

Project Number: **T 1045**
Project Title: **ROSIN Railway Open System Interconnection Network**
Deliverable Type *: **PU (Public Usage of results)**

Date of Delivery: **31st December 1996**
Title: **ROSIN Data Representation and Notation**
Work-Package: **WP04**
Nature **: **RE**
Author(s): **Hubert Kirrmann**
ABB Corporate Research Ltd,
Ch-5405 Baden - Switzerland
tel. +41 56 486 80 71 - fax +41 56 493 36 62
e-mail: hubert.kirrmann@chcrc.abb.ch

Abstract:

This Deliverable defines a formalism for describing the encoding of any interface data, including the transmission level, based on ISO's an Abstract Syntax Notation ASN.1.

These *explicit encoding rules* allow to define each transmitted bit on the line, while retaining the power of expression and the formalism of ASN.1.

The notation describes the primitive and constructed data types used in most field busses such as IEC 1158, WorldFIP, Profibus, IEC 870, Fieldbus Foundation and LON.

The explicit encoding rules allow a compact encoding suited for the transmission of real-time data. It defines the mapping to the application programming, to the storage and to the transmission on the bus.

Keyword List: Data representation, Abstract Syntax Notation, Encoding Rules.

* Type: PU-public, LI-limited, RP-restricted

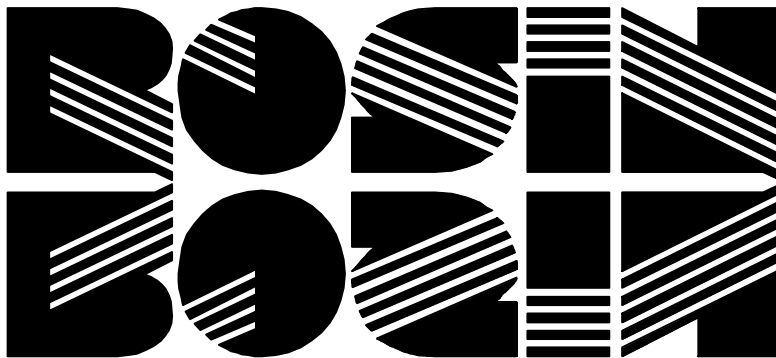
** Nature: PR-Prototype, RE-Report, SP-Specification, TO-Tool, OT-Other

TELEMATICS APPLICATIONS Programme

Telematics for Transport

TR 1045 - ROSIN

Railway Open System Interconnection Network



“Data Representation and Notation”

Deliverable D04.01, V2.1
DEL01/ABB/WP04/V2.1 /Aug98

August 1998

Table Of Contents

| | |
|---|-----------|
| Introduction | 8 |
| Scope and Object..... | 8 |
| References..... | 8 |
| Terms and Definitions | 9 |
| Legends and Abbreviations | 9 |
| Data Representation Language | 10 |
| General | 10 |
| Levels of data representation..... | 10 |
| Horizontal and vertical type conversion | 11 |
| Choice of the language | 12 |
| General transmission format..... | 13 |
| General storage format | 13 |
| General representation rules | 15 |
| Alignment | 15 |
| Graphical and textual representation | 17 |
| Presentation and encoding of transmitted and stored data | 20 |
| Purpose..... | 20 |
| Data ordering | 20 |
| Transmission format..... | 20 |
| Bus and Traffic Store format | 20 |
| Application data format | 20 |
| General rules | 20 |
| Relationship with ASN.1 | 21 |
| Notation for the primitive types | 22 |
| Notation for the boolean type..... | 22 |
| Notation for the antivalent type | 23 |
| Notation for the unsigned integer types | 23 |
| Notation for the integer type..... | 24 |
| Notation for the enumerated type..... | 25 |
| Notation for the binary coded decimal type..... | 26 |
| Notation for the unipolar types | 27 |
| Notation for the bipolar types | 27 |

| | |
|--|-----------|
| Notation for the real type | 28 |
| Notation for the character type | 29 |
| Notation for the Unicode character type | 29 |
| Notation for the uncommitted types | 30 |
| Structured types | 30 |
| General..... | 30 |
| Notation for the record types | 30 |
| Notation for the bitset types..... | 31 |
| Notation for the array type..... | 33 |
| Notation for the choice types..... | 35 |
| Notation for the set types | 38 |
| Alignment | 40 |
| Notation for special types..... | 40 |
| Notation for the string type | 41 |
| Notation for the TIMEDATE48 type | 41 |
| Notation for the TIME64 type..... | 42 |
| Notation for the ASN.1 boolean8 type..... | 42 |
| Notation for Little-Endian types | 42 |
| Type identifier..... | 44 |
| Data types for analog values..... | 45 |
| Usage and restrictions..... | 45 |
| SI-Unit code..... | 45 |
| Data representation for physical units..... | 50 |
| Representation as floating point numbers..... | 50 |
| Representation as fractionals..... | 50 |
| Mapping fixed-point to analog values..... | 50 |
| Small fractionals | 51 |
| Expressing the fractional variables representation in a unified way..... | 51 |
| Application Data Types | 52 |
| ROSIN Variable Types | 52 |
| Proposed additional RVTs for rail transportation | 52 |
| Global Positioning Group | 53 |
| Diverse | 53 |
| Differences with other standards..... | 54 |

| | |
|--|-----------|
| Differences with ASN.1 | 54 |
| Differences with WorldFIP | 55 |
| Expressing EN50170 / Fieldbus Foundation /IEC 1158-6 | 55 |
| Differences with IEC 870-5-4 | 58 |
| LONWorks Data Types | 59 |
| Differences with XDR | 60 |
| Conclusion..... | 61 |
| Appendix A: format mapping in TCN | 62 |
| Bus and application data formats..... | 62 |
| Big-endian and Little-endians processors..... | 62 |
| Bit numbering in assembly language | 62 |
| Operations on bit fields | 63 |
| Little-endian and Big-endian controller | 64 |
| Optimisation for Little-Endian processors | 66 |
| Class 1 device connection | 68 |
| MVB transmission format | 68 |
| WTB transmission format..... | 69 |
| Variable identification in TCN | 70 |
| PV_Name definition | 70 |
| Encoding of the PV_NAME | 70 |
| Example of PV_Name and TCN data types | 73 |
| Appendix B: format mapping in LON | 75 |
| Example of SNVT Object description..... | 80 |
| Neuron-C definition..... | 80 |
| Same SNVT examples in ROSIN Explicit Notation | 81 |
| Reformatting LonTalk Network Management messages..... | 81 |
| General structure of a LonTalk PDU..... | 82 |
| General structure of a LonTalk Network Variable PDU | 82 |
| General structure of a LonTalk Message PDU | 82 |
| Example: LonTalk Network Management commands | 83 |
| Example: Query_ID Request/Response | 84 |
| Graphical representation of Query_ID Request/Response..... | 85 |

Summary Information

Deliverable D04.01 - ROSIN Data Representation

WP 04 / Version 1.3 / April 1997

Classification

This report is: **public**.

Contributors to this report

This report has been prepared by ABB Corporate Research, Participant in WP04. Contributions of ADtranz and Siemens to the first draft are acknowledged.

History

| | | |
|------|-------|--|
| V0.1 | Dec96 | first draft, distributed at Milan, 6 Dec 1996 |
| V1.0 | Dec96 | DEL01/ABB/WP04/V1.0/Dec96, revised |
| V1.1 | Mar97 | DEL01/ABB/WP04/V1.1/Mar96 considers input from Milan meeting, April 6th. |
| V1.2 | Apr97 | DEL01/ABB/WP04/V1.2/Apr96 includes LON reformatting |
| V1.3 | May97 | DEL01/ABB/WP04/V1.3/Apr96 Small changes after St Ouen April 24, includes TCIP types |
| V2.0 | May98 | DEL01/ABB/WP04/V2.0/May98 revised to include application data types from document Unitype and consider optional fields encoding. |
| V2.1 | Aug98 | DEL01/ABB/WP04/V2.1/Aug98 revised to remove TCN dependencies where unnecessary. |

Executive Summary

In data communication, interoperability of interconnected devices of different origin require a precise definition of the exchanged data, from the application down to the bit level.

Each variable and each object is identified by a network-wide name to which an object type, an object encoding and an object semantic is attached.

A formal data representation allows to describe variables within devices and messages exported by the different devices as a sequence of named data types.

A formal data representation allows interoperability with other networks.

A formal data type representation is the base of the specification of application data, which groups such as the UIC 5R, the IEC WG22 or the US TCIP develop.

It is always possible to represent the exchanged data structures in a graphical form, which is quite intuitive, but unfortunately not machine-readable, which makes error-prone the process of translating the graphical representation to the programming language representation.

This document introduces a formalism of data representation based on ISO's Abstract Syntax Notation Nr. 1 (ASN.1), which allows to express any encoding rules. In particular, this language can to express data structures found in most busses, including TCN, WorldFIP, Profibus, Fieldbus Foundation and LON.

This language is independent from the programming language used by the application.

To precise the conventions from the application to the bus, this document explains how to map bus data formats to the application data form at four different levels:

- on the bus
- in the Traffic Memory (Traffic Store in TCN, Buffer Memory in WorldFIP).
- in the Application Memory
- in the application programming language.

Note - this document does not supersede the current IEC 9/413/CDV, it is intended to clarify it using a different notation and precise the mapping.

When enough people do the same error, it's not an error anymore, it's a standard.

Introduction

Scope and Object

The ROSIN Work Package develops a framework for a Standard Application Interface dealing with the data communications between programmable electronic equipment and on-board computers in Railways vehicles.

Interoperability of devices of different origin require that all devices adhere to the same data representation and formatting of transmitted data. It is assumed that the semantics of the data are known to all communicating parties.

This document specifies the data representation for Variables and Messages exchanged between devices over an arbitrary network.

To this effect, this document defines an Abstract Syntax Notation based on ISO 8824 and a set of compact and encoding rules.

This document defines how variables are identified within a dataset.

This document defines the mapping between the data on the bus and the data in memory.

This document indicate differences with existing, similar standards.

References

| | |
|-----------------|---|
| ISO 3309 | Information Processing Systems - Data communication - High-level data link control procedures - Frame structure, 3rd edition 1984-10-01 |
| ISO 8859-1 | Character Representation. |
| ISO/IEC 8824-1 | Information technology - Abstract Syntax Notation One (ASN.1): Specification of Abstract Syntax Notation (ASN.1) 1995-10-15. |
| ISO/IEC 8824-2 | Information technology - Open Systems Interconnection - Abstract Syntax Notation (ASN.1): Information object specification 1995-10-15. |
| ISO/IEC 8824-3 | Information technology - Open Systems Interconnection - Abstract Syntax Notation (ASN.1): Constraint specification 1995-10-15. |
| ISO/IEC 8824-4 | Information technology - Open Systems Interconnection - Abstract Syntax Notation (ASN.1): Parametrization of ASN.1 specifications 1995-10-15. |
| ISO/IEC 8825-1 | Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules (BER) 1990-12-15. |
| ISO/IEC 8825-2 | Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER) 1996-08-01. |
| IEEE Std 754 | Standard for Binary Floating-Point Arithmetic, IEEE, 1981. |
| IEC 9/413/CDV | Train Communication Network, IEC V 1375, 1996. |
| SNVT | The SNVT Master List and Programmer's Guide, document 005-0027-01, revision J, March 1996 of Echelon Corporation. |
| LonTalk | LonTalk Protocol Specification, Revision 3.0, 078-0125-01A, Echelon Corporation. |
| IEEE 268A- 1982 | Metric leaflet |

| | |
|------------------|--|
| ISO 31/0 - 1981 | General Principles) |
| ISO 31-I - 1978 | Quantities and units of space and time) |
| ISO 31-II - 1978 | Quantities and Units fro periodic and related phenomenon |
| ISO 31/5 - 1979 | Quantities of electricity and magnetism |
| NIST 330 - 1991 | United States National Institute of Standards and Technology, Special Edition 330, "The International System of Units (SI)" |

Terms and Definitions

| | |
|--------------|--|
| bit offset | the position of a bit within a transmitted or stored data structure, counted in bits with respect to the beginning of that structure, the first transmitted or stored bit of the structure having bit offset 0 in this specification. |
| octet offset | the position of an 8-bit unit within a transmitted or stored data structure, counted in multiples of 8 bits, bit 0-7 belonging to octet offset = 0 in this specification. |
| bit number | within an 8-bit data structure, the position of a bit counted from the least significant bit (LSB) of an 8-bit integer which would occupy this structure, this bit having bit number 0 in this specification. within a 16-bit data structure, the bit number is counted from the least significant bit of a 16-bit integer which would occupy this structure, this bit having bit number 0. |

Legends and Abbreviations

| | |
|-------|---|
| ASN.1 | Abstract Notation Number 1, defined in ISO 8824 |
| BER | Basic Encoding Rules, defined in ISO 8825 |
| EPRI | Electric Power Research Institute |
| FIP | Factory Instrumentation Protocol, a field bus defined as part of EN 50170 |
| HDLC | High-level Data Link Control, a set of standardised protocols, including ISO 3309 for data transmission |
| IEC | International Electrotechnical Committee, based in Geneva |
| IEEE | Institute of Electrical and Electronics Engineers, Piscataway, New York. |
| ISO | International Standards Organisation, based in Geneva |
| LON | Local Operating Network, as defined by Echelon Corporation, Palo Alto. |
| MVB | Multifunction Vehicle Bus, the vehicle bus of the IEC TCN |
| NIST | National Institute of Standards and Technology |
| SNVT | Standard Network Variable Types, see document [SNVT] |
| TCN | Train Communication Network |
| UCA | Utility Communication Architecture, an EPRI project. |
| WTB | Wire Train Bus, the train bus of the IEC Train Communication Network |

Data Representation Language

General

Levels of data representation

Interoperability of devices of different origin requires agreement on the semantics of the application data and on their transmission format.

Within a device, data are copied several times between bus and application, first by the Bus Controller, which copies them from the bus to a communication memory and then by the Application Processor which copies them from the communication memory to the Application Memory in a format suitable for the application, as Figure 1 shows.

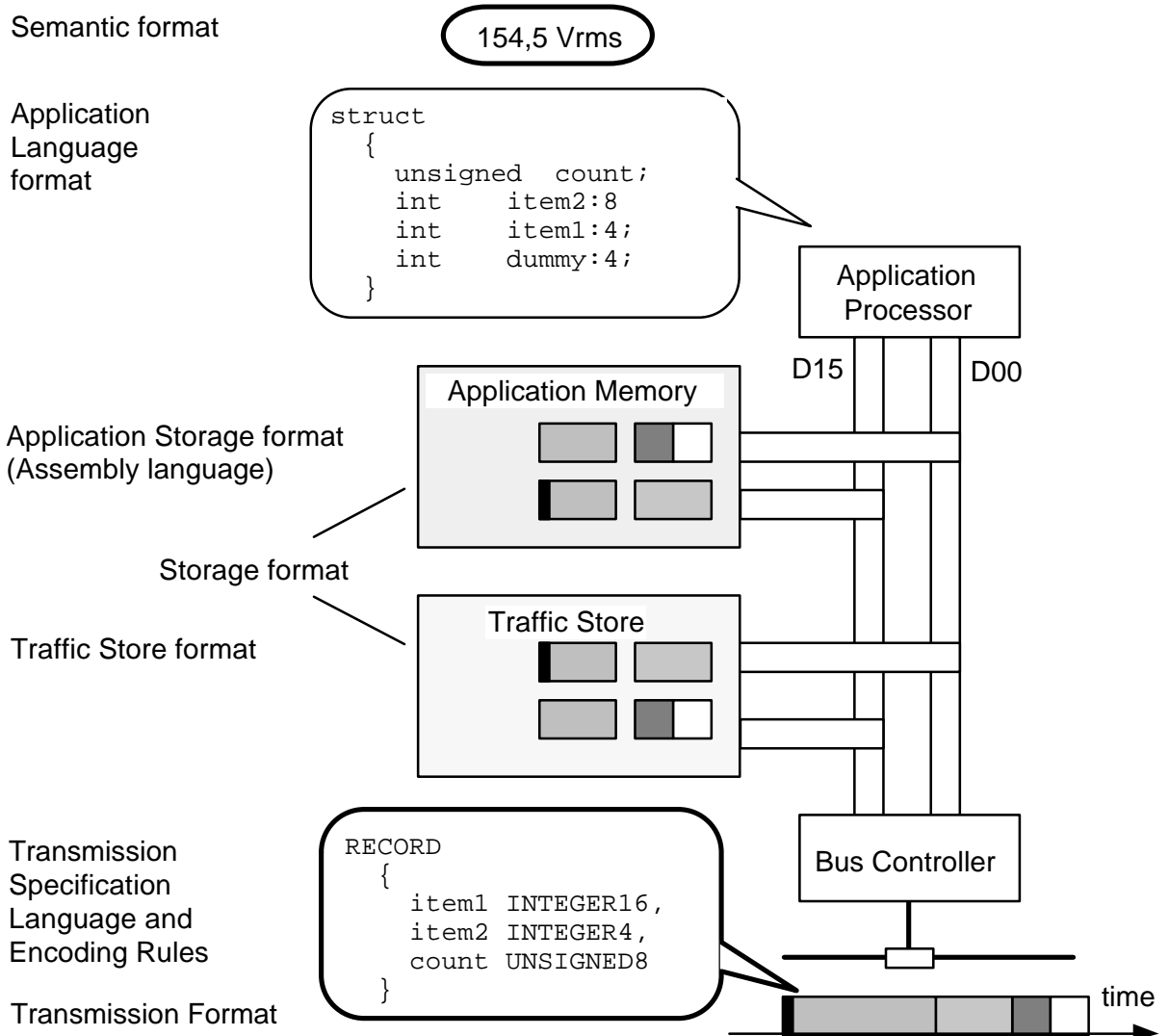


Figure 1 - Levels of data representation

1 - Sometimes, a communication processor copies data from the traffic store to an intermediate storage or directly to the Application Memory. This step is ignored here.

2 - The Traffic Store may be part of the processor's memory or a separate dual-access memory.

Therefore, the same data must be defined at several interfaces, as Figure 1 shows:

- 1) bus transmission format (what a logic analyser sees on the bus)
- 2) transmission format (what should be transmitted and how)
- 3) Traffic Store format (what a memory dump sees in the Traffic Store)
- 4) processor format (what a memory dump sees in the Application Memory)
- 5) application language format (what the programmer defines in 'C' or ADA)
- 6) semantic format (what data mean physically to the application, e.g. a door control).

Although these levels are independent, it is necessary to specify the mapping of the different data representations to each other at each copying operation to ensure end-to-end understanding. This mapping constitutes the "vertical" type conversion.

Horizontal and vertical type conversion

A basic assumption in an interoperable system is that the source dictates the data representation, all sinks have to convert the source data representation to their own.

Within an homogeneous bus, this aspect is often ignored, but emerges as soon as interconnected devices obey to different data representations. In this case, only the data representation on the bus is relevant, the application representation is not visible to other devices.

