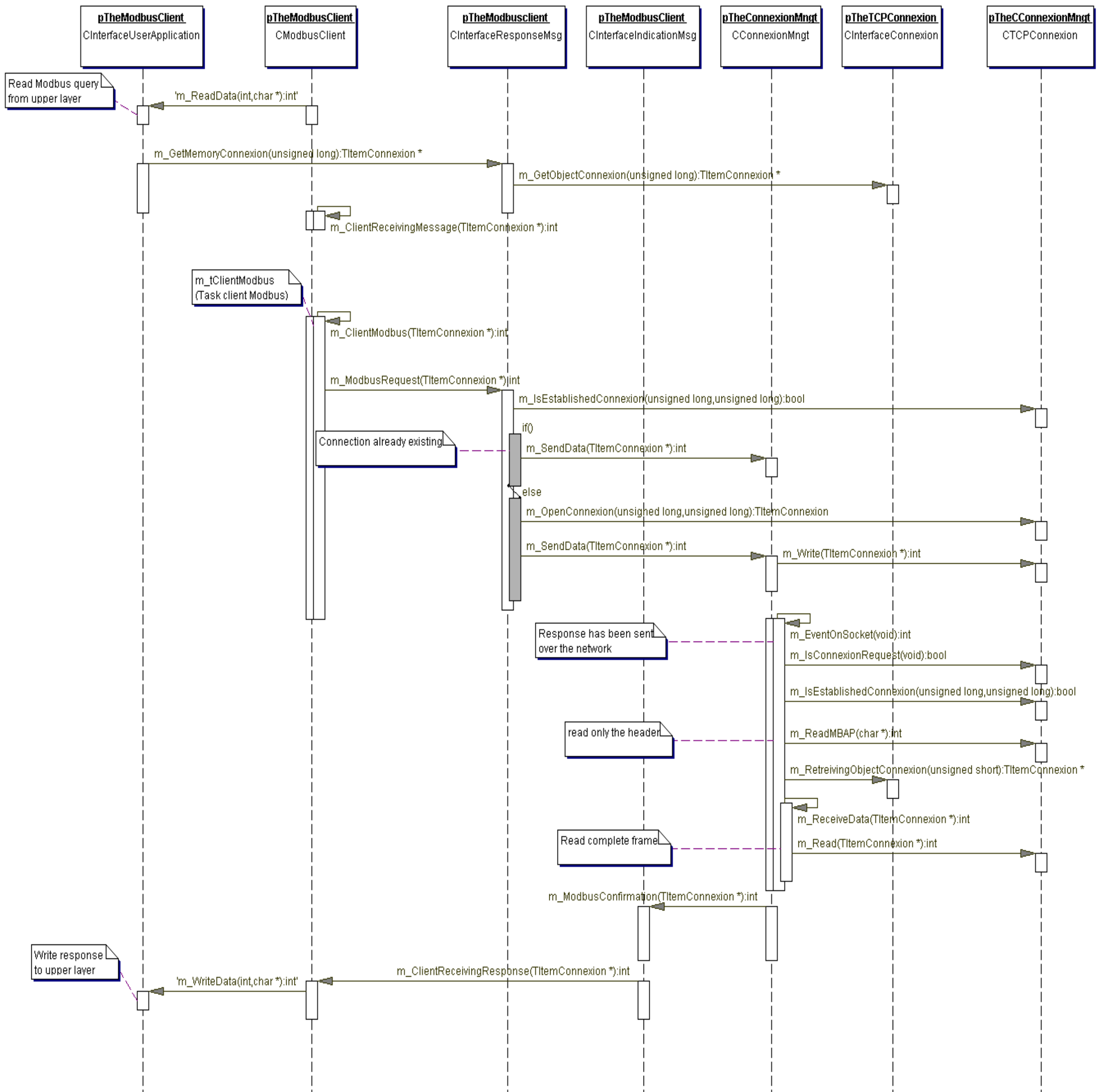


Figure A.23 – MODBUS client sequence diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dhabi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

General comments for a better understanding of the **Client** sequence diagram:

First step: A Reading query comes from User Application (method m_Read).

Second Step: The 'Client' task receives the MODBUS query (method **m_ClientReceivingMessage**). This is the entry point of the Client. To associate the query with the corresponding response when it will arrive, the Client uses a Transaction resource (Class **CTransaction**). The MODBUS query is sent to the TCP_Management by calling the class interface **CInterfaceResponseMsg** (method m_MODBUSRequest)

Third Step: If the connection is already established there is nothing to do on connection, the message can be send over the network. Otherwise, a connection must be opened before the message can be sent over the network.
At this time the client is waiting for a response (from a remote server)

Fourth step: Once a response has been received from the network, the TCP/IP stack receives data (method **m_EventOnSocket** is implicitly called).
If the connection is already established, then the MBAP is read for retrieving the connection object (connection object gives memory resource and other information).
Data coming from network is read and confirmation is sent to the client task via the class Interface **CInterfaceIndicationMsg** (method **m_MODBUSConfirmation**). Client task receives the MODBUS Confirmation (method **m_ClientReceivingResponse**).
Finally the response is–written to the user application (method m_WriteData), and transaction resource is freed.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Hereafter is an example of a MODBUS Server exchange.

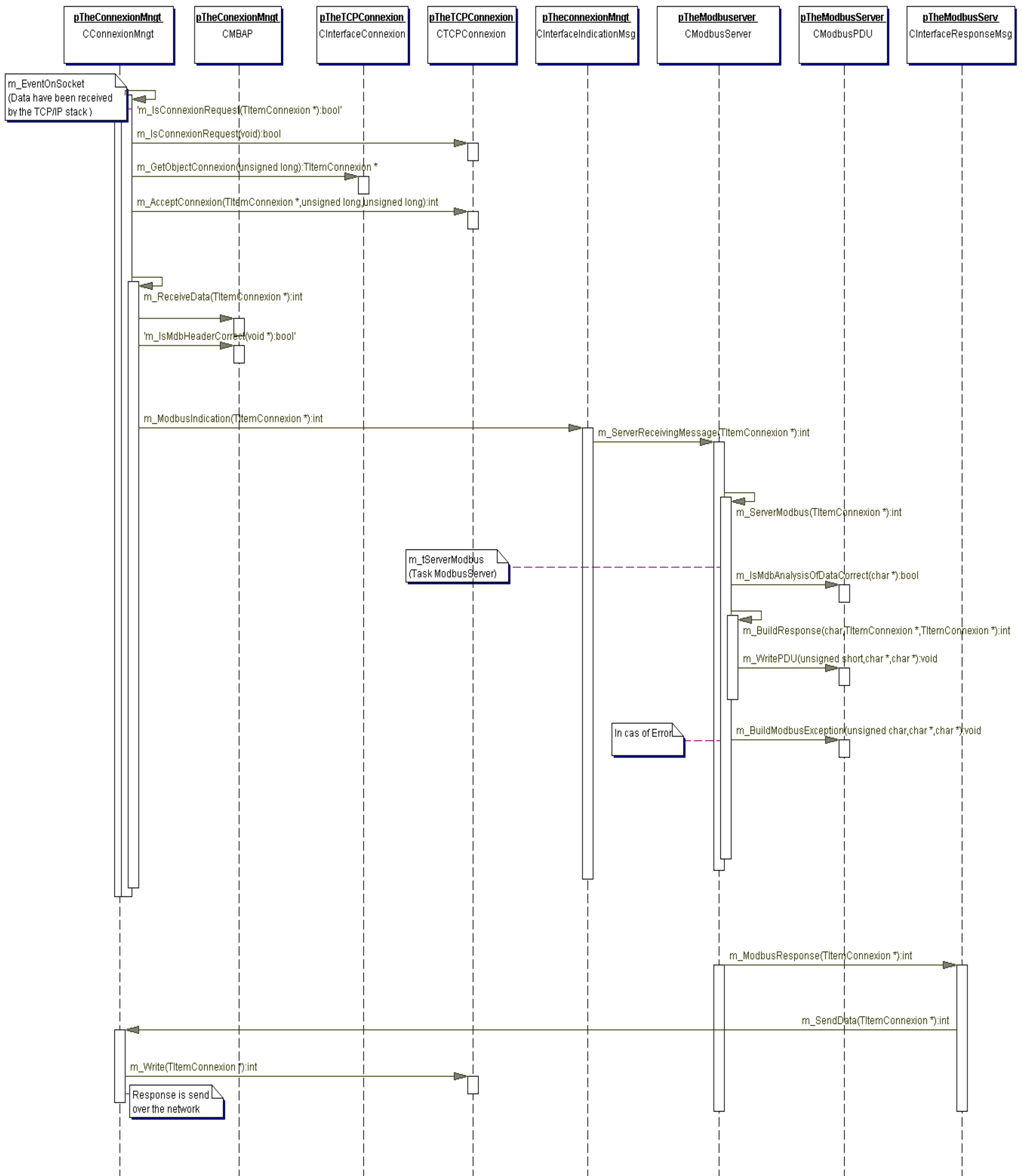


Figure A.24 – MODBUS server Diagram

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

General comments for a better understanding of the **Server** sequence diagram:

First step: a client has sent a query (MODBUS query) over the network.
The TCP/IP stack receives data (method **m_EventOnSocket** is implicitly called).

Second step: The query may be a connection request or not (method **m_IsConnexionRequest**).
If the query is a connection request, the connection object and buffers for receiving and sending the MODBUS frame are allocated (method **m_GetObjectConnexion**). Just after, the connection access control must be checked and accepted (method **m_AcceptConnexion**)

Third step: If the query is a MODBUS request, the complete MODBUS Query can be read (method **m_ReceiveData**). At this time the MBAP must be analyzed (method **m_IsMdbHeaderCorrect**). The complete frame is sent to the Server task via the **CinterfaceIndicationMessaging** Class (method **m_MODBUSIndication**). Server task receives the MODBUS Query (method **m_ServerReceivingMessage**) and analyses it.
If an error occurs (function code not supported, etc), a MODBUSException frame is built (**m_BuildMODBUSException**), otherwise the response is built.

Fourth Step: The response is sent over the network via the **CinterfaceResponseMessaging** (method **m_MODBUSResponse**). Treatment on the connection object is done by the method **m_SendData** (retrieve the connection descriptor, etc) and data is sent over the network.

A.5.4 CLASSES AND METHODS DESCRIPTION

A.5.4.1 MODBUS server class

Class CMODBUSServer

---

class **CMODBUSServer**

Stereotype implementationClass

Provides methods for managing MODBUS Messaging in Server Mode

---

Field Summary	
protected char	**GlobalState** state of the MODBUS Server

Constructor Summary	
CMODBUSServer(TConfigureObject * lnkConfigureObject) Constructor : Create internal object	

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Method Summary	
protected void	m_InitServerFunctions(void) Function called by the constructor for filling array of functions 'm_ServerFunction'
bool	m_Reset(void) Method for Reseting Server, return true if reseted
int	m_ServerReceivingMessage(TItemConnexion * lnkMODBUS) Interface with CindicationMsg::m_MODBUSIndication for receiving Query from NetWork return negative value if problem
bool	m_Start(void) Method for Starting Server, return true if Started
bool	m_Stop(void) Method for Stopping Server, return true if Stopped
protected void	m_tServerMODBUS(void) Server MODBUS task ...

A.5.4.2 MODBUS Client Class

Class CMODBUSClient

---

class **CMODBUSClient**

Provides methods for managing MODBUS Messaging in Client Mode

Stereotype implementationClass

Field Summary	
protected char	GlobalState State of the MODBUS Client

Constructor Summary	
CMODBUSClient(TConfigureObject * lnkConfigureObject) Constructor : Create internal object , initialize to 0 variables.	

Method Summary	
int	m_ClientReceivingMessage(TItemConnexion * lnkMODBUS) Interface provided for receiving message from application Layer Typically : Call CinterfaceUserApplication::m_Read for reading data call CInterfaceConnexion::m_GetObjectConnexion for getting memory for a transaction Return negative value if problem
int	m_ClientReceivingResponse(TitemConnexion * lnkTItemConnexion) Interface with CindicationMsg::m_Confirmation for receiving response from network return negative value if problem
bool	m_Reset(void) Method for Reseting component return true if reseted
bool	m_Start(void) Method for Starting component return true if started
bool	m_Stop(void) Method for Stoppping component return true if stopped
protected void	m_tClientMODBUS(void) Client MODBUS task....

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.5.4.3 Interface Classes

A.5.4.3.1 Interface Indication class

Class CInterfaceIndicationMsg

Direct Known Subclasses:

CConnexionMngt

---

class **CInterfaceIndicationMsg**

Class for sending message from TCP_Management to MODBUS Server or Client

Stereotype interface

Method Summary	
int	**m_MODBUSConfirmation**(TItemConnexion * lnkObject) Method for Receiving incoming Response, calling the Client : could be by reference, by Message Queue, Remote procedure Call, ...
int	**m_MODBUSIndication**(TItemConnexion * lnkObject) Method for reading incoming MODBUS Query and calling the Server : could be by reference, by Message Queue, Remote procedure Call, ...

A.5.4.3.2 Interface Response Class

Class CInterfaceResponseMsg

Direct Known Subclasses:

CMODBUSClient, CMODBUSServer

---

class **CInterfaceResponseMsg**

Class for sending response or sending query to TCP_Management from Client or Server

Stereotype interface

Method Summary	
TitemConnexion *	**m_GetMemoryConnexion**(unsigned long IPDest) Get an object ITemConnexion from memory pool Return -1 if not enough memory
int	**m_MODBUSRequest**(TItemConnexion * lnkCMODBUS) Method for Writing incoming MODBUS Query Client to ConnexionMngt : could be by reference, by Message Queue, Remote procedure Call, ...
int	**m_MODBUSResponse**(TItemConnexion * lnkObject) Method for writing Response from MODBUS Server to ConnexionMngt could be by reference, by Message Queue, Remote procedure Call, ...

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.5.4.4 Connexion Management class

Class CConnexionMngt

class **CConnexionMngt**

Class that manages all TCP Connections

Stereotype implementationClass

Field Summary	
protected char	**GlobalState** Global State of the Component ConnexionMngt
Int	**NbConnectionSupported** Global number of connections
Int	**NbLocalConnection** Number of connections opened by the local Client to a remote Server
Int	**NbRemoteConnection** Number of connections opened by a remote Client to the local Server

Constructor Summary	
CconnexionMngt(TConfigureObject * lnkConfigureObject) Constructor : Create internal object , initialize to 0 variables.	

Method Summary	
int	**m_EventOnSocket**(void) wake-up
bool	**m_IsConnectionAuthorized**(unsigned long IPAdress) Return true if new connection is authorized
int	**m_ReceiveData**(TItemConnexion * lnkConnexion) Interface with CTCPConnexion::write method for reading data from network return negative value if problem
bool	**m_Reset**(void) Method for Resetting ConnectionMngt component return true if Reset
int	**m_SendData**(TItemConnexion * lnkConnexion) Interface with CTCPConnexion::read method for sending data to the network Return negative value if problem
bool	**m_Start**(void) Method for Starting ConnectionMngt component return true if Started
bool	**m_Stop**(void) Method for Stopping component return true if Stopped

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Annex B of Section 1
(Informative)

MODBUS RESERVED FUNCTION CODES, SUBCODES AND MEI TYPES

The following function codes and subcodes shall not be part of this PAS and these function codes and subcodes are specifically reserved. The format is function code/subcode or just function code where all the subcodes (0-255) are reserved: 8/19; 8/21-65535, 9, 10, 13, 14, 41, 42, 90, 91, 125, 126 and 127.

Function Code 43 and its MEI Type 14 for Device Identification and MEI Type 13 for CANopen General Reference Request and Reponse PDU are the currently available Encapsulated Interface Transports in this PAS.

The following function codes and MEI Types shall not be part of the IEC published Specification derived from this document and these function codes and MEI Types are specifically reserved: 43/0-12 and 43/15-255. In this PAS, a User Defined Function code having the same or similiar result as the Encapsulated Interface Transport is not supported.

Annex C of Section 1
(Informative)

CANOPEN GENERAL REFERENCE COMMAND

Refer to the MODBUS-IDA web site or the CiA (CAN in Automation) website for a copy and terms of use that cover Function Code 43 MEI Type 13.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Section 2 – Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification Version 1.0

2 RTPS

2.1 Basic Concepts

2.1.1 Introduction

With the explosion of the Internet, the TCP/UDP/IP protocol suite has become the underlying framework upon which all Internet-based communications are built. Their success attests to the generality and power of these protocols. However, these transport-level protocols are too low level to be used directly by any but the simplest applications. Consequently, higher-level protocols such as HTTP, FTP, DHCP, DCE, RTP, DCOM, and CORBA have emerged. Each of these protocols fills a niche, providing well-tuned functionality for specific purposes or application domains.

In network communications, as in many fields of engineering, it is a fact that "one size does not fit all." Engineering design is about making the right set of trade-offs, and these trade-offs must balance conflicting requirements such as generality, ease of use, richness of features, performance, memory size and usage, scalability, determinism, and robustness. These trade-offs must be made in light of the types of information flow (e.g. periodic, one-to-many, request-reply, events), and the constraints imposed by the application and execution platforms.

The Real-Time Publish-Subscribe (RTPS) Wire Protocol provides two main communication models: the publish-subscribe protocol, which transfers data from publishers to subscribers; and the Composite State Transfer (CST) protocol, which transfers state.

The RTPS protocol is designed to run over an unreliable transport such as UDP/IP. The broad goals for the RTPS protocol design are:

- Plug and play connectivity so that new applications and services are automatically discovered and applications can join and leave the network at any time without the need for reconfiguration.
- Performance and quality-of-service properties to enable best-effort and reliable publish- subscribe communications for real-time applications over standard IP networks.
- Configurability to allow balancing the requirements for reliability and timeliness for each data delivery.
- Modularity to allow simple devices to implement a subset and still participate in the network.
- Scalability to enable systems to potentially scale to very large networks.
- Extensibility to allow the protocol to be extended and enhanced with new services without breaking backwards compatibility and interoperability.
- Fault tolerance to allow the creation of networks without single points of failure.
- Type-safety to prevent application programming errors from compromising the operation of remote nodes.

This PAS defines the message formats, interpretation, and usage scenarios that underlie all messages exchanged by applications that use the RTPS protocol.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.1.2 The RTPS Object Model

Figure 31 shows the object model that underlies the RTPS Protocol.

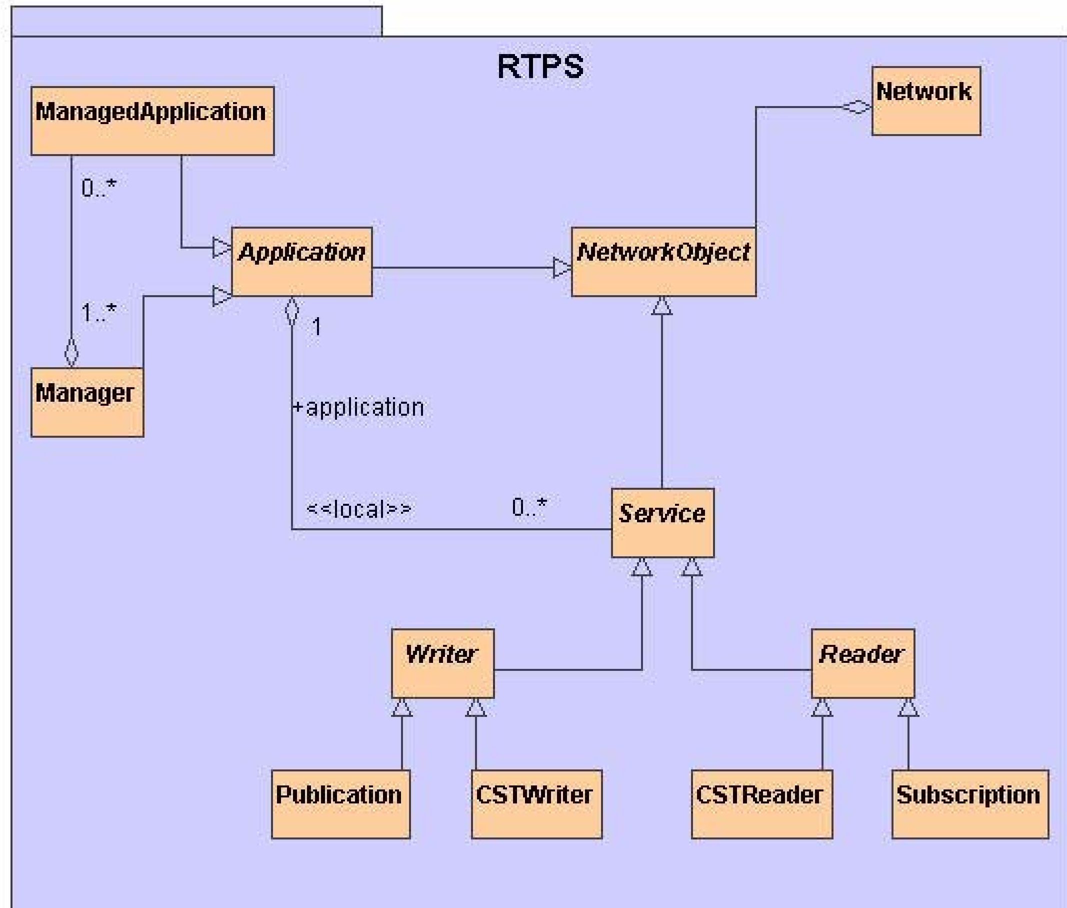


Figure 31 – Object model

The RTPS Protocol runs in a **Network** of **Applications**. An **Application** contains local **Services** through which the application sends or receives information using the RTPS Protocols. The **Services** are either **Readers** or **Writers**. **Writers** provide locally available data (a composite state or a stream of issues) on the network. **Readers** obtain this information from the network.

There are two broad classes of **Writers**: **Publications** and **CSTWriters**. A **Publication** is a **Writer** that provides issues to one or more instances of a **Subscription** using the publish-subscribe protocol and semantics.

The presence of a **Publication** in an **Application** indicates that the **Application** is willing to publish issues to matching subscriptions. The attributes of the **Publication** service object describe the contents (the *topic*), the type of the issues, and the quality of the stream of issues that is published on the **Network**.

There are two broad classes of **Readers**: **Subscriptions** and **CSTReaders**. A **Subscription** is a **Reader** that receives issues from one or more instances of **Publication**, using the publish-subscribe service.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The presence of a **Subscription** indicates that the **Application** wants to receive issues from **Publications** for a specific *topic* on the **Network**. The **Subscription** has attributes that identify the contents (the *topic*) of the data, the *type* of the issues and the quality with which it wants to receive the stream of issues.