Within a network, different busses are connected over gateways. A gateway understands the data representation on both sides, since they may be different (Figure 2).

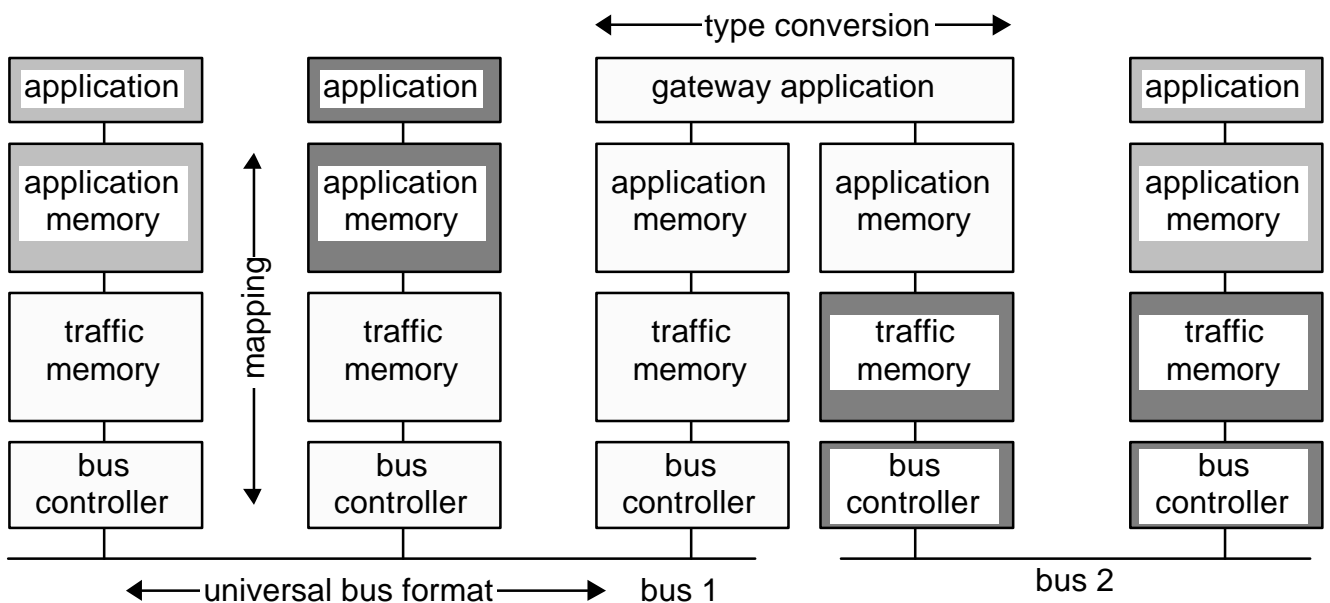


Figure 2 - Horizontal and Vertical type conversion

In industrial networks, a gateway is either transmitting data transparently, in which case the gateway recognises the headers and routing information, but otherwise does no data interpretation (it is a router) or the gateway filters information, in which case it has a precise knowledge of the information.

To this effect, it is necessary to introduce a data representation language which can be used at any level, for the vertical as well as for the horizontal interfaces.

Choice of the language

Some bus standards use a data representation language derived from a computer language. However, the representation of a variable varies from language to language and even within the same language. For instance, a Pascal compiler may store a boolean TRUE value as "0", while a "C" compiler for the same processor may store it as 'FF'H. Some machines store a boolean in 8 bits or even 32 bits. A "short" in the C-language can be 8 or 16 bits, depending on the compiler, a "short" in Java has 16 bits.

The numbering of items within a set can vary widely from one language to another. Figure 3 shows the same access to a bitset defined in C for Intel and ADA for Motorola.

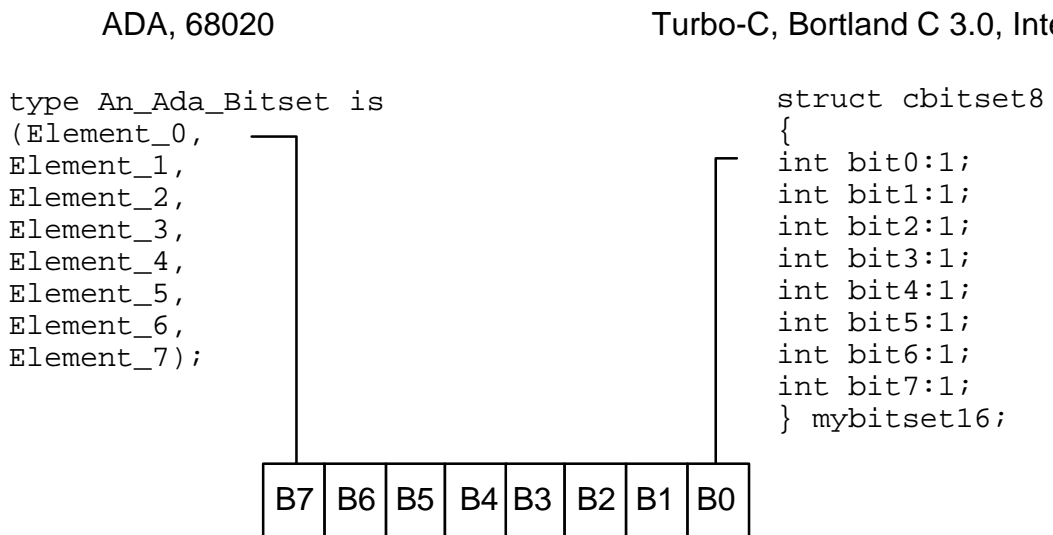


Figure 3 - Accessing bits in a Bitset

A programmer must assume that the way data structures are stored in memory depends on the processor, the language, the compiler and the compiler options

This is why the application routines which access the bus must be specific of the compiler and must know the type of the accessed data.

Therefore, the specification language defined here is independent of any programming language, and it is no programming language. It bases on the ISO/IEC Abstract Syntax Notation One, which is widely used in data communication. It has been extended to avoid the rigid data typing and tagging of ASN.1. Although transmitting each value with its data type greatly simplifies the work of the decoder, this method is unacceptable in real-time industrial networks, where bandwidth and/or response time are limited.

General transmission format

Correct interpretation of data requires that each transmitted bit be exactly identified even when data structures are not yet defined.

Data are considered to be transmitted as a *bit field*, each bit being specified by its *bit offset*, counted in bits since the first transmitted bit, which has bit offset 0 (Figure 4).

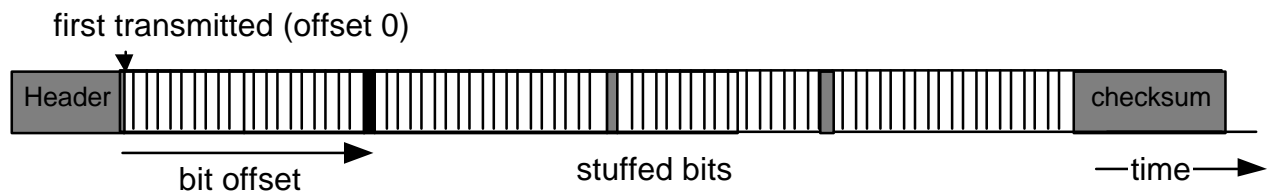


Figure 4 - Transmitted data treated as bit field

The transmission format is normally not accessible by software, since it requires direct observation of the bus, it is only relevant to the Bus Controller.

The Bus Controller copies data from a memory to the bus and vice-versa, without interpreting the contents. The Bus Controller could omit the overhead in the frames and count only the bits it passes to the Traffic Store.

As a first rule, bits are supposed to be transmitted in the same order as they are numbered on the bus or within the memory.

Although many controllers do not follow this rule (such as HDLC or UART which transmit octets with their least significant bit first), these exceptions will be ignored, since it does not matter how bits are shuffled on the bus as long as all controllers do it the same way.

Also, at a given abstraction layer, only the bits relevant to that abstraction layer are considered. For instance, flags and stuffed bits are not visible at the link layer interface.

Process Variables are transmitted in one frame and therefore the bit numbering on the bus and the bit numbering within memory correspond.

Messages are transmitted in several packets, in which case the communication software removes the headers of the individual packets and discards possible retransmissions - only the data transmitted to the application are considered as a single bit field.

To settle one fixed point for data representation in this scheme, the following rule is used:

Data formats will only be considered in the communication memory, considering that the mapping between memory and bus is defined precisely for each particular bus.

General storage format

The following conventions hold for the data representation in memory, regardless of the contents.

A bit within an arbitrary data structure is uniquely identified its *bit offset* with respect to the beginning of the data structure, the first bit of the structure having bit offset 0.

A memory is organised as a sequence of 8-bit words, called octets, located at consecutive and increasing memory addresses, the first octet of the sequence has the lowest address and has an octet offset of 0.

The difference between the address at one octet and at the beginning of the sequence is the *octet offset*.

Data structures may be nested. A data structure is uniquely identified by the offset of the first bit of that data structure with respect to the next level of nesting.

Within an octet, bits are *numbered* considering that an 8-bit unsigned integer occupies the word, and that the most significant bit (MSB) of the octet has the *lowest* offset, 0.

This numbering is the same, whether data are considered to be 8-bit, 16-bit or 32-bit units, as Figures 5a and 5b show.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|--------------|----|----|----|----|----|----|----|----|---------------------|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | lowest address |
| 1 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | next higher address |
| 2 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | |
| 3 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | |
| 4 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | |
| octet offset | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | |

Figure 5a - Bit offset in memory (8-bit wise).

Note - this numbering scheme is consequently big-endian and allows to consider non-aligned structures. It differs from the naming used in most processors, where the MSB is termed 7 or 15 depending on the word size, from IEC 870-5, where the MSB is named 8, from Profibus (EN 50170-3), where the MSB is named 7, and others. The bit naming makes only a difference where the bits are individually named, such as in BITSET data types. Bit *name* and bit *number* are two separate concepts.

| | lowest address | | | | | | | | next higher address | | | | | | | |
|--------------|----------------|----|----|----|----|----|----|----|---------------------|----|----|----|----|----|----|----|
| offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 2 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 4 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| octet offset | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |

Figure 5b - Bit offset in memory (16-bit wise)

Note - The bit numbering differs from the numbering used in several processors. This numbering corresponds to the offset used in bit string operations in a Motorola-style processor.

information about the data type is not sent with the data. It is assumed that types and their meaning are defined and agreed between the users in a specific application.

Alignment

Transmitted or stored data can have an imposed structure in different ways:

- a) A Bus Controller usually transmits data by multiple of 8 bits or of 16 bits and therefore transmission must be done in multiples of that value.
- b) The processor may impose an additional structuring (e.g. by 32-bits) because of alignment restrictions.

- c) A compiler may impose alignment rules, even if the processor does not (e.g. the compiler can prohibit unaligned structures in Intel processors).

This hidden structuring is considered in the encoding rules, with the objective of increasing the processing efficiency.

Since processors often access the memory contents by units of 16 bits or 32 bits, it is a good practice to place the first octet containing the first application data bit at an octet address which is divisible by 2, or preferably by 4.

When transferring messages, the application fetches the data from its own Application Memory, where the communication software copies it, and therefore the location of the message is under control of the application. If the application specifies a dynamic buffer allocation, the communication software will locate the first octet of the message at an address divisible by 4.

Alignment must be considered when defining efficient transmission formats.

This abstract syntax places no restrictions on alignment.

Graphical and textual representation

The graphical representation provides an intuitive, yet not formal, view of the data structure. It is helpful for documentation of work and can be readily deduced from the textual representation.

In the graphical representation, data are ordered by rows of 8 or 16 bits, corresponding to a preferred alignment in memory.

Rows are identified by their octet offset (0, 2, ...) as shown in the margin of Table 1.

Bits in a row are identified by their bit offset with respect to the beginning of the row.

Data are transmitted as a sequence of bits, in order of reading (from left to right, and then from top to bottom).

Variables are identified by their bit offset with respect to the beginning of the structure.

Example - the variable "tv" has bit offset 24, parameter 2 has bit offset 64.

Repetitive structures are marked by a bracket in the left margin.

Repetitive structures can be nested.

The header of a structured type is shaded, it does not occupy a memory location.

Table 1 shows an example of a message structure.

Table 1- Example of message structure

| | first transmitted octet | | | | | | | | next transmitted octet | | | | | | | |
|--------|--|-----|---------|---|---|---|---|-------------|------------------------|--------------|----|----|--------|----|----|----|
| offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | snu | gni | node_id | | | | | sta_or_func | | | | | | | | |
| 2 | next_station_id | | | | | | | tv | d1 | topo_counter | | | | | | |
| 4 | tnm_code | | | | | | | action_code | | | | | | | | |
| 6 | parameter1 | | | | | | | | | | | | | | | |
| 8 | Parameter2 | | | | | | | parameter3 | | | | | param4 | | | |
| | parameter5: RECORD | | | | | | | | | | | | | | | |
| | dummy5 | | | | | | | nr_elem5 | | | | | | | | |
| | ARRAY [nr_elem5] OF | | | | | | | | | | | | | | | |
| | parameter 5.1 (1st part of an array element) | | | | | | | | | | | | | | | |
| | parameter 5.2 (2nd part of an array element) | | | | | | | | | | | | | | | |
| | parameter6: STRING32 -- size is implicit | | | | | | | | | | | | | | | |
| | parameter 6 | | | | | | | | | | | | | | | |

with:

```
Action_Code ::=      ENUM8
{
  PAUSE              (0),          -- action "pause" has code 0
  RESTART_ONLY      (3),          -- action "restart_only" has code 3
  CLEAR_AND_RESTART (1),          -- action "clear and restart" has code 1
  BOOT_ALL          (2),          -- action "boot all" has code 2
  CONTINUE_WORK     (7)          -- action "continue" has code 7
}
-- all other codes are undefined
```

and:

```
Parameter5 ::= RECORD
{
  dummy5           WORD8,        -- used for alignment
  nr_elem5         UNSIGNED8,    -- array elements for the following
  parameter5       ARRAY [nr_elem5] OF
  {
    RECORD          -- these are the array elements
    {
      parameter5_1  UNIPOLAR2_16,
      parameter5_2  BIPOLAR4_16
    }
  }
}
```

Presentation and encoding of transmitted and stored data

The present Chapter has been copied from the IEC FDIS 61375-2 document (Train Communication Network, Real-Time Protocols”, which will be the reference in case of discrepancy. This was done to keep all relevant information in one document.

Purpose

This Chapter specifies data types and defines an abstract syntax notation to express how these data types are transmitted or stored. This notation relies on ISO 8824 (ASN.1), but includes additional constructs to describe existing data structures for retrofit devices and to transmit data structures with implicit typing, such as needed for bandwidth-limited channels.

This notation serves therefore a purpose similar to ISO 8824 and ISO 8825 together.

Data ordering

Transmission format

This Standard prescribes the order in which bits and words are transmitted. To this effect, it defines a number of primitive types and structured types. The meaning of the data is outside the scope of this document.

Bus and Traffic Store format

This standard assumes that the communication system copies the data from one memory to another memory transparently. These traffic memories are octet-oriented, i.e. data within these memories are identified by their octet address and within an octet by their offset with respect to the most significant bit of the octet, which has offset zero.

The actual data transfer is not specified, but the most probable is that octets are transmitted in order of increasing addresses.

Application data format

This Standard does not specify the Application data format. The interface procedures are expected to convert the data formats of the Application to theses used for storage or transmission and vice-versa.

General rules

1. Data structures shall be numbered from left to right and from top to bottom, in the order of reading an English text. The first item on the top and to the left has offset zero.

2. Memory shall be treated as an array of octets, and shall be transmitted in order of increasing address, regardless of the size of the transmitted units (by octets, by 32-bit words, etc.). The first octet has octet offset zero.
3. Bits within a data structure shall be identified by their offset with respect to the beginning of the structure. If this structure would contain an unsigned integer, the most significant bit of that integer would have offset zero. This bit is considered to be the 'leftmost' one when reading the data structure.
4. All data shall be transmitted most significant octet first (Big-Endian)
5. The order of bit transmission within an octet is considered a bus issue, invisible to the programmer. In general, character-oriented protocols such as Profibus or WTB first transmit the least significant bit of an octet (offset 7), while bit-oriented busses such as WorldFIP or MVB transmit it last.
6. Information about the data type is not sent with the data. Types are expected to be defined and agreed beforehand between the users of the network in a specific application.
7. The elements of a structured type (Record, Sequence) shall be transmitted in the order they are declared.
8. Arrays shall be transmitted in order of increasing index. Multi-dimensional arrays are transmitted in the order their indices are listed (e.g. ARRAY OF [row, column] is transmitted row by row).
9. To ease implementation, a variable shall be stored at an offset address which is a multiple of its size (alignment).
10. Variable length data (open arrays, records, sets,...) shall not be used as Process Variables, but may be transmitted in messages.

Relationship with ASN.1

The ISO standard 8824:1990 'Abstract Syntax Notation' defines the Abstract Syntax Notation One (ASN.1) to express data structures in machine-readable form.

Although ASN.1 does not impose a transfer syntax, it cannot express the compact encoding rules frequently used by programmers where bandwidth or where time is limited. In addition, it cannot express already existing encodings which do not obey to its structuring method.

Therefore, this standard defines an abstract syntax based on ASN.1, which expresses at the same time the data encoding and the bit-by-bit content of the data.

The following keywords have been added to ASN.1:

| | | | |
|-------|-------------|-------|------|
| ALIGN | ANTIVALENT2 | ARRAY | BCD4 |
|-------|-------------|-------|------|

| | | | |
|--------------|-------------|-------------|-----------|
| BIPOLAR2_16 | BIPOLAR4_16 | BITSET# | BITSET_L# |
| BOOLEAN1 | BOOLEAN8 | ENUM# | INDIRECT |
| INTEGER# | INTEGER_L# | ONE_OF | REAL32 |
| REAL64 | RECORD | SOME_OF | STOP |
| STRING# | TIME64 | TIMEDATE48 | UNICODE16 |
| UNIPOLAR2_16 | UNSIGNED# | UNSIGNED_L# | WORD# |

This notation uses compact encoding rules:

- it assumes that all user-defined types are recognised by the destination.
- it uses primitive types of fixed size (in ASN.1, an integer can be of any size)
- it includes the size of the elements explicitly in a dedicated field where needed.
- it introduces own keywords to avoid confusion with ASN.1 where the semantic is similar, but different (for instance ONE_OF instead of CHOICE, SOME_OF instead of SET).
- it uses no implicit type-tagging, except for the ONE_OF and SOME_OF types, where tagging is explicitly done by a dedicated field.
- it has no optional fields (except in SOME_OF).
- it is not aligned, although alignment can be specified.

The following rules for notation are used:

- Keywords, including basic types, and constant identifiers are entirely in upper case.
- Type identifiers begin with an Uppercase letter
- Field identifiers begin with a lower case letter

Notation for the primitive types

Notation for the boolean type

Definition

A primitive type with two distinguished values, TRUE and FALSE.

NOTES

1 - This is the ASN.1 definition of a 'BooleanType'.

2 - This type is used to represent binary inputs and outputs (relay, led, micro switch, etc.).

Syntax

```
BooleanType ::= BOOLEAN1
```

Encoding

A variable of boolean type shall be encoded as one bit:

| 1st | interpretation |
|-----|----------------|
| | |
| 0 | FALSE |
| 1 | TRUE |

Notation for the antivalent type

Definition

A primitive type with four distinguished values.

NOTES

- 1 - This is not an ASN.1 type.
- 2 - Variables of this type are used as check variable for other variables or for critical Booleans.

Syntax

```
AntivalentType ::= ANTIVALENT2
```

Encoding

A variable of antivalent type shall be transmitted as 2 bits, the first corresponding to the boolean meaning of the variable and the second to its inverse.

It may take one of four states, as follows:

| 0 | 1 | interpretation |
|----------|-------|----------------|
| 2^{+1} | 2^0 | |
| 0 | 0 | ERROR |
| 0 | 1 | FALSE |
| 1 | 0 | TRUE |
| 1 | 1 | UNDEFINED |

NOTE - the ERROR and UNDEFINED states may be interpreted as legal states by an application.

Notation for the unsigned integer types

Definition

A primitive type with distinguished values which are positive whole numbers, including zero (as a single value), having a fixed size in bits defined by the suffix #.

NOTE - This is a ASN.1 'IntegerType', restricted to a fixed size # and non-negative values.

Syntax

UnsignedType ::= UNSIGNED#, (# = any unsigned integer).

Encoding

An unsigned integer shall be transmitted in binary representation, most significant bit first.

When the carried value has a smaller size than the UNSIGNED# type, it shall be right-justified and extended to the left with zeroes.

UNSIGNED8 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |

Range: 0..255

UNSIGNED16 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|----------|----------|----------|----------|----------|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |

Range: 0..65535

UNSIGNED32 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 2^{31} | 2^{30} | 2^{29} | 2^{28} | 2^{27} | 2^{26} | 2^{25} | 2^{24} | 2^{23} | 2^{22} | 2^{21} | 2^{20} | 2^{19} | 2^{18} | 2^{17} | 2^{16} |
| 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

Range: $0..+2^{32}-1$

Notation for the integer type

Definition

A primitive type with distinguished values which are positive and negative whole numbers, including zero (as a single value), having a fixed size in bits defined by the suffix #.

NOTE - This is a ASN.1 'integer type', restricted to a fixed size of #.

Syntax

IntegerType ::= INTEGER#, (# = any unsigned integer).

Encoding

The value shall be represented in binary 2's complement, with the first transmitted bit being the sign bit.

When the carried value has a smaller size than the INTEGER# type, it shall be right-justified and sign-extended to the left (if it is negative, with '1', otherwise, with '0')

INTEGER8 encoding

| | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| sign | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |

Range: -128 .. +127

EXAMPLE: '1111 1110'B = - 2

INTEGER16 encoding

| | | | | | | | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| sign | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |

Range: $-2^{15} .. 2^{15} - 1$

INTEGER32 encoding

| | | | | | | | | | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| sign | 2^{30} | 2^{29} | 2^{28} | 2^{27} | 2^{26} | 2^{25} | 2^{24} | 2^{23} | 2^{22} | 2^{21} | 2^{20} | 2^{19} | 2^{18} | 2^{17} | 2^{16} |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |

Range: $-2^{31} .. +2^{31} - 1$

Notation for the enumerated type

Definition

A primitive type whose values are given distinct identifiers as part of the type notation, having a fixed size in bits defined by the suffix #.

NOTE - This is an ASN.1 ENUMERATED type, restricted to a fixed size of #.

Syntax

EnumeratedType ::= ENUM#{Enumeration}

with

(# = any unsigned integer)

Enumeration ::= NamedNumber | Enumeration, NamedNumber

and

NamedNumber ::= identifier (UnsignedNumber) | identifier (DefinedValue)

Values can be listed in any order.

EXAMPLE:

```
Day_Of_Week_Type ::= ENUM4
{
  monday           (1),
  tuesday          (2),
  wednesday        (3),
  thursday         (4),
  friday           (5),
  saturday         (6),
  sunday           (7),
  undefined        (0)
}
```

Value '2' means 'TUESDAY'.

Encoding

Values of ENUM# shall be represented by an unsigned integer occupying the same place.

ENUM4 encoding

| 0 | 1 | 2 | 3 |
|-------|-------|-------|-------|
| 2^3 | 2^2 | 2^1 | 2^0 |

Range: 0..15

EXAMPLE: '0001'B means 'Monday' in the above example.

ENUM8 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |

Range: 0..255

EXAMPLE: '0000 0001'B means 'Monday' in the above example (considering it is ENUM8 rather than ENUM4).

Notation for the binary coded decimal type

Definition

A 4-bit unsigned integer expressing a decimal digit between 0 and 9.

NOTE - This type does not exist in ASN.1

Syntax

```
BinaryCodedDecimalType ::= BCD4
```

Encoding

A BCD4 shall be encoded as an unsigned integer occupying the same place.

| 0 | 1 | 2 | 3 |
|-------|-------|-------|-------|
| 2^3 | 2^2 | 2^1 | 2^0 |

Range: 0..9 (other values undefined)

EXAMPLE: '0111'B = 7.

NOTE - some undefined values may be used, for instance to designate the sign or another arithmetic operator.

Notation for the unipolar types

Definition

primitive types with distinguished values which are non-negative, whole numbers divided by a fixed power of two, expressing a value in percent of a span.

NOTE - These types do not exist in ASN.1, they are expressed in IEC 870 as 'unsigned fixed point number'

Syntax

UnipolarType ::= UNIPOLAR2_16

NOTES

1 - The number before the comma gives the number of power of 2 forming the integer part

2 - The epsilon factor is equal to the value of the smallest power of two in the word.

Encoding

A variable of unipolar type shall be transmitted as an unsigned integer.

UNIPOLAR2_16 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|--------------|-------|----------|-----------------|----------|----------|----------|----------|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|
| 2^1 | 2^0 | 2^{-1} | 2^{-2} | 2^{-3} | 2^{-4} | 2^{-5} | 2^{-6} | 2^{-7} | 2^{-8} | 2^{-9} | 2^{-10} | 2^{-11} | 2^{-12} | 2^{-13} | 2^{-14} |
| integer part | | | fractional part | | | | | | | | | | | | |

Span: 0 .. 400% - epsilon

Notation for the bipolar types

Definition

Primitive types with distinguished values which are positive or negative, whole numbers (including zero) divided by a fixed power of two, expressing a value in percent of a span.

NOTE - These types do not exist in ASN.1, they are expressed in IEC 870 as 'signed fixed point number'

Syntax

`BipolarType ::= BIPOLAR2_16 | BIPOLAR4_16`

NOTES

- 1 - The number before the comma gives the number of power of 2 forming the integer part.
- 2 - The epsilon factor is equal to the value of the smallest power of two in the word.

Encoding

BIPOLAR2_16 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|--------------|-------|----------|----------|-----------------|----------|----------|----------|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|
| sign | 2^0 | 2^{-1} | 2^{-2} | 2^{-3} | 2^{-4} | 2^{-5} | 2^{-6} | 2^{-7} | 2^{-8} | 2^{-9} | 2^{-10} | 2^{-11} | 2^{-12} | 2^{-13} | 2^{-14} |
| integer part | | | | fractional part | | | | | | | | | | | |

Span: -200%..+200%-epsilon

BIPOLAR4_16 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|--------------|-------|-------|-------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|-----------|-----------|
| sign | 2^2 | 2^1 | 2^0 | 2^{-1} | 2^{-2} | 2^{-3} | 2^{-4} | 2^{-5} | 2^{-6} | 2^{-7} | 2^{-8} | 2^{-9} | 2^{-10} | 2^{-11} | 2^{-12} |
| integer part | | | | fractional part | | | | | | | | | | | |

Span: -800%..+800%-epsilon

Notation for the real type

Definition

A primitive type whose distinguished values are members of the set of real numbers.

Syntax

`RealType ::= REAL32`

Encoding

This type shall be encoded as IEEE 754 prescribes for Short Real Number (32-bit).

NOTES

- 1 - This is an ASN.1 'RealType', restricted to the IEEE 754 Short Real Number format.
- 2 - The 64-bit floating point number of IEEE 754 (REAL64) is not considered useful in this context.

| | | | | | | | | | | | | | | | | |
|-----------------|--------------------------------|----|----|----|----|----|--------------------------------|----------|----------|----|----|----|----|----|------------------|--|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |
| sign | 2 ⁷ biased exponent | | | | | | 2 ⁰ 2 ⁻¹ | | mantissa | | | | | | 2 ⁷ | |
| 2 ⁻⁸ | | | | | | | | mantissa | | | | | | | 2 ⁻²³ | |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | |

Range: $\pm 3.37 \times 10^{+38}$

Notation for the character type

Definition

A primitive type whose distinguished values are members of the set of characters defined in ISO 8859-1.

Syntax

CharacterType ::= CHARACTER8

Encoding

Characters shall be transmitted in one octet, without parity bit.

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

EXAMPLE - '01100001'B = character 'a' according to ISO 8859-1.

Notation for the Unicode character type

Definition

A primitive type whose distinguished values are members of the set of characters defined in ISO 10646-X.

Syntax

UnicodeType ::= UNICODE16

Encoding

Unicode characters shall be transmitted in two octets.

| | | | | | | | | | | | | | | | |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 2 ¹⁵ | 2 ¹⁴ | 2 ¹³ | 2 ¹² | 2 ¹¹ | 2 ¹⁰ | 2 ⁹ | 2 ⁸ | 2 ⁷ | 2 ⁶ | 2 ⁵ | 2 ⁴ | 2 ³ | 2 ² | 2 ¹ | 2 ⁰ |

Notation for the uncommitted types

Definition

An uncommitted type of undefined contents, but of fixed size.

Syntax

AnyType ::= WORD#, (# = any unsigned integer)

Encoding

A variable of uncommitted type has no prescribed encoding.

Bits shall be named according to the power of two of a variable of type UNSIGNED# which would occupy that place.

NOTE - This naming is in the reverse direction as the offset within the same word.

WORD8 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |

WORD16 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|----------|----------|----------|----------|----------|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |

Structured types

General

Five different structured types are defined:

1. RECORD (variable length),
2. ARRAY (fixed length or variable length),
3. BITSET# (fixed length),
4. ONE_OF (variable length)
5. SOME_OF (variable length).

Notation for the record types

Definition

A structured type defined by referencing a fixed, ordered list of types; each value of the new type is an ordered list of values, one from each of the component types

NOTES

1 - This type is an ASN1 'Sequence Type' with no optional types.

2 - It is recommended to observe alignment when defining a RECORD, i.e. all elements should be located at an offset with respect to the beginning of the record which is a multiple of their size.

Syntax

```
RecordType ::= RECORD { ElementTypeList }
```

with

```
ElementTypeList ::= ElementType | ElementTypeList, ElementType
```

and

```
ElementType ::= identifier Type | Type
```

The elements of a RECORD shall be identified by the identifier of the RECORD field followed by a dot and the subfield identifier, which may itself be a structured type.

EXAMPLE:

file.date.day

Encoding

Elements of a RECORD shall be transmitted in the order of their declaration.

EXAMPLE - A value of type Date32 is represented as follows:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|-------|---|---|---|-----|---|----|----|----|----|----|----|
| year | | | | | | | | | | | | | | | |
| dummy | | | | month | | | | day | | | | | | | |

```
Date32 ::= RECORD
{
  year      INTEGER16,
  dummy     WORD4,
  month     UNSIGNED4,
  day       UNSIGNED8
}
```

'dummy' was introduced to align the variable 'day' on a 16-bit word boundary.

Notation for the bitset types

Definition

an ARRAY [#] of BOOLEAN1, having a fixed size in bits defined by the suffix #.

NOTE - this type corresponds to BITSTRING in ASN.1.

Syntax

```
BitsetType ::= BITSET# {NamedBitList}
```

with

```
NamedBitList ::= NamedBit | NamedBitList, NamedBit
```

and

```
NamedBit ::= identifier (number) | identifier (DefinedValue)
```

1. The value of each 'number' or 'DefinedValue' appearing in the 'NamedBitList' shall be different, and is the offset of a distinguished bit in a bitset value.
2. Each 'identifier' appearing in the 'NamedBitList' shall be different.
3. All elements are implicitly of type BOOLEAN1. The DefinedValue can only be TRUE (1) or FALSE (0).
4. Elements shall be declared in order of increasing offset.
5. If all elements of the BITSET are declared, 'number' can be omitted. This should be the normal case.

Encoding

Elements of a bitset shall be transmitted in order of declaration.

NOTE - The name of the bitset elements may be different from their offset. In assembly language programming, the bit with offset 0 is usually named 'B7' in an 8-bit computer, or 'B15' in a 16-bit computer.

BITSET8 encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-----|---|---|---|---|---|---|-----|
| 1st | | | | | | | 8th |

Range: 8-bit Set of Boolean

EXAMPLE:

```
AccessType8 ::= BITSET8
{
  system      (0),          -- first bit of the bitset (MSB)
  owner       (1),
  group       (2),
  world       (3),
}
```

is equivalent to:

```
AccessType8 ::= BITSET8
{
```

```

system,          -- first bit of the bitset (MSB)
owner,
group,
world,
reserved4
reserved5
reserved6
reserved7       -- 8th or last bit of the bitset (LSB)
}
    
```

An UNSIGNED8 occupying that space with a value of '80'H means that 'system' is the only member of the set.

BITSET16 encoding

| | | | | | | | | | | | | | | | | |
|-----|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|-----------|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1st | | | | | | | | | | | | | | | | |

EXAMPLE:

```

AccessType ::= BITSET16 { system (0), owner (1), group (2), world (3)}
    
```

Value '0110 0000 0000 0000'B means that 'owner' and 'group' are members of the set.

BITSET32 encoding

| | | | | | | | | | | | | | | | | |
|-----|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1st | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | |
| | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

BITSET64 encoding

| | | | | | | | | | | | | | | | | |
|-----|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|-----------|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1st | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | |

Notation for the array type

Definition

A structured type, defined by referencing a single existing type; each value of the new type is an ordered list of zero, one or more values of the existing type. The position of each value is identified by an index. The number of values is indicated by either a

constant or a field of the embedding structure. The number of values may be omitted if a stop element is supplied.

NOTE - An ARRAY is an ASN1 'SequenceOf Type' with a number of elements indicated by a constant, a dedicated variable or not at all (stop element).

Syntax

```
ArrayType ::= ARRAY [IndexList] OF Type
```

```
IndexList ::= Index | IndexList, Index
```

```
Index ::=
```

```
    number | DefinedValue |
    identifier |
    identifier UnsignedType |
    UnsignedType |
    STOP = Value
```

The number, DefinedValue or identifier specify the size of the array in number of elements (0 for a void array). Its type shall be an unsigned integer.

If an unsigned type with a defined identifier is indicated, this declares the corresponding field.

If the identifier names a field declared outside of the array, this field shall be located in the embedding data structure at the same level of nesting, or be a subfield of a field located at the same level of nesting, in which case the full path name shall be indicated.

If a stop value is defined to close an open array, the value shall be of the same type as the array element.

The size may be given by an arithmetic expression.

Encoding

Arrays shall be transmitted in order of increasing index.

Multi-dimensional arrays shall be transmitted in the order their indices are listed.

NOTE - ARRAY OF [row, column] is transmitted row by row.

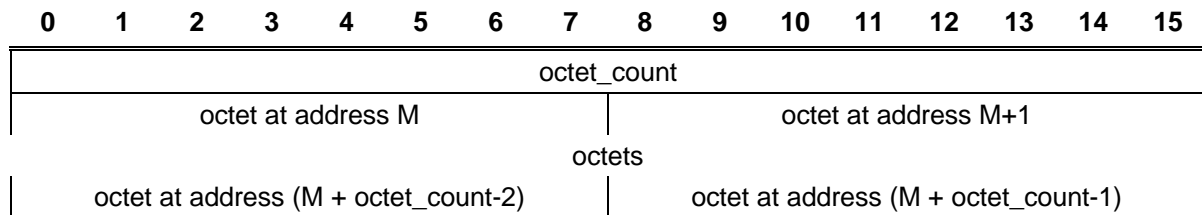
Arrays of octets (uncommitted contents, e.g. memory dump) shall be transmitted in increasing memory address (or index) of the Application Memory.

All elements of the array shall be transmitted, even those which are not significant.

EXAMPLES

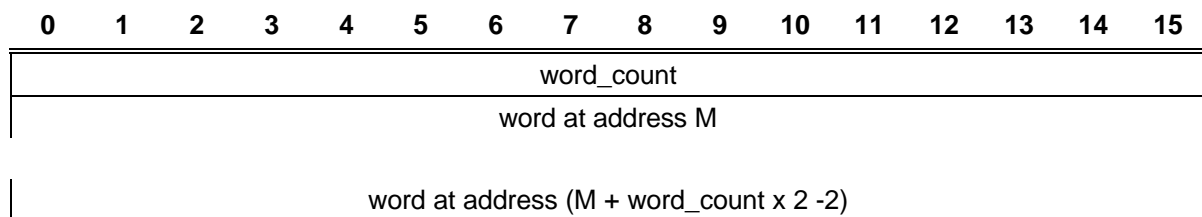
1 - Transmission of an octet memory dump:

```
DumpOctetType ::= ARRAY [octet_count UNSIGNED16] OF WORD8.
```



2 - Transmission of the same memory dump by words of 16 bits, 'word_count' having half the value of 'octet_count' of the preceding example:

DumpWordType ::= ARRAY [word_count UNSIGNED16] OF WORD16.



3 - Number of elements given by a field in the nesting data structure (at an unspecified offset):

DumpOctetType ::= ARRAY [array_count] OF WORD8.

4 - Number of elements given by a field of a structured value in the nesting structure:

HeaderType ::= RECORD

```
{
  name           ARRAY [32] OF CHARACTER8
  bodysize       UNSIGNED16,
  ...
}
```

FrameType ::= RECORD

```
{
  header         HeaderType
  body           ARRAY [ header.bodysize ] OF CHARACTER8
}
```

5 - Character strings, in which the stop character is a 'space':

ProfibusString ::= ARRAY [STOP = '20'H] OF CHARACTER8.

Notation for the choice types

Definition

A structured type, defined by referencing a fixed, unordered, list of distinct types; each value of the new type is a value of (exactly) one of the component types.

NOTE - this type corresponds to the ASN1 'ChoiceType', but has a dedicated tag.