The **CSTWriter** and **CSTReader** are the equivalent of the **Publication** and **Subscription**, respectively, but are used solely for the state-synchronization protocol and are provided so that applications have a means to exchange state information about each other.

Every **Reader** (**CSTReader** or **Subscription**) and **Writer** (**CSTWriter** or **Publication**) is part of an **Application**. The **Application** and its **Readers** and **Writers** are local, which is indicated in Figure 31 by the keyword "local" on the relationship between an **Application** and its **Services**.

There are two kinds of **Applications**: **Managers** and **ManagedApplications**. A **Manager** is a special **Application** that helps applications automatically discover each other on the **Network**. A **ManagedApplication** is an **Application** that is managed by one or more **Managers**. Every **ManagedApplication** is managed by at least one **Manager**.

The protocol provides two types of functionality:

- **Data Distribution: The RTPS protocol specifies the message formats and communication protocols that support the publish-subscribe protocol (to send issues from Publications to Subscriptions) and the Composite State Transfer (CST) protocol (to transfer state from a CSTWriter to a CSTReader) at various service levels.**
- **Administration:** The RTPS protocol defines a specific use of the CST protocol that enables **Applications** to obtain information about the existence and attributes of all the other **Applications** and **Services** on the **Network**. This *metatraffic* enables every **Application** to obtain a complete picture of all **Applications**, **Managers**, **Readers** and **Writers** in the **Network**. This information allows every **Application** to send the data to the right locations and to interpret incoming packets.

2.1.3 The Basic RTPS Transport Interface

RTPS is designed to run on an unreliable transport mechanism, such as UDP/IP. The protocols implement reliability in the transfer of issues and state.

RTPS takes advantage of the multicast capabilities of the transport mechanism, where one message from a sender can reach multiple receivers.

RTPS is designed to promote determinism of the underlying communication mechanism. The protocol also provides an open trade-off between determinism and reliability.

2.1.3.1 The Basic Logical Messages

The RTPS protocol uses five logical messages:

- **ISSUE**: Contains the Application's **UserData**. **ISSUE**s are sent by **Publications** to one or more **Subscriptions.**
- **VAR**: Contains information about the attributes of a **NetworkObject**, which is part of a composite state. **VAR**s are sent by **CSTWriters** to **CSTReaders**.
- **HEARTBEAT**: Describes the information that is available in a **Writer**. **HEARTBEAT**s are sent by a **Writer** (**Publication** or **CSTWriter**) to one or more **Readers** (**Subscription** or **CSTReader**).
- **GAP**: Describes the information that is no longer relevant to **Readers.**
- **ACK**: Provides information on the state of a **Reader** to a **Writer**.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Each of these logical messages are sent between specific **Readers** and **Writers** as follows:

- **Publication** to **Subscription(s)**: **ISSUE**s and **HEARTBEAT**s
- **Subscription** to a **Publication**: **ACK**s
- **CSTWriter** to a **CSTReader**: **VAR**s, **GAP**s and **HEARTBEAT**s
- **CSTReader** to a **CSTWriter**: **ACK**s

Readers and **Writers** are both senders and receivers of RTPS **Messages**. In the protocol, the logical messages **ISSUE**, **VAR**, **HEARTBEAT**, **GAP** and **ACK** can be combined into a single message in several ways to make efficient use of the underlying communication mechanism. clause 2.3 explains the format and construction of a **Message**.

2.1.3.2 Underlying Wire Representation

RTPS uses the CDR (Common Data Representation) as defined by the Object Management Group (OMG) to represent all basic data and structures.

Annex A of this section describes CDR and the specific choices that RTPS made in its usage of CDR.

2.1.4 Notational Conventions

2.1.4.1 Name Space

All the definitions in this PAS are part of the "RTPS" name-space. To facilitate reading and understanding, the name-space prefix has been left out of the definitions and classes in this PAS. For example, an implementation of RTPS will typically provide the service **RTPSPublication** or **RTPS::Publication**; however, in this document we will use the more simple **Publication**.

2.1.4.2 Representation of Structures

The following sections often define structures, such as:

```
typedef struct {
    octet[3] instanceId;
    octet    appKind;
} AppId;
```

These definitions use the OMG IDL (Interface Definition Language). When these structures are sent on the wire, they are encoded using the corresponding CDR representation.

Annex A of this section shows what standards describe this notation.

2.1.4.3 Representation of Bits and Bytes

This PAS often uses the following notation to represent an octet or byte:

```
+-+-+-+-+-+-+-+-+
|7|6|5|4|3|2|1|0|
+-+-+-+-+-+-+-+-+
```

In this notation, the leftmost bit (bit 7) is the most significant bit ("MSB") and the rightmost bit (bit 0) is the least significant bit ("LSB").

Streams of bytes are ordered per lines of 4 bytes each as follows:

```
0...2...........7...............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   first byte  |               |               |    4th byte   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                      -----------stream------------->>>>
```

In such representation, the byte that comes first in the stream is on the left. The bit on the extreme left is the MSB of the first byte; the bit on the extreme right is the LSB of the 4th byte.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

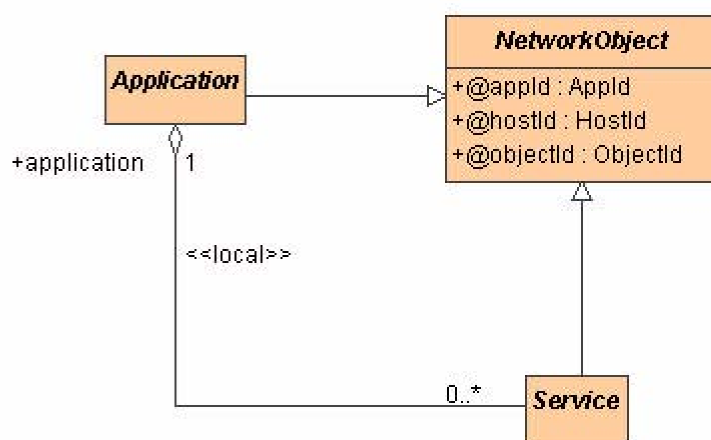
2.2 Structure Definitions

This subclause defines the Globally Unique ID (GUID) used to reference objects in a **Network** and the basic structures used in the protocol (to represent bitmaps, sequence numbers, etc.). These structures will be used in the following subclauses where the RTPS **Message** is defined.

2.2.1 Referring to Objects: the GUID

The GUID (Globally Unique Id) is a unique reference to an **Application** or a **Service** on the **Network**.

The GUID is built as a 12-octet triplet: <**HostId** hostId, **AppId** appId, **ObjectId** objectId>. The GUID should be a globally unique reference to one specific **NetworkObject** within the **Network**.



The **HostId** and **AppId** are defined as follows:

```
typedef octet[4] HostId;

typedef struct {
    octet[3] instanceId;
    octet    appKind;
} AppId;
```

where *appKind* is one of the following:

```
0x01  ManagedApplication
0x02  Manager
```

An implementation based on this version (1.0) of the protocol will consider anything other than the above two to be an unknown class.

The unknown *hostId* and *appId* are defined as follows:

```
#define HOSTID_UNKNOWN { 0x00, 0x00, 0x00, 0x00 }
#define APPID_UNKNOWN { 0x00, 0x00, 0x00, 0x00 }
```

2.2.1.1 The GUIDs of Applications

Every **Application** on the **Network** has GUID <hostId, appId, OID_APP>, where the constant OID_APP is defined as follows.

```
#define OID_APP {0x00,0x00,0x01,0xc1}
```

The implementation is free to pick the *hostId* and *appId*, as long as the last octet of the *appId* identifies the *appKind* and as long as every **Application** on the **Network** has a unique GUID.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

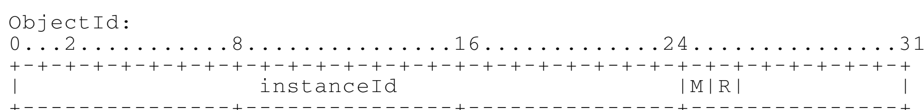
2.2.1.2 The GUIDs of the Services within an Application

The **Services** that are local to the **Application** with GUID <hostId, appId, OID_APP> have GUID <hostId, appId, *objectId*>. The *objectId* is the unique identification of the **NetworkObject** relative to the **Application**. The *objectId* also encapsulates what kind of **NetworkObject** this is, whether the object is a user-object or a meta-object and whether the *instanceId* is freely chosen by the middleware or is a *reserved* instanceId, which has special meaning to the protocol. One example of a reserved (protocol defined) objectId is OID_APP, which is used in the GUID of **Applications**.

The **ObjectId** structure is defined as follows:

```
typedef struct {
    octet[3] instanceId;
    octet    objKind;
} ObjectId;

#define OBJECTID_UNKNOWN { 0x0, 0x0, 0x0, 0x0 }
```

For *objKind*, the two most significant bits indicate whether the object is meta-level or user-level (M- bit) and whether its *instanceId* is chosen or reserved (R-bit), respectively.



M=1 The **NetworkObject** is a meta-object: it can be reached through the meta-ports of the **Application** to which it belongs (see 2.4).

R=1 The instanceId is reserved; it has a special meaning to the protocol. Subclause 2.5 lists all reserved instanceId's.

The last six bits of the objectId define the class to which the object belongs (**Application**, **Publication**, **Subscription**, **CSTWriter**, or **CSTReader**). Table 1 provides an overview. The meaning of the message IDs is fixed in this major version (1). New *objKinds* may be added in higher minor versions as the RTPS object-model is extended with new classes.

Table 1 – objKind octet of an objectId

Class of Object	Normal User-object	Reserved User-object	Normal Meta-object	Reserved Meta-object
unknown	0x00	0x40	0x80	0xc0
Application	0x01	0x41	0x81	0xc1
CSTWriter	0x02	0x42	0x82	0xc2
Publication	0x03	0x43	0x83	0xc3
Subscription	0x04	0x44	0x84	0xc4
CSTReader	0x07	0x47	0x87	0xc7

2.2.2 Building Blocks of RTPS Messages

This section describes the basic structures that are used inside RTPS **Messages**.

2.2.2.1 VendorId

This structure identifies the vendor of the middleware implementing the RTPS protocol and allows this vendor to add specific extensions to the protocol. The vendor ID does not refer to the vendor of the device or product that contains RTPS middleware.

```
typedef struct {
    octet major;
    octet minor;
} VendorId;
```

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The currently assigned vendor IDs are listed in Table 2.

Table 2 – Vendor IDs

Major	Minor	Name
0x00	0x00	VENDOR_ID_UNKNOWN
0x01	0x01	Real-Time Innovations, Inc., CA, USA

2.2.2.2 ProtocolVersion

The following structure describes the protocol version.

```
typedef struct {
    octet major;
    octet minor;
} ProtocolVersion;
```

Implementations following this version of the PAS implement protocol version 1.0 (major = 1, minor = 0).

```
#define PROTOCOL_VERSION_1_0 { 0x1, 0x0 }
```

2.2.2.3 SequenceNumber

A sequence number, N, is a 64-bit signed integer, that can take values in the range: $-2^{63} <= N <= 2^{63}-1$.

On the wire, it is represented using two 32-bit integers as follows:

```
typedef struct {
    long high;
    unsigned long low;
} SequenceNumber;
```

Using this structure, the sequence number is: $high * 2^{32} + low$.

The sequence number, 0, and negative sequence numbers are used to indicate special cases:

```
#define SEQUENCE_NUMBER_NONE     0
#define SEQUENCE_NUMBER_UNKNOWN -1
```

2.2.2.4 Bitmap

Bitmaps are used as parts of several messages to provide binary information about individual sequence numbers within a range. The representation of the **Bitmap** includes the length of the **Bitmap** in bits and the first **SequenceNumber** to which the **Bitmap** applies.

```
Bitmap:
0...2...........8...............16.............24...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                    SequenceNumber bitmapBase                  +
|                                                               |
+---------------+---------------+---------------+---------------+
|                    long      numBits                          |
+---------------+---------------+---------------+---------------+
|                    long      bitmap[0]                        |
+---------------+---------------+---------------+---------------+
|                    long      bitmap[1]                        |
+---------------+---------------+---------------+---------------+
|                               ...                             |
+---------------+---------------+---------------+---------------+
|                    long      bitmap[M-1]  M = (numBits+31)/32 |
+---------------+---------------+---------------+---------------+
```

Given a **Bitmap,** *bitmap*, the boolean value of the bit pertaining to SequenceNumber *N*, where *bitmapBase* <= *N* < *bitmapBase*+*numBits,* is:

```
bit(N) = bitmap[deltaN/32]  &  (1 << (31 - deltaN%32) )
```

where

```
deltaN = N - bitmapBase
```

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The bitmap does not indicate anything about sequence numbers outside of the range [bitmapBase, bitmapBase+numBits-1].

A valid bitmap must satisfy the following conditions:

- o bitmapBase >= 1
- o 0 <= numBits <= 256
- o there are M=(numBits+31)/32 longs containing the pertinent bits

This document uses the following notation for a specific bitmap:

```
bitmapBase/numBits:bitmap
```

In the bitmap, the bit corresponding to sequence number *bitmapBase* is on the left. The ending "0" bits can be represented as one "0".

For example, in bitmap “1234/12:00110”, bitmapBase=1234 and numBits=12. The bits apply as follows to the sequence numbers:

Table 3 – Example of bitmap: meaning of "1234/12:00110”

Sequence	Bit
1234	0
1235	0
1236	1
1237	1
1238-1245	0

2.2.2.5 NtpTime

Timestamps follow the NTP standard and are represented on the wire as a pair of integers containing the high- and low-order 32 bits:

```
typedef struct {
    long seconds;           // time in seconds
    unsigned long fraction; // time in seconds / 2^32
NtpTime;
```

Time is expressed in seconds using the following formula:

```
seconds + (fraction / 2^(32))
```

The RTPS protocol does not require a concept of absolute time.

2.2.2.6 IPAddress

An IP address is a 4-byte unsigned number:

```
typedef unsigned long IPAddress
```

An IP address of zero is an invalid IP address:

```
#define IPADDRESS_INVALID 0
```

The mapping between the dot-notation "a.b.c.d" of an IP address and its representation as an unsigned long is as follows:

```
IPAddress ipAddress = ( ( ( a * 256 + b ) * 256 ) + c ) * 256 + d
```

For example, IP address "127.0.0.1" corresponds to the unsigned long number 2130706433 or 0x7F000001.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.2.2.7 Port

A port number is a 4-byte unsigned number:

```
typedef unsigned long Port
```

The port number zero is an invalid port-number:

```
#define PORT_INVALID 0
```

If a port number represents an IPv4 UDP port, only the range of unsigned short numbers from 0x1 to 0x0000ffff is valid.

2.3 RTPS Message Format

2.3.1 Overall Structure of RTPS Messages

The overall structure of a **Message** includes a leading **Header** followed by a variable number of **Submessages**. Each **Submessage** starts aligned on a 32-bit boundary with respect to the start of the **Message**.

```
Message:
0...2...........7...............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header                                                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Submessage                                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.................................................................
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Submessage                                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A **Message** has a well-known length. This length is not sent explicitly by the RTPS protocol but is part of the underlying transport with which **Messages** are sent. In the case of UDP/IP, the length of the **Message** is the length of the UDP payload.

2.3.2 Submessage Structure

The general structure of each **Submessage** in a **Message** is as follows:

```
Submessage:
0...2...........7...............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| submessageId  |      flags  |E|   ushort octetsToNextHeader   |
+---------------+---------------+---------------+---------------+
|                                                               |
~                       contents of submessage                  ~
|                                                               |
+---------------+---------------+---------------+---------------+
```

This general structure cannot change in this major version (1) of the protocol. Subclauses 2.3.2.1 through 2.3.2.3 describe the meaning of the three fields of the submessage header: *submessageId*, *flags* and *octetsToNextHeader*.

2.3.2.1 submessageId in the Submessage Header

This octet identifies the kind of **Submessage**. Submessages with IDs 0x00 to 0x7f (inclusive) are protocol-specific. They are defined as part of the RTPS protocol. Version 1.0 defines the following submessages:

```
enum SubmessageId {
PAD        = 0x01,
VAR        = 0x02,
ISSUE      = 0x03,
ACK        = 0x06,
HEARTBEAT  = 0x07,
GAP        = 0x08,
INFO_TS    = 0x09,
INFO_SRC   = 0x0c,
INFO_REPLY = 0x0d,
INFO_DST   = 0x0e
};
```

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The meaning of the submessage IDs cannot be modified in this major version (1). Additional submessages can be added in higher minor versions. Submessages with ID's 0x80 to 0xff (inclusive) are vendor-specific; they will not be defined by the protocol. Their interpretation is dependent on the *vendorId* that is current when the submessage is encountered. Section describes how the current *vendorId* is determined. The current list of *vendorId*'s is provided in 2.2.2.1 .

2.3.2.2 Flags in the Submessage Header

The least-significant bit (LSB) of the flags is always present in all **Submessages** and represents the endianness used to encode the information in the **Submessage**. E=0 means big-endian, E=1 means little-endian.

Other bits in the flag have interpretations that depend on the type of **Submessage**.

In the following descriptions of the **Submessages**, the character '**X**' is used to indicate a flag that is unused in version 1.0 of the protocol. RTPS implementations of version 1.0 should set these to zero when sending and ignore these when receiving. Higher minor versions of the protocol can use these flags.

2.3.2.3 octetsToNextHeader in the Submessage Header

The final two octets of the **Submessage** header contain the number of octets from the first octet of the contents of the submessage until the first octet of the header of the next **Submessage**. The representation of this field is a CDR unsigned short (ushort). If the **Submessage** is the last one in the **Message**, the *octetsToNextHeader* field contains the number of octets remaining in the **Message**.

2.3.3 How to Interpret a Message

The interpretation and meaning of a **Submessage** within a **Message** may depend on the previous **Submessages** within that same **Message**. Therefore the receiver of a **Message** must maintain state from previously deserialized **Submessages** in the same **Message**.

sourceVersion The major and minor version with which the following submessages need to be interpreted.

sourceVendorId The vendor identification with which the following vendor-specific extensions need to be interpreted.

sourceHostId, sourceAppId The originator's host and application identifiers. The following submessages need to be identified as if they are coming from this host and application.

destHostId, destAppId The destination's host and application identifiers. The following submessages need to be identified as if they are meant for this host and application.

unicastReplyIPAddress, unicastReplyPort An explicit IP address and port that provides an additional direct way for the receiver to reply directly to the originator over unicast.

multicastReplyIP Address, multicastReplyPort An explicit IP address and port that provides an additional direct way for the receiver to reach the originator (and potentially many others) over multicast.

haveTimestamp, timestamp The timestamp applying to all the following submessages.

2.3.3.1 Rules Followed By A Message Receiver

The following algorithm outlines the rules that a receiver of any **Message** must follow:

1. If a 4-byte **Submessage** header cannot be read, the rest of the **Message** is considered invalid.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2. The last two bytes of a **Submessage** header, the *octetsToNextHeader* field, contains the number of octets to the next **Submessage**. If this field is invalid, the rest of the **Message** is invalid.
3. The first byte of a **Submessage** header is the *submessageId*. A **Submessage** with an unknown ID must be ignored and parsing must continue with the next **Submessage**. Concretely: an implementation of RTPS 1.0 must ignore any **Submessages** with IDs that are outside of the **SubmessageId** list used by version 1.0. IDs in the vendor-specific range coming from a *vendorId* that is unknown must be ignored and parsing must continue with the next **Submessage**.
4. The second byte of a **Submessage** header contains flags; unknown flags should be skipped. An implementation of RTPS 1.0 should skip all flags that are marked as "**X**" (unused) in the protocol.
5. A valid *octetsToNextHeader* field must *always* be used to find the next **Submessage**, even for **Submessages** with unknown IDs.
6. A known but invalid **Submessage** invalidates the rest of the **Message**. Subclauses 2.3.5 through 2.3.14 each describe known **Submessage** and when it should be considered invalid.

Reception of a valid header and/or submessage has two effects:

— It can change the state of the receiver; this state influences how the following **Submessages** in the **Message** are interpreted. Subclauses 2.3.5 through 2.3.14 show how the state changes for each **Submessage**. In this version of the protocol, only the **Header** and the **Submessages INFO_SRC**, **INFO_REPLY** and **INFO_TS** change the state of the receiver.

— The **Submessage**, interpreted within the **Message**, has a logical interpretation: it encodes one of the five basic RTPS messages: **ACK**, **GAP**, **HEARTBEAT**, **ISSUE** or **VAR**.

Subclauses 2.3.5 through 2.3.14 describe the detailed behavior of the **Header** and every **Submessage**.

2.3.4 Header

This is the **Header** found at the beginning of every **Message**.

2.3.4.1 Format

```
0...2...........7...............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      'R'      |      'T'      |      'P'      |      'S'      |
+---------------+---------------+---------------+---------------+
| ProtocolVersion version       | VendorId vendorId             |
+---------------+---------------+---------------+---------------+
| HostId hostId                                                 |
+---------------+---------------+---------------+---------------+
| AppId appId                                                   |
+---------------+---------------+---------------+---------------+
```

2.3.4.2 Validity

A **Header** is *invalid* when any of the following are true:

— The **Message** has less than the required number of octets to contain a full **Header**.

— Its first four octets are not '**R**' '**T**' '**P**' '**S**'.

— The major protocol version is larger than the major protocol version supported by the implementation.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.3.4.3 Change in State of the Receiver

```
sourceHostId   = Header.hostId
sourceAppId    = Header.appId
sourceVersion  = Header.version
sourceVendorId = Header.vendorId
haveTimestamp  = false
```

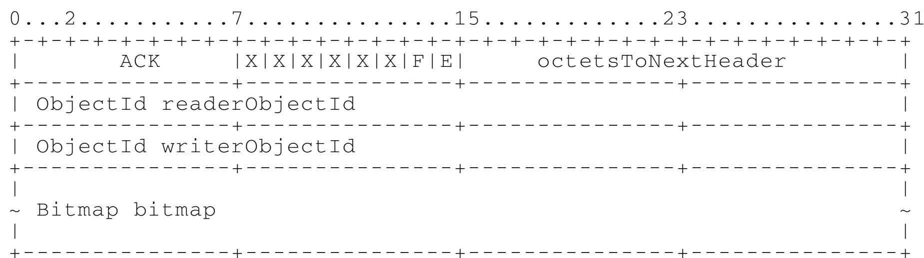
2.3.4.4 Logical Interpretation

None

2.3.5 ACK

This submessage is used to communicate the state of a **Reader** to a **Writer**.

2.3.5.1 Submessage Format