Syntax

```
OneOfType ::= ONE_OF [identifier | identifier EnumeratedType]
           {AlternativeTypeList}
```

with

```
AlternativeTypeList ::= ElementType | ElementTypeList, ElementType
```

and

```
ElementType ::= identifier [tag] Type | [tag] Type
```

and

```
tag ::= UnsignedNumber | DefinedValue | identifier
```

If a named variable is used as a tag, this variable shall be located in the structure embedding the element.

If the tag variable is located at the same nesting level as the choice, only the name of the variable shall be included.

If the variable is at another level of nesting, the path to the same level of nesting shall be included.

EXAMPLES:

1 - The tag is a number (not recommended since this number must be defined in different places):

```
Commands ::= ONE_OF [choice_var ENUM8]
{
  [3]           OpenSequence,
  [2]           CloseSequence,
  [5]           StandbySequence
}
```

2 - The tag is an enumeration type located in the 16 bits before the choice:

```
CommandType ::= ENUM16
{
  OPEN           (3),
  CLOSE          (2),
  STANDBY        (5)
}
```

```
Commands ::= ONE_OF [choice_var CommandType]
{
  [OPEN]         OpenSequence,
  [CLOSE]        CloseSequence,
  [STANDBY]      StandbySequence
}
```

3 - The tag is defined at the same level of nesting in the embedding structure:

```
Commands ::= ONE_OF [choice_var]
{
  [OPEN]           OpenSequence,
  [CLOSE]          CloseSequence,
  [STANDBY]        StandbySequence
}
```

```
Command_Frame ::= RECORD
{
  choice_var       CommandType,
  ...
  command          Commands;
  ...
}
```

4 - The tag is defined in a subfield of a field located at the same level of nesting:

```
Commands ::= ONE_OF [Command_Frame.header.choice_var]
{
  [OPEN]           OpenSequence,
  [CLOSE]          CloseSequence,
  [STANDBY]        StandbySequence
}
```

```
Command_Frame ::= RECORD
{
  header           RECORD
  {
    ....addresses  ...
    choice_var     CommandType
    ....
  }
  commands         Commands
}
```

NOTE - Relative paths (e.g. -/header) are not recommended.

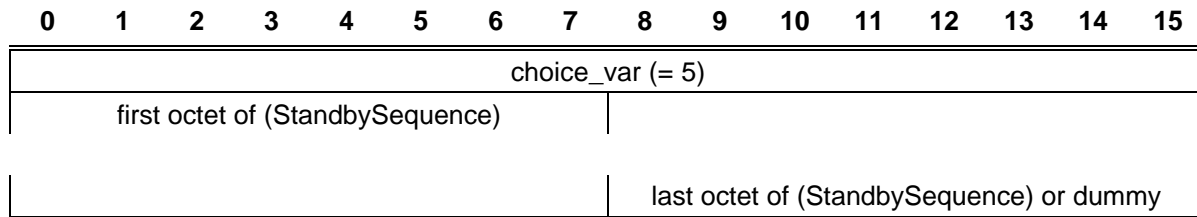
Encoding

A ONE_OF shall be encoded by transmitting before the value the tag field indicating which choice has been made.

The size of the transmitted value is either implicit or indicated in the type itself.

NOTE - A ONE_OF is a SOME_OF with only one element.

EXAMPLE - A particular value of the above Commands choice will be transmitted as:



Notation for the set types

Definition

A structured type, defined by referencing a fixed, unordered, list of distinct types, some of which may be declared as optional); each value of the new type is an unordered list of values, one for each of the transmitted component types.

NOTE - this type corresponds to the ASN1 'SetType', but has a dedicated tag.

Syntax

```
SetType ::= SOME_OF { ElementTypeList }
```

with

```
ElementTypeList ::= ElementType | ElementTypeList, ElementType
```

and

```
ElementType ::= [tag] NamedType
```

and

```
tag ::= identifier | identifier ElementType | ElementType
```

In case the tag is an identifier without an Element Type, the corresponding tag variable shall be included in a data structure at the same level of nesting and be identified by its full path name.

EXAMPLES

1 - Tag is an unsigned integer

```
MemberType ::= SOME_OF [UNSIGNED8]
{
  OPENSEQ           [3]           Type_OpenSequence,
  CLOSESEQ          [2]           Type_CloseSequence,
  STANDBY           [5]           Type_StandbySequence
}
```

2 - Omission of the

If the members of the SOME_OF type are fixed in number, the reference name can be omitted to every member whose purpose is evident from its type.

EXAMPLE:

```
MemberType ::= SOME_OF [UNSIGNED8]
{
```

```

[3]          Type_OpenSequence,
[2]          Type_CloseSequence,
[5]          Type_StandbySequence
}

```

If **SOME_OF** uses as a selector an enumerated type, the enumeration constants shall be put in bracket:

```

MemberType          ENUM8
{
  OPENSEQ           (3),
  CLOSESEQ          (2),
  STANDBY           (5)
}
..
CommandsType ::= SOME_OF [MemberType]
{
  [OPENSEQ]        Type_OpenSequence,
  [CLOSESEQ]       Type_CloseSequence,
  [STANDBY]        Type_StandbySequence
}

```

If **SOME_OF** uses as a selector a bitset variable , this variable shall be defined previously within the embedding data structure and referred to by its full path name:

```

MembersType          BITSET8
{
  OPENSEQ           (3),
  CLOSESEQ          (2),
  STANDBY           (5)
}
...
CommandsType ::= SOME_OF [members]
{
  [OPENSEQ]        Type_OpenSequence,
  [CLOSESEQ]       Type_CloseSequence,
  [STANDBY]        Type_StandbySequence
}

Commands_Frame ::= RECORD
{
  ...
  members          MembersType,
  ...
  commands         CommandsType
  ...
}

```

Encoding

A set is encoded by transmitting before each transmitted type in the set the tag which indicates which type is being transmitted next, the particular tag value 'FF'H (all ones) closing the transmitted set.

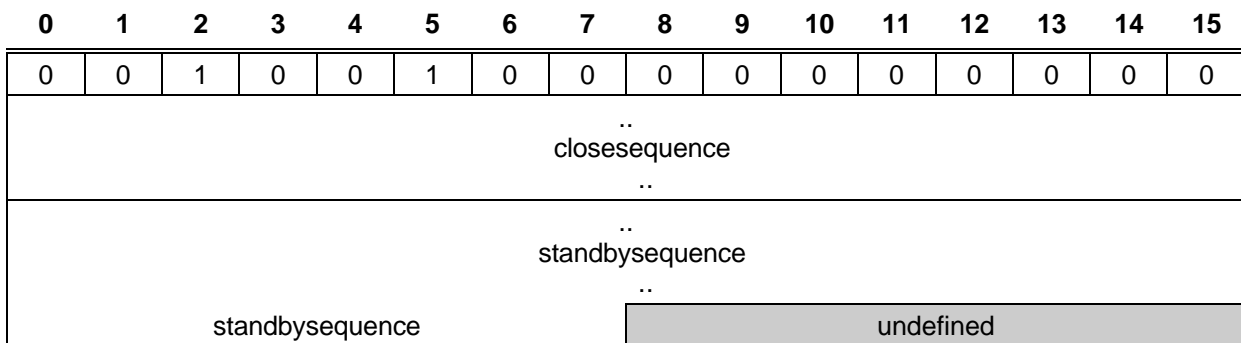
If the tag is replaced by a bitset, the bitset shall be transmitted before the set and the different members of the set shall be transmitted contiguously.

EXAMPLES

1 - A particular value of the above MemberType set will be transmitted as:



2 - If the tag is replaced by a bitset, the coding will be:



Alignment

In any type, it may be necessary to add padding bits to align the next field on a 16 or a 32-bit boundary (or any other boundary). To express this, the qualifier ALIGN is used after the type. The padding bits are not defined (they are 0 by default).

EXAMPLE - The following defines an array of characters which is aligned on a 32-bit boundary, regardless of the value of 'count'.

```
AlignedString ::= ARRAY ALIGN 32 [count] OF CHARACTER8.
```

Notation for special types

Some structured types have a special type designator.

Notation for the string type

STRING# is an ARRAY [] OF CHARACTER8, in which the stop element shall be the character '00'H, the actual size of the string is deduced from the number of significant characters, although the number of transmitted characters may be larger

EXAMPLE - A text string of type STRING32 is represented by an ARRAY [32 STOP='00'H] OF CHARACTER8.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------------------------|---|---|---|---|---|---|---|-------------------------|---|----|----|----|----|----|----|
| 1st character or '00'H | | | | | | | | 2nd character or '00'H | | | | | | | |
| characters ... | | | | | | | | | | | | | | | |
| last character or '00'H | | | | | | | | 32nd character or '00'H | | | | | | | |

Notation for the TIMEDATE48 type

Definition

A structured type expressing the absolute time in number of seconds since Universal Co-ordinated Time (UTC), 00:00:00, 1st January 1970 (Unix and ANSI-C format).

NOTE - This type is used for distribution of the actual time, event tagging, synchronisation.

Syntax

```
TimeDate48 ::= RECORD
{
  seconds      UNSIGNED32, -- elapsed since 1970, January 1st, 00:00
  ticks       UNSIGNED16 -- fraction of seconds (1 tick = 1/65536s)
}
```

Encoding

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----------------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| seconds (most significant) | | | | | | | | | | | | | | | |
| seconds (least significant) | | | | | | | | | | | | | | | |
| ticks = 1/65536 second | | | | | | | | | | | | | | | |

Time can be represented to a granularity of 15,3 μ s (=1/65536 s).

The range is 136 years.

The precision of the fractional part shall be at least 10 bits.

Unused low order bits shall be set to zero.

NOTE – A TimeDate48 variable will wrap around in the year 2038, on January 19, 3:14:07 UTC. This wrap-around should be considered in the test of the software. In fact, new developments should consider the year 2038 as being the reference date, rather than 1970.

Notation for the TIME64 type

Definition

A structured type expressing the absolute time (UTC) in seconds since 1900, January 1st, 00:00. This time is not compensated by leap seconds.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|-----------------------------|---|----|----|----|----|----|----|
| | | | | | | | | seconds (most significant) | | | | | | | |
| | | | | | | | | seconds (least significant) | | | | | | | |
| | | | | | | | | ticks = 1/65536 second | | | | | | | |
| | | | | | | | | chirp = 0,232 microsecond | | | | | | | |

NOTE

1 – This time definition is taken from Internet's RFC1305, which defines the synchronisation protocol for a distributed clock systems. It is different from UNIX time, which is based on the year 1970.

2 - A TIME64 variable will wrap around in the year 2036. This wrap-around should be considered in the test of the software. This time definition can therefore also be considered as defining the time remaining until January 2036.

Notation for the ASN.1 boolean8 type

Definition

A primitive type with two distinguished values, TRUE and FALSE.

NOTE - This is the ASN.1 'BooleanType'.

Syntax

Boolean8Type ::= BOOLEAN8

Encoding

A variable of boolean8 type shall be encoded on 8 bits, '00000000'B being interpreted as FALSE and any other value as TRUE.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|-------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FALSE |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | TRUE |

Notation for Little-Endian types

For compatibility with foreign devices, the following types allow to express data structures in Little-Endian notation. These types should not be used in new projects:

| | |
|-------------|---|
| ENUM_L16 | Little-Endian representation of a 16-bit enumeration type |
| INTEGER_L16 | Little-Endian representation of a 16-bit integer |
| INTEGER_L32 | Little-Endian representation of a 32-bit integer |

UNSIGNED_L16 Little-Endian representation of a 16-bit unsigned

UNSIGNED_L32 Little-Endian representation of a 32-bit unsigned

NOTE - Other types may be constructed on the same pattern.

Type identifier

When it becomes necessary to exchange type information, e.g. during commissioning, the type shall be encoded as indicated by Table 2 as two 8-bit words, one expressing the size (Var_Size) and the other the type (Var_Type).

Var_Size indicates the number of bits in simple types and the number of elements in arrays.

Table 2 - Var_Size and Var_Type encoding

| Var_Size | Var_Type | Data type |
|----------|----------|---|
| '00'H | '00'H | BOOLEAN1 |
| | '01'H | ANTIVALENT2 |
| | '02'H | BCD4 or ENUM4 |
| | '03'H | reserved |
| | '04'H | BITSET8 |
| | '05'H | UNSIGNED8 or ENUM8 |
| | '06'H | INTEGER8 |
| | '07'H | CHARACTER8 (ARRAY [0..0] OF WORD8) |
| '01'H | '04'H | BITSET16 |
| | '05'H | UNSIGNED16 or ENUM16 |
| | '06'H | INTEGER16 |
| | '08'H | BIPOLAR2_16 |
| | '09'H | UNIPOLAR2_16 |
| | '0A'H | BIPOLAR4_16 |
| '02'H | '03'H | REAL32 |
| | '04'H | BITSET32 |
| | '05'H | UNSIGNED32 or ENUM32 |
| | '06'H | INTEGER32 |
| '03'H | '02'H | TIMEDATE48 |
| '04'H | '04'H | BITSET64 |
| | '05'H | UNSIGNED64 |
| | '06'H | INTEGER64 |
| n-1 | '07'H | ARRAY [n] OF WORD8 (odd number of octets) |
| | '0F'H | ARRAY [n] OF WORD8 (even number of octets) |
| n-1 | '0D'H | ARRAY [n] OF UNSIGNED16 (n= number of array elements) |
| | '0E'H | ARRAY [n] OF INTEGER16 (n= number of array elements) |
| n-1 | '0B'H | ARRAY [n] OF UNSIGNED32 (n= number of array elements) |
| | '0C'H | ARRAY [n] OF INTEGER32 (n= number of array elements) |

Types not appearing in Table 2 are reserved for future use.

Data types for analog values

Usage and restrictions

Primitive types such as INTEGER16 or REAL32 represent a basic set on which application types are based, but they do not imply an application meaning.

For instance, a REAL32 could represent a temperature in celcius or a voltage in volts.

Therefore, it is necessary to define for each variable type, i.e. at the limit, for each variable, the exact meaning.

As an intermediate step, we define for each physical magnitude an analog data type, to be used whenever a variable of that physical kind appears, independently of its usage.

For instance, a voltage is represented identically, whether it is a line voltage for a locomotive or a supply voltage for an integrated circuit. This allows the connected tool to reference the correct unit system in all cases.

By contrast, the meaning of binary variables, for instance counter values or incremental position values is so specific that an application data type is preferable (see below).

Whenever analog variables represent physical units, they shall be represented in the International System (SI) units. Exception arise in retrofit installations.

Consequently, even frequently used metric values like rpm (rotations per minute) or KWh (kilowatt-hour) should be converted to SI (radian per second, joule). Conversion from SI units (e.g. metre) to local units (yards, miles, state miles,...) is handled by the device.

SI-Unit code

The following classification is taken from ISO 31 and NIST. Although variables are used in railways do not use all physical units, a complete list is included here as a reference.

To each physical unit corresponds one SI_code. The unit code has been taken from UCA when it comes to non-electrical units, and from IEC when it comes to electrical units.

| SI_code | | Base Units | | | |
|---------|-----|---------------------|----------------------------|---------|------|
| ISO | UCA | Quantity | Unit Name | Symbol | Unit |
| | 0 | | unknown | | |
| | 1 | none | dimensionless, not defined | | (1) |
| | 2 | length | meter | d, l, w | m |
| | 3 | mass | kilogram | m | kg |
| | 4 | time | second | t | s |
| | 5 | current | ampere | I | A |
| | 6 | temperature | kelvin | T | K |
| | 7 | amount of substance | mole | | mol |

| | | | | | |
|--|---|--------------------|---------|--|----|
| | 8 | luminous intensity | candela | | cd |
|--|---|--------------------|---------|--|----|

| Derived Units with special names | | | | | |
|----------------------------------|-----|-----------------------|--|--------|--------------------------------|
| ISO | UCA | Quantity | Unit Name | Symbol | Unit |
| | 10 | plane angle | radian | | rad (1) |
| | 11 | solid angle | steradian | | sr (1) |
| | 33 | frequency | hertz (1/s) | f | Hz |
| | 32 | force | newton (kg m / s ²) | F | N |
| | 39 | pressure | pascal (N / m ²) | p | Pa |
| | 31 | energy | joule (N m) | W | J |
| | 38 | power | watt (J /s) | P | W |
| | 26 | electric charge | coulomb (As) | Q | C |
| | 29 | electric potential | volt (W/A) | U | V |
| | 30 | electric resistance | ohm (V/A) | R | Ω |
| | 25 | electric capacitance | farad (C/V) | C | F |
| | 27 | electric conductance | siemens (A/V) | G | S |
| | 36 | magnetic flux | weber (V s) | H | Wb |
| | 37 | magnetic flux density | tesla (V s/ m ²) | B | T |
| | 28 | electric inductance | henry (Wb/A) | L | H |
| | 23 | relative temperature | degrees Celcius | T | °C |
| | 35 | luminous flux | lumen (cd sr) | | lm |
| | 34 | illuminance | lux (lm / m ²) | | lx |
| | 22 | activity | becquerel (l/s) | | Bq |
| | 21 | absorbed dose | gray (J/Kg) | | Gy |
| | 24 | dose equivalent | seivert (J/kg) | | Sv |
| | | angular speed | radian per second | ω | s ⁻¹ |
| | | angular acceleration | radian per second squared | | s ⁻² |
| | 41 | area | square meter (m ²) | A | m ² |
| | 42 | volume | cubic meter (m ³) | V | m ³ |
| | 43 | velocity | meters per second (m / s) | v | ms ⁻¹ |
| | 44 | acceleration | meters per second ² (m / s ²) | a | ms ⁻² |
| | | torque | newton-meter | T | Nm |
| | 45 | volumetric flow rate | cubic meters per second (m ³ / s) | | m ³ s ⁻¹ |
| | 46 | fuel efficiency | meters per cubic meter (m / m ³) | | ms ³ |
| | 47 | moment of mass | kilogram meter (kg m) | | kgm |

| | | | | | |
|--|----|------------------------|--------------------------------|--|--------------------------|
| | 48 | density | kilogram per cubic meter | | kg / m^3 |
| | 49 | kinematic viscosity | square meter per second | | m^2 / s |
| | | dynamic viscosity | newton-second per square meter | | Ns/m^2 |
| | 50 | thermal conductivity | watt / meter kelvin | | $\text{W}/(\text{m K})$ |
| | 51 | heat capacity, entropy | joule / kelvin | | J / K |

| SI_code | | Derived Units without special name | | | |
|---------|-----|------------------------------------|---------------------------------|--------|-----------------------------------|
| ISO | UCA | Quantity | Unit Name | Symbol | Unit |
| | | specific heat | joule per kilogram - kelvin | | $\text{J}/(\text{kgK})$ |
| | | fluence | kilojoule per square meter | | kJ/m^2 |
| | | radiant intensity | watt per steradian | | W/sr |
| | | radiance | watt per square meter steradian | | $\text{W}/(\text{m}^2 \text{sr})$ |
| | | fluence rate | watt per square meter | | W/m^2 |
| | | luminance | candela per square meter | | cd/m^2 |
| | | wave number | reciprocal meter | | m^{-1} |
| | 52 | concentration | parts per million (ppm) | | (1) |
| | | amplitude level difference | $\ln (F_1/F_2)$, neper | | Np (1) |
| | | amplitude level difference | $20 \log (F_1/F_2)$, (decibel) | | dB (1) |
| | | power level difference | $0,5 \ln (F_1/F_2)$, neper | | Np (1) |
| | | power level difference | $10 \log (F_1/F_2)$, (decibel) | | dB (1) |

| SI_code | | Electrical Units (IEEE 268A, ISO 31/V-1979) | | | |
|---------|-----|---|--------------------------|-----------|-----------------------|
| IEC | UCA | Quantity | Unit Name | Symbol | Unit |
| | | | | | |
| 67 | 5 | electric current | ampere | I | A |
| 52 | 26 | electric charge | coulomb (As) | Q | C |
| 53 | 26 | volume charge density | coulomb per cubic meter | ρ | C/m^3 |
| 54 | | surface charge density | coulomb per square meter | σ | C/m^2 |
| 55 | | electric field strength | volt per meter | E | V/m |
| 56 | | electric potential | volt | V, ϕ | V |
| 57 | 29 | potential difference, tension | volt | U, u | V |
| 58 | | electromotive force | volt | e | V |
| 59 | | electric flux | (not IEC) | ψ | C |

| | | | | | |
|----|----|-------------------------------|--------------------------|--------------|------------------|
| 60 | | displacement | coulomb per square meter | D | C/m ² |
| 61 | 25 | electric capacitance | farad (C/V) | C | F |
| 62 | | permittivity | farad per meter | ϵ | F/m |
| 63 | | relative permittivity | dimensionless | ϵ_r | (1) |
| | | ... | | | |
| 70 | | magnetic field strength | ampere per meter | H | A/m |
| 71 | | magnetic potential difference | ampere (turns) | U_m | A |
| | | ... | | | |

| IEC | UCA | Quantity | Unit Name | Symbol | Unit |
|-----|-----|---------------------------|--|-------------------|------------------|
| 72 | | magnetomotive force | ampere (turns) | F | A |
| 72a | | current linkage | ampere (turns) | θ | A |
| 73 | 37 | magnetic flux density | tesla (Wb / m ²) | B | T |
| 74 | 36 | magnetic flux | weber (V s) | ϕ | Wb |
| 75 | | magnetic vector potential | | | |
| 76 | 28 | electric inductance | henry (Wb/A) | L | H |
| 77 | | mutual inductance | henry (Wb/A) | M | H |
| | | ... | | | |
| 80 | | permeability (absolute) | henry per meter | μ | H/m |
| 81 | | relative permeability | dimensionless | μ_r | (1) |
| | | ... | | | |
| 87 | 30 | electric resistance | ohm (V/A) | R | Ω |
| 88 | | resistivity | ohm-meter | ρ | Ω m |
| 89 | 27 | electric conductance | siemens (A/V) | G | S |
| 90 | | conductivity | siemens per meter | γ | S / m |
| | | ... | | | |
| 93 | | electric impedance | ohm | Z | Ω |
| 94 | | electric reactance | ohm | X | Ω |
| 97 | | electric admittance | siemens | Y | S |
| 98 | | electric susceptance | siemens | B | S |
| 99 | 62 | real electric power | watt | P | W |
| 99 | 61 | apparent power | volt ampere | S, P _s | VA |
| 99 | 63 | reactive power | volt ampere reactive | Q,...P. | VA _r |
| | 65 | power factor | Cos θ | | (1) |
| | | electric energy | joule | W | J |
| | 66 | volt seconds | volt seconds (W s / A) | | Vs |
| | 67 | volt squared | volt square (W ² / A ²) | | V ² |
| | 69 | ampere squared | ampere square (A ²) | | A ² |
| | 70 | ampere squared time | ampere square second (A ² s) | | A ² t |

| Non-SI units, use with caution | | | | | |
|--------------------------------|------|-----------------------|-------------------|--------|------|
| | Code | Quantity | Unit Name | Symbol | Unit |
| | 71 | apparent hourly power | volt ampere hours | | VAh |

| | | | | | |
|--|----|-----------------------|----------------------------|--|------|
| | 72 | real hourly power | watt hours | | Wh |
| | 73 | reactive hourly power | volt ampere reactive hours | | VArh |

(1) dimensionless

Data representation for physical units

Representation as floating point numbers

Variables representing physical units should preferably be expressed in a floating-point representation (REAL32), which is the default representation.

This is the recommended practice for all new projects.

In this case, the physical variable shall be referenced as: RVT_<unit name>.

Example:

```
TypeTrueAirSpeed ::= RVT_speed.
```

Note - The unit name rather than the unit was chosen since the unit can be ambiguous.

Representation as fractionals

In some applications, it is preferable to transmit the variables as fractional (fixed-point integers) rather than floating point. There are two main reasons for this:

1. saving bandwidth (a 16-bit analog value representing the range of an A/D converter takes only half the place of a REAL32. At the same time, this data representation can be used for local computations. For this reason, the small fractionals (UNIPOLAR2_16 or BIPOLAR2_16) have been introduced.

For instance, a voltage (0..6553,5 V) is represented as a UNIPOLAR2_16 in steps of 1,0 V, 100% being equal to 1638,4 V). Voltages in the range smaller as 1,0 V cannot be represented correctly.

1. keeping a constant precision independent of the magnitude, as is the case for quantified variables, such as time, money, energy quantity (for billing purpose), distance (travelled by a vehicle). In this case, the large 32 bit, 48 bit or 64 bit fractionals can be used.

For instance, a vehicle may travel during its life 1 000 000 km, but the required precision for daily position and for positioning is 0,1 km. A UNIPOLAR32 allows to represent with a resolution of 1,0 m up to 4 294 967,295 km.

It should however be reminded that the conversion from floating point to fixed point is only possible within a certain range of values, and that technical processes often take values not foreseeable by their designers (cf. the Ariane V501 failure).

The wrap-around situation must be considered in the acceptance test.

Mapping fixed-point to analog values

The type definition of a fractional, or fixed point number, must consider the range of the values. This range can be expressed by an offset and a span, as illustrated by Figure 2.

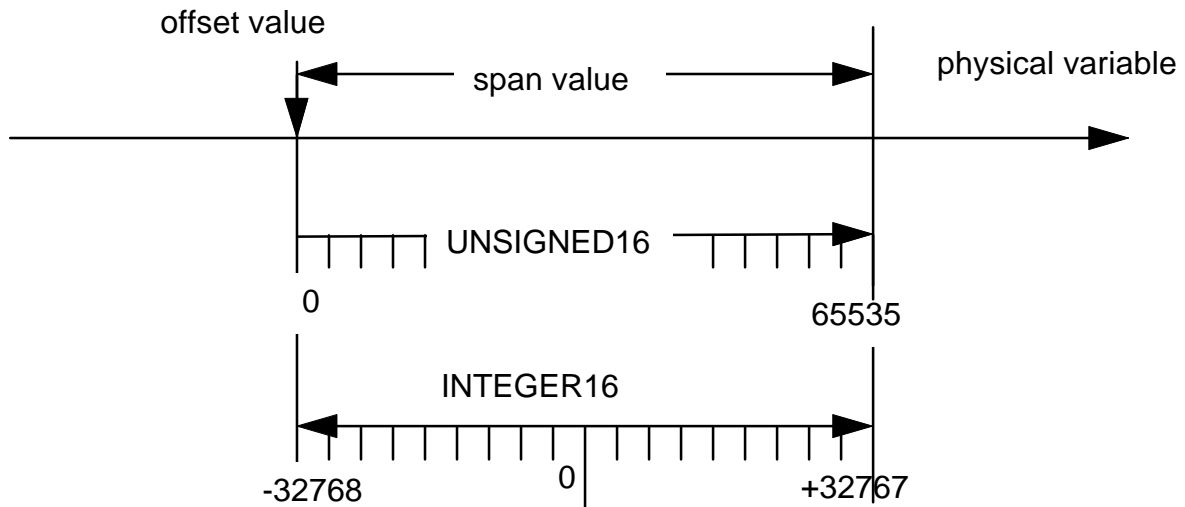


Figure 2 - Mapping from REAL to INTEGER and UNSIGNED

Example: var_type = INTEGER8, span = 100 m/s, offset = 0 means that 0000'0000 represents 0 m/s and 1111'1111 represents 100 m/s.

As an additional convention, the extreme values include the minimum value but exclude the maximum value. The application may decide that the extreme values represent out-of-range.

Example: 0000'0000 represents a speed between 0 and 0,394 m/s, 1111'1111 a speed between 99.61 m/s and 99.9999 m/s, which, in practice includes 100 m/s).

Small fractionals

The value 100% should be chosen low enough to avoid overflow in unforeseen situations, but not so low that its resolution is impaired.

The data type of the variable shall express the offset and range:

Examples:

SI_speed_[0,0..100,0[_UNIPOLAR2_16 is the same as:

SI_speed_[0,0..400,0[_UNSIGNED16

SI_voltage_[-10,0..+10,0[_BIPOLAR2_16

Expressing the fractional variables representation in a unified way

To each variable type, a descriptor allows to map the name to a string, as fo

| | |
|--------------------------------------|---------------------|
| variable_name: STRING32 | |
| (CHARACTER8) | CHARACTER8 or '00'H |
| properties | standard_type |
| variable_type: STRING32 | |
| (CHARACTER8) | CHARACTER8 or '00'H |
| variable_unit: STRING32 | |
| (CHARACTER8) | CHARACTER8 or '00'H |
| real_min | |
| (minimum value of physical variable) | |
| real_max | |
| (maximum value of physical variable) | |
| real_resolution | |
| (resolution of physical variable) | |
| integer_min | |
| (minimum value of physical variable) | |
| Integer_max | |
| (minimum value of physical variable) | |
| bus_id | port_address |
| var_size | var_type |
| var_offset | |
| chk_offset | |

When an equipment exports its variables, it indicates for each variable the name and the representation. Although it would be possible to install a world-wide service for mapping the variable name to a unit (e.g. always map "RVT_apparent_speed" to a REAL32 in m/s), it is simpler to let the device indicate which unit is used and how it is mapped.

Application Data Types

Applications exchange variables under the premise that their semantics have been agreed beforehand. Indeed, variables of the same unit can express different physical variables, for instance root-mean-square current, peak current or instantaneous current.

Therefore, a device exports or imports variables with a certain application data type, which is mapped to a physical unit (if it exists) and which is mapped to a basic data type.

Note that there is no suffix expressing the data size when this size is 32 bits.

ROSIN Variable Types

ROSIN Variable Types (RVT) is a generalisation of the physical variable representation.

RVTs base on a small number of basic types.

It adds the physical values according to the SI-Units to them.

It also defines a quantity of structured data types, such as timestamps.

Finally, RVT define the name structure to be used to describe each variable in an equipment, e.g. air_condition.inlet.temperature.

Example

RVT_average_speed (implicitly a REAL32)

RVT_distance_m64(covered distance in m with a 64 bit resolution)

RVT_angular_acceleration

Proposed additional RVTs for rail transportation

To respond to railways specific variables, the RVTs have been extended by the following types.

Global Positioning Group

The following are RVTs subject to inclusion for railways application.

| Measurement | Name | Range (Resolution) | Based on | ID |
|------------------|------------------|---------------------------|----------|-----|
| GPS longitude | RVT_longitude | -3,14159.. +3,14159 rad | REAL32 | tbd |
| GPS latitude | RVT_latitude | -3,14159..+3,14159 rad | REAL32 | tbd |
| GPS altitude | RVT_altitude | m | REAL32 | tbd |
| GPS velocity | RVT_velocity | m/s | REAL32 | tbd |
| GPS heading | RVT_heading | rad | REAL32 | tbd |
| GPS odometer | RVT_odometer | cm | REAL32 | tbd |
| GPS pitch | RVT_pitch32 | -3.14159E1...+3.14159 rad | REAL32 | tbd |
| GPS acceleration | RVT_acceleration | m/s ² | REAL32 | tbd |

Diverse

| Measurement | Name | Range (Resolution) | Based on | ID |
|------------------------------|-------------|-------------------------------|------------------|-----|
| Scaled Bipolar | RVT_sb16 | -327,67% ..+327,67% | BIPOLAR2_16 | tbd |
| Scaled Unipolar | RVT_su16 | 0 ..+655,35% | UNIPOLAR2_1 6 | tbd |
| Angle_circle (360 degree) | RVT_angle16 | 0..6.28 rad (100% = 1,57 rad) | UNIPOLAR2_1 6 | tbd |

Expressing other standards in ROSIN

Expressing ASN.1

ASN.1 is an ISO/IEC standard notation which is general and more complex than ROSIN, but it cannot express existing data structured in a way not following its notation.

- ASN.1 always precedes each value by its tag (which can imply a type) and its size.
- ASN.1 allows primitive types of variable size (an INTEGER can be 8, 16, 32,...bits).
- ASN.1 puts constraints on types to control the value size.
- ASN.1 selects elements by their tag (not by a separate field).
- ASN.1 lacks some primitive data types used in ROSIN (such as UNIPOLARs).

Note - in theory, ASN.1 does not depend on a particular encoding. In practice, ASN.1 assumes a tagged encoding and controls its encoding by keywords such as IMPLICIT.

Table - Equivalents in ASN.1 and ROSIN

| ASN.1 | ROSIN |
|--|--|
| INTEGER (-128..+127) | INTEGER8 |
| INTEGER (-32768..+32767) | INTEGER16 |
| INTEGER (-2 ³¹ ..+2 ³¹ -1) | INTEGER32 |
| UNSIGNED (0..255) | UNSIGNED8 |
| UNSIGNED (0..65535) | UNSIGNED16 |
| UNSIGNED (0..4294967296) | UNSIGNED32 |
| SEQUENCE | RECORD (uses no tagging nor size) |
| SEQUENCE OF | ARRAY (fixed size or variable size) (explicit size or stop) |
| SET OF | BITSET (there is no SET OF except for Booleans) |
| CHOICE (defined by tag) | ONE_OF (explicit choice variable) |
| SET (defined by tag) | SOME_OF (explicit tag variable) |

There exist several encoding rules for ASN.1: BER is universal, but dispendious (3 octets for a Boolean). PER (in its aligned form) allows about the same compactness as ROSIN, but is unable to express constructs frequently used by programmers.

Differences with WorldFIP

- WorldFIP uses the ASN.1 syntax, with special encoding rules for MPS variables.
- WorldFIP numbers bits within an octet starting with 8 as MSB (ROSIN: 0).
- WorldFIP uses the BER (8825) encoding, consisting of an 8-bit tag, an 8-bit size and the value.
- WorldFIP introduces the following additional types or type redefinition:

| FIP | ROSIN |
|---|---|
| BOOLEAN | BOOLEAN8 (FALSE = 0, TRUE <> 0) |
| Unsigned4 | UNSIGNED4 |
| Unsigned8 | UNSIGNED8 |
| Unsigned16 | UNSIGNED16 |
| Symbol | OPEN ARRAY [15] OF CHAR; |
| OCTETSTRING | OPEN ARRAY OF WORD8 |
| GENERALIZED TIME (YYYYMMDDhhmmss local time) | STRING14 |
| FLOATING POINT | REAL64 |
| BCD | BCD8 (most significant nibble = '0000'B) |
| COMPACTBOOLEAN (first bit is LSB of first octet) | BITSET (first bit is MSB of first octet) |

All FIP types can be expressed with the ROSIN syntax.

Expressing EN50170 / Fieldbus Foundation /IEC 1158-6

The different field busses (Profibus, WorldFIP, Fieldbus Foundation, IEC 1158-6) use practically the same notation under the name of FER (Fieldbus encoding rules), TER (Traditional encoding rules), BER (Buffer encoding rules) or MER (Message encoding rules). The differences with ROSIN are:

- IEC 1158-6 uses the ASN.1 notation, but has special encoding rules (an ASN.1 parser cannot be used).
- IEC 1158-6 numbers bits within an octet starting with 8 (ROSIN: 0).
- IEC 1158-6 encodes Booleans as one octet.
- IEC 1158-6 has no ANTIVALENT2, BCD, ENUM#, Analog types, CHARACTER8.
- IEC 1158-6 closes strings of characters with a "space" (ROSIN: '00'H, like 'C').

- IEC 1158-6 restricts Bitsets to multiples of 8 bits (same numbering as in ROSIN)
- IEC 1158-6 counts TimeOfDay starting from 01 January 1984 (ROSIN: 01.01.1970).
- EN50170 counts Time starting from 01 January 1972 (ROSIN: 01.01.1970).
- IEC 1158-6 allows Optional fields in type SEQUENCE.

| Profibus / FF/IEC 1158-6 | ROSIN |
|---|--|
| Boolean | BOOLEAN8 (false = '00'H, TRUE = 'FF'H) |
| Integer | INTEGER16 |
| Unsigned | UNSIGNED16 |
| Floating-Point | REAL64 |
| Visible-String | OPEN ARRAY OF CHAR, space closes |
| Octet-String | OPEN ARRAY OF WORD8 |
| Date | WORD56 (see below) |
| TimeOfDay | STRING6 (see below) |
| Time | UNSIGNED48 (1/32s since 1 Jan 1972) |
| TimeDifference | like TimeOfDay |
| BCD | BCD8 (most significant nibble = '0000'B) |
| BitString | BITSET (only multiples of 8 bits) |
| SEQUENCE (indicates number of elements, not size) | RECORD (size is implicit) |
| SEQUENCE OF | ARRAY |
| CHOICE | ONE_OF |

The structured types in Profibus or Fieldbus Foundation can be expressed in the ROSIN syntax, as the following example shows:

```
Type_FF_Date ::= RECORD
{
  milliseconds  UNSIGNED16,      -- 0..59999
  reserved1     WORD2,
  minutes       UNSIGNED6,       -- 0..59
  su            ENUM1
  {
    STANDARD (0),
    SUMMER   (1)
  },
  reserved2     WORD2,
  hours         UNSIGNED6,       -- 0..23
  day_of_week   UNSIGNED3,       -- 1..7, 1 = Monday
  day_of_month  UNSIGNED5,       -- 1..31
  reserved3     WORD2,
```

```

months      UNSIGNED6,      -- 1..12
reserved4   WORD2,
years       UNSIGNED7      -- 0..99 (without century)
}

```

```

Type_FF_TimeOfDay ::= RECORD
{
  reserved      WORD4
  milliseconds  UNSIGNED28  -- since midnight
  days          UNSIGNED16   -- since 01 January 1984
}

```

The tag used in Profibus/Fieldbus Foundation and TER is expressed as:

```

FieldbusTag ::=      RECORD
{
  prim_cons      ENUM
  {
    PRIMITIVE    (0),
    CONSTRUCTED  (1)
  },
  tag            UNSIGNED3,  -- extended if 7
  length         UNSIGNED4,  -- extended if 15
  ONE_OF [tag]
  {
    ext_tag      [7] UNSIGNED8
  },
  ONE_OF [length]
  {
    ext_length   [15]          UNSIGNED8
  }
}

```

Conversely, EN50170 types can be included in the ROSIN notation by preceding their type with the following prefixes:

“FIP_” for FIP (FER, BER, MER)

“PFB_” for Profibus

“FBF_” for Fieldbus Foundation

“CAN_” for Can Application Layer type.