```
0...2...........7..............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       ACK     |X|X|X|X|X|X|F|E|     octetsToNextHeader        |
+---------------+---------------+---------------+---------------+
| ObjectId readerObjectId                                       |
+---------------+---------------+---------------+---------------+
| ObjectId writerObjectId                                       |
+---------------+---------------+---------------+---------------+
|                                                               |
~ Bitmap bitmap                                                 ~
|                                                               |
+---------------+---------------+---------------+---------------+
```

2.3.5.2 Validity

This submessage is *invalid* when any of the following is true:

- *octetsToNextHeader* is too small.
- *bitmap* is invalid.

2.3.5.3 Change in State of the Receiver

None

2.3.5.4 Logical Interpretation

Table 4 – Interpretation of ACK Submessage

Field	Value
FINAL-bit	ACK.F
readerGUID	<sourceHostId, sourceAppId, ACK.readerObjectId>
writerGUID	<destHostId, destAppId, ACK.writerObjectId>
replyIPAddressPortList	{ unicastReplyIPAddress:unicastReplyPort, multicastReplyIPAddress:multicastReplyPort }
bitmap	ACK.bitmap

FINAL-bit ACK.F : When the F-bit is set, the application sending the ACK does not expect a response to the ACK.

readerGUID <sourceHostId, sourceAppId, ACK.readerObjectId> : The GUID of the **Reader** that acknowledges receipt of certain sequence numbers and/or requests to receive certain sequence numbers.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

writerGUID <destHostId, destAppId, ACK.writerObjectId> : The GUID of the **Writer** that the reader has received these sequence numbers from and/or wants to receive these sequence numbers from.

replyIPAddressPortList { unicastReplyIPAddress : unicastReplyPort, multicastReplyIPAddress : multicastReplyPort } : This is an additional list of addresses that the receiving application can use to respond to this ACK.

bitmap ACK.bitmap : A "0" in this bitmap means that the corresponding sequence-number is missing. A "1" in the bitmap conveys no information, that is, the corresponding sequence number may or may not be missing. By sending an ACK, the readerGUID object acknowledges receipt of all messages up to and including the sequence number (bitmap.bitmapBase -1).

2.3.6 GAP

This submessage is sent from a **CSTWriter** to a **CSTReader** to indicate that a range of sequence numbers is no longer relevant. The set may be a contiguous range of sequence numbers or a specific set of sequence numbers.

2.3.6.1 Submessage Format

```
0...2...........7..............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      GAP      |X|X|X|X|X|X|X|E|    octetsToNextHeader         |
+---------------+---------------+---------------+---------------+
| ObjectId readerObjectId                                       |
+---------------+---------------+---------------+---------------+
| ObjectId writerObjectId                                       |
+---------------+---------------+---------------+---------------+
|                                                               |
+ SequenceNumber firstSeqNumber                                 +
|                                                               |
+---------------+---------------+---------------+---------------+
|                                                               |
~ Bitmap bitmap                                                 ~
|                                                               |
+---------------+---------------+---------------+---------------+
```

2.3.6.2 Validity

This submessage is *invalid* when any of the following are true:

- *octetsToNextHeader* is too small.
- *bitmap* is invalid.
- *firstSeqNumber* is 0 or negative.

2.3.6.3 Change in State of the Receiver

None

2.3.6.4 Logical Interpretation

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Table 5 – Interpretation of GAP Submessage

Field	Value
readerGUID	<destHostId, destAppId, GAP.readerObjectId>
writerGUID	<sourceHostId, sourceAppId, GAP.writerObjectId>
ACKIPAddressPortList	{ unicastReplyIPAddress:unicastReplyPort }
gapList	{ GAP.firstSeqNumber, GAP.firstSeqNumber+1,..., GAP.bitmap.bitmapBase-1 } *and* all sequence numbers that have a corresponding bit set to 1 in the bitmap

readerGUID <destHostId, destAppId, GAP.readerObjectId> : The GUID of the **CSTReader** for which the *gapList* is meant. The GAP.readerObjectId can be **OBJECTID_UNKNOWN**, in which case the **GAP** applies to all **Readers** within the **Application** <destHostId, destAppId>.

writerGUID <sourceHostId, sourceAppId, GAP.writerObjectId> : The GUID of the **CSTWriter** to which the *gapList* applies.

ACKIPAddressPortList { unicastReplyIPAddress : unicastReplyPort } : If the **CSTReader** that receives this submessage needs to reply with an **ACK** submessage, then this **ACK** can be sent to one of the explicit destinations in this list.

gapList The list of sequence numbers that are no longer available in the *writerObject*. This list is the union of:

- All the sequence numbers in the range from *GAP.firstSeqNumber* up to *GAP.bitmap.bitmapBase* - 1. This list is empty if the *firstSeqNumber* is greater than or equal to the *bitmapBase* of the *bitmap*. *GAP.firstSeqNumber* should always be greater than or equal to 1.

and

- The sequence numbers that have the corresponding bit in the *bitmap* set to 1.

2.3.6.5 Example

A **GAP** with:

- *firstSeqNumber* = 12
- *bitmap* = 17/5:0011101

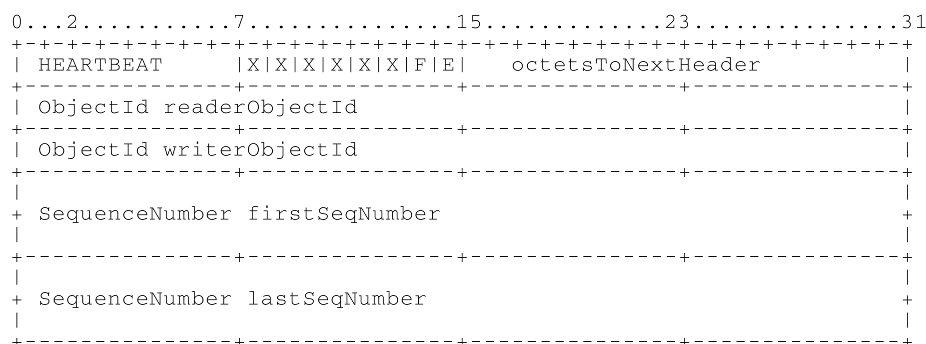
means that the *gapList* = {12, 13, 14, 15, 16, 19, 20, 22}.

2.3.7 HEARTBEAT

This message is sent from a **Writer** to a **Reader** to communicate the sequence numbers of data that the **Writer** has available.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.3.7.1 Submessage Format



```
0...2...........7..............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| HEARTBEAT     |X|X|X|X|X|X|F|E|   octetsToNextHeader          |
+---------------+---------------+---------------+---------------+
| ObjectId readerObjectId                                       |
+---------------+---------------+---------------+---------------+
| ObjectId writerObjectId                                       |
+---------------+---------------+---------------+---------------+
|                                                               |
+ SequenceNumber firstSeqNumber                                 +
|                                                               |
+---------------+---------------+---------------+---------------+
|                                                               |
+ SequenceNumber lastSeqNumber                                  +
|                                                               |
+---------------+---------------+---------------+---------------+
```

2.3.7.2 Validity

This submessage is *invalid* when any of the following are true:

- *octetsToNextHeader* is too small.
- *firstSeqNumber* is less than 0.
- *lastSeqNumber* is less than 0.
- *lastSeqNumber* is strictly less than *firstSeqNumber*.

2.3.7.3 Change in State of the Receiver

None

2.3.7.4 Logical Interpretation

Table 6 – Interpretation of HEARTBEAT Submessage

Field	Value
FINAL-bit	HEARTBEAT.F
readerGUID	<destHostId, destAppId, HEARTBEAT.readerObjectId>
writerGUID	<sourceHostId, sourceAppId, HEARTBEAT.writerObjectId>
ACKIPAddressPortList	{ unicastReplyIPAddress:unicastReplyIPPort }
firstSeqNumber	HEARTBEAT.firstSeqNumber
lastSeqNumber	HEARTBEAT.lastSeqNumber

FINAL-bit HEARTBEAT.F : When the F-bit is set, the application sending the **HEARTBEAT** does not require a response.

readerGUID <destHostId, destAppId, HEARTBEAT.readerObjectId> : The **Reader** to which the heartbeat applies. The HEARTBEAT.readerObjectId can be **OBJECTID_UNKNOWN**, in which case the **HEARTBEAT** applies to all **Readers** of that *writerGUID* within the **Application** <destHostId, destAppId>.

writerGUID <sourceHostId, sourceAppId, HEARTBEAT.writerObjectId> : The **Writer** to which the **HEARTBEAT** applies.

ACKIPAddressPortList { unicastReplyIPAddress : unicastReplyIPPort } : An additional list of destinations where responses (**ACKs**) to this submessage can be sent.

firstSeqNumber HEARTBEAT.firstSeqNumber : The first sequence number, *firstSeqNumber*, that is still available and meaningful in the *writerObject*. This field must be greater than or

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

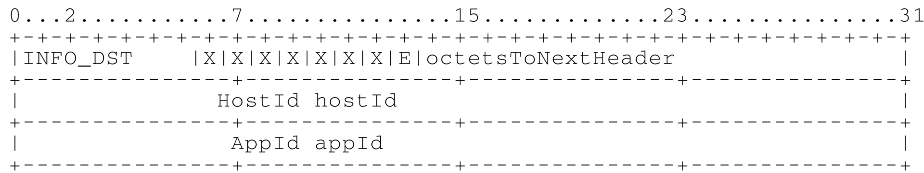
equal to zero. If it is equal to **SEQUENCE_NUMBER_NONE**, the **Writer** has no data available.

lastSeqNumber HEARTBEAT.lastSeqNumber : The last sequence number, *lastSeqNumber*, that is available in the **Writer**. This field must be greater than or equal to *firstSeqNumber*. If *firstSeqNumber* is **SEQUENCE_NUMBER_NONE**, *lastSeqNumber* must also be **SEQUENCE_NUMBER_NONE**.

2.3.8 INFO_DST

This submessage modifies the logical destination of the submessages that follow it.

2.3.8.1 Submessage Format



```
0...2...........7...............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|INFO_DST      |X|X|X|X|X|X|X|E|octetsToNextHeader              |
+---------------+---------------+---------------+---------------+
|               HostId hostId                                    |
+---------------+---------------+---------------+---------------+
|                AppId appId                                     |
+---------------+---------------+---------------+---------------+
```

2.3.8.2 Validity

This submessage is *invalid* when:

– *octetsToNextHeader* is too small.

2.3.8.3 Change In State Of The Interpreter

```
if(INFO_DST.hostId != HOSTID_UNKNOWN) {
    destHostId = INFO_DST.hostId
} else {
    destHostId = hostId of application receiving the message
}

if(INFO_DST.appId != APPID_UNKNOWN) {
    destAppId = INFO_DST.appId
} else {
    destAppId = appId of application receiving the message
}
```

In other words, an INFO_DST with a HOSTID_UNKNOWN means that any host may interpret the following submessages as if they were meant for it. Similarly, an INFO_DST with a APPID_UNKNOWN means that any application may interpret the following submessages as if they were meant for it.

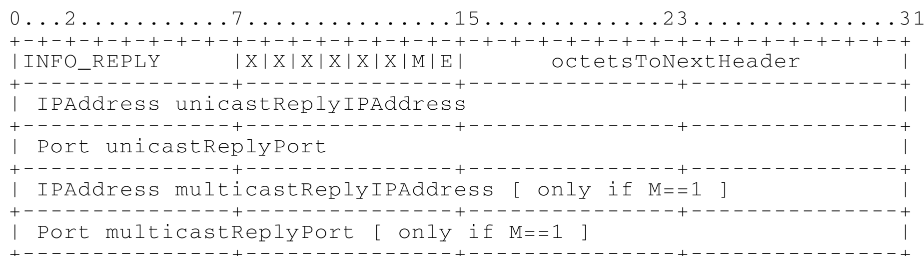
2.3.8.4 Logical Interpretation

None; this only affects the interpretation of the submessages that follow it.

2.3.9 INFO_REPLY

This submessage contains explicit information on where to send a reply to the submessages that follow it within the same message.

2.3.9.1 Submessage Format



```
0...2...........7...............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|INFO_REPLY     |X|X|X|X|X|X|M|E|      octetsToNextHeader       |
+---------------+---------------+---------------+---------------+
| IPAddress unicastReplyIPAddress                                |
+---------------+---------------+---------------+---------------+
| Port unicastReplyPort                                          |
+---------------+---------------+---------------+---------------+
| IPAddress multicastReplyIPAddress [ only if M==1 ]             |
+---------------+---------------+---------------+---------------+
| Port multicastReplyPort [ only if M==1 ]                       |
+---------------+---------------+---------------+---------------+
```

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.3.9.2 Validity

This submessage is *invalid* when the following is true:

- *octetsToNextHeader* is too small.

2.3.9.3 Change in State of the Receiver

```
if ( INFO_REPLY.unicastReplyIPAddress != IPADDRESS_INVALID) {
    unicastReplyIPAddress  = INFO_REPLY.unicastReplyIPAddress;
}
unicastReplyPort  = INFO_REPLY.replyPort
if ( M==1 ) {
    multicastReplyIPAddress  = INFO_REPLY.multicastReplyIPAddress
    multicastReplyPort       = INFO_REPLY.multicastReplyPort
} else {
    multicastReplyIPAddress  = IPADDRESS_INVALID
    multicastReplyPort       = PORT_INVALID
}
```

2.3.9.4 Logical Interpretation

None, this only affects the interpretation of the submessages that follow it.

2.3.10 INFO_SRC

This submessage modifies the logical source of the submessages that follow it.

2.3.10.1 Submessage Format

```
0...2...........7...............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| INFO_SRC      |X|X|X|X|X|X|X|E|   octetsToNextHeader          |
+---------------+---------------+---------------+---------------+
| IPAddress appIPAddress                                        |
+---------------+---------------+---------------+---------------+
| ProtocolVersion version       | VendorId vendorId             |
+---------------+---------------+---------------+---------------+
| HostId hostId                                                 |
+---------------+---------------+---------------+---------------+
| AppId appId                                                   |
+---------------+---------------+---------------+---------------+
```

2.3.10.2 Validity

This submessage is *invalid* when the following is true:

- *octetsToNextHeader* is too small.

2.3.10.3 Change in State of the Receiver

```
sourceHostId             = INFO_SRC.hostId
sourceAppId              = INFO_SRC.appId
sourceVersion            = INFO_SRC.version
sourceVendorId           = INFO_SRC.vendorId
unicastReplyIPAddress    = INFO_SRC.appIPAddress
unicastReplyPort         = PORT_INVALID
multicastReplyIPAddress  = IPADDRESS_INVALID
multicastReplyPort       = PORT_INVALID
haveTimestamp            = false
```

2.3.10.4 Logical Interpretation

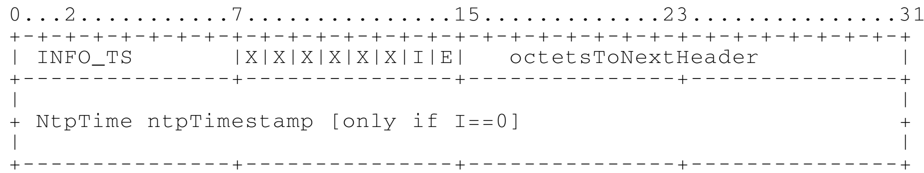
None, this only affects the interpretation of the submessages that follow it.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.3.11 INFO_TS

This submessage is used to send a timestamp which applies to the submessages that follow within the same message.

2.3.11.1 Submessage Format



```
0...2...........7...............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| INFO_TS       |X|X|X|X|X|X|I|E|   octetsToNextHeader          |
+---------------+---------------+---------------+---------------+
|                                                               |
+ NtpTime ntpTimestamp [only if I==0]                           +
|                                                               |
+---------------+---------------+---------------+---------------+
```

2.3.11.2 Validity

This submessage is *invalid* when the following is true:

– *octetsToNextHeader* is too small.

2.3.11.3 Change in State of the Receiver

```
if (INFO_TS.I==0) {
    haveTimestamp   =  true
    timestamp       = INFO_TS.ntpTimestamp
} else {
    haveTimestamp   =  false
}
```

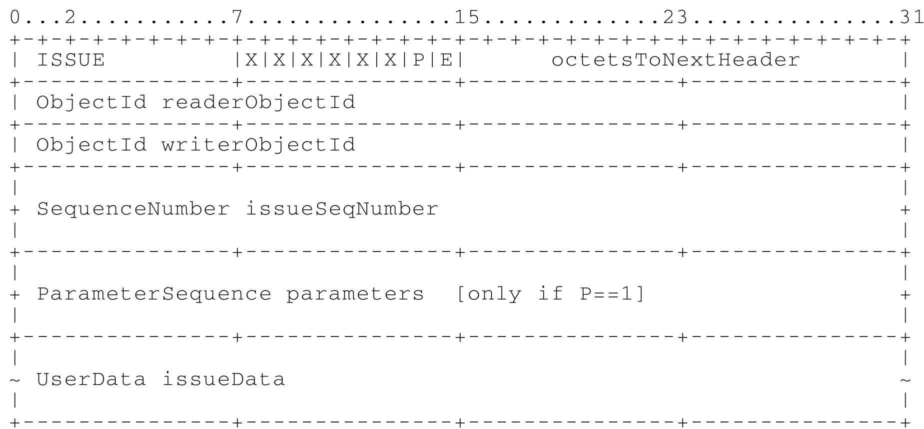
2.3.11.4 Logical Interpretation

None, this only affects the interpretation of the submessages that follow it.

2.3.12 ISSUE

This submessage is used to send issues from a **Publication** to a **Subscription**.

2.3.12.1 Submessage Format