In general, one should only include a specific type when there is no corresponding ROSIN type.

Expressing IEC 870-5-4

- IEC 870 is a consequent little-endian, it transmits all data structures starting with the least significant element. This applies equally to INTEGER or to BCD numbers.
- IEC 870-5-4 numbers octets starting with 1, bit 8 within an octet is transmitted first.
- IEC 870 is bit-oriented, not octet-oriented.
- IEC 870 defines additional application types. These types can be mapped to ROSIN types as Table 3 shows.
- IEC 870 uses the ANTIVALENT2 type to represent four types:
 - DoubleCommand,
 - PersistentRegulatingCommand,
 - RegulatingStepCommand and
 - DoublePointInformation.

Table 3 - ANTIVALENT2 in IEC 870-5

| 1st | 2nd | Double Command | Persistent Regulating Command | Regulating Step Command | Double-Point Information |
|---|---|-------------------|-------------------------------------|-------------------------------|-----------------------------|
| $\begin{matrix} \wedge 1 \\ 2 \end{matrix}$ | $\begin{matrix} \wedge 0 \\ 2 \end{matrix}$ | | | | |
| 0 | 0 | not permitted | not permitted | not permitted | indeterminate |
| 0 | 1 | OFF | Lower | Next Step Lower | determined OFF |
| 1 | 0 | ON | Higher | Next Step Higher | determined OFF |
| 1 | 1 | not permitted | not permitted | not permitted | not permitted |

IEC870 types shall be referred to by the prefix: "IEC68705_"

Expressing LONWorks

LONWorks uses a small number of basic types, on which application types are built. The mapping of the primitive types of the LONWorks to ROSIN is shown in Table 4.

Table 4 - LONWorks to ROSIN mapping

| LONWorks | ROSIN |
|----------------|------------|
| unsigned short | UNSIGNED8 |
| signed short | INTEGER8 |
| unsigned long | UNSIGNED16 |
| signed long | INTEGER16 |
| float | REAL32 |

LONWorks define Standard Network Variable Types (SNVTs) mapped on these primitive types (most SNVTs are also primitive types).

Each of these types has an own SNVT identifier (SNVTID). Currently, the highest SNVT defined is 113.

SNVTs can be referred in the ROSIN notation by including the prefix "SNVT_".

Examples:

SNVT_amp (-3276,8..3276,7 A) has SNVTID = 1 and is based on BIPOLAR2_16 (100% = 1638,4 A).

SNVT_amp_f (-1.0×10^{38} .. 1.0×10^{38} A) has SNVTID = 48 and is based on REAL32.

```

TractionCurrent ::= {
    motorcurrent1    SNVT_amp_f,
    exitationcurrent SNVT_amp,
    status           ENUM2,
}
    
```

LONWorks expresses data formats in a syntax based on Neuron-C, which uses a Motorola-style bit and byte ordering, different from other C compilers.

SNVTs use for most physical variables the floating point and several fractional representation, generally as binary multiple of a decimal value.

SNVT should not be used for new projects, but they are defined for retrofit purposes.

Appendix B describes in more detail the mapping between the LON and ROSIN data types and shows an example of conversion between Neuron-C and the ROSIN Syntax.

Expressing XDR

SUN's XDR languages is oriented towards a high-speed communication medium. It is also a trickt big-endian. Its basic unit is 32-bit in size. The mapping is shown in Table 5:

Table 5 - XDR to ROSIN mapping

| XDR | ROSIN |
|---------------------------------------|---|
| boolean | BOOLEAN32 (FALSE=0, TRUE=1) |
| integer | INTEGER32 |
| unsigned | UNSIGNED32 |
| hyper integer | INTEGER64 |
| hyper unsigned | UNSIGNED64 |
| enumeration | ENUM32 |
| float | REAL32 |
| double | REAL64 |
| string | RECORD { size UNSIGNED32, body ARRAY [size ALIGNED32] OF CHARACTER8 } |
| opaque identifier [n] (fixed size) | ARRAY [n ALIGNED32] OF WORD8 |
| opaque identifier <n> (variable size) | RECORD { size UNSIGNED32, body ARRAY [size ALIGNED32] OF WORD8 } |

XDR's 32-bit alignment makes it little suitable for process control.

Conclusion

The ROSIN data representation is able to represent the data types used by most field busses. It can therefore be used as ‘lingua franca’ of the data exchange.

The correct mapping between bus transmission and application programming, it is important to check the format mapping at each intermediate level (bus, Traffic Store, processor, high-level language).

A programmer should not assume that transmission is correct for all types when a number of types have been transmitted correctly.

In addition, a programmer should be aware that correct transmission of all types is no proof that the transmission will be correctly understood by a third party - the same error on both sink and source can remain undetected for a long time.

Therefore, a conformance test of types should be introduced.

Baden, 1998 May 25.

Appendix A: Format mapping in Bus Controllers

The bus traffic is seldom directly observable. A logic analyser connected to a bus sees a sequence of bits, possibly interrupted by checksum or bit stuffing bits.

The Bus Controller converts the bus traffic to a memory format when it translates frames, removes headers and checksums and possibly stuffed bits and stores them in the Traffic Store.

Since the base of the data representation is a memory format, the relationship between bus transmission format and Traffic Store format must be defined for the Bus Controller.

This Annex defines the data traffic on the bus for the MVB and the WTB and explains the mapping.

Bus and application data formats

Big-endian and Little-endians processors

Just as there exists two ways of writing (e.g. from left to right in English or the reverse in Arabic), there exists two primitive numberings for data representation: *big-endian* processors (such as Motorola) store the most significant part of a 16-bit or 32-bit integer at the lowest memory address, while *little-endian* processors (such as Intel) store the least significant octet of that integer at the lowest memory address.

This is only the most visible expression of an underlying convention: Little-endians assume that the most significant item of an integer receives the highest number, while Big-Endians assume that the most significant item receives the lowest number - both views are equally respectable, although none is consistent in reality.

For instance, in an Intel processor (little-endian), the least significant bit of a 16-bit word is transmitted over AD0 and goes to a memory location with an even address.

By contrast, the least significant bit in a Motorola processor is carried by line D0 and goes to a memory location with an odd address.

Note - Intel processors do not require words to be aligned, i.e. the least significant bit can be at an odd address and the most significant bit at the next higher even address. Non-alignment is ignored here. Compilers can be compelled to use only aligned data types.

The only structure which is common to both Intel and Motorola processors is an ARRAY OF CHARACTER8.

This convention has a huge importance since the processor, the Bus Controller and the bus traffic are not necessarily of the same type.

Bit numbering in assembly language

In assembly language, the programmer operates with a bit number which is not necessarily identical to the bit offset.

Big-endian processors are not consequent when naming the bits within an 8-bit, a 16-bit or a 32-bit word. All manufacturers (except Texas Instruments' 9900) number the bits within a word assuming that the word is an unsigned integer and give each bit position the name of the corresponding power of 2. This makes that the least significant bit always receives bit number 0. This bit is transmitted over the processor's parallel bus line called

AD0 (Intel) or D0 (Motorola). The Intel 80x86 and Motorola 68xxx follow the same convention for numbering the bits within a word (Figure A-1):

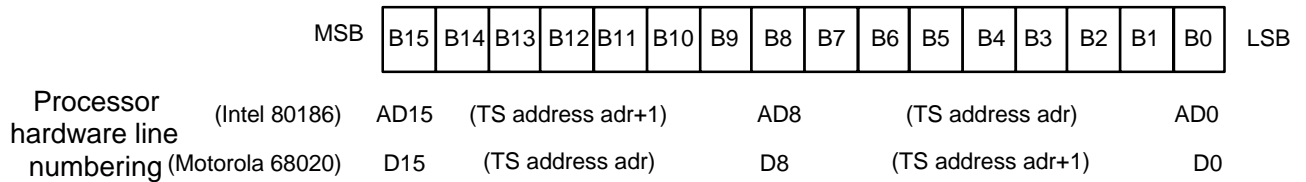


Figure A-1 - 16-bit bitset operations in Intel and Motorola processors

B0 belongs to an odd address at Motorola, and to an even address at Intel, though.

Motorola processors, when performing logical operations such as BSET (Test a Bit and Set) count the bit number from the right.

Example - BSET #0, D6 sets the least significant bit of register D6, i.e. B0 (Motorola)

Operations on bit fields

However, when operating with bit fields, the bit offset is not identical with the bit number as Figure A-2 shows.

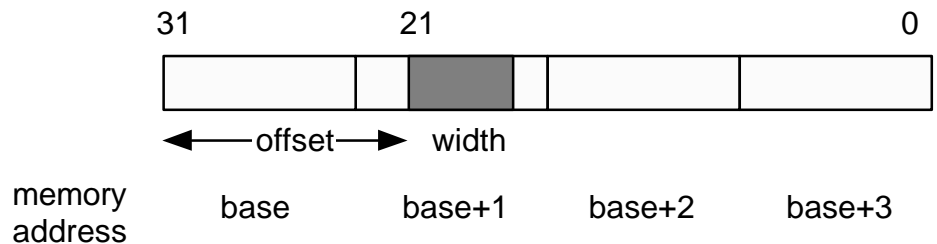


Figure A-2 - Big Endian bit field in a 32-bit word (Motorola)

“For bit fields, the most significant bit is addressed as bit 0 and the least significant bit is addressed as the width of the field minus 1” [Motorola 68020 User’s Manual]

The assembly-language programmer is well aware of this difference. However, the high-level programmer does not know how the C compiler places the bits.

Conversely, little-endian processors count the bit offset from the left. Here, bit ordering and bit offset are identical (Figure A-3).

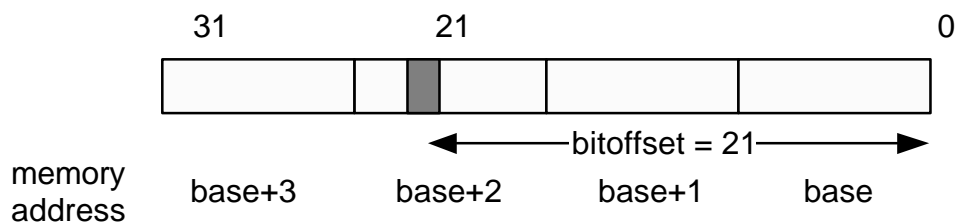


Figure A-3 - Little Endian bit field in a 32-bit word (Intel)

“The addressed bit is numbered (Offset MOD 8) within the byte at address (BitBase + (BitOffset DIV 8)), where DIV is signed division with rounding towards negative infinity, and MOD returns a positive number. [Intel 386 Programmer’s Reference Manual]

To make the distinction clear, one should speak of *bit number* when considering the power of 2 and of *bit offset* when considering the bit position.

Little-endian and Big-endian controller

Bus Controllers can be configured to serve either a little-endian or a big-endian processor.

An 8-bit Bus Controller can be configured for either:

- 1) *direct access*: the first transmitted octet is taken from the even memory address, the second from the odd memory address; or
- 2) *swapped access*: the first transmitted octet is taken from the odd memory address.

In both cases, the data comes over the D7-D0 lines.

A 16-bit controller always fetches one 16-bit word at a time and can be configured for one of two options:

- 1) *direct access*: the first transmitted octet is taken from the D15-D8 lines, the second transmitted octet from the D7-D0 lines.
- 2) *swapped access*: the first transmitted octet is taken from the D7-D0 lines, the second transmitted octet from the D15-D8 lines.

The effect of “swapped” and “direct” access is the same both for the 8-bit and for the 16-bit controller. In the following, the 16-bit case will be discussed.

Figure A-4 shows two processors connected to the bus over a Bus Controller, the format on the bus being the big-endian format.

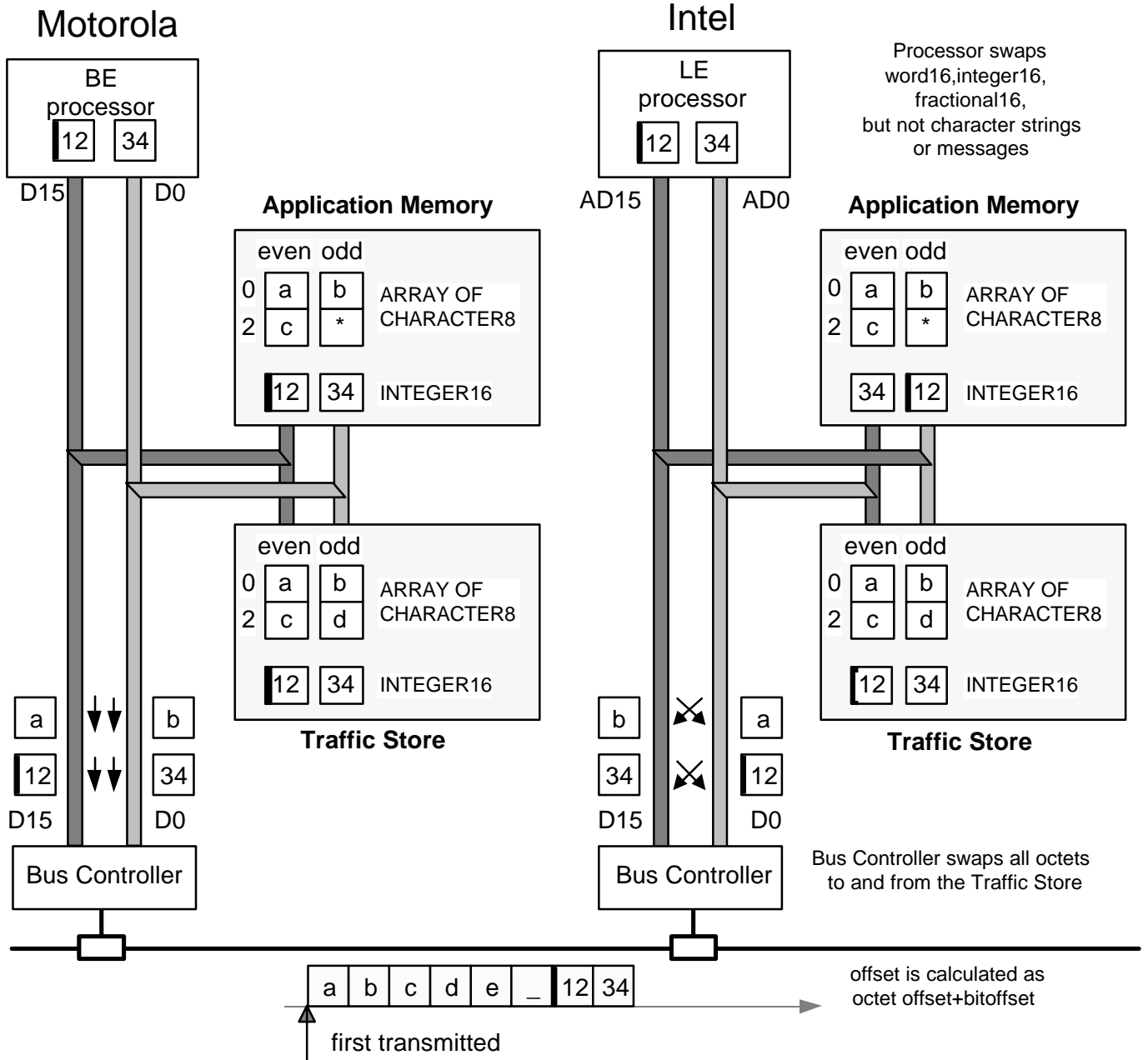


Figure A-4 - Little-Endian and Big-Endian transmission

For big-endian processors, the Bus Controller operates in direct access. The processor copies octets without swapping them since the data format is the same in the Traffic Store and in the Application Memory.

For little-endian processors, the Bus Controller swaps all bytes since its byte lanes are connected to the opposite memory lanes. Little-endian processors copy strings of

characters directly, but swap bytes when accessing 16-bit numerical values, or invert the byte sequence when accessing 32-bit numerical values.

Optimisation for Little-Endian processors

The swapping of 16-bit numerical values penalises the little-endian processors. If the Bus Controller could know that it is transmitting numerical values, it could cease to swap bytes and the processor would not need to swap them either.

Some Bus Controllers allow to swap or not depending on the port address, for instance, to swap bytes in ports containing octet strings, but not in ports containing 16-bit units, as shown in Figure A-5.

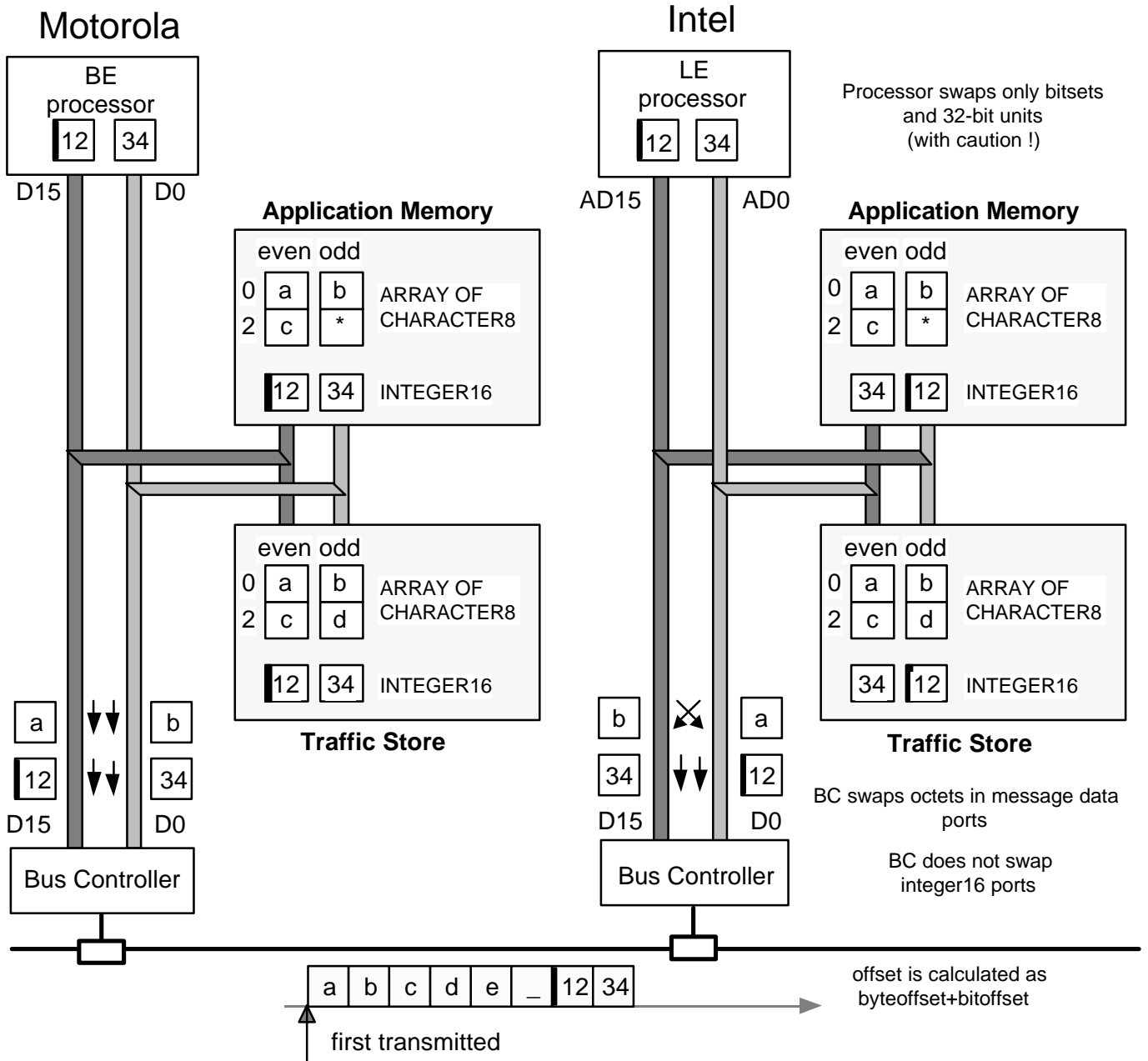


Figure A-5 - Optimised access for Little-Endian processors.

However, this scheme only works if the ports are homogeneous, i.e. contain an array of 16-bit numerical values. In practice, ports contain a number of different types and swapping indifferently could lead to inconsistencies.

The ability of a Bus Controller to swap bytes depending on the port address should be used with caution, since it avoids swapping by the application when the port contains only 16-bit numerical data (integer, unsigned or fractional number) located at a word boundary (even address). In addition, it breaks the rule that data ordering is the same on the bus and on the Traffic Store and therefore hampers monitoring.

Class 1 device connection

Class 1 devices have no processor, the Bus Controller dictates the data representation.

In direct access, the first transmitted bit (the one with offset 0) is sent to the data line D15, as shown in Figure A-5.

In swapped access, the first transmitted bit (the one with offset 0) is sent to the data line D7, as shown in Figure A-5 for the Little-Endian processor. Swapped access mode is not recommended.

MVB transmission format

On MVB, frames start with a start delimiter, followed by a number (1,2,4,8,16 or 32) of 16-bit words. After each group of 64 bits or at the end of the frame, an 8-bit checksum is inserted, as is shown in Figure A-6.

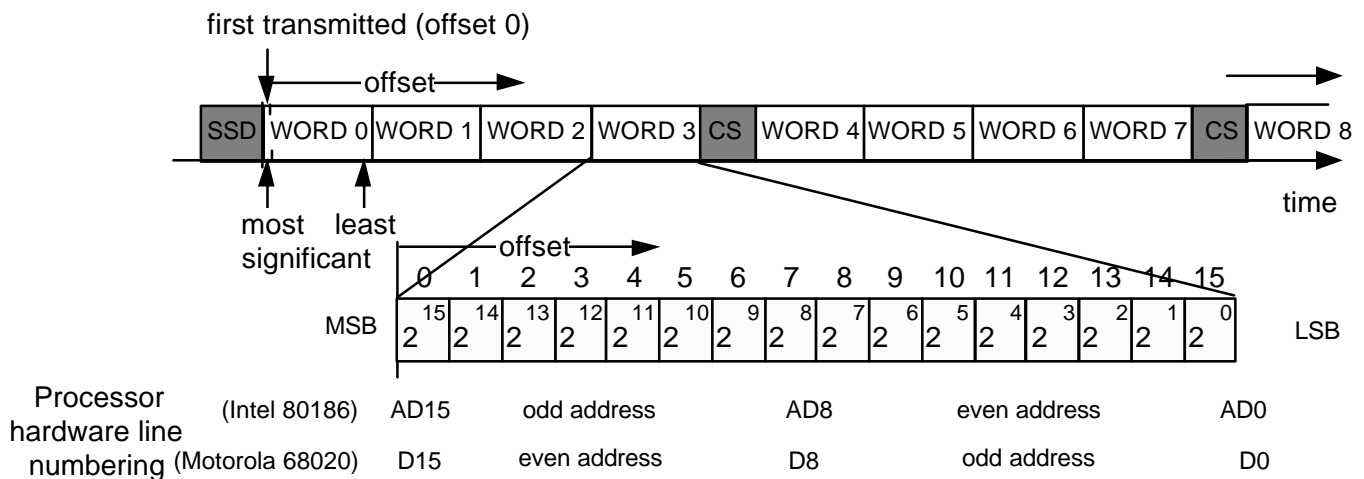


Figure A-6 - MVB data frame

Words are numbered ignoring delimiters and checksum, from the first word, which has number 0, to the last word.

Within a word, treated as an unsigned 16-bit integer, the most significant bit is transmitted first. It has bit offset 0 within that word.

The transmitted units can also be treated as octets, and in that case the octets are numbered starting from 0.

Within an octet, treated as an unsigned integer, the most significant bit is transmitted first. It has bit offset 0 within that octet.

Bits are named (not located) by assuming that a 16-bit integer would be transmitted in the word and by giving each bit a number expressing the power of 2 of that bit.

This naming of bits follows the naming convention of the bus lines leading to the Bus Controller chip. It is not identical with the bit offset, as is shown in Figure A-6.

Note - Different Bus Controllers could name the bits and lines differently.

On the MVB, a bit is defined by its offset, which is counted from the first transmitted bit.

The offset is the concatenation of the octet offset and of the bit offset within an octet.

WTB transmission format

The WTB uses standard HDLC controllers coupled with a Manchester encoder. Frames consist of a preamble, an opening flag, a sequence of octets, a 16-bit checksum and a closing flag. Since flags are legal bit sequences which could appear in the body of the frame, the Bus Controller inserts bits into the frame to avoid flags to appear and removes them at the destination. By some particularity of the HDLC transmission format (shared by most UART protocols), the bits within an octet are transmitted starting with bit 0, which would be the least significant bit of an octet, as Figure A-7 shows.

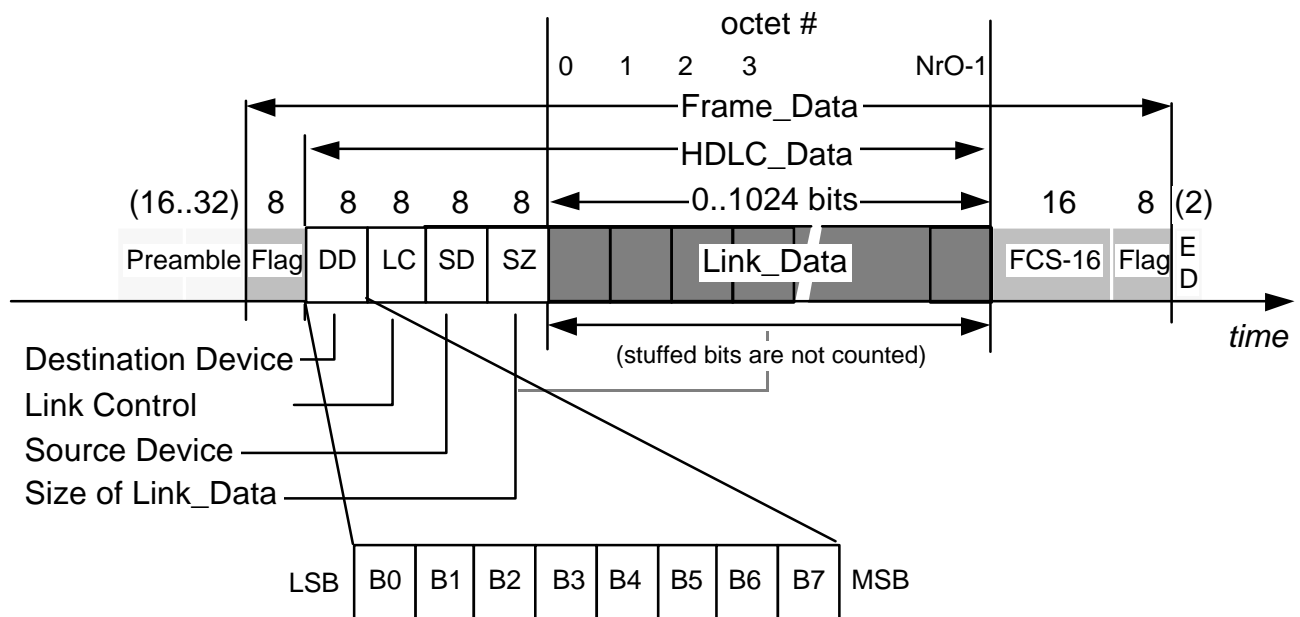


Figure A-7 - WTB data frame

The first transmitted bit within an octet is bit B0, the one carried by the D0 line of the controller chip. This is an internal convention of HDLC, not relevant to the programmer, which should consider transmission of a sequence of octets.

In the Traffic Store, only a sequence of octets appear, in the same order as on the bus.

HDLC controllers do not support swapped access. The octets in the Traffic Store reflect the octets transmitted over the bus and they have the same format. A little-endian processor must swap octets in all 16-bit data structures.

Variable identification in TCN

PV_Name definition

Each Process_Variable is identified within a device by a unique identifier, called its PV_Name, consisting of the following elements:

- Traffic_Store_Id;
- Port_Address;
- Var_Offset;
- Var_Size;
- Var_Type;
- Chk_Offset.

Note - the PV_Name points to the value and to its associated check variable (and possibly to a variable descriptor). Since variables are not allowed to overlap each other, they are uniquely defined by their dataset and offset. The Var_Size, Var_Type and Chk_Offset are included in the PV_Name to speed up operations.

Traffic_Store Identifier

The Traffic_Store_Id identifies one of the 16 Traffic_Stores within a device.

Port_Address

The Port_Address identifies one of 4096 Ports within a Traffic_Store.

Var_Offset

The Var_Offset expresses the bit offset of the variable with respect to the beginning of the dataset in the port.

Var_Type

The Var_Type expresses the type of the variable, according to Table 2.

Var_Size

The Var_Size expresses the size in octets of the variable. For variables of less than one octet, the size is given by the type.

Chk_Offset

The Chk_Offset expresses the bit offset of the check variable (an ANTIVALENT2) associated with this variable (if no check variable is used, the Chk_Offset is set to the last bit position in the dataset, an illegal position since it is a 2-bit data type).

Encoding of the PV_NAME

The PV_Name has been optimised to speed up access to the variables, It should not be used at the application level, but be generated automatically by the configuration tools.

There may exist several ways to represent the PV_Name (for instance depending on access type). The preferred encoding of PV_Name for individual access is shown in Figure A-8:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|------------------|---|---|---|--------------|-----------|---|---|---|---|----|----------------|----|----|----|----|
| 0 | Traffic_Store_Id | | | | Port_Address | | | | | | | | | | | |
| 2 | Var_Size | | | | | Var_Octet | | | | | | Var_Bit_Number | | | | |
| 4 | Var_Type | | | | | Chk_Octet | | | | | | Chk_Bit_Number | | | | |

Figure A-8 - PV_Name encoding

The structure of a PV_Name is expressed as:

```
Pv_Name ::= RECORD
{
  bus_id           UNSIGNED4,  -- 0..15
  port_id         UNSIGNED12, -- 0..4095
  var_size        UNSIGNED6,  -- in bits for primitive types
  var_octet_offset UNSIGNED7, -- in octets from first = 0
  var_bit_number  UNSIGNED3,  -- counted from msb of octet
  var_type        UNSIGNED6,  -- TCN code
  chk_octet_offset UNSIGNED7, -- in octets from first = 0
  chk_bit_number  UNSIGNED3   -- counted from msb of octet
}
```

To speed up programming, Var_Offset and Chk_Offset consist each of two fields, Var_Octet and Var_Bit_Number. The Var_Octet is the octet offset with respect to the beginning of the dataset, the first transmitted or stored octet being octet 0.

The Var_Bit_Number is the bit number, i.e. the number of bits the variable must be shifted right so that the variable is right-justified in the character. It is not identical with the bit offset within the octet.

Example:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

Figure A-9 - PV_Name as memory dump

The Process_Variable identified by the PV_Name illustrated in Figure A-9:

- is located in Traffic_Store 3, at Port_Address '1BA'H (= 442), at bit offset '0F8'H (= 248) bits;
- it is of type INTEGER8 (Var_Size = 0 x WORD16, Var_Type = 6);
- its associated 2-bit Check_Variable is located in the same Dataset at octet offset 0, bit number 4, i.e. bit offset 2.

Caution - Var_Bit_Number differs from bit offset for data structures smaller than 8 bits.

When used in cluster or set access, the PV_Name is encoded differently, with an octet for each the type and the size, the octet offset and the bit number. In clusters, the Chk_Variable appears as a normal ANTIVALENT2, it is not included for each variable explicitly since the same Chk_Variable can protect several Variables.

Example of PV_Name and TCN data types

Table 5 shows the memory dump of different variables which all have Var_Offset = 0.

Table 6 shows different variables with Var_Offset = 12.

Table 5 - Data representation of variables with Var_Offset =0

0 is first bit in Traffic Store (first transmitted on MVB)

| offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |
|-------------|----------|----------|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|---------------------------|
| BOOLEAN1 | B | | | | | | | | | | | | | | | | Var_Bit_Number = 7 |
| ANTIVALENT2 | B | BN | | | | | | | | | | | | | | | Var_Bit_Number = 6 |
| BCD4 | 2^3 | 2^2 | 2^1 | 2^0 | | | | | | | | | | | | | Var_Bit_Number = 4 |
| ENUM4 | 2^3 | 2^2 | 2^1 | 2^0 | | | | | | | | | | | | | Var_Bit_Number = 4 |
| BITSET8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | | | | | | | | | Var_Bit_Number = 0 |
| UNSIGNED8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | | | | | | | | | (holds for all following) |
| ENUM8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | | | | | | | | | |
| INTEGER8 | sign | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | | | | | | | | | |
| BITSET16 | b15 | b14 | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | |
| UNSIGNED16 | 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | |
| INTEGER16 | sign | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | |
| BITSET32 | b31 | b30 | b29 | b28 | b27 | b26 | b25 | b24 | b23 | b22 | b21 | b20 | b19 | b18 | b17 | b16 | |
| | b15 | b14 | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | |
| UNSIGNED32 | 2^{31} | 2^{30} | 2^{29} | 2^{28} | 2^{27} | 2^{26} | 2^{25} | 2^{24} | 2^{23} | 2^{22} | 2^{21} | 2^{20} | 2^{19} | 2^{18} | 2^{17} | 2^{16} | |
| | 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | |
| INTEGER32 | sign | 2^{30} | 2^{29} | 2^{28} | 2^{27} | 2^{26} | 2^{25} | 2^{24} | 2^{23} | 2^{22} | 2^{21} | 2^{20} | 2^{19} | 2^{18} | 2^{17} | 2^{16} | |
| | 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | |
| REAL32 | sign | 2^7 | biased exponent | | | | | 2^0 | 2^{-1} | mantissa | | | | | | 2^{-7} | |
| | 2^{-8} | mantissa | | | | | | | | | | | | | | | |
| TIMEDATE48 | 2^{47} | 2^{46} | 2^{45} | 2^{44} | 2^{43} | 2^{42} | 2^{41} | 2^{40} | 2^{39} | 2^{38} | 2^{37} | 2^{36} | 2^{35} | 2^{34} | 2^{33} | 2^{32} | |
| | 2^{31} | 2^{30} | 2^{29} | 2^{28} | 2^{27} | 2^{26} | 2^{25} | 2^{24} | 2^{23} | 2^{22} | 2^{21} | 2^{20} | 2^{19} | 2^{18} | 2^{17} | 2^{16} | |
| | 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | |

Table 6 - Data representation of variables with Var_Offset =12

| | 1st | | | | | | | | | | | | | | | |
|-------------|-----|---|---|---|---|---|---|---|---|---|----|----|-------|-------|-------|-------|
| offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| BOOLEAN1 | | | | | | | | | | | | | B | | | |
| ANTIVALENT2 | | | | | | | | | | | | | B | BN | | |
| BCD4 | | | | | | | | | | | | | 2^3 | 2^2 | 2^1 | 2^0 |

In Table 4, the BOOLEAN1 has Var_Bit_Number = 3, the ANTIVALENT2 has Var_Bit_Number = 2 and the BCD4 has Var_Bit_Number = 0.

Appendix B: format mapping in LON

LONWorks' Standard Network Variable Types are defined in document [SNVT]. The following Table B-1 is an extract from [SNVT] for informative purposes. With respect to the original Table, the units have all being converted to metric form, the range is expressed in % and non-metric units have been dropped.

In addition, the ROSIN Type on which they are based is mentioned.