```
0...2...........7...............15.............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ISSUE         |X|X|X|X|X|X|P|E|      octetsToNextHeader       |
+---------------+---------------+---------------+---------------+
| ObjectId readerObjectId                                       |
+---------------+---------------+---------------+---------------+
| ObjectId writerObjectId                                       |
+---------------+---------------+---------------+---------------+
|                                                               |
+ SequenceNumber issueSeqNumber                                 +
|                                                               |
+---------------+---------------+---------------+---------------+
|                                                               |
+ ParameterSequence parameters  [only if P==1]                  +
|                                                               |
+---------------+---------------+---------------+---------------+
|                                                               |
~ UserData issueData                                            ~
|                                                               |
+---------------+---------------+---------------+---------------+
```

2.3.12.2 Validity

This submessage is *invalid* when any of the following are true:

– *octetsToNextHeader* is too small.

– *issueSeqNumber* is either not strictly positive (1,2,...) or is not **SEQUENCE_ NUMBER_UNKNOWN**.

– the parameter sequence is invalid.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.3.12.3 Change in State of the Receiver

None

2.3.12.4 Logical Interpretation

Table 7 – Interpretation of ISSUE Submessage

Field	Value
subscriptionGUID	<destHostid, destAppId, ISSUE.readerObjectId>
publicationGUID	<sourceHostId, sourceAppId, ISSUE.writerObjectId>
issueSeqNumber	ISSUE.issueSeqNumber
(parameters)	ISSUE.parameters (iff ISSUE.P==1)
ACKIPAddressPortList	{ unicastReplyIPAddress:unicastReplyPort }
(timestamp)	timestamp (present iff haveTimestamp == true)
issueData	ISSUE.issueData

subscriptionGUID <destHostid, destAppId, ISSUE.readerObjectId> : The **Subscription** for which the ISSUE is meant. The ISSUE.readerObjectId can be **OBJECTID_UNKNOWN**, in which case the **ISSUE** applies to all **Subscriptions** within the **Application** <destHostId, destAppId>.

publicationGUID <sourceHostId, sourceAppId, ISSUE.writerObjectId> : The **Publication** object that originated this issue.

issueSeqNumber ISSUE.issueSeqNumber : The sequence number of this issue; this should either be a strictly positive number (1,2,3,...) or the special sequence-number **SEQUENCENUMBER_UNKNOWN**. The latter may be used by a simple publication that does not number consecutive issues.

parameters (optional) ISSUE.parameters : This is present iff P == 1. These parameters will allow future extensions of the protocol. An implementation of RTPS 1.0 can ignore the contents of this **ParameterSequence**.

ACKIPAddressPortList { unicastReplyIPAddress : unicastReplyPort } : The destinations to which the **Publication** can send an **ACK** message in response to this **ISSUE**.

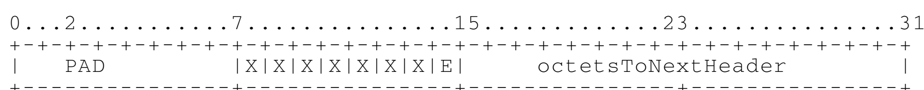
timestamp (optional) Timestamp of this issue. This is present iff Timestamp == true.

issueData ISSUE.issueData : The actual user data in this issue.

2.3.13 PAD

This submessage has no meaning.

2.3.13.1 Submessage Format



--`,````...```...`...``,``-`-`-,,`,,`,`,,`---

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.3.13.2 Validity

This submessage is always valid.

2.3.13.3 Change in State of the Receiver

None

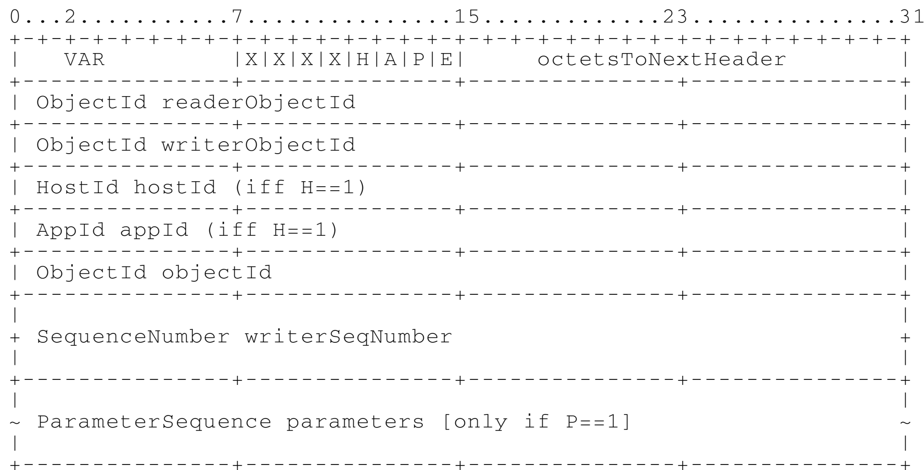
2.3.13.4 Logical Submessage Generated On Reception

None; the receiver skips the **PAD** using *octetsToNextHeader*.

2.3.14 VAR

This submessage is used to communicate information about a **NetworkObject** (which is part of the Composite State). It is sent from a **CSTWriter** to a **CSTReader**.

2.3.14.1 Submessage Format



2.3.14.2 Validity

This submessage is *invalid* when any of the following are true:

- *octetsToNextHeader* is too small.
- *writerSeqNumber* is not strictly positive (1, 2, ...) or is **SEQUENCE_NUMBER_ UNKNOWN**.
- the parameter sequence is invalid.

2.3.14.3 Change in State of the Receiver

None

2.3.14.4 Logical interpretation

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Table 8 – Interpretation of VAR Submessage

Field	Value
readerGUID	<destHostId, destAppId, VAR.readerObjectId>
writerGUID	<sourceHostId, sourceAppId, VAR.writerObjectId>
objectGUID	<VAR.hostId, VAR.appId, VAR.objectId> (iff H == 1) <sourceHostId, sourceAppId, VAR.objectId> (iff H == 0)
writerSeqNumber	VAR.writerSeqNumber
(timestamp)	current.timestamp if curent.haveTimestamp == true
(parameters)	VAR.parameters and VAR.P
ALIVE-bit	VAR.A

ACKIPAddressPortList	{
	unicastReplyIPAddress:unicastReplyIPPort,
	writer->IPAddressPortList()
	}

readerGUID <destHostId, destAppId, VAR.readerObjectId> : The **Reader** to which the heartbeat applies. The VAR.readerObjectId can be **OBJECTID_UNKNOWN**, in which case the **VAR** applies to all **Readers** of that *writerGUID* within the **Application** <destHostId, destAppId>.

writerGUID <sourceHostId, sourceAppId, VAR.writerObjectId> : The **CSTWriter** that sent the information.

objectGUID <VAR.hostId, VAR.appId, VAR.objectId> (iff H == 1) or <sourceHostId, sourceAppId, VAR.objectId> (iff H == 0) : The object this information (contained in the parameters) is about.

writerSeqNumber VAR.writerSeqNumber : Incremented each time a change in the Composite State provided by the **CSTWriter** occurs. This should be a strictly positive number (1, 2, ...). Or, the special sequence number, **SEQUENCE_NUMBER_UNKNOWN**, may be sent to indicate that the sender does not keep track of the sequence number.

timestamp (optional) current.timestamp : This is present iff curent.haveTimestamp == true. Timestamp of the new parameters sent with this submessage.

parameters (optional) VAR.parameters : This is present iff VAR.P == 1. Contains information about the object.

ALIVE-bit VAR.A : See 2.7 .

ACKIPAddressPortList { unicastReplyIPAddress : unicastReplyIPPort, writer->IPAddressPortList() } : Where to sent **ACKs** in reply to this submessage.

2.3.15 Versioning and Extensibility

An implementation based on this version (1.0) of the protocol should be able to process RTPS messages not only with the same major version (1) but possibly higher minor versions.

2.3.15.1 Allowed Extensions Within This Major Version

Within this major version, future minor versions of the protocol can augment the protocol in the following ways:

- Additional submessages with other *submessageIds* can be introduced and used anywhere in an RTPS message. Therefore, a 1.0 implementation should skip over unknown submessages (using the *octetsToNextHeader* field in the submessage header).

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

- Additional fields can be added to the end of a submessage that was already defined in the current minor version. Therefore, a 1.0 implementation should skip over possible additional fields in a submessage using the *octetsToNextHeader* field.
- Additional object-kinds and built-in objects with new IDs can be added; these should be ignored by the 1.0 implementation.
- Additional parameters with new IDs can be added; these should be ignored by the 1.0 implementation.

All such changes require an increase of the minor version number.

2.3.15.2 What Cannot Change Within This Major Version

The following items cannot be changed within the same major version:

- A submessage cannot be deleted.
- A submessage cannot be modified except as described in 2.3.15.1 .
- The meaning of the *submessageIds* (described in 2.3.2.1) cannot be modified.

All such changes require an increase of the major version number.

2.4 RTPS and UDP/IPv4

This section describes the mapping of RTPS on UDP/IP v4.

2.4.1 Concepts

2.4.1.1 RTPS Messages and the UDP Payload

When RTPS is used over UDP/IP, a **Message** is the contents (payload) of exactly one UDP/IP Datagram.

2.4.1.2 UDP/IP Destinations

A UDP/IP destination consists of an **IPAddress** and a **Port**. This document uses notation such as "12.44.123.92:1024" or "225.0.1.2:6701" to refer to such a destination. The IP address can be a unicast or multicast address.

2.4.1.3 Note On Relative Addresses

The RTPS protocol often sends IP addresses to a sender of **Messages**, so that the sender knows where to send future **Messages**. These destinations are always interpreted locally by the sender of UDP datagrams. Certain IP addresses, such as "127.0.0.1" have only relative meaning (i.e. they do not refer to a unique host).

2.4.2 RTPS Packet Addressing

The following subclauses describe how a sending application can construct a list of **IPAddress:Port** pairs that it can use to send **Messages** to remote **Services**. Every **Service** has a method, **IPAddressPortList()**, that represents this list. This **IPAddressPortList** is gathered by combining four sources:

- The well-known ports of the **Network**.
- The attributes of the **Application** in which the **Service** exists, as well as whether the **Application** is a **Manager** or a **ManagedApplication**.
- Whether the **Service** is user-level or meta-level (M-bit in the GUID).
- Additional attributes of the **Service** itself.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The sender's implementation is free to send the information to any valid destination(s) in this list and is encouraged to make good choices, depending on its network interfaces, resources or optimization concerns.

2.4.2.1 Well-known Ports

At the **Network** level, RTPS uses the following three well-known ports:

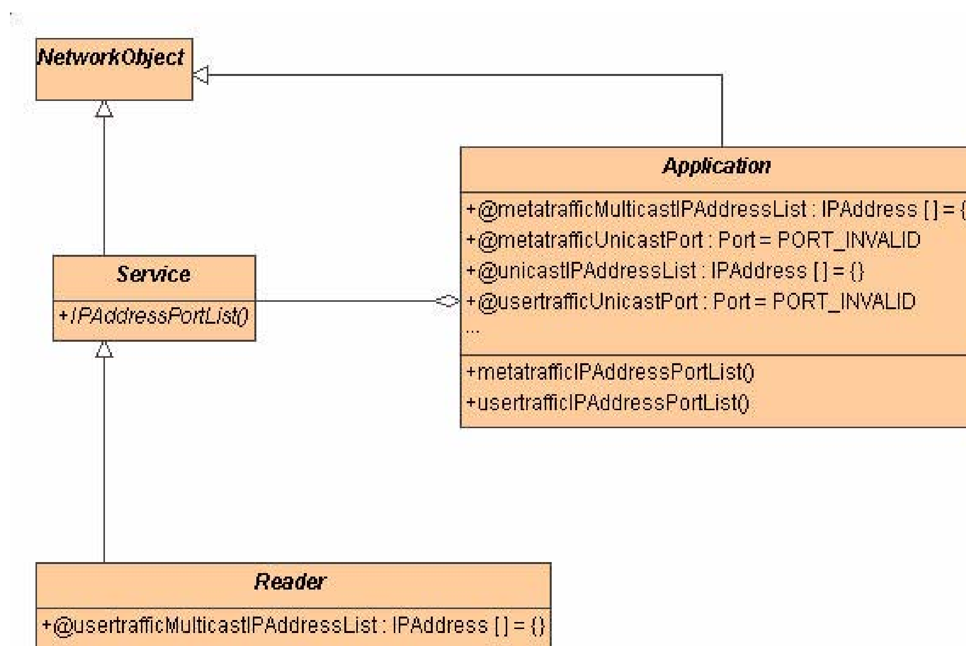
```
wellknownManagerPort = portBaseNumber + 10 * portGroupNumber

wellknownUsertrafficMulticastPort =
                        1 + portBaseNumber + 10 * portGroupNumber
wellknownMetatrafficMulticastPort =
                        2 + portBaseNumber + 10 * portGroupNumber
```

Within a **Network**, all applications need to use the same ***portBaseNumber***. Applications that want to communicate with each other use the same ***portGroupNumber***; applications that need to be isolated from each other use a different ***portGroupNumber***.

Each application needs to be configured with the correct ***portBaseNumber*** and ***portGroupNumber***.

Except for the rules stated above, RTPS does not define which ***portBaseNumber*** and ***portGroupNumber*** are used nor how the **Applications** participating in a **Network** obtain this information.

2.4.2.2 Relevant Attributes of an Application



The relevant attributes of an **Application** are:

unicastIPAddressList These are the unicast IP addresses of the **Application**; they are the unicast IP addresses of the host on which the **Application** runs (there can be multiple addresses on a multi-NIC host). Depending on the network topology, a sending application might only be able to address that application on a subset of these IP addresses.

metatrafficMulticastIPAddressList For the purposes of meta-traffic, an **Application** can also accept **Messages** on this set of multicast addresses.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

usertrafficUnicastPort and **metatrafficUnicastPort** Every **Application** has exactly two application-dependent ports where it receives unicast user-traffic and unicast meta-traffic, respectively. A datagram sent to one of the application's unicast IP addresses and to one of these ports should only be received by one **Application**.

These attributes define two lists of UDP destinations. The first list, represented by the method **usertrafficAddressPortList()**, is used for user data; the second list, **metatrafficAddressPortList()**, is used for the RTPS metatraffic. These lists are defined as follows:

```
Application::metatrafficIPAddressPortList() =
{
unicastIPAddressList[] : metatrafficUnicastPort,
metatrafficMulticastIPAddressList[] :
wellknownMetatrafficMulticastPort
}

Application::usertrafficIPAddressPortList() =
{
unicastIPAddressList[] : usertrafficUnicastPort
}
```

RTPS messages sent to the multicast destinations can be received by multiple applications on multiple hosts.

2.4.2.3 Manager

For the special case of a **Manager**, these lists are defined as follows:

```
Manager::metatrafficIPAddressPortList() =
{
unicastIPAddressList[]              : wellknownManagerPort,
metatrafficMulticastIPAddressList[] : wellknownManagerPort
}
```

A manager receives all data on one well-known port, the ***wellknownManagerPort***.

```
Manager::usertrafficIPAddressPortList() = NULL
```

A **Manager** does not handle user data, only meta-data.

2.4.2.4 Definition of the IPAddressPortList()

A distinction needs to be made between a **Reader** and a **Writer**.

A **Writer** that is a meta-object is addressed through the metatraffic ports of the **Application** to which it belongs; if the **Writer** is a user-object, it is addressed through its **Application**'s user-data ports:

```
iff user-object
Writer::IPAddressPortList() = Application()->usertrafficIPAddressPortList()

iff meta-object
Writer::IPAddressPortList() = Application()->metatrafficIPAddressPortList()
```

Note that the *GUID* of the object immediately shows whether the object is a meta-object or a user- object.

A **Reader** (such as a **Subscription**) has an additional attribute: *usertrafficMulticastIPAddressList*.

The *IPAddressPortList* of a **Reader** is defined as follows:

```
iff user-object
Reader::IPAddressPortList() =
    {
       Application()->usertrafficIPAddressPortList(),
       usertrafficMulticastIPAddressList[] : wellknownUsertrafficMulticastPort
    }

iff meta-object
Reader::IPAddressPortList() = Application()->metatrafficIPAddressPortList()
```

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A user-level **Reader** can be addressed by unicast over the destination in the *usertrafficIPAddressPortList* of the **Application** to which it belongs or by sending UDP multicast to the additional multicast addresses the **Reader** provides at the *wellknownUsertrafficMulticastPort*.

A meta-**Reader** is addressed through the *metatrafficIPAddressPortList* of the application to which it belongs.

2.4.3 Possible Destinations for Specific Submessages

This section lists the UDP/IP destinations to which the basic **Submessages** (**ACK**, **HEARTBEAT**, **GAP**, **ISSUE** and **VAR**) can be sent.

2.4.3.1 Possible Destinations of an ACK

An **ACK** is usually sent to one of the known ports of the **Writer** (this could be a **Publication** or a **CSTWriter**) for which the **ACK** is meant (these ports are defined in 2.4.2.4 as *writer->IPAddressPortList()*).

An **ACK** can also be sent in response to a **VAR**, **HEARTBEAT**, **GAP** or **ISSUE**. The logical interpretation of these submessages explicitly contains an *ACKIPAddressPortList*, which contains possible additional destinations where such an **ACK** can be sent.

2.4.3.2 Possible Destinations of a GAP

A **GAP** is normally sent to a **CSTReader** which can be addressed through the *reader->IPAddressPortList()*, defined in 2.4.2.4 .

A **GAP** can also be sent in response to an **ACK**, in which case the **GAP** can be sent to one of the destinations in the logical *replyIPAddressPortList* of the **ACK**.

2.4.3.3 Possible Destinations of a HEARTBEAT

A **HEARTBEAT** is sent to a **Reader**, *reader*, (either a **CSTReader** or a **Subscription**); which can be addressed on *reader->IPAddressPortList()*, defined in 2.4.2.4 .

A **HEARTBEAT** can also be sent in response to an **ACK**, in which case the **HEARTBEAT** can be sent to one of the destinations in the logical *replyIPAddressPortList* of the **ACK**.

2.4.3.4 Possible Destinations of an ISSUE

To address a **Subscription**, *sub*, (a subclass of a **Reader**), this submessage needs to be sent to one of the destinations in *sub->IPAddressPortList()*.

An **ISSUE** can also be sent in response to an **ACK**, in which case the **ISSUE** can also be sent to one of the destinations in the logical *replyIPAddressPortList* of the **ACK**.

2.4.3.5 Possible Destinations of a VAR

To address a **Reader**, *reader*, the **VAR** is sent to one of the address/ports in *reader->IPAddressPortList()*.

A **VAR** can also be sent in response to an **ACK**, in which case the **VAR** can also be sent to one of the destinations in the logical *replyIPAddressPortList* of the **ACK**.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.5 Attributes of Objects and Metatraffic

2.5.1 Concept

Figure 32 shows an overview of all the attributes of the **NetworkObjects**. Some of the attributes are *frozen*, indicated by the symbol "@" in front of them. The value of a *frozen* attribute cannot change during the life of the object. All attributes of a **NetworkObject** (except for its GUID) have default values.

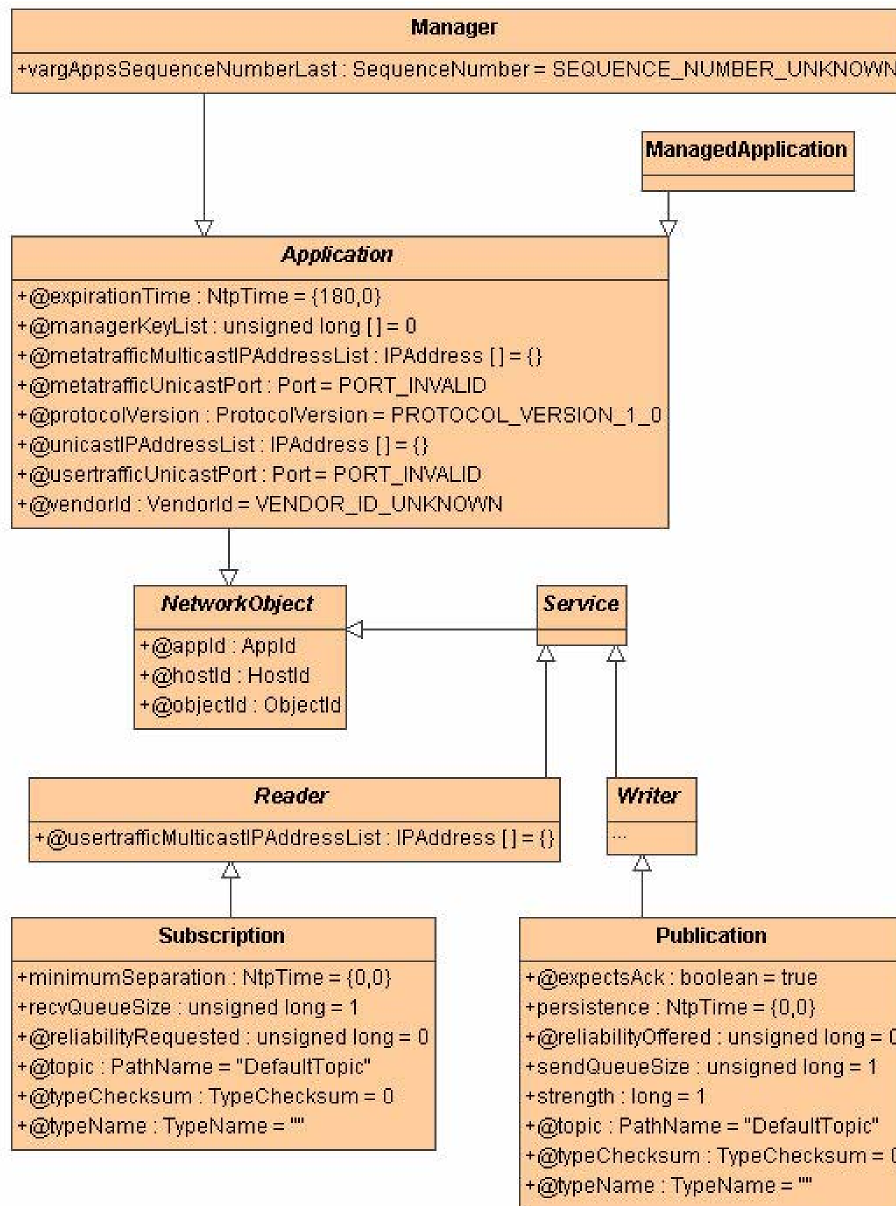


Figure 32 – Object attributes

The protocol uses the CST protocol to convey information about the creation, destruction and attributes of **Objects** (**Applications** and their **Services**) on the **Network**.

On the wire, the attributes of the objects are encoded in the **ParameterSequence** that is part of the **VAR** submessage (see 2.3.14). The information in the **ParameterSequence** applies to the object with GUID *objectGUID*. This GUID immediately encodes the class of the object and, therefore, the relevant attributes of the object and their default values.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

When the parameter sequence does not contain information about an attribute that is part of the class, the receiving application may assume that attribute has the default value.

The semantics of these classes and their attributes cannot be changed in this major version (1) of the protocol. Higher minor versions can extend this model in two ways:

- New classes may be added.
- New attributes may be added to the existing classes.

Table 9 – ManagedApplication Attributes

Attributes	Type	Default
unicastIPAddressList [ ]	IPAddress	{ }
@protocolVersion	ProtocolVersion	PROTOCOL_VERSION_1_0
@vendorId	VendorId	VENDOR_ID_UNKNOWN
@expirationTime	NtpTime	{180, 0}
@managerKeyList	unsigned long	0
@metatrafficMulticastIPAddressList [ ]	IPAddress	{ }
@metatrafficUnicastPort	Port	PORT_INVALID
@usertrafficUnicastPort	Port	PORT_INVALID

Table 9 shows the attributes of a **ManagedApplication**. The convention followed is that a preceding "@" denotes that the attribute is *frozen* and thus cannot be changed. A trailing "[ ]" denotes an array that indicates that the attribute can be repeated. A **Manager** submessage has the contents described in Table 9 and another attribute described in Table 10. The description of the types are included in 6.1.

Table 10 – Manager Submessage attributes (in addition to Table 2.5.1)

Attributes	Type	Default
vargAppsSequenceNumberLast	SequenceNumber	SEQUENCE_NUMBER_UNKNOWN

The next two tables represent the **Publication** and **Subscription** attributes, respectively.

Table 11 – Publication attributes

Attributes	Type	Default
@topic	PathName	"DefaultTopic"
@typeName	TypeName	""
@typeChecksum	TypeChecksum	0
strength	long	1
persistence	NtpTime	{0, 0}
@expectsAck	boolean	true
sendQueueSize	unsigned long	1
@reliabilityOffered	unsigned long	0

--`,````,,,```,,,`,,`,`-`-`,,`,,`,`,,`---

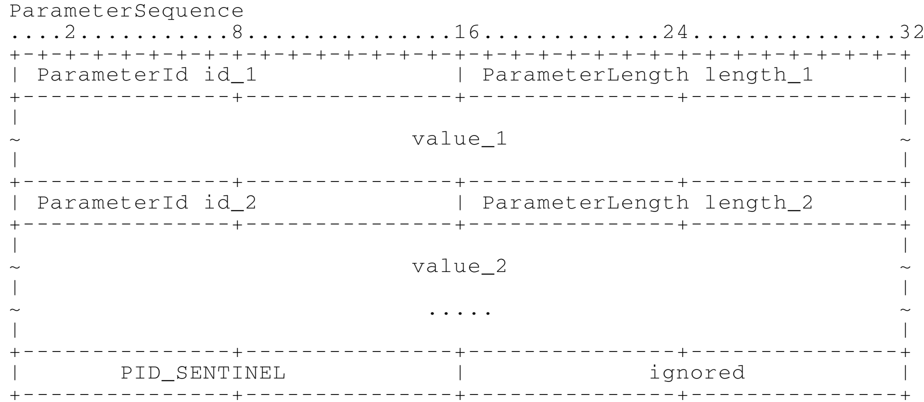
Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.5.2 Wire Format of the ParameterSequence

A **ParameterSequence** is a sequence of **Parameters**, terminated with a sentinel. Each **Parameter** starts aligned on a 4-byte boundary with respect to the start of the **ParameterSequence**. The representation of each parameter starts with a **ParameterId** (identifying the parameter), followed by a **ParameterLength** (the number of octets from the first octet of the value to the ID of the next parameter), followed by the value of the parameter itself.

When an attribute is a list (indicated by the "**[ ]**" after the type-name in the object model), the elements of the array are represented in the parameter sequence by listing the individual elements with the same (repeated) parameter ID.



ParameterId and **ParameterLength** are unsigned shorts:

```
typedef unsigned short ParameterId;
typedef unsigned short ParameterLength;
```

The parameter length is the number of octets following the length of the parameter to reach the ID of the next parameter (or the ID of the sentinel). Because every **ParameterId** starts on a 4-byte boundary, the **ParameterLength** is always a multiple of four.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.5.3 ParameterID Definitions

Table 12 – ParameterID Values

ID	Name	Used For Fields
0x0000	PID_PAD	-
0x0001	PID_SENTINEL	-
0x0002	PID_EXPIRATION_TIME	Application::expirationTime : NtpTime
0x0003	PID_PERSISTENCE	Publication::persistence : NtpTime
0x0004	PID_MINIMUM_SEPARATION	Subscription::minimumSeparation : NtpTime
0x0005	PID_TOPIC	Publication::topic : PathName, Subscription::topic : PathName
0x0006	PID_STRENGTH	Publication::strength : long
0x0007	PID_TYPE_NAME	Publication::typeName : TypeName, Subscription::typeName : TypeName
0x0008	PID_TYPE_CHECKSUM	Publication::typeChecksum : TypeChecksum, Subscription::typeChecksum : TypeChecksum
0x0009	RTPS_PID_TYPE2_NAME	
0x000a	RTPS_PID_TYPE2_CHECKSUM	
0x000b	PID_METATRAFFIC_ MULTICAST_IPADDRESS	Application::metatrafficMulticastIPAddressList: IPAddress[]
0x000c	PID_APP_IPADDRESS	Application::unicastIPAddressList : IPAddress[]
0x000d	PID_METATRAFFIC_ UNICAST_PORT	Application::metatrafficUnicastPort : Port
0x000e	PID_USERDATA_ UNICAST_PORT	Application::userdataUnicastPort :Port
0x0010	PID_EXPECTS_ACK	Publication::expectsAck : boolean
0x0011	PID_USERDATA_ MULTICAST_IPADDRESS	Reader::userdataMulticastIPAddressList : IPAddress[]
0x0012	PID_MANAGER_KEY	Application::managerKeyList : unsigned long []
0x0013	PID_SEND_QUEUE_SIZE	Publication::sendQueueSize : unsigned long
0x0015	PID_PROTOCOL_VERSION	Application::protocolVersion : ProtocolVersion
0x0016	PID_VENDOR_ID	Application::vendorId : VendorId
0x0017	PID_VARGAPPS_SEQUENCE_ NUMBER_LAST	Manager::vargAppsSequenceNumberLast : SequenceNumber
0x0018	PID_RECV_QUEUE_SIZE	Subscription::recvQueueSize : unsigned long
0x0019	PID_RELIABILITY_ OFFERED	Publication::reliabilityOffered : unsigned long
0x001a	PID_RELIABILITY_ REQUESTED	Subscription::reliabilityRequested : unsigned long

Future minor versions of the protocol can add new parameters up to a maximum parameter ID of 0x7fff. The range 0x8000 to 0xffff is reserved for vendor-specific options and will not be used by any future versions of the protocol.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.5.4 Reserved Objects

2.5.4.1 Description

To ensure the automatic discovery of **Applications** and **Services** in a **Network**, every **Manager** and every **ManagedApplication** contains a number of special built-in **NetworkObjects**, which have reserved objectId's.

These special objects fall into these categories:

- The **Application** itself is a **NetworkObject** with a special GUID (the instance of the **Application** is called *applicationSelf*). In addition, every **Application** has a **CSTWriter** (*writerApplicationSelf*) that disseminates the attributes of the local **Application** on the **Network**.
- Several objects are dedicated to the discovery of **Managers** and **ManagedApplications** on the **Network**. Every **ManagedApplication** has the **CSTReaders** *readerApplications* and *readerManagers*, through which the existence and attributes of the remote **ManagedApplications** and remote **Managers**, respectively, are obtained. Every **Manager** has the corresponding **CSTWriters** *writeApplications* and *writeManagers*.
- As seen in Figure 33, every **ManagedApplication** has, among others, two instances of a **CSTReader** (*readerPublications* and *readerSubscriptions*) and two instances of a **CSTWriter** (*writerPublications* and *writerSubscriptions*). Through the **CSTReaders**, the **ManagedApplication** can receive information about the existence and attributes of all the remote **Publications** and **Subscriptions** in the **Network**. Through the **CSTWriters**, the **ManagedApplication** can send out information about its local **Publications** and **Subscriptions**.

Future versions of the protocol may add additional special objects and expand the list of reserved objectId's within the same major version number.

Subclause 2.8 describes in detail what Messages are exchanged between these special objects.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.5.4.2 Overview: Special Objects in a ManagedApplication

Every **ManagedApplication** contains the following special objects seen in Figure 33.

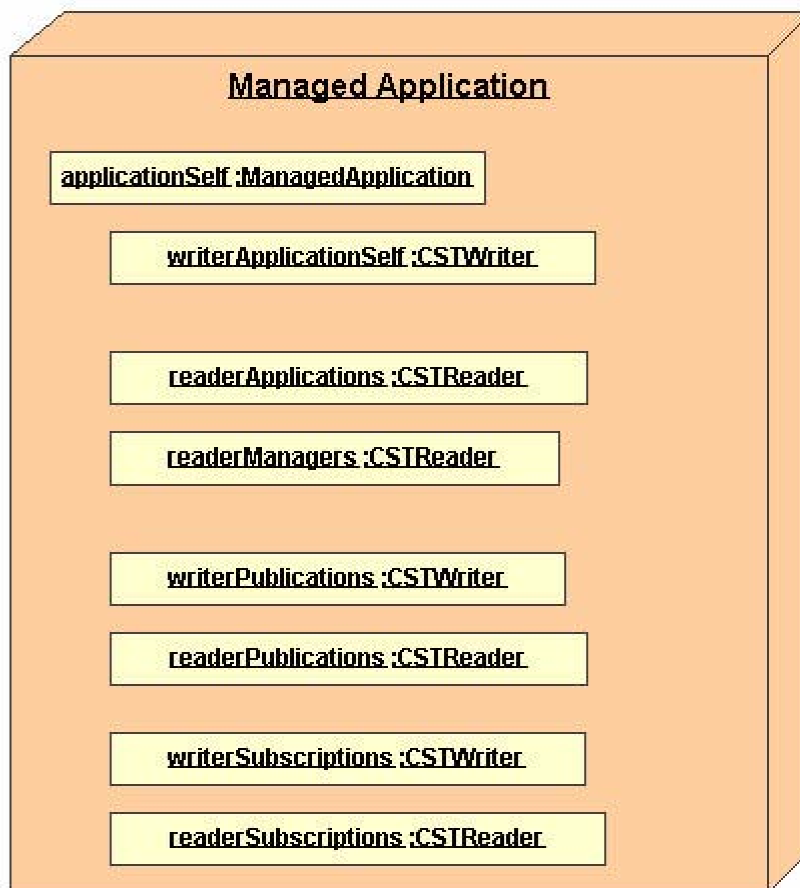


Figure 33 – Special objects of a Managed Application

applicationSelf :ManagedApplication The attributes of the **ManagedApplication** itself.

writerApplicationSelf :**CSTWriter** A **Writer** that makes the attributes of the application itself available.

readerApplications :**CSTReader** The **Reader** through which the application receives the attributes of other **Applications** on the **Network**.

readerManagers :CSTReader The **Reader** through which the application receives the attributes of **Managers** on the **Network**.

readerPublications :**CSTReader** The **Reader** through which the application receives information about remote **Publications** that exist on the **Network**.

writerPublications :**CSTWriter** The **Writer** that makes the attributes of the local **Publications** (contained in the local application) available on the **Network**.

readerSubscriptions:**CSTReader** The **Reader** through which the application receives information about remote **Subscriptions** that exist on the **Network**.

writerSubscriptions :**CSTWriter** The **Writer** that makes the attributes of the local **Subscriptions** (contained in the local application) available on the **Network**.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

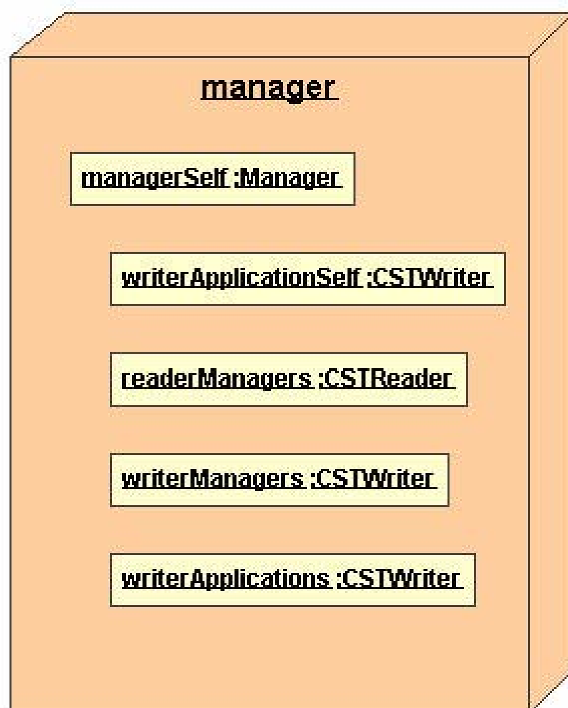


Figure 34– Special objects of a Manager

2.5.4.3 Overview: Special Objects in a Manager

Every **Manager** contains the following special objects seen in Figure 34.

managerSelf :Manager The attributes of the **Manager** itself.

writerApplicationSelf :**CSTWriter** A **Writer** that makes the attributes of the application itself available.

readerManagers :**CSTReader** The **Reader** through which the **Manager** discovers the other **Managers** on the **Network**.

writerManagers :CSTWriter The **Writer** through which a **Manager** provides information on all the other **Managers** in the **Network** to its managees.

writerApplications :CSTWriter The **Writer** through which a **Manager** provides information on all its managees.

2.5.4.4 Reserved ObjectIds

Table 13 lists the current reserved *objectId*s. All these objects are also meta-objects; so the M-bit and R-bit are set in the objectId. The meaning of these objects cannot change in this major version (1) of the protocol but future minor versions may add additional reserved *objectId*'s.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Table 13 – Predefined InstanceIds

Predefined instanceId	objectId of this Built-in Object	Description
applicationSelf	(OID_APP) = {00,00,01,c1}	The Application (ManagedApplication or Manager) itself.
writerApplicationSelf	(OID_WRITE_APPSELF) = {00,00,08,c2}	The CSTWriter which makes the attributes of the local Application available on the Network. Every Application has one of these.
writerApplications	(OID_WRITE_APP) = {00,00,01,c2}	Every Manager has this CSTWriter, to make the attributes of the ManagedApplications that are its managees available on the Network.
readerApplications	(OID_READ_APP) = {00,00,01,c7}	Every Manager has such a CSTReader, through which it reads the managees from Managers.
writerManagers	(OID_WRITE_MGR) = {00,00,07,c2}	Every Manager has this CSTWriter containing the other Managers.
readerManagers	(OID_READ_MGR) = {00,00,07,c7}	CSTReader through which an Application obtains information about the attributes of the Managers on the Network.
writerPublications	(OID_WRITE_PUBL) = {00,00,03,c2}	Every ManagedApplication makes its local Publications available through this CSTWriter.
readerPublications	(OID_READ_PUBL) = {00,00,03,c7}	This CSTReader reads the attributes of remote Publications. It is present in every ManagedApplication.
writerSubscriptions	(OID_WRITE_SUBS) = {00,00,04,c2}	Every ManagedApplication makes its local Subscriptions available through this CSTWriter.
readerSubscriptions	(OID_READ_SUBS) = {00,00,04,c7}	This CSTReader reads the attributes of remote Subscriptions. It is present in every ManagedApplication.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.5.5 Examples

2.5.5.1 Examples of GUIDs

Table 14 – Interpretation of Sample GUIDs

<hostId,	appId,	objectId>	Interpretation
{aa,bb,cc,dd}	{11,22,33,01}	{00,00,07,03}	A user-level object of class Publication.
	{11,22,33,02}	{00,00,07,c2}	A meta-CSTWriter that resides on a Manager; the object has a special instanceId: it is the CSTWriter of all Man agers for which the Manager keeps information.
	{11,22,33,01}	{00,00,17,c2}	This is a special instanceId; the object is a meta-level CSTWriter, however, version 1.0 does not define this special instanceId (a higher-level minor-version might define it). An implementation of version 1.0 should classify this GUID as UNKNOWN.
	{11,22,33,01}	{ee,ee,ee,02}	A user-level CSTWriter in an Application.
	{11,22,33,01}	{dd,dd,dd,82}	A meta-level CSTWriter in an Application.
	{11,22,33,01}	{00,00,01,c1}	A special meta-object of kind Application: the special instanceId "000001c1" is defined to refer to the applica tion itself, <{aa,bb,cc,dd},{11,22,33,01}>.
	{11,22,33,02}	{00,00,01,c1}	The same objectId as the previous example; the only difference is that the receiver knows from the appId that it is dealing with a special application, a Manager.
	{11,22,33,17}	{00,00,01,c1}	Should be classified as UNKNOWN, because the kind of application ("17") is unknown.
	{11,22,33,01}	{00,00,01,40}	Should be classified as UNKNOWN because the kind of objectId is unknown.
{00,00,00,00}	{11,22,33,01}	{00,00,01,c1}	Should be classified as UNKNOWN because the hostId is unknown.

Table 14 shows some examples of GUIDs and their interpretations.

2.5.5.2 Examples of ParameterSequences

Suppose an application receives a **VAR** submessage for an object with GUID <{11,22,33,44},{55,66,77,02},{00,00,01,c1}>. This GUID indicates this is a **Manager** (the kind of the appId is 0x02).

Suppose the parameter list in the **VAR** submessage contains a parameter sequence with the contents listed in Table 15.

--`,````,,,```,,,`,,,``,``-`-`,,`,,`,`,,`---

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Table 15 – Example VAR Submessage

Parameter ID	Value
PID_EXPIRATION_TIME	{10,0}
PID_APP_IPADDRESS	206.197.67.102
PID_APP_IPADDRESS	206.167.12.12
PID_METATRAFFIC_UNICAST_PORT	1051
PID_USERDATA_UNICAST_PORT	1052
PID_TOPIC	"abc"
0x00a0	123456
0x9001	abcdef

This means that the application knows that the remote **Manager** object with GUID <{11,22,33,44},{55,66,77,02},{00,00,01,c1}> has the attributes listed in Table 16.

Table 16 – Example Manager Attributes

Attribute	Contents
expirationTime	{10,0}
managerKey	0
metatrafficMulticastIPAddressList	{}
metatrafficUnicastPort	1051
usertrafficUnicastPort	1052
protocolVersion	PROTOCOL_VERSION_1_0
unicastIPAddressList	{ 206.197.67.102, 206.167.12.12}
vendorId	VENDORID_UNKNOWN
vargAppsSequenceNumberLast	SEQUENCE_NUMBER_UNKNOWN

Note that the application uses default values for those attributes for which it has not explicitly received information.

The receiving application ignores the last three parameters in the parameter sequence of Table 15:

- The parameter **PID_TOPIC** is a known parameter; but in version 1.0 of the protocol, it does not change a known attribute of a **Manager**; this parameter should be ignored. This is not an error.
- The parameter with ID 0x00a0 is an unknown parameter that might have been added in a higher minor version of the protocol; this parameter should be ignored. This is not an error.
- The parameter with ID 0x9001 is a vendor-specific parameter: if the application does not know about this vendor-specific extension, this parameter should be ignored. This is not an error.

--`,````,,,``,,,`,,,``,`-`-`,,`,,`,`,,`---
Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

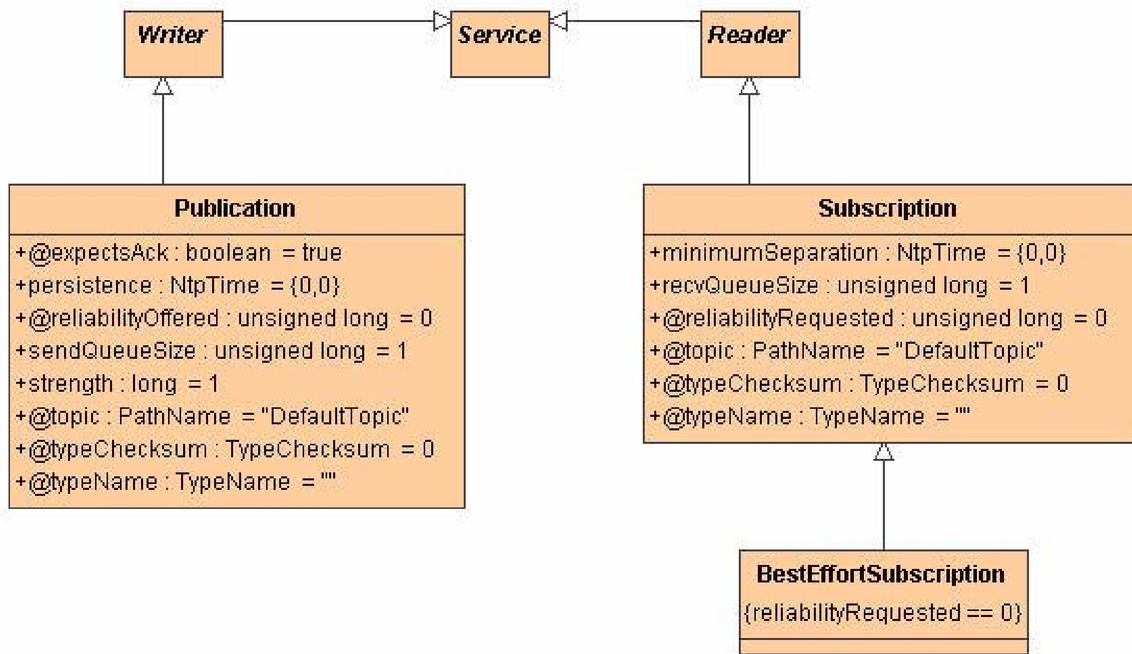
2.6 Publish-Subscribe Protocol

This subclause describes the Publish-Subscribe Protocol, which sends issues containing **UserData** from **Publications** to **Subscriptions**. The section separately describes the protocols for the case of best-effort publish-subscribe and reliable publish-subscribe and shows the representation of **UserData** and the related type-checking.

2.6.1 Publication and Subscription Objects

2.6.1.1 Object Model

The following figure shows the relevant aspects of the RTPS object model. This subclause only describes the simple case of best-effort **Subscriptions** (the **Subscription** attribute *reliabilityRequested* is 0).



2.6.1.1.1 Topic And Type Properties

Every **Publication** and **Subscription** has the following three properties:

topic The name of the information in the **Network** that is published or subscribed to.

typeName The name of the type of this data.

typeChecksum A checksum that identifies the CDR-representation of the data.

The types and meaning of these attributes is described in detail in Section 2.6.2 .

A **Publication** and **Subscription** "match" when the following conditions are satisfied:

- They have the same value for the attribute *topic*.
- They have the same value for the attribute *typeName* or this string is the empty string for one of the two objects.
- They have the same value for the attribute *typeChecksum* or this number is 0 for one of the two objects.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.6.1.1.2 Subscription Properties: minimumSeparation

The *minimumSeparation* is the minimum time between two consecutive issues received by the **Subscription**. It defines the maximum rate at which the **Subscription** is prepared to receive issues. **Publications** sending to this **Subscription** should try to send issues so that they are spaced at least this far apart.

2.6.1.1.3 Publication Properties: strength, persistence

The *strength* is the precedence of the issue sent by the **Publication**; the *persistence* indicates how long the issue is valid. *Strength* and *persistence* allow the receiver to arbitrate if issues are received from several matching publications.

2.6.1.1.4 Reliability

Publications can offer multiple reliability policies ranging from best-efforts to strict (blocking) reliability. Subscriptions can request multiple policies of desired reliability and specify the relative precedence of each policy. Publications will automatically select among the highest precedence requested policy that is offered by the publication.

The reliability policies offered by the publication are part of the publication declaration and are listed with using the parameter PID_PUBL_RELIABILITY_OFFERED. The reliability policies requested by the subscription are part of the subscription declaration and are listed with using the parameter PID_SUBS_RELIABILITY_REQUESTED.

The relative order of each PID_SUBS_RELIABILITY_REQUESTED in the subscription declaration indicates relative precedence. The policies are ordered in decreasing order of precedence, that is, starting with the highest precedence requested policy.

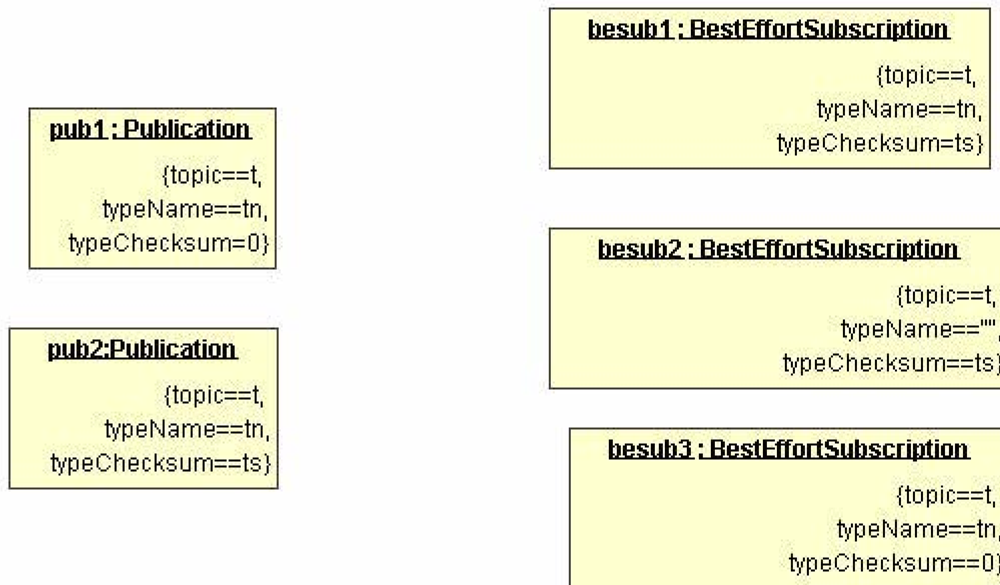
Version 1.0 of the RTPS protocol defines two reliability policies: best-efforts and strict.

Value	Name
0	PID_VALUE_RELIABILITY_BEST_EFFORTS
1	PID_VALUE_RELIABILITY_STRICT

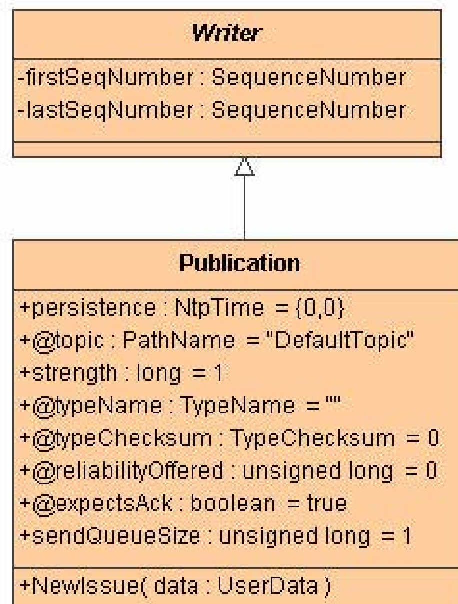
Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.6.1.1.5 Deployment

The following figure shows a possible deployed system of **Publications** and **Subscriptions**: for the following description, only matching objects matter. In RTPS, there can be multiple matching **Publications** and **Subscriptions** on the **Network**.



2.6.1.2 Publication Behavior Towards Best-Effort Subscriptions



The **Publication** is given user data by the application (represented by the method **NewIssue()**, which gives the **UserData** to the **Publication**). The **Publication** maintains a queue called the **sendQueue** with space for **sendQueueSize** issues. Every time a new issue is

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

given to the **Publication**, it places it in the **sendQueue** and increases the *lastModificationSeqNumber*.

The **Publication** sends this **UserData** to all the *matching* **Subscriptions** on the **Network** using the **ISSUE** submessage.

2.6.1.2.1 Contents of the Publication Message

A **Publication** puts the information from Table 2.6.1 in the **ISSUE** submessage.

Table 17 – ISSUE generated by an RTPSPublication Publication

Field in ISSUE Submessage	Contents
subscriptionGUID	< HOSTID_UNKNOWN, APPID_UNKNOWN, OBJECTID_UNKNOWN> (the issues sent by a best-effort publication will be usable by all interested subscriptions)
publicationGUID	< pub->hostId, pub->appId, pub->objectId >
issueSeqNumber	pub->lastModificationSequenceNumber
(parameters)	NONE
(timestamp)	optional timestamp of the issue
issueData	user data

2.6.1.2.2 When to Send an Issue

The publication needs to try to minimize latency while also trying to respect the *minimumSeparation* of the subscriptions.

2.6.1.2.3 Best-Effort Subscriptions

A best-effort subscription is a completely passive element that receives **Messages** containing **ISSUE**s from matching publications; it does not send messages itself.

2.6.1.3 Publication Behavior Towards Strict-Reliable Subscriptions

The **Publication** is given user data by the application (represented by the method **NewIssue()**, which gives the **UserData** to the **Publication**).

The **Publication** maintains a queue called the **sendQueue** with space for **sendQueueSize** issues. Every time a new issue is given to the **Publication**, it attempts to place it in the **sendQueue**. The attempt will succeed if either the queue has space available, or else there are some issues that can be removed from the queue. Otherwise the attempt will fail.

If the attempt succeeds, the *lastModificationSeqNumber* is increased, and the issue is associated with that sequence number.

If the attempt fails the **Publication** will block until it is possible to remove at least one issue from the queue.

The **Publication** keeps track of all the *matching* strict-reliable **Subscriptions** on the **Network. The Publication** keeps track of the issues (identified by the associated sequenceNumber) that have been acknowledged by each strict-reliable **Subscription**.

Issues can only be removed from the **sendQueue** if they have been acknowledged by all **Active** strict-reliable **Subscriptions** on the **Network.**

A strict-reliable **Subscription** is **Active** if and only if the Publication receives timely ACK messages from it in response to the **HEARTBEAT** messages it sends. The actual timing of

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

HEARBEAT messages sent and the elapsed time required to declare a **Subscription** non-**Active** is middleware dependent. It is expected that the implementation will allow the application developer to tune the behavior to the specific timing and reliability requirements of the application.

The **Publication** sends this **UserData** to all the *matching* **Subscriptions** on the **Network** using the **ISSUE** submessage.

The **Publication** sends **HEARTBEAT** messages to all *matching* strict-reliable **Subscriptions** on the **Network.**

HEARTBEAT messages sent to strict-reliable **Subscriptions** that have not acknowledged all issues in the sendQueue must have the FINAL-bit unset.

HEARTBEAT messages sent to strict-reliable **Subscriptions** that have acknowledged all issues in the sendQueue can have the FINAL-bit set or unset. The decision is middleware specific.

2.6.1.3.1 When to Send an Issue

The publication needs to try to minimize latency while also trying to respect the *minimumSeparation* of the subscriptions.

2.6.1.3.2 When to Send a HEARTBEAT

The timing of **HEARTBEAT** messages is middleware dependent. However, the publication must continue sending **HEARTBEAT** messages to all **Active** strict-reliable subscriptions that have not acknowledged all issues up to and including the one with sequence number *lastModificationSeqNumber*.

2.6.1.3.3 Strict-Reliable Subscriptions

Strict-reliable **Subscriptions** receives **Messages** containing **ISSUE**s and **HEARTBEAT**s from matching publications and send back **ACK Messages** in response.

2.6.1.3.4 When to Send an ACK

Strict-reliable **Subscriptions** should only send **ACK Messages** in response to **HEARTBEAT**s.

If the **HEARTBEAT** does not have the FINAL-bit set, then the subscription must send an **ACK Message** back.

If the **HEARTBEAT** does has the FINAL-bit set, then the subscription should only send an **ACK Message** back if it has not received all issues up to **HEARTBEAT**'s lastSeqNumber.

The strict-reliable **Subscriptions** can choose to send the **ACK Messages** back immediately in response to the **HEARTBEAT**s or else it can schedule the response for a certain time in the future. It can also coalesce related responses so there need not be a one-to-one correspondence between a HEARTBEAT and an ACK response. These decisions and the timing specifics are middleware dependent.

2.6.1.3.5 Contents of the ACK Message

A **Subscription** puts the information from Table 18 in the **ACK** submessage. In this table HEARTBEAT stands for the heartbeat message that triggered the ACK as a response.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Table18 – ACK Sent By a Subscription in Response to a HEARTBEAT Sent By a Matching Publication

Field in ISSUE Submessage	Contents
readerGUID	< sub->hostId, sub->appId, sub->objectId >
writerGUID	< pub->hostId, pub->appId, pub->objectId >
Bitmap	The specifics of the bitmap are middleware-dependent. However, it must meet the following constraints: 1. Bitmap.bitmapBase>= HEARTBEAT.firstSeqNum. 2. The Subscriber has received all issues up-to and including Bitmap.bitmapBase-1 3. Bits are only set to "0" if the Subscription is missing the corresponding sequence numbers.

2.6.1.3.6 Contents of the HEARTBEAT Message

A **Publication** puts the information from Table 19 in the **HEARTBEAT** submessage sent to strict-reliable subscriptions.

Table 19 – HEARTBEAT Sent By a Publication to a Matching Strict-Reliable Subscription

Field in ISSUE Submessage	Contents
readerGUID	This can take several forms to indicate whether the message is directed to a specific subscription or to all subscriptions. The distinction is based on whether the objectId portion is OBJECTID_UNKNOWN. If the objectId=OBJECTID_UNKNOWN then the reader GUID is: < HOSTID_UNKNOWN, APPID_UNKNOWN, OBJECTID_UNKNOWN> This indicates the heartbeat applies to all subscriptions. If the objectId!=OBJECTID_UNKNOWN then the readerGUID is: < sub->hostId, sub->appId, sub->objectId > This indicates that the heartbeat applies to one specific subscription.
writerGUID	< pub->hostId, pub->appId, pub->objectId >
firstSeqNumber	The first sequence number available to the Subscription. This sequence number must be greater or equal to (lastSeqNumber-sendQueueSize). It may not be exactly this because either not enough issues have been published to fill the sendQueue, or else some middleware-specific option causes certain issues to expire their validity.
lastSeqNumber	pub->lastModificationSequenceNumber

2.6.2 Representation of User Data

UserData is sent in the **ISSUE** submessage from a **Publication** to one or more **Subscriptions**.

The *topic* of that data is an attribute of the **Publication** and **Subscription**. The type of this *topic* attribute is **PathName**.

To ensure type-consistency between the **Publication** and **Subscription**, both have additional attributes *typeName* (of type **TypeName**) and *typeChecksum* (of type **TypeChecksum**).

The following sections describe the encapsulation of user data in CDR format in the **ISSUE**, and the meaning of the **TypeName** and **TypeChecksum** structures and the **PathName**

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

structure that is used in the *topic*.

2.6.2.1 Format of Data in UserData

User data is represented on the wire in CDR format, as specified in Annex A: CDR for RTPS. The endianness used in the representation of the user data is defined by the endianness of the submessage: the E-bit present in every submessage (see 2.3.2.2). For purposes of alignment when encoding/decoding user data elements that need 8-byte alignment, the CDR stream will be reset at the start of the **UserData** block.

The RTPS protocol assumes that the sender and receiver of **UserData** know the format of the type, so that they can serialize and deserialize the data in the correct CDR format. RTPS does not define how the sender and receiver get this type information but does define optional mechanisms to check whether the types are consistent.

2.6.2.2 TypeName

TypeName is a string composed of up to **TYPENAME_LEN_MAX** characters.

```
#define TYPENAME_LEN_MAX 63
typedef string<TYPENAME_LEN_MAX> TypeName;
```

The RTPS protocol does *not* define the relationship between this type-name and the CDR type of the issues. The contents of the type-name can be used freely by the application level. The RTPS mechanism only checks that the *typeName* of **Publication** and **Subscription** are equal. The middleware should not allow communication if the strings are not equal in length and contents.

The default **TypeName** is the empty string (""). The empty string means that the type-name is unknown and that type-checking should not be done.

2.6.2.3 TypeChecksum

The *typeChecksum* is used to verify that the format of the user data is consistent. It is a 4-byte unsigned number:

```
typedef unsigned long TypeChecksum;
```

In contrast to the **TypeName**, the RTPS protocol defines the relationship between the CDR type of the data and the number in the checksum. The default checksum is the number 0, which means that the checksum has not been generated and cannot be used to check type-safety. If both the sender and receiver declare the checksum to be something other than 0, the RTPS mechanism should only allow communication if the numbers are equal. Future versions will expand on how this field is generated.

2.6.2.4 PathName

The **PathName** is a string with a maximum length of 255 characters:

```
#define PATHNAME_LEN_MAX 255
typedef string<PATH_LEN_MAX> PathName;
```

This is the type of the field *topic* in a **Publication** and **Subscription**.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.7 CST Protocol

The Composite State Transfer (CST) protocol transfers Composite State from **CSTWriters** to **CSTReaders**.

2.7.1 Object Model

Figure 35 shows the relevant aspects of the RTPS object model.

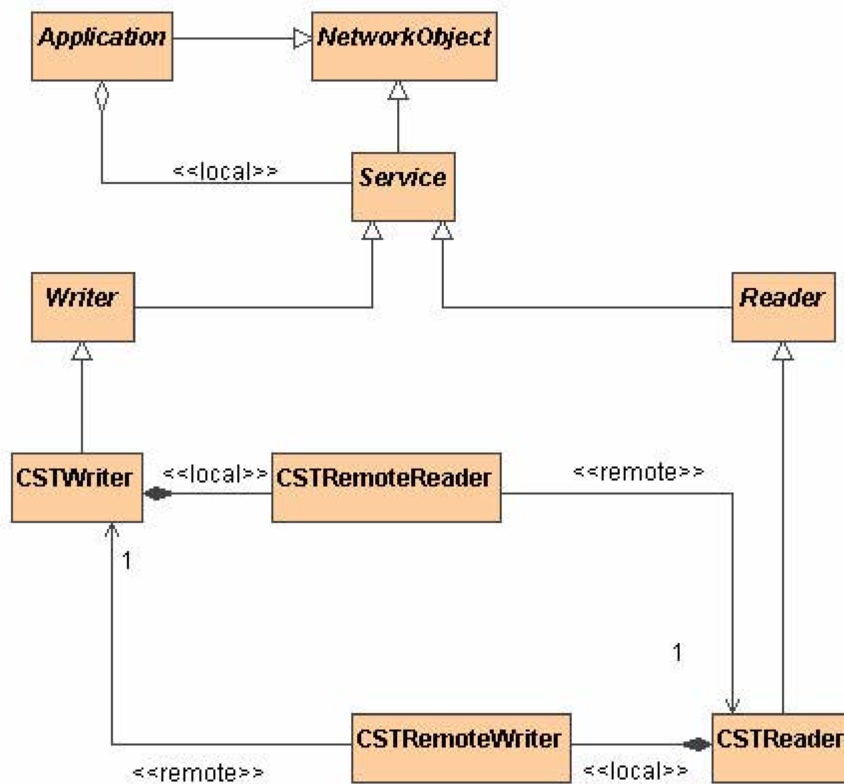


Figure 35 – CST Protocol Object Model

The classes **CSTWriter** and **CSTReader** and their base-classes are part of the RTPS object model described in earlier sections. To facilitate the description of the CST protocol, two classes are added: **CSTRemoteReader** and **CSTRemoteWriter**.

A **CSTWriter** locally instantiates a **CSTRemoteReader** for each remote **CSTReader** that it transfers information to. Because the CST protocol allows one **CSTWriter** to transfer data concurrently to many **CSTReaders**, the **CSTWriter** can have several local **CSTRemoteReaders**.

The complementary class on the reader's side is the **CSTRemoteWriter**. A **CSTReader** has a local **CSTRemoteWriter** for each remote **CSTWriter** it receives data from.

The **CSTRemoteWriter** and **CSTRemoteReader** are *not* **NetworkObjects**; they do not have a GUID and are therefore not remotely accessible.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.7.2 Structure of the Composite State (CS)

The goal of the CST protocol is to transfer *Composite State* (**CS**) from **CSTWriters** to the interested **CSTReaders**. This **CS** is composed of the attributes of a set of **NetworkObjects**.

The initial **CS** is an empty set. This **CS** can change dynamically through the following three kinds of **CSChanges**:

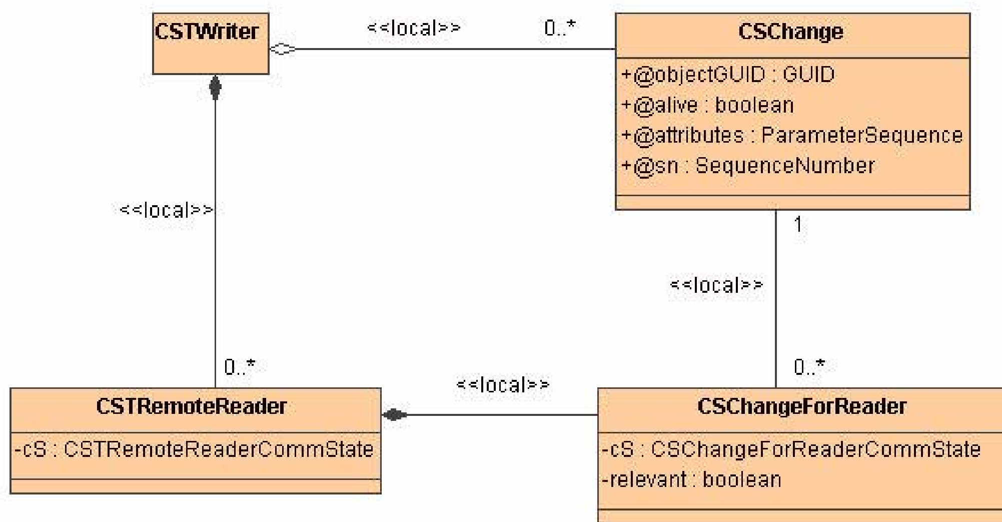
- A new **NetworkObject** (with a new unique GUID) is added to the **CS** of the **CSTWriter**.
- A **NetworkObject** is removed from the **CS** of the **CSTWriter**.
- One or more attributes of a **NetworkObject** within the **CS** change.

The goal of the CST protocol is to allow the **CSTReaders** to reconstruct the **CS** in the **CSTWriter**: the full set of **NetworkObjects** in the **CS** and their attributes. The CST protocol is aimed at transferring the *current* **CS** and avoids transferring the entire history of **CSChanges** that led to the *current* **CS**.

2.7.3 CSTWriter

2.7.3.1 Overview

The following subclauses describe the behavior of the **CSTWriter**, the **CSTChangeForReader** and the **CSTRemoteReader**.



2.7.3.2 CSTWriter Behavior

The current **CS** of the **CSTWriter** is represented by a sequence of **CSChanges**. The CSChanges are sequentially ordered by their SequenceNumber.

Every change in the **CS** of the **CSTWriter** creates a new CSChange with a new SequenceNumber. The *objectGUID* of the new **CSChange** is the GUID of the **NetworkObject** that the change in the **CS** applies to. The *attributes* of that **NetworkObject** are represented as a **ParameterSequence in the CSChange**. The *alive* boolean is set to FALSE iff the **CSChange** represents the removal of the **NetworkObject** from the set of objects in the **CS**.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.7.3.3 CSChangeForReader Behavior

The **CSTChangeForReader** keeps track of the communication state (attribute *cS*) and relevance (attribute *relevant*) of each **CSChange** with respect to a specific remote **CSTReader**.

This *relevant* boolean is set to TRUE when the **CSChangeForReader** is created; it can be set to FALSE when the **CSChange** has become irrelevant for the remote Reader because of later **CSChanges**. This can happen, for example, when an attribute of a **NetworkObject** changes several times: in that case a later **CSChange** can make a previous **CSChange** irrelevant because a **Reader** is only interested in the latest attributes of the **NetworkObject**. It is the responsibility of the **CSTRemoteReader** to use this argument correctly so that the **CSTReader** can reconstruct the correct **CS** from the *relevant* **CSChanges** it receives.

Figure 36 shows the Finite State Machine representing the state of the attribute *cS* of the **CSChangeForReader**.

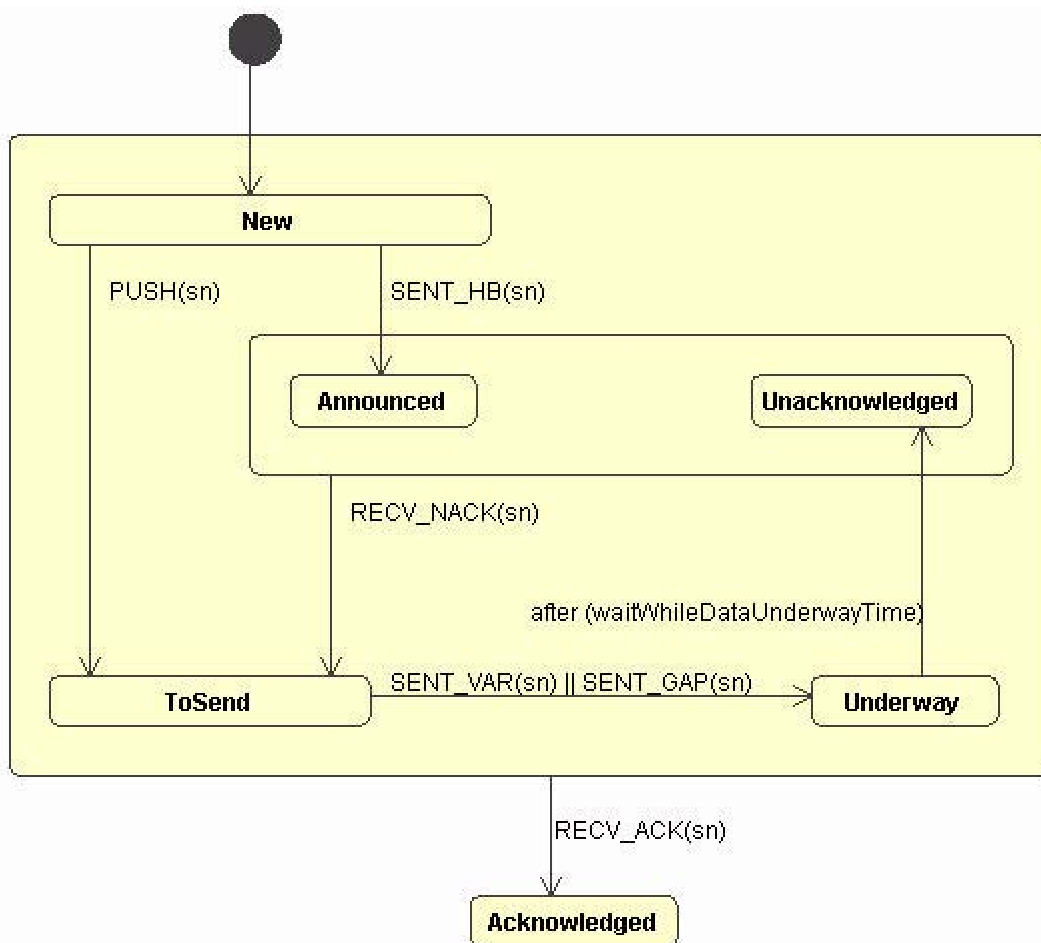


Figure 36 – State of Attribute cS for CSChangeForReader

The states have the following meanings:

<**New**> a **CSChange** with **SequenceNumber** *sn* is available in the **CSTWriter** but this has not been announced or sent to the **CSTReader** yet.

<**Announced**> the existence of this SequenceNumber has been announced.

<**ToSend**> it is time to send either a **VAR** or **GAP** with this *sn* to the **CSTReader**.

<**Underway**> the **CSChange** has been sent but the Writer will ignore new requests for this CSChange.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

<Unacknowledged> the CSChange should have been received by the CSTReader, but this has not been acknowledged by the CSTReader. As the message could have been lost, the **CSTReader** can request the **CSChange** to be sent again.

<Acknowledged> the **CSTWriter** *knows* that the **CSTReader** has received the **CSChange** with **SequenceNumber** *sn*.

The following describes the main events in this Finite State Machine. Tthis FSM just keeps track of the state of the **CSChangeForReader**; it does not imply any specific actions:

SENT_HB(sn) : The **CSTWriter** has sent a **HEARTBEAT** with firstSeqNumber <= sn <= lastSeqNumber; which means that the **CSChange** has been announced to the **CSTReader**.

RECV_NACK(sn) : The **CSTWriter** has received an **ACK** where *sn* is inside the bitmap of the **ACK** and has a bitvalue of 0.

SENT_VAR(sn) : The **CSTWriter** has sent a **VAR** for *sn*. The CSTReader will use the received VAR to adjust its local copy of the CS.

SENT_GAP(sn) : The **CSTWriter** has sent a **GAP** where *sn* is in the **GAP**'s *gapList*, which means that the *sn* is irrelevant to the **CSTReader**.

RECV_ACK(sn) : The **CSTWriter** has received an **ACK** with bitmap.bitmapBase > *sn*. This means the **CSChange** with **SequenceNumber** *sn* has been received by the **CSTReader**.

PUSH(sn) : A **CSTWriter** can push out **CSChanges** that have not been requested *explicitly* by the reader, by moving them directly from the state **<New>** to the state **<ToSend>**.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.7.3.4 CSTRemoteReader Behavior

Each **CSTRemoteReader** has a communication state *cS*, which represents the current behavior of the **CSTWriter** with respect to one remote **CSTReader**. The behavior of the **CSTReader** is partly influenced by the attribute *fullAcknowledge*.

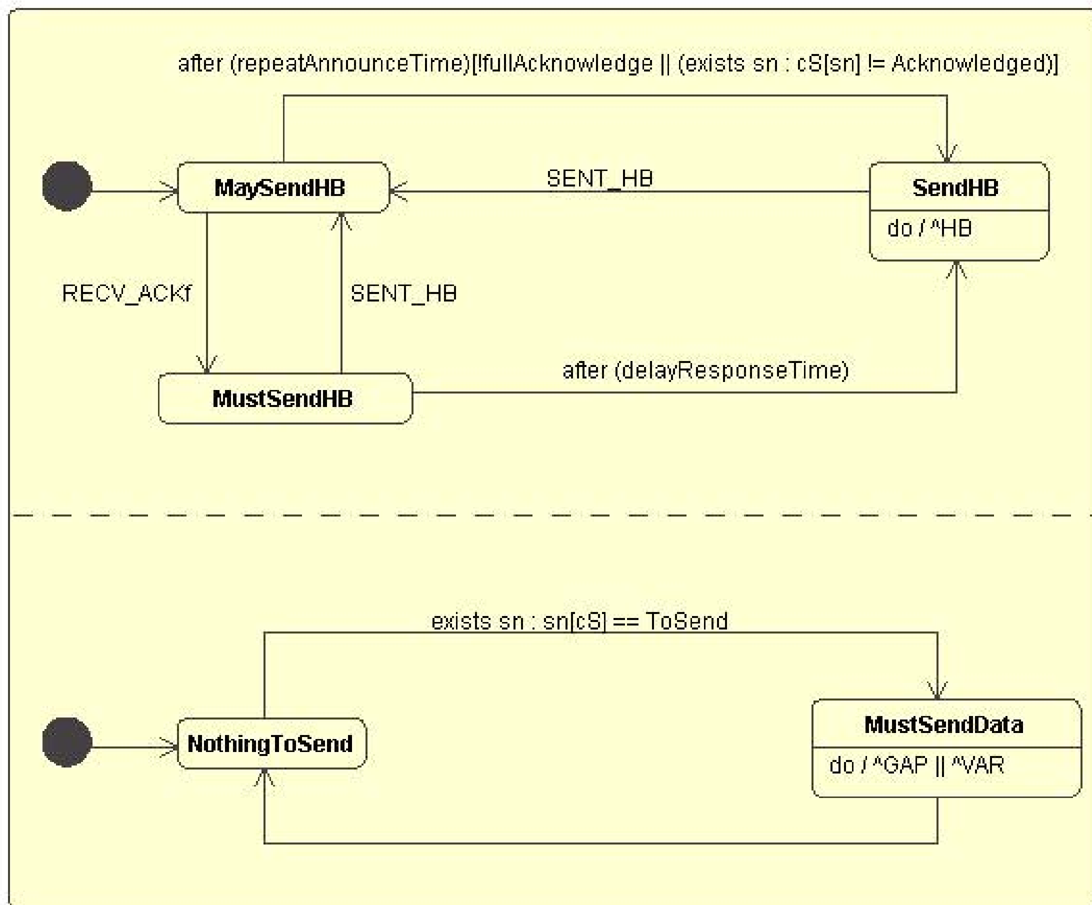


Figure 37 – CSTRemoteReader

The following is an overview of the most important abbreviations used in Figure 37 to represent events:

RECV_ACKf : an **ACK** was received from the **CSTReader** with FINAL-bit==FALSE.

SENT_HB : a HB was sent to the CSTReader

^VAR : this is an action: send a VAR submessage

^HB : this is an action: send a HB submessage

^GAP : this is an action: send a GAP submessage

The overall behavior of the CSTRemoteReader is modelled by two concurrent FSMs.

The bottom FSM deals with sending data: GAPs or VARs. Whenever there are CSChanges in state **<ToSend>**, the CSTRemoteReader is in state **<MustSendData>**. In this state, the **CSTWriter** will send **VARs** for *relevant* **CSChanges** and will include the *irrelevant* **CSChanges** in the *gapList* of a **GAP**. 2.7.5.2 and 2.7.5.3 show the contents of the **VAR** and **GAP**.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The most efficient **CSTWriter** will send the **VAR**s consecutively and in order (lowest sequence-numbers first) to facilitate the reconstruction of the **CS** by the **CSTReader**, but this is not a requirement. Likewise, the **CSTReader** will deal more efficiently with the **CSTWriter** that sends a **GAP** before **VAR** if there is a gap in the sequence numbers of the **VAR**, since the **CSTReader** then knows that sequence number is irrelevant. A possible sequence of submessages might be: GAP(1->100) VAR(101) VAR(102) GAP(103,105) VAR(104) VAR(106).

The top FSM shows the heartbeating behavior of a CSTWriter. In case an ACK without FINAL-bit is received, the CSTWriter must send a heartbeat within the delayResponseTime. In addition, a CSTWriter must regularly announce itself by sending a heartbeat. In case the CST protocol is in "fullAcknowledge" mode, the heartbeating only is necessary when there are unacknowledged CSChanges.

2.7.3.5 Timing Parameters on the CSTWriter side

The behavior is determined by the following timing parameters:

CSTWriter::waitWhileDataUnderwayTime: The CSTWriter is allowed to ignore NACKs for data that it considers to be underway to the CSTReader. The size of this window is the "waitWhileDataUnderwayTime". The window could be the **CSTWriter**'s estimate of the time it takes a message (a **VAR** or **GAP**) to be sent by the **CSTWriter** to the **CSTReader**, plus the time it takes for the **CSTReader** to process the message and immediately send a response (an **ACK**) to the **CSTWriter**, plus the time it takes the **CSTWriter** to receive and process this **ACK**. A larger *waitWhileDataUnderwayTime* will cause the CST protocol to slow down and be less aggressive; a lower time might cause data to be sent unnecessarily. *waitWhileDataUnderwayTime* can be 0 seconds.

CSTWriter::repeatAnnouncePeriod: This is the period with which the **CSTWriter** will announce its existence and/or the availability of new **CSChanges** to the **CSTReader**. This period determines how quickly the protocol recovers when an announcement of data is lost. **CSTWriter::repeatAnnouncePeriod** cannot be 0 nor INFINITY for the protocol to function correctly.

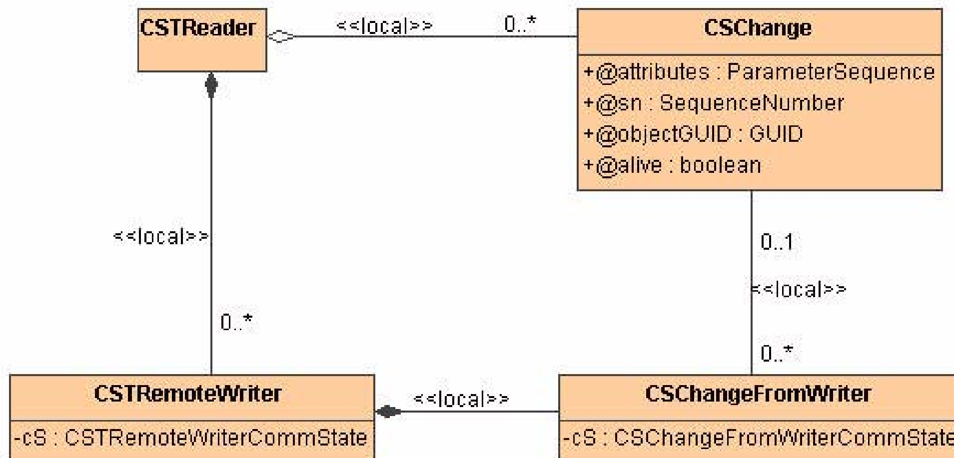
CSTWriter::responseDelayTime: This is the time the **CSTWriter** waits before responding to an incoming message. Higher values allow the **CSTWriter** to combine more information in one **Message** or to service many concurrent **CSTReaders** more efficiently. **CSTWriter::delayResponseTime** can be 0 seconds.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.7.4 CSTReader

2.7.4.1 Overview

The following subclauses describe the behavior of the **CSTWriter**, the **CSTChangeForReader** and the **CSTRemoteReader**. The **CSTReader** receives **CSChangeFromWriters** from the **CSTWriter**. In case a **VAR** was received for the **CSChangeFromWriters**, the **CSTReader** will store the contents of the **VAR** in an associated **CSChange**. The **CSTReader** should be able to reconstruct the current **CS** of a specific **CSTWriter** by interpreting all consecutive **CSChanges**.



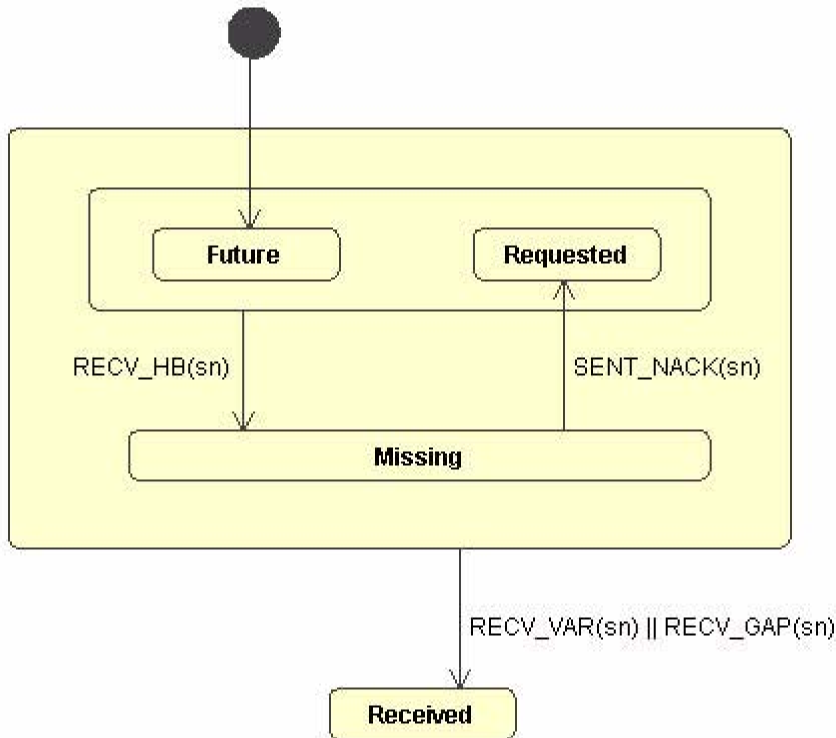
In the current version of the protocol, the **CSTReader** should reconstruct the **CS** for each **CSTRemoteWriter**. Future versions of the protocol will specify the correct interpretation in the case that several **CSTRemoteWriters** provide information on the *same* **NetworkObject**.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.7.4.2 CSTReader Behavior

As in the case of the **CSTWriter**, the **CSTReader** maintains a state **CSTRemoteWriterCommState** *cS* per **CSTRemoteWriter**, as well as a state **CSChangeFromWriterCommState** *cS* for most **CSChanges** (since there may be a **CSChangeFromWriter** that has no corresponding **CSChange**, for example, a **GAP** message).

2.7.4.3 CSChangeFromWriter Behavior



Here is the meaning of the abbreviated events in this FSM:

RECV_HB(sn) : the **CSTReader** received a **HEARTBEAT** with firstSeqNumber <= sn <= lastSeqNumber

SENT_NACK(sn) : the **CSTReader** sent an **ACK** with sn inside the bitmap-range and with bit-value 0

RECV_GAP(sn) : the **CSTReader** received a **GAP** with sn in the gapList

RECV_VAR(sn) : the **CSTReader** received a **VAR** for sequenceNumber sn

The four states have the following meaning:

<Future> : A CSChange with SequenceNumber *sn* may not be used yet by the CSTWriter.

<Missing>: The *sn* is available in the **CSTWriter** and is needed to reconstruct the **CS**.

<Requested>: The *sn* was requested from the **CSTWriter**, a response might be pending or underway

<Received> : The *sn* was received: as a **VAR** if the *sn* is relevant to reconstruct the **CS** or as a **GAP** if the sn is irrelevant.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.7.4.4 CSTRemoteWriter Behavior

The abbreviations used for events are as follows:

RECV_HBf : received a **HEARTBEAT** from the **CSTWriter** with FINAL-bit==FALSE

The abbreviations used for the actions are as follows:

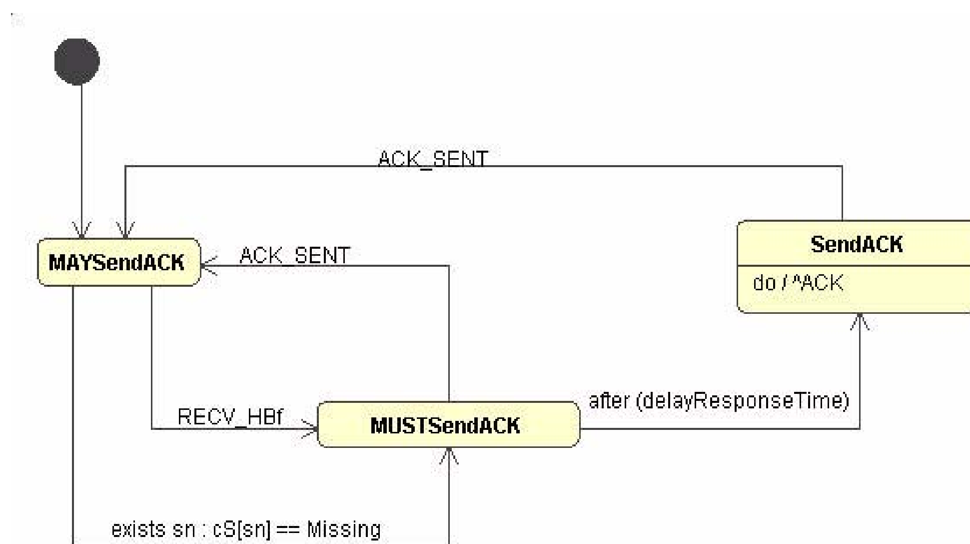
^ACK : send an **ACK** to the **CSTWriter**

In these **ACK**s, the ACK.bitmap.bitmapBase always is the *lowest* sequenceNumber whose corresponding **CSChangeFromReader** is not in state <**Received**>. This can be 0 (SEQUENCE_NUMBER_NONE). The **CSTReader** can choose the length of the bitmap as this will determine how much CSChanges move to <ToSend> state on the CSTWriter side and how much information the CTSReader will receive from the **CSTWriter**. The bitmap can only contain "0"'s when the corresponding **CSChangeFromReaders** are in state <**Missing**> (or <**Future**>). See Section **Error! Reference source not found.** for a description of the further fields in these **ACK**s.

The CSTRemoteWriter must send an ACK in two cases:

1. First, when a **HEARTBEAT** with the FINAL-bit==FALSE ("RECV_HBf") is received, the **CSTReader** *must* respond with an ACK that has the FINAL-bit==TRUE. The CSTReader can delay its response.

2. Second, when the CSTReader has evidence of Missing data, it needs to request the data by sending the appropriate ACK.



2.7.4.5 Timing Parameters on the CSTReader side

The timing parameters of the **CSTReader** are:

CSTReader::responseDelayTime: how long the **CSTReader** waits before sending a response to a HEARTBEAT to the **CSTWriter**.

2.7.5 Overview of Messages used by CST

This subclause gives an overview of the contents of the **Messages** that a **CSTReader** and **CSTWriter** exchange and the contents of the various fields of the **Messages**.

The submessages may need to be preceded by other messages that modify the context (see subclause 2.3.3).

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Table 20 – ACKs—Sent from a CSTReader to a CSTWriter

Field	Value in the CST protocol
FINAL-bit	see description of the behavior of the CSTRemoteWriter
readerGUID	reader.GUID
writerGUID	writer.GUID
replyIPAddressPortList	**required: must explicitly contain *all* destinations of the reader (reader.IPAddressPortList())**
bitmap	see description of the behavior of the CSTRemoteWriter

The only logical **SubMessage** that a **CSTReader** *reader* sends to a **CSTWriter** *writer* are **ACKs**. As shown in the table above, in this version of the protocol, the *replyIPAddressPortList* must be set explicitly to all the destinations of the *reader*.

2.7.5.1 HEARTBEATs—Sent from a CSTWriter to a CSTReader

The **HEARTBEATs** sent by the **CSTWriter** *writer* to the **CSTReader** *reader* always have the contents listed in this table.

Field	Value in the CST protocol
FINAL-bit	see description of the behavior of the CSTRemoteReader
readerGUID	reader.GUID
writerGUID	writer.GUID
ACKIPAddressPortList	optional destinations of the writer
firstSeqNumber	writer.firstSeqNumber
lastSeqNumber	writer.lastSeqNumber

2.7.5.2 GAPs—Sent from a CSTWriter to a CSTReader

The contents of the **GAPs** sent by the **CSTWriter** *writer* to the **CSTReader** *reader* is shown in the table. The contents of the *gapList* is described in the detailed description of the behavior of the **CSTRemoteReader**.

Field	Value in the CST protocol
readerGUID	reader.GUID
writerGUID	writer.GUID
ACKIPAddressPortList	optional; additional destinations of the writer
gapList	see description of the behavior of the CSTRemoteReader

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

2.7.5.3 VARs—Sent from a CSTWriter to a CSTReader

A **VAR** encodes the contents of a specific **CSChange** *cSChange* and is sent from the **CSTWriter** *writer* to the **CSTReader** *reader*.

Field	Value in the CST protocol
readerGUID	reader.GUID
writerGUID	writer.GUID
objectGUID	cSChange.GUID
writerSeqNumber	cSChange.sn
(timestamp)	optional timestamp
(parameters)	cSChange.attributes (iff cSChange.alive==TRUE)
ALIVE-bit	csChange.alive
ACKIPAddressPortList	optional; additional destinations of the writer

2.8 Discovery with the CST Protocol

RTPS defines mechanisms that allow every **Application** to automatically discover other relevant **Applications** and their **Services** in the **Network**. These mechanisms use the CST Protocol that is described in the previous section.

2.8.1 Overview

The **Manager** that manages a **ManagedApplication** is called the **Application's** MOM (My Own Manager). The other **Managers** in the **Network** are the **Application's** OAMs (Other Applications' Manager).

Figure 38 provides an overview of the protocols used for the discovery:

- The **Inter-Manager Protocol** allows **Managers** to discover each other in the **Network**. This protocol is described in 2.8.3 .
- The **Manager-Discovery Protocol** allows every **ManagedApplication** to discover other **Managers** in the **Network**: the **ManagedApplication** receives this information from its MOM. This protocol is described in 2.8.5 .
- The **Registration Protocol** allows **Managers** to find their managees and obtain their managees' state. This protocol is described in 2.8.4 .
- The **Application-Discovery Protocol** allows every **ManagedApplication** to discover other **ManagedApplications** on the **Network**. This protocol is described in 2.8.6 .
- The **Services-Discovery Protocol** allows every **ManagedApplication** to find out about the **Services** (the **Publications** and **Subscriptions**) in the other **ManagedApplications** on the **Network**. This protocol is described in 2.8.7 .

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

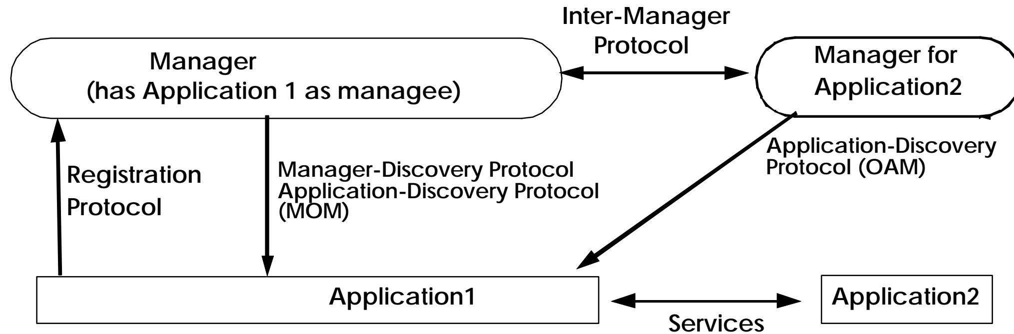


Figure 38 – Relation ship betwen Applications and managers

The discovery protocol uses reserved objects described in 2.5.4 .

2.8.2 Managers Keep Track of Their Managees

Every **Manager** keeps track of its managees and their attributes. To provide this information on the **Network**, every **Manager** has a special **CSTWriter** *writerApplications*.

The Composite State that the **CSTWriter** *writerApplications* provides are the attributes of all the **ManagedApplications** that the **Manager** manages (its managees).

2.8.3 Inter-Manager Protocol

Every **Manager** has a special **CSTWriter** *writerApplicationSelf* through which the **Manager** makes its own state available on the **Network**. The **CS** of the *writerApplicationSelf* contains the attributes of only one **NetworkObject**: the **Manager** itself.

The attribute *vargAppsSequenceNumberLast* of the **Manager** is equal to the *lastModificationSeqNumber* of the **CSTWriter** *writerApplications*. Whenever the **Manager** accepts a new **ManagedApplication** as its managee, whenever the **Manager** loses a **ManagedApplication** as a managee or whenever an attribute of a managee changes, the **CS** of the *writerApplications* changes and the **Manager**'s *vargAppsSequenceNumberLast* is updated.

Formally: for every **Manager** manager : manager.vargAppsSequenceNumberLast = manager.writerApplications.lastModificationSeqNumber.

Every **Manager** has the special **CSTReader** *readerManagers* through which the **Manager** obtains information on the state of all other **Managers** on the **Network**.

The communication between the Manager::writerApplicationSelf and Manager::readerManagers uses the CST Protocol that was described in the previous subclause, with a specific configuration.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

The Manager::writerApplicationSelf needs to be configured with the destinations (IP-addresses) of the Manager::readerManagers on the **Network**. This configuration is necessary to bootstrap the plug-and-play mechanism of RTPS. In case multicast is used, one single multicast address is sufficient: this is the multicast-address the **Managers** will then use to discover each other on the **Network**.

To support the automatic dynamic discovery and aging of **Managers**, the Manager::writerApplicationSelf *must* announce its presence repeatedly: the value of the repeatAnnouncePeriod timing- parameter of the **Manager's** writerApplicationSelf must be small relative to the expirationTime of the **Manager**.

Similarly, the readerManagers **CSTReader** will only consider the remote **Manager** alive within the expirationTime of the Manager. If no **Message** is received from the **Manager's** *writerApplicationSelf* during the expirationTime, the remote **Manager** must be considered dead; the **CSTReader** should behave as if it received a **CSChange** with the ALIVE-bit set to FALSE.

Because the CST Protocol for the inter-management traffic relies on repetitive messages, the fullAcknowledge attribute of the CSTReader and CSTWriter must be FALSE.

Here is a summary of the inter-manager protocol:

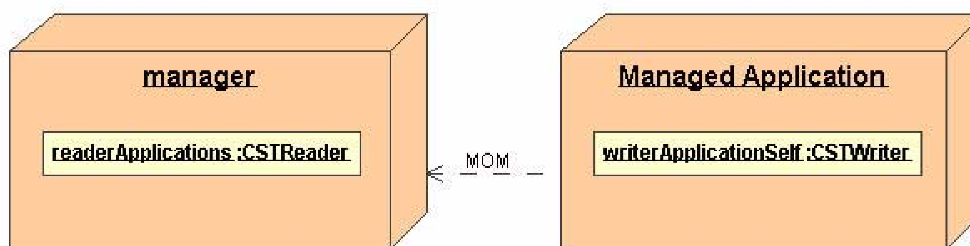
Initial Condition: New **Managers** know how to reach other *potential* managers on the **Network**.

Protocol: CST Protocol between Manager::writerApplicationSelf and Manager::readerManagers with repetition (repeatAnnouncePeriod of the writerApplicationSelf must be sufficiently high) and no acknowledgements (fullAcknowledge == FALSE).

Final Condition: Every **Manager** has the state of all other **Managers** on the **Network**. Repeated keep-alive HEARTBEATING is needed.

2.8.4 The Registration Protocol

The **registration protocol** enables managees to discover their **Managers** in the **Network**.



Initial Condition: The **ManagedApplication** is configured with a way to contact the *readerApplications* of its *potential* **Managers** (this configuration can be one single multicast address that will be used for the discovery of managers by applications). In addition, the **ManagedApplication** and **Manager** are configured with a *managerKeyList* which makes it possible for **Applications** and **Managers** to decide which **Managers** will manage which **Applications**.

Final Condition: Every **Manager** knows all its **Managees** and their attributes.

Protocol: CST Protocol (with sufficient repeatAnnouncePeriod and fullAcknowledge==FALSE) between the **ManagedApplication**'s *writerApplicationSelf* and the **Manager**'s *readerApplications*.

The **ManagedApplication** has a special **CSTWriter** *writerApplicationSelf*. The Composite State

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

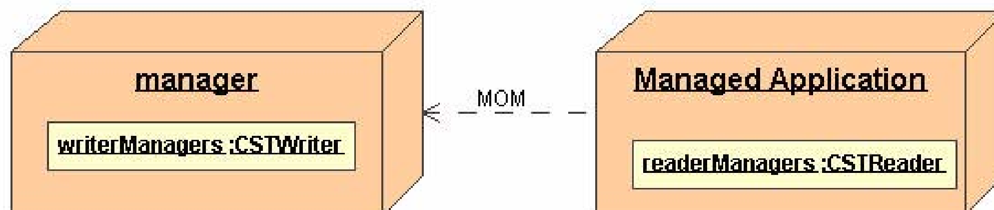
of the ManagedApplication::writerApplicationSelf contains only one **NetworkObject**: the application itself. As is the case for the *writerApplicationSelf* of the **Manager**, the *writerApplicationSelf* of the **ManagedApplication** must be configured to announce its presence repeatedly (the repeatAnnouncePeriod of that writer must be smaller than expirationTime of the **ManagedApplication**) and does not request nor expect acknowledgements (fullAcknowledge==FALSE).

A **Manager** that discovers a new **ManagedApplication** through its readerApplications must decide whether it must manage this **ManagedApplication** (become its MOM) or not (stay an OAM). For this purpose, the attribute *managerKeyList* of the **Application** is used: if one of the **ManagedApplication's** keys (in the attribute *managerKeyList*) is equal to one of the **Manager's** keys, the **Manager** accepts the **Application** as a managee and becomes its MOM. If none of the keys are equal, the managed application is ignored: the **Manager** will not manage this **Application** and stay an OAM for the **Application**. The *managerKey* 0x7F000001 (IP loopback) has a special meaning: the **Manager** will accept the **ManagedApplication** with key 0x7F000001 as a managee when that **ManagedApplication** runs on the same host as the **Manager**.

The application state in the **Manager** is only temporary. This approach is completely similar to the repeatAnnouncePeriod mechanism of **Managers** described in 2.8.3 . The duration of the lease is based on the value of the **ManagedApplication**'s expirationTime. The repeatAnnouncePeriod of the *writerApplicationSelf* must be small enough so that the **Manager** receives regular messages from the **ManagedApplication**. If the **Manager** has not received a **Message** from the **ManagedApplication** during the expirationTime of that **ManagedApplication**, it considers the **ManagedApplication** dead and behaves as if a CSChange has been received declaring the **Application** dead.

2.8.5 The Manager-Discovery Protocol

With the Manager-Discovery protocol, a **Manager** will send the state of all **Managers** in the **Network** to all its managees.



Initial Condition: Every **Manager** has obtained the state of other **Managers** (using the inter-manager protocol) and knows its managees.

Protocol: CST Protocol between Manager::writerManagers and ManagedApplication::readerManagers.

Final Condition: Every managee of every **Manager** has the state of all **Managers** on the **Network**.

2.8.6 The Application Discovery Protocol

Initial Condition: The **Managers** have discovered their managees and the **ManagedApplications** know all **Managers** in the **Network** (they got this information from their MOMs).

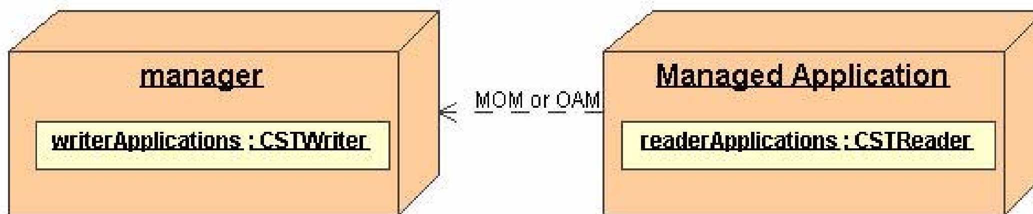
Protocol: The CST Protocol is used between the writerApplications of the **Managers** and the

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

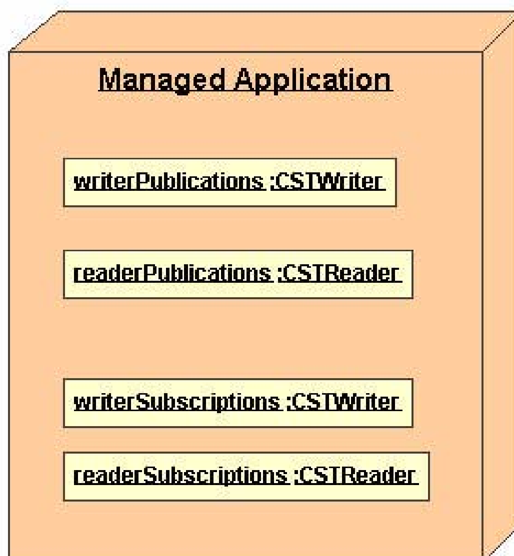
readerApplications of the **ManagedApplications**.

Final Condition: The **ManagedApplications** have discovered the other **ManagedApplications** in the **Network**.



2.8.7 Services Discovery Protocol

This subclause describes how the **ManagedApplications** transfer information to each other about their local **Services**.



As mentioned previously, every **ManagedApplication** has two special **CSTWriters**, *writerPublications* and *writerSubscriptions*, and two special **CSTReaders**, *readerPublications* and *readerSubscriptions*.

The Composite State that the **CSTWriters** make available on the **Network** are the attributes of all the local **Publication** and **Subscriptions**. The **CSTWriter** *writerPublications*/*Subscriptions* needs to instantiate a local **CSTRemoteReader** for each remote **ManagedApplication** on the **Network**.

Similarly, the **CSTReaders** *writerPublication/Subscription* need to instantiate a **CSTRemoteWriter** for each remote **ManagedApplication** on the **Network**.

Once **ManagedApplications** have discovered each other, they use the standard CST protocol through these special **CSTReaders** and **CSTWriter** to transfer the attributes of all **Publications** and **Subscriptions** in the **Network**.

Because all **CSTRemoteReaders** and **CSTRemoteWriters** for Service-discovery are known (as a result of Application-Discovery), the CST Protocol must support the acknowledgement of received issues (fullAcknowledge==TRUE) and repeated heartbeating should be turned off (repeatAnnouncePeriod==INFINITE).

Initial Condition: The **ManagedApplications** have discovered each other on the **Network.**

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Protocol: CST Protocol from *writerPublications* to *readerPublications* and from *writerSubscriptions* to *readerSubscriptions* (repeatAnnouncePeriod==INFINITE and fullAcknowledge==TRUE).

Final Condition: The **ManagedApplications** know about each other's **Services**.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Annex A of Section 2
(informative)

CDR for RTPS

The following is a summary of the CDR data format and the OMG IDL syntax to the extent that they are used by the RTPS protocol and its description in this PAS.

The authoritative source of the CDR specification and OMG IDL is the CORBA protocol (available through the Object Management Group). In the CORBA V2.3.1 spec, the relevant sections are 15.3 (General Inter-ORB Protocol—CDR Transfer Syntax) and 3.10 (OMG IDL Syntax and Semantics— Type Declaration). Unless mentioned explicitly, CDR for RTPS follows the CDR standard for GIOP version 1.1.

RTPS makes some additional restrictions on CDR and makes concrete choices where CDR for GIOP 1.1 is not fully defined. Notable are the implementation of the wide characters and strings (wchar and wstring) and the definition of the RTPSIdentifier, which only allows certain characters.

A.1 Primitive Types

A.1.1 Semantics

```
OMG IDL-name            size     meaning
octet                   1        8 uninterpreted bits
boolean                 1        TRUE or FALSE
unsigned short          2        integer N, 0 <= N < 2^16
short                   2        integer N, -2^15 <= N < 2^15
unsigned long           4        integer N, 0 <= N < 2^32
long                    4        integer N, -2^31 <= N < 2^31
unsigned long long      8        integer N, 0 <= N < 2^64
long long               8        integer N, -2^63 <= N < 2^63
float                   4        IEEE single-precision fp number
double                  8        IEEE double-precision fp number
char                    1        a character following ISO8859-1
wchar                   2        a wide-character following UNICODE
```

Remarks:

- CDR defines some additional primitive types, such as "long double"; these are currently disallowed by RTPS.
- CDR leaves the width of the wchar open; RTPS gives it a fixed length of two bytes.

A.1.2 Encoding

CDR has both a big-endian ("BE") and a little-endian ("LE") encoding. The sender is allowed to choose the encoding. The receiver needs to know which encoding has been used by the sender to unpack the data correctly. This endianness-bit is transmitted as part of the RTPS protocol.

A.1.3 octet

```
BE/LE
0...2...........7
+-+-+-+-+-+-+-+-+
|7|6|5|4|3|2|1|0|
+-+-+-+-+-+-+-+-+
```

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.1.4 boolean

```
TRUE BE/LE
0...2...........7
+-+-+-+-+-+-+-+-+
|0|0|0|0|0|0|0|1|
+-+-+-+-+-+-+-+-+

FALSE BE/LE
0...2...........7
+-+-+-+-+-+-+-+-+
|0|0|0|0|0|0|0|0|
+-+-+-+-+-+-+-+-+
```

A.1.5 unsigned short

```
BE
0...2...........7...............15
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|MSB            |            LSB|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
LE
0...2...........7...............15
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            LSB|MSB            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A.1.6 short

A short has the same encoding as an unsigned short, but uses 2's complement representation.

A.1.7 unsigned long

```
BE
0...2...........7...............15.............23..............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|MSB            |MSB    X       |MSB     Y      |            LSB|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

LE
0...2...........7...............15............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            LSB|MSB    Y       |MSB     X      |MSB            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A.1.8 long

A long has the same encoding as an unsigned long, but uses 2's complement representation.

A.1.9 unsigned long long

```
BE
0...2...........7...............15............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|MSB            |MSB    A       |MSB    B       |MSB    C       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|MSB    D       |MSB    E       |MSB    F       |            LSB|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

LE
0...2...........7...............15............23...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            LSB|MSB    F       |MSB     E      |MSB     D      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|MSB    C       |MSB    B       |MSB     A      |MSB            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

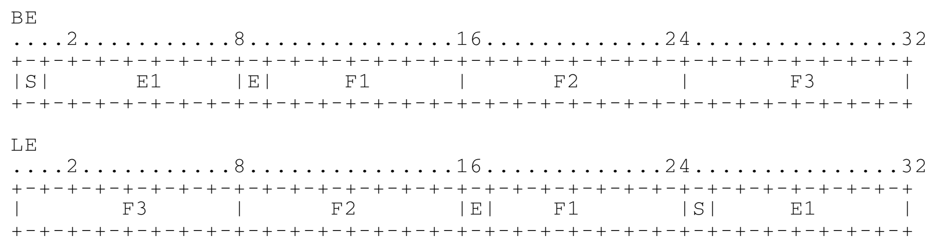
A.1.10 long long

A long long has the same encoding as an unsigned long long, but uses 2's complement representation.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

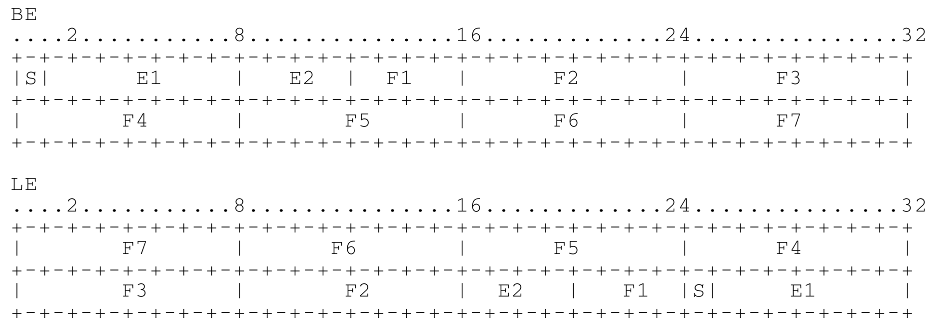
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.1.11 float



```
BE
....2...........8..............16............24...............32
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|     E1      |E|     F1      |       F2      |       F3      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

LE
....2...........8..............16............24...............32
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       F3      |       F2      |E|     F1      |S|     E1      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A.1.12 double



```
BE
....2...........8..............16............24...............32
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|     E1      |   E2  |  F1   |       F2      |       F3      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       F4      |       F5      |       F6      |       F7      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

LE
....2...........8..............16............24...............32
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       F7      |       F6      |       F5      |       F4      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       F3      |       F2      |   E2   |   F1  |S|     E1      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A.1.13 char

A character has the same encoding as an octet.

A.1.14 wchar

A wide-character occupies two octets and follows UNICODE encoding.

A.2 Constructed Types

A.2.1 Alignment

In CDR, only the primitive types listed in SectionA.1 have alignment constraints. The primitive types need to be aligned on their length. For example, a long must start on a 4-byte boundary. The boundaries are counted from the start of the CDR stream.

A.2.2 Identifiers

An identifier is a sequence of ASCII alphabetic, numeric and underscore characters. The first character must be an ASCII alphabetic character.

A.2.3 List of constructed types

RTPS supports the following subset of CDR constructed types:

struct structure

array fixed size array (the length is part of the type)

sequence variable size array (the maximum length is part of the type)

enum enumeration

string string of 1-byte characters

wstring string of wide characters

NOTE There are some additional constructed types in CDR, such as unions and fixed-point decimal types; these are currently not supported in RTPS.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

A.2.4 Struct

A structure has a name (an identifier) and an ordered sequence of elements. Each element has a name (an identifier) and a type. In OMG IDL, a structure is defined by the keyword "struct", followed by an identifier and a sequence of the elements of the structure. An example of the definition of a structure named "myStructure" in OMG IDL is:

```
struct myStructure {
    long long l;
    unsigned short s;
    myType t;
}
```

In CDR, the components of such a structure are encoded in the order of their declaration in the structure. The only alignment requirements are at the level of the primitive types.

A.2.5 Enumeration

An enumeration has a name (an identifier) and an ordered set of case-keywords which also are identifiers. In OMG IDL, an enumeration is defined by the keyword "enum", followed by an identifier and a list of identifiers in the enumeration. For example:

```
enum myEnumeration { case1, case2, case3 }
```

In CDR, enumerations are encoded as unsigned longs, where the identifiers in the enumeration are numbered from left to right, starting with 0.

A.2.6 Sequence

A sequence is a variable number of elements of the same type. Optionally, the type can specify the maximum number of elements in the sequence. OMG IDL uses the keyword "sequence". The syntax for an unbounded sequence of floats is:

```
sequence<float>
```

The syntax for a sequence of unsigned long longs with a maximum length is:

```
sequence<unsigned long long, MAX_NUMBER_OF_ELEMENTS>
```

In CDR, sequences are encoded as the number of elements (as an unsigned long) followed by each of the elements in the sequence.

A.2.7 Array

Arrays have a fixed and well-known number of elements of the same type. In OMG IDL, an array is defined using the symbols "[" and "]", following the C/C++ style. An example is:

```
float[17]
```

In CDR, arrays are encoded by encoding each of its elements from low to high index. In multi-dimensional arrays, the index of the last dimension varies most quickly.

A.2.8 String

A string is an optionally bounded sequence of characters. In OMG IDL, a string of unbounded length is identified by the keyword "string"; a bounded string is specified as follows:

```
string<MAX_LENGTH>
```

MAX_LENGTH is the maximum number of actual characters in the string (not including a possible terminating zero). For example: the string "Hello" can be stored in a variable of type string<5>.

On the wire, strings are encoded as an unsigned long (indicating the number of octets that follow to encode the string), followed by each of the characters in the string and a terminating zero. For example, the string "Hello" is encoded as the unsigned long 6 followed

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

by the octets ’H’, ’e’, ’l’, ’l’, ’o’, 0.

A.2.9 Wstring

A wide-string is a string of wide-characters. In OMG IDL, unbounded and bounded strings are specified, respectively, as follows:

```
wstring
wstring<MAX_LENGTH>
```

In CDR (GIOP 1.1), a wide-string is encoded as an unsigned long indicating the length of the string on octets or unsigned integers (determined by the transfer syntax for wchar), followed by the individual wide characters. Both the string length and contents include a terminating NULL.

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Standards Survey

The IEC would like to offer you the best quality standards possible. To make sure that we continue to meet your needs, your feedback is essential. Would you please take a minute to answer the questions overleaf and fax them to us at +41 22 919 03 00 or mail them to the address below. Thank you!

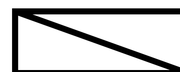
Customer Service Centre (CSC)
International Electrotechnical Commission
3, rue de Varembé
1211 Genève 20
Switzerland

or

Fax to: **IEC**/CSC at +41 22 919 03 00

Thank you for your contribution to the standards-making process.

Nicht frankieren
Ne pas affranchir



Non affrancare
No stamp required

A Prioritaire

RÉPONSE PAYÉE

SUISSE

Customer Service Centre (CSC)
International Electrotechnical Commission
3, rue de Varembé
1211 GENEVA 20
Switzerland

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Q1 Please report on **ONE STANDARD** and **ONE STANDARD ONLY**. Enter the exact number of the standard: *(e.g. 60601-1-1)*

....

Q2 Please tell us in what capacity(ies) you bought the standard *(tick all that apply).* I am the/a:

- purchasing agent ❑
- librarian ❑
- researcher ❑
- design engineer ❑
- safety engineer ❑
- testing engineer ❑
- marketing specialist ❑
- other...

Q3 I work for/in/as a: *(tick all that apply)*

- manufacturing ❑
- consultant ❑
- government ❑
- test/certification facility ❑
- public utility ❑
- education ❑
- military ❑
- other...

Q4 This standard will be used for: *(tick all that apply)*

- general reference ❑
- product research ❑
- product design/development ❑
- specifications ❑
- tenders ❑
- quality assessment ❑
- certification ❑
- technical documentation ❑
- thesis ❑
- manufacturing ❑
- other...

Q5 This standard meets my needs: *(tick one)*

- not at all ❑
- nearly ❑
- fairly well ❑
- exactly ❑

Q6 If you ticked NOT AT ALL in Question 5 the reason is: *(tick all that apply)*

- standard is out of date ❑
- standard is incomplete ❑
- standard is too academic ❑
- standard is too superficial ❑
- title is misleading ❑
- I made the wrong choice ❑
- other ....

Q7 Please assess the standard in the following categories, using the numbers:
(1) unacceptable,
(2) below average,
(3) average,
(4) above average,
(5) exceptional,
(6) not applicable

- timeliness ....
- quality of writing....................................
- technical contents...................................
- logic of arrangement of contents ....
- tables, charts, graphs, figures...............
- other ....

Q8 I read/use the: *(tick one)*

- French text only ❑
- English text only ❑
- both English and French texts ❑

Q9 Please share any comment on any aspect of the IEC that you would like us to know:

....
....
....
....
....
....
....
....
....
....
....
....

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS

Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST

ISBN 2-8318-7740-7

ICS 25.040.40; 35.240.50

Typeset and printed by the IEC Central Office
GENEVA, SWITZERLAND

Copyright International Electrotechnical Commission
Provided by IHS under license with IEC
No reproduction or networking permitted without license from IHS
Licensee=Technip Abu Dabhi/5931917101
Not for Resale, 02/22/2006 23:21:46 MST