Table B-1 - LONWorks' Standard Network Data Types

| Measurement | Name | Range (Resolution) | Based on | ID |
|-----------------------|------------------|--|--------------|-----|
| Angular velocity | SNVT_angle_vel_f | -1,0 x10 ^{^38} .. 1,0 10 ^{^38} rad/s | REAL32 | 50 |
| | SNVT_angle_vel | -3276,8 .. 3276,7 rad/s, (0,1 /s) | BIPOLAR2_16 | 4 |
| Area | SNVT_area | 0,0 .. 13,1068 m ² , (200 mm ²), [100% = 32,768 m ²] | UNIPOLAR2_16 | 110 |
| Character | SNVT_char_ascii | 0..255 | CHARACTER8 | 7 |
| Concentration | SNVT_ppm | 0,0 .. 65'535,0 part per million [100% = 16'384 ppm] | UNIPOLAR2_16 | 29 |
| | SNVT_ppm_f | 0,0 .. 1,0 x 10 ^{^38} ppm | REAL32 (>0) | 58 |
| Count, event | SNVT_count | 0 .. 65'535 | UNSIGNED16 | 8 |
| Count, incremental | SNVT_count_inc | -32'768 .. 32'767 | INTEGER16 | 9 |
| | SNVT_count_inc_f | -1,0 x10 ^{^38} .. 1,0 x10 ^{^38} | REAL32 | 52 |
| Current | SNVT_amp_f | A | REAL32 | 48 |
| | SNVT_amp | -3276,8 .. 3276,7 A(0.1 A) [100% = 1638,4 A] | BIPOLAR2_16 | 1 |
| | SNVT_amp_mil | -3276,8 .. 3276,7 mA (0.1 mA) [100% = 1638,4 mA] | BIPOLAR2_16 | 2 |
| Density | SNVT_density | 0 .. 32767,5 kg/m ³ (0,5 kg/m ³) [100% = 8192 kg/m ³] | UNIPOLAR2_16 | 100 |
| | SNVT_density_f | kg/m ³ | REAL32 | 101 |
| Flow | SNVT_flow | 0 .. 65,535 dm ³ /s (1,0 dm ³ /s) [100% = 16'384 dm ³ /s] | UNIPOLAR2_16 | 15 |
| | SNVT_flow_f | dm ³ /s | REAL32 | 53 |
| | SNVT_flow_mil | 0 .. 65,535 milliliter/s | UNIPOLAR2_16 | 16 |

| | | | | |
|-----------|------------------|--|--------------|----|
| | | (1 mm ³ /s) [100% = 16'384 mm ³ /s] | | |
| Frequency | SNVT_freq_f | hertz | REAL32 | 75 |
| | SNVT_freq_hz | 0 .. 6553,5 Hz (0,1 Hz) [100% = 16'384 Hz] | UNIPOLAR2_16 | 76 |
| | SNVT_freq_kilohz | 0 .. 6553,5 kHz (0,1 kHz) [100% = 16'384 kHz] | UNIPOLAR2_16 | 77 |
| | SNVT_freq_milhz | 0 .. 6,5535 Hz (0.0001 Hz) [100% = 16'384 mHz] | UNIPOLAR2_16 | 78 |

| Measurement | Name | Range (Resolution) | Based on | ID |
|-------------------|------------------|---|--------------|-----|
| Gain | SNVT_muldiv | to be completed | | 91 |
| Grammage | SNVT_grammage | 0 .. 6,553.5 gsm (0.1 gsm) | UNIPOLAR2_16 | 71 |
| | SNVT_grammage_f | -1,0 x10 ^{^38} .. 1,0 x10 ^{^38} gsm | REAL32 | 72 |
| HVAC mode | SNVT_hvac_mode | see Enum Lists below | | 108 |
| HVAC override | SNVT_hvac_overid | to be completed | | 111 |
| Humidity | SNVT_lev_percent | See Level, percent below | | 81 |
| Illumination | SNVT_lux | 0 .. 6'535,0 lux (1,0 lux) [100% = 16'384,0 lux] | UNIPOLAR2_16 | 79 |
| Length | SNVT_length | 0 .. 6'553,5 metre, (0,1 m) [100% = 1'638,4 m] | UNIPOLAR2_16 | 17 |
| | SNVT_length_f | metre | REAL32 | 54 |
| | SNVT_length_kilo | 0 .. 6'533,5 km (0.1 km) [100% = 1'638,4 km] | UNIPOLAR2_16 | 18 |
| | SNVT_length_micr | 0 .. 6'553,5 µm (0,1 µm) [100% = 1'638,4 µm] | UNIPOLAR2_16 | 19 |
| | SNVT_length_mil | 0 .. 6'533,5 mm (0,1 mm) [100% = 1'638,4 mm] | UNIPOLAR2_16 | 20 |
| Level, continuous | SNVT_lev_cont | 0 .. 100 % (0.5%) [caution: 100% full range !] | UNIPOLAR2_16 | 21 |
| | SNVT_lev_cont_f | 0 .. 100 % | REAL32 | 55 |
| Level, discrete | SNVT_lev_disc | see Enum Lists below | | 22 |
| Level, percent | SNVT_lev_percent | -163,84% .. 163,83% (0.005% or 50 ppm) | BIPOLAR2_16 | 81 |
| Magnetic cards | SNVT_magcard | to be completed | | 86 |
| | SNVT_ISO_7811 | Use SNVT_magcard instead | UNSIGNED16 | 80 |
| Mass | SNVT_mass | 0 .. 6'553,5 gram (0,1 g) | UNIPOLAR2_16 | 23 |
| | SNVT_mass_f | gram | REAL32 | 56 |

| | | | | |
|----------------|-----------------|--------------------------------------|--------------|----|
| | SNVT_mass_kilo | 0 .. 6'553,5 kg (0,1 kg) | UNIPOLAR2_16 | 24 |
| | SNVT_mass_mega | 0 .. 6'553,5 tonne (0,1 tonne) | UNIPOLAR2_16 | 25 |
| | SNVT_mass_mil | 0 .. 6'553,5 milligram (0,1 mg) | UNIPOLAR2_16 | 26 |
| Multiplier | SNVT_multiplier | 0 .. 32,7675 (0,0005) | | 82 |
| Phase/rotation | SNVT_angle | 0 .. 65,535 radian (0,001 radian) | UNIPOLAR2_16 | 3 |
| | SNVT_angle_f | radians | REAL32 | 49 |

| Measurement | Name | Range (Resolution) | Based on | ID |
|---------------------|-----------------|---|--------------|-----|
| Power | SNVT_power | 0 .. 6'553,5 watt (0,1 W) | UNIPOLAR2_16 | 27 |
| | SNVT_power_f | watt | | 57 |
| | SNVT_power_kilo | 0 .. 6'553,5 kW (0,1 kW) | UNIPOLAR2_16 | 28 |
| Power factor | SNVT_pwr_fact | -1,0 .. 1,0 (0,00005) | BIPOLAR2_16 | 98 |
| | SNVT_pwr_fact_f | -1,0 .. 1,0 | BIPOLAR2_16 | 99 |
| Preset | SNVT_preset | to be completed | | 94 |
| Pressure - gauge | SNVT_press | -3,276,8 .. 3'276,7 kilopascal (0,1 kPa) | BIPOLAR2_16 | 30 |
| Pressure - absolute | SNVT_press_f | pascal | REAL32 | 59 |
| Pressure - gauge | SNVT_press_p | -32'768 .. 32'766 pascal (1 Pa) | BIPOLAR2_16 | 113 |
| Resistance | SNVT_res | 0 .. 6'553,5 ohm (0,1 W) | UNIPOLAR2_16 | 31 |
| | SNVT_res_f | Ohm | REAL32 | 60 |
| | SNVT_res_kilo | 0 .. 6'553,5 k (0,1 k) | UNIPOLAR2_16 | 32 |
| Sound level | SNVT_sound_db | -327,68 .. 327,67 decibel (0,01 dB) | BIPOLAR2_16 | 33 |
| | SNVT_sound_db_f | decibel | REAL32 | 61 |
| Speed | SNVT_speed | 0 .. 6'553,5 metre/s (0,1 m/s) | UNIPOLAR2_16 | 34 |
| | SNVT_speed_f | m/s | REAL32 | 62 |
| | SNVT_speed_mil | 0 .. 65,535 m/s (0,001 m/s) | UNIPOLAR2_16 | 35 |
| Temperature | SNVT_temp | -274 .. 6'279,5 °C (0,1 °C) | BIPOLAR2_16 | 39 |
| | SNVT_temp_p | -273,17 .. +327,66 °C (0,01 °C) | BIPOLAR2_16 | 105 |
| | SNVT_temp_f | °C | REAL32 | 63 |
| Volume | SNVT_vol | 0 .. 6'553,5 litre (0,1 l) | UNIPOLAR2_16 | 41 |
| | SNVT_vol_f | litre (dm ³) | REAL32 | 65 |
| | SNVT_vol_kilo | 0 .. 6'553,5 m ³ (0,1 m ³) | UNIPOLAR2_16 | 42 |
| | SNVT_vol_mil | 0 .. 6'553,5 millilitre (0,1 ml) | UNIPOLAR2_16 | 43 |
| Voltage | SNVT_volt | -3'276,8 .. 3'276,7 volt (0,1 V) | BIPOLAR2_16 | 44 |

| | | | |
|----------------|---|-------------|----|
| SNVT_volt_f | volt | REAL32 | 66 |
| SNVT_volt_kilo | -3'276,8 .. 3'276,7 kilovolt (0,1 kV) | BIPOLAR2_16 | 46 |
| SNVT_volt_mil | -3'276,8 .. 3'276,7 millivolt (0,1 mV) | BIPOLAR2_16 | 47 |

| Measurement | Name | Range (Resolution) | Based on | ID |
|-------------------|------------------|-----------------------------|--------------|-----|
| Time of day | SNVT_date_time | Use SNVT_timestamp instead | | 12 |
| Time - elapsed | SNVT_time_f | second | REAL32 | 64 |
| | SNVT_elapsed_tm | to be completed | | 87 |
| | SNVT_time_sec | 0,0 .. 6553,4 s (0,1 s) | UNIPOLAR2_16 | 107 |
| | SNVT_time_passed | Use SNVT_elapsed_tm instead | | 40 |
| Time stamp | SNVT_time_stamp | to be completed | | 84 |
| Translation table | SNVT_trans_table | to be completed | | 96 |

1 SNVT_temp represents tenths of a degree Celsius above -274 o C. To get SNVT_temp units define a constant: C_to_K equal to 2740 which is added to temperature expressed in tenths of degrees C.

2 To be used for heating, ventilation and air conditioning applications.

The type color is defined as:

```
SNVT_color ::= RECORD {
    l_star          UNIPOLAR2_16,
    a_star          BIPOLAR2_16,
    b_star          BIPOLAR2_16
}
```

Example of SNVT Object description

Neuron-C definition

Example of SNVT_alarm (taken form [SNVT])

```
typedef struct {
    char location [ LOCATION_LEN ];
    unsigned long object_id;
    alarm_type_t alarm_type;
    priority_level_t priority_level;
    unsigned long index_to_SNVT;
    unsigned value [ 4 ];
    unsigned long year;
    unsigned short month;
    unsigned short day;
    unsigned short hour;
    unsigned short minute;
}
```

```

unsigned short second;
unsigned long millisecond;
unsigned alarm_limit [ 4 ];
} SNVT_alarm;

```

Same SNVT examples in ROSIN Explicit Notation

```

SNVT_alarm ::= RECORD
{
  location          ARRAY [ LOCATION_LEN ] OF CHARACTER8; -- constant size
  object_id         UNSIGNED16;
  alarm_type        Alarm_Reason;      -- this is an ENUM8, see below
  priority_level    Priority_Level;
  index_to_SNVT     UNSIGNED16;
  value             ARRAY [4] OF UNSIGNED8;
  year              UNSIGNED16;
  month             UNSIGNED8;
  day               UNSIGNED8;
  hour              UNSIGNED8;
  minute           UNSIGNED8;
  second           UNSIGNED8;
  millisecond       UNSIGNED16;
  alarm_limit       ARRAY [4] OF UNSIGNED8;
}

```

```

Alarm_Reason ::= ENUM8
{
  AL_NO_CONDITION, (0)      -- No alarm condition present
  AL_ALM_CONDITION, (1)    -- Unspecified alarm condition present
  AL_TOT_SVC_ALM_1, (2)    -- Total/service interval alarm 1
  AL_TOT_SVC_ALM_2, (3)    -- Total/service interval alarm 2
  AL_TOT_SVC_ALM_3, (4)    -- Total/service interval alarm 3
  AL_LOW_LMT_CLR_1, (5)    -- Alarm low limit alarm clear 1
  AL_LOW_LMT_CLR_2, (6)    -- Alarm low limit alarm clear 2
  AL_HIGH_LMT_CLR_1, (7)   -- Alarm high limit alarm clear 1
  AL_HIGH_LMT_CLR_2, (8)   -- Alarm high limit alarm clear 2
  AL_LOW_LMT_ALM_1, (9)    -- Alarm low limit alarm 1
  AL_LOW_LMT_ALM_2, (10)   -- Alarm low limit alarm 2
  AL_HIGH_LMT_ALM_1, (11)  -- Alarm high limit alarm 1
  AL_HIGH_LMT_ALM_2 (12)   -- Alarm high limit alarm 2
};

```

Reformatting LonTalk Network Management messages

The following shows how the TCN Encoding Rules are applied to the LonTalk Network Management messages. Only one example of message, taken from [LonTalk] is shown.

General structure of a LonTalk PDU

```

LON_APDU ::= RECORD
{
  apdu_var_msg      ENUM1
  {
    VARIABLE        (0)          -- network variable
    MESSAGE          (1)          -- application or network management
  },
  body              ONE_OF [apdu_var_msg]
  {
    [VARIABLE]      Lnv_Pdu      -- see type below
    [MESSAGE]       Msg_Pdu     -- see type below
  }
}

```

General structure of a LonTalk Network Variable PDU

```

Lnv_Pdu ::= RECORD
{
  direction         ENUM1        -- this is the second bit of the first byte
  {
    INCOMING        (0)          -- incoming network variable
    OUTGOING        (1)          -- outgoing network variable
  },
  nv_selector       UNSIGNED14,
  data              ARRAY [INDIRECT nv_selector] OF WORD8
                  -- size is implicit from the selector
}

```

General structure of a LonTalk Message PDU

```

UpperLowerCount ::= RECORD      -- auxiliary variable
Msg_Pdu ::= ONE_OF
{
  Msg_Request_Pdu,              -- there is no way to distinguish them
  Msg_Response_Pdu             -- except by looking at the context
}

Msg_Request_Pdu ::= ONE_OF [ENUM1] -- begins with '0'B (apl | nm)
{
  [0] Apl_Pdu,                  -- begins with '00'B (appl | nmd resp)
  [1] ONE_OF [ENUM1]           -- begins with '01'B (nm_req | nd_req)
  {
    [1] Lnm_Request_Pdu        -- begins with '011'B
    [0] ONE_OF [ENUM1]         -- begins with '010'B (nd | foreign) {
      [1] Lnd_Request_Pdu,     -- begins with '0101'B
      [0] Foreign_Pdu          -- begins with '0100'B
    }
  }
}

```

Example: LonTalk Network Management commands

In the LonTalk handbook, the command is listed in the body although it is a selector. The same command is returned in the response. Since the diagnostic messages have the same format as the network management commands, they are included in this enumeration:

```

LNM_Command ::= ENUM5
{
  query_id          (1),          -- (0 unused)
  response_to_query (2),
  update_domain    (3),
  leave_domain     (4),
  update_key       (5),
  update_addr      (6),
  query_address    (7),
  query_nv_cnfg    (8),
  update_group_add (9),
  query_domain     (10),
  update_nv_cnfg   (11),
  set_node_mode    (12),
  read_memory      (13),
  write_memory     (14),
  checksum         (15),
  install          (16),
  memory_refresh   (17),
  query_SNVT       (18),
  nv_fetch         (19),
  router_mode      (20),          -- begins router configuration messages
  router_table_clear (21),
  groupsubnet_table_download (22),
  group_forward    (23),
  subnet_forward   (24),
  group_noforward (25),
  subnet_noforward (26),
  groupsubnet_table_report (27),
  ROUTER_STATUS    (28),
  ESCAPE           (29),          -- code 0x7D somehow embedded
  Router_half_ESCAPE (30),      -- code 31 unused
}

```

```

LNM_Request_PDU ::= RECORD
{
  nm_command      LNM_Command,
  ONE_OF [nm_command] -- add 96 for first byte
  {
    [query_id]      NM_Query_Id_Request,
    [response_to_query] NM_ResponseToQuery,
    [update_domain] NM_UpdateDomain,
  }
}

```

```

    [leave_domain]  NM_LeaveDomain,
    ...
  }
}

LND_Command ::= ENUM4
{
  query_status      (1), -- add 80 for first byte
  proxy             (2),
  clear_status      (3),
  query_xcvr_status (4)
                                -- codes from 5 to 15 reserved for nd
}

LND_request_PDU ::= RECORD
{
  nd_command        LND_Command
  ONE_OF [nd_command]
  {
    [QUERY_STATUS]  ND_Query_Status_Request
    [PROXY]         ND_Proxy_Command,
    [CLEAR_STATUS]  ND_Clear_Status_Command,
    [QUERY_XCVR_STATUS] ND_Query_Xcrr_Status_Request
    ...
  }
}

```

Example: Query_ID Request/Response

```

Address_Mode ::= ENUM8
{
  ABSOLUTE           (0),
  READ_ONLY_RELATIVE (1),
  CONFIG_RELATIVE    (2)
}

NM_Query_Id_Request ::= RECORD -- cf. B.1.1 in Motorola LonWorks Technology
{
  selector          ENUM8      -- could also be expressed as an BITSET8
  {
    UNCONFIGURED     (0)      --
    SELECTED         (1)
    SELECTED_UNCNFG  (2)      --
  },
  mode              Address_Mode -- 8 bits
  offset            WORD16     -- msb first
  count             UNSIGNED8,  -- nr elements for the open array
  data              OPEN ARRAY [count] OF WORD8
}

```

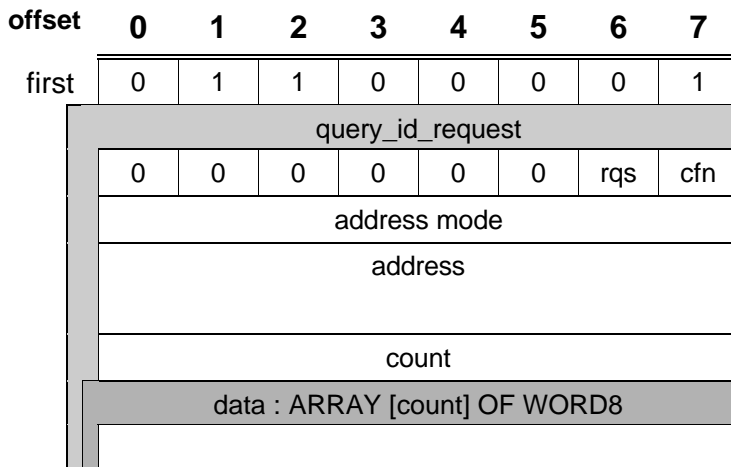
```

}

NM_Query_Id_Response ::= RECORD
{
  neuron_ID          ARRAY [NEURON_ID_LENGTH] OF WORD8,    -- = 6
  id_string          ARRAY [ID_STR_LEN] OF WORD8            -- = 8
}

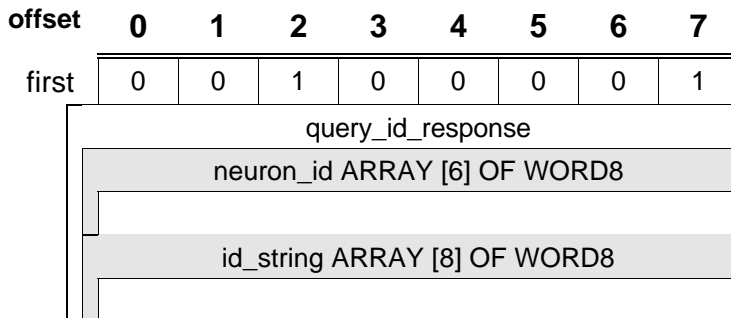
```

Graphical representation of Query_ID Request/Response



NM message, Query_ID = 97
 this field is a title, it is not transmitted
 (selector)

this field is a title, it is not transmitted
 this field is repeated "count" times



(assumes success)
 this field is a title, it is not transmitted
 this field is a title, it is not transmitted
 this field is repeated 6 times
 this field is a title, it is not transmitted
 this field is repeated 8 